# Oracle® Warehouse Builder

Data Modeling, ETL, and Data Quality Guide

11*g* Release 2 (11.2)

**E10935-03**

August 2010

ORACLE®

Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide, 11*g* Release 2 (11.2)

E10935-03

Copyright © 2000, 2010, Oracle and/or its affiliates. All rights reserved.

Contributing Authors: Padmaja Potineni, Vishwanath Sreeraman

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this software or related documentation is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications which may create a risk of personal injury. If you use this software in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of this software. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software in dangerous applications.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

This software and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

# Contents

**Preface** ..... xxxiii

Audience ..... xxxiii
Documentation Accessibility ..... xxxiii
Conventions ..... xxxiv
Getting Help ..... xxxiv
Related Documents ..... xxxv

## 1 Designing Source and Target Schemas

**Designing Target Schemas** ..... 1-1
Creating Target Modules ..... 1-1
Designing Relational Target Schemas ..... 1-2
Designing Dimensional Target Schemas ..... 1-3
**Configuring Data Objects** ..... 1-4
**Validating Data Objects** ..... 1-4
Viewing Validation Results ..... 1-5
Editing Invalid Objects ..... 1-6
**Generating Data Objects** ..... 1-6
Viewing Generation Results and Generated Scripts ..... 1-7
Saving Generated Scripts to a File ..... 1-7

## 2 Creating Relational Data Objects

**Overview of Data Objects** ..... 2-1
Supported Data Types ..... 2-3
About Object Class Definition ..... 2-7
About First Class Objects (FCOs) ..... 2-7
About Second Class Objects (SCOs) ..... 2-7
About Third Class and Fourth Class Objects ..... 2-7
Naming Conventions for Data Objects ..... 2-8
Using the Data Viewer to View Data Stored in Data Objects ..... 2-9
About Error Tables ..... 2-9
Defining Error Tables for Data Objects ..... 2-9
Error Table Columns ..... 2-10
**Defining Tables** ..... 2-10
Creating Table Definitions ..... 2-11
Name Tab ..... 2-11

Columns Tab ..... 2-12
Keys Tab ..... 2-13
Indexes Tab ..... 2-13
Partitions Tab ..... 2-13
Attribute Sets Tab ..... 2-13
Data Rules Tab ..... 2-13
Editing Table Definitions ..... 2-13
Renaming Tables ..... 2-14
Adding, Modifying, and Deleting Table Columns ..... 2-14
Adding, Modifying, and Deleting Table Constraints ..... 2-14
Adding, Modifying, and Deleting Attribute Sets ..... 2-14
Reordering Columns in a Table ..... 2-15
**Defining Views** ..... 2-15
Creating View Definitions ..... 2-15
Name Tab ..... 2-16
Columns Tab ..... 2-16
Query Tab ..... 2-16
Keys Tab ..... 2-17
Attribute Sets Tab ..... 2-17
Data Rules Tab ..... 2-17
Editing View Definitions ..... 2-17
Renaming Views ..... 2-17
Adding, Modifying, and Deleting View Columns ..... 2-17
Adding, Modifying, and Deleting View Constraints ..... 2-17
Adding, Modifying, and Deleting Attribute Sets ..... 2-18
**Defining Materialized Views** ..... 2-18
Creating Materialized View Definitions ..... 2-18
Columns Tab ..... 2-19
Query Tab ..... 2-19
Keys Tab ..... 2-19
Indexes Tab ..... 2-19
Partitions Tab ..... 2-19
Attribute Sets Tab ..... 2-19
Data Rules Tab ..... 2-20
Editing Materialized View Definitions ..... 2-20
Renaming Materialized Views ..... 2-20
Adding, Modifying, and Deleting Materialized View Columns ..... 2-20
Adding, Modifying, and Deleting Materialized View Constraints ..... 2-20
Adding, Modifying, and Deleting Attribute Sets ..... 2-20
**Defining Constraints** ..... 2-20
About Constraints ..... 2-20
Creating Constraints ..... 2-21
Defining Primary Key Constraints ..... 2-22
Defining Foreign Key Constraints ..... 2-22
Defining Unique Key Constraints ..... 2-22
Defining Check Constraints ..... 2-23
Editing Constraints ..... 2-23

**Defining Indexes** ... 2-24
Creating Indexes ... 2-24
**Defining Partitions** ... 2-25
Range Partitioning ... 2-26
Example of Range Partitioning ... 2-27
Hash Partitioning ... 2-27
Hash by Quantity Partitioning ... 2-28
List Partitioning ... 2-28
Composite Partitioning ... 2-30
About the Subpartition Template ... 2-31
Creating Custom Subpartitions ... 2-31
Index Partitioning ... 2-32
Index Performance Considerations ... 2-33
Configuring Partitions ... 2-33
**Defining Attribute Sets** ... 2-33
Creating Attribute Sets ... 2-34
Editing Attribute Sets ... 2-34
**Defining Sequences** ... 2-35
About Sequences ... 2-35
Creating Sequence Definitions ... 2-35
Editing Sequence Definitions ... 2-35
Name Tab ... 2-36
Columns Tab ... 2-36
**Defining User-Defined Types** ... 2-36
About Object Types ... 2-36
Defining Object Types ... 2-37
Name Tab ... 2-37
Columns Tab ... 2-37
Editing Object Types ... 2-38
About Varrays ... 2-38
Defining Varrays ... 2-39
Name Tab ... 2-39
Details Tab ... 2-39
Editing Varrays ... 2-40
About Nested Tables ... 2-40
Defining Nested Tables ... 2-40
Name Tab ... 2-40
Details Tab ... 2-41
Editing Nested Tables ... 2-41
**Defining Queues** ... 2-41
Creating Queue Table Definitions ... 2-42
Defining the Payload Type of Queue Tables ... 2-42
Editing Queue Tables ... 2-43
Creating Advanced Queue Definitions ... 2-43
Specifying the Queue Table on which the AQ is Based ... 2-44
Editing Advanced Queue Definitions ... 2-44
Creating Queue Propagations ... 2-45

Selecting a Target Queue for Propagation ..... 2-45
Editing Queue Propagations ..... 2-45
**Configuring Relational Data Objects** ..... 2-46
Configuring Target Modules ..... 2-46
Deployment System Type ..... 2-46
Generation Preferences ..... 2-47
Generation Target Directories ..... 2-47
Identification ..... 2-48
Run Time Directories ..... 2-48
Tablespace Defaults ..... 2-48
Configuring Tables ..... 2-48
Error Table ..... 2-49
Foreign Keys ..... 2-49
Identification ..... 2-49
Parallel ..... 2-50
Performance Parameters ..... 2-50
Partition Parameters ..... 2-50
Storage Space ..... 2-50
Change Data Capture ..... 2-50
Configuring Materialized Views ..... 2-51
Materialized View Parameters ..... 2-51
Materialized View Log Parameters ..... 2-53
Fast Refresh for Materialized Views ..... 2-53
Configuring Views ..... 2-54
Configuring Sequences ..... 2-54
Configuring Advanced Queues ..... 2-54
Configuring Queue Tables ..... 2-55
Configuring Queue Propagations ..... 2-55
**Creating Relational Data Objects in Microsoft SQL Server and IBM DB2 UDB** ..... 2-57
Rules for Naming Objects in IBM DB2 UDB ..... 2-57
Rules for Naming Objects in Microsoft SQL Server ..... 2-58

## 3 Defining Dimensional Objects

**Overview of Dimensional Objects** ..... 3-1
Overview of Dimensions ..... 3-2
Overview of Surrogate Identifiers ..... 3-3
Overview of Slowly Changing Dimensions ..... 3-3
Overview of Defining Type 2 Slowly Changing Dimensions ..... 3-4
Overview of Hierarchy Versioning ..... 3-5
Overview of Defining Type 3 Slowly Changing Dimensions (SCDs) ..... 3-6
Overview of Cubes ..... 3-7
Orphan Management for Dimensional Objects ..... 3-7
Orphan Management While Loading Data Into Dimensional Objects ..... 3-8
Orphan Management While Removing Data From Dimensional Objects ..... 3-8
Error Tables ..... 3-8
**Overview of Implementing Dimensional Objects** ..... 3-9
Relational Implementation of Dimensional Objects ..... 3-9

Binding ..... 3-10
Auto Binding ..... 3-10
Manual Binding ..... 3-11
Unbinding ..... 3-12
ROLAP Implementation of Dimensional Objects ..... 3-12
MOLAP Implementation of Dimensional Objects ..... 3-13
Analytic Workspace ..... 3-13
Deployment Options for Dimensional Objects ..... 3-13
**Creating Dimensions** ..... 3-14
Dimension Example ..... 3-14
Creating Dimensions Using the Create Dimension Wizard ..... 3-15
Name and Description Page ..... 3-15
Storage Type Page ..... 3-15
Dimension Attributes Page ..... 3-17
Levels Page ..... 3-18
Level Attributes Page ..... 3-18
Slowly Changing Dimension Page ..... 3-19
Pre Create Settings Page ..... 3-19
Dimension Creation Progress Page ..... 3-20
Summary Page ..... 3-20
Defaults Used By the Create Dimension Wizard ..... 3-20
Storage ..... 3-20
Dimension Attributes ..... 3-21
Hierarchies ..... 3-21
Level Attributes ..... 3-21
Slowly Changing Dimensions ..... 3-21
Orphan Management Policy ..... 3-22
Implementation Objects ..... 3-22
Creating Dimensions Using the Dimension Editor ..... 3-22
Name Tab ..... 3-23
Storage Tab ..... 3-24
Attributes Tab ..... 3-25
Levels Tab ..... 3-25
Hierarchies Tab ..... 3-26
SCD Tab ..... 3-27
Orphan Tab ..... 3-28
Specifying the Default Parent for Orphan Rows ..... 3-28
Physical Bindings Tab ..... 3-29
Limitations of Deploying Dimensions to the OLAP Catalog ..... 3-30
Using Control Rows ..... 3-30
Determining the Number of Rows in a Dimension ..... 3-31
**Creating Slowly Changing Dimensions** ..... 3-31
Creating Type 2 Slowly Changing Dimensions Using the Dimension Editor ..... 3-32
Type 2 Slowly Changing Dimension Dialog Box ..... 3-32
Updating Type 2 Slowly Changing Dimensions ..... 3-33
Creating Type 3 Slowly Changing Dimensions Using the Dimension Editor ..... 3-35
Type 3 Slowly Changing Dimension Dialog Box ..... 3-36

**Editing Dimension Definitions** ... 3-36
**Configuring Dimensions** ... 3-37
Specifying How Dimensions are Deployed ... 3-38
**Creating Cubes** ... 3-39
About Calculated Measures in Cubes ... 3-39
Standard Calculation ... 3-39
Custom Expression ... 3-41
Cube Example ... 3-42
Using the Create Cube Wizard to Create Cubes ... 3-42
Name and Description Page ... 3-42
Storage Type Page ... 3-42
Dimensions Page ... 3-44
Measures Page ... 3-44
Summary Page ... 3-45
Defaults Used by the Create Cube Wizard ... 3-45
Using the Cube Editor to Create Cubes ... 3-45
Name Tab ... 3-46
Storage Tab ... 3-46
Dimensions Tab ... 3-47
Advanced Dialog Box ... 3-48
Measures Tab ... 3-49
Calculated Measure Wizard ... 3-50
Define Calculated Measure Details ... 3-50
Reviewing the Summary Information ... 3-50
Aggregation Tab ... 3-50
Precomputing ROLAP Cubes ... 3-50
Orphan Tab ... 3-51
Physical Bindings Tab ... 3-51
Cubes Stored in Analytic Workspaces ... 3-52
Ragged Cube Data ... 3-52
Defining Aggregations ... 3-52
Auto Solving MOLAP Cubes ... 3-52
Solving Cube Measures ... 3-53
Solving Cubes Independent of Loading ... 3-53
Parallel Solving of Cubes ... 3-54
Output of a MOLAP Cube Mapping ... 3-54
Restriction on Referencing a Non-Leaf Level ... 3-54
**Editing Cube Definitions** ... 3-55
**Configuring Cubes** ... 3-55
Specifying How Cubes are Deployed ... 3-56
**Creating Time Dimensions** ... 3-57
Creating a Time Dimension Using the Time Dimension Wizard ... 3-57
Name and Description Page ... 3-58
Storage Page ... 3-58
Data Generation Page ... 3-58
Levels Page (Calendar Time Dimension Only) ... 3-59
Levels Page (Fiscal Time Dimension Only) ... 3-59

Pre Create Settings Page ..... 3-60
Time Dimension Progress Page ..... 3-60
Summary Page ..... 3-60
Defaults Used by the Time Dimension Wizard ..... 3-60
Editing Time Dimension Definitions ..... 3-61
Name Tab ..... 3-61
Storage Tab ..... 3-62
Attributes Tab ..... 3-62
Levels Tab ..... 3-63
Hierarchies Tab ..... 3-63
Modifying the Implementation of Time Dimensions ..... 3-63
**Populating Time Dimensions** ..... 3-64
Dynamically Populating Time Dimensions ..... 3-64
Overlapping Data Populations ..... 3-65

## 4 Overview of Transforming Data

**About Data Transformation in Oracle Warehouse Builder** ..... 4-1
**About Mappings** ..... 4-2
**About Operators** ..... 4-2
Types of Operators ..... 4-3
Source and Target Operators ..... 4-3
Transformation Operators ..... 4-4
Pre/Post Processing Operators ..... 4-5
Pluggable Mapping Operators ..... 4-5
Real-time Data Warehousing Operators ..... 4-6
**About Transformations** ..... 4-6
Types of Transformations ..... 4-6
Predefined Transformations ..... 4-6
Custom Transformations ..... 4-7
**About Transformation Libraries** ..... 4-8
Types of Transformation Libraries ..... 4-8
Accessing Transformation Libraries ..... 4-9

## 5 Creating PL/SQL Mappings

**Overview of Oracle Warehouse Builder Mappings** ..... 5-1
Types of Mappings ..... 5-2
PL/SQL Mappings ..... 5-3
SQL*Loader Mappings ..... 5-3
SAP ABAP Mappings ..... 5-3
Code Template (CT) Mappings ..... 5-3
About Chunking for PL/SQL Mappings ..... 5-4
Types of Chunking ..... 5-4
Methods of Performing Parallel Chunking ..... 5-5
**Overview of the Mapping Editor** ..... 5-5
Mapping Editor Canvas ..... 5-6
Logical View ..... 5-6

Execution View ..... 5-6
Execution View Menu and Toolbars ..... 5-7
Mapping Editor Display Options ..... 5-7
**Example: Defining a Simple PL/SQL Mapping** ..... 5-7
**Steps to Perform Extraction, Transformation, and Loading (ETL) Using Mappings** ..... 5-9
**Defining Mappings** ..... 5-11
Rules for Naming Mappings ..... 5-12
**Adding Operators to Mappings** ..... 5-13
Using the Add Operator Dialog Box to Add Operators ..... 5-15
Create Unbound Operator with No Attributes ..... 5-15
Select from Existing Repository Object and Bind ..... 5-15
Using Pseudocolumns ROWID and ROWNUM in Mappings ..... 5-15
**Connecting Operators, Groups, and Attributes** ..... 5-16
Connecting Operators ..... 5-17
Connecting Groups ..... 5-17
Connecting Attributes ..... 5-17
Using the Mapping Connection Dialog Box ..... 5-18
Attribute Group to Connect ..... 5-19
Connection Options ..... 5-20
Messages ..... 5-20
Connections ..... 5-20
**Editing Operators** ..... 5-21
Name Tab ..... 5-22
Groups Tab ..... 5-22
Input and Output Tabs ..... 5-22
Using Display Sets ..... 5-24
Defining Display Sets ..... 5-24
Selecting a Display Set ..... 5-25
**Setting Mapping Properties** ..... 5-25
Specifying the Order in Which Target Objects in a Mapping Are Loaded ..... 5-26
Reset to Default ..... 5-26
**Configuring Mappings** ..... 5-27
Steps to Configure Mappings ..... 5-27
**Synchronizing Operators and Workspace Objects** ..... 5-28
Synchronizing a Mapping Operator with its Associated Workspace Object ..... 5-29
Synchronizing All Operators in a Mapping ..... 5-30
Synchronizing a Workspace Object with a Mapping Operator ..... 5-30
Steps to Synchronize a Workspace Object with a Mapping Operator ..... 5-31
Advanced Options for Synchronizing ..... 5-32
Matching Strategies ..... 5-32
Match by Object Identifier ..... 5-33
Match by Bound Name ..... 5-33
Match by Position ..... 5-33
**Example: Using a Mapping to Load Transaction Data** ..... 5-33
**Example: Using the Mapping Editor to Create Staging Area Tables** ..... 5-37
**Using Pluggable Mappings** ..... 5-38
Creating Pluggable Mappings ..... 5-39

Creating Standalone Pluggable Mappings ..... 5-39
Signature Groups ..... 5-40
Input Signature ..... 5-40
Output Signature ..... 5-40
Creating Pluggable Mapping Folders ..... 5-41
Creating User Folders Within Pluggable Mapping Libraries ..... 5-41
**Copying Operators Across Mappings and Pluggable Mappings** ..... 5-42
Limitations of Copying Operators, Groups, and Attributes ..... 5-43
**Grouping Operators in Mappings and Pluggable Mappings** ..... 5-44
Steps to Group Operators in Mappings and Pluggable Mappings ..... 5-44
Viewing the Contents of a Folder ..... 5-45
Steps to Ungroup Operators in Mappings and Pluggable Mappings ..... 5-45
Spotlighting Selected Operators ..... 5-45
**Locating Operators, Groups, and Attributes in Mappings and Pluggable Mappings** ..... 5-46
Steps to Perform a Regular Search ..... 5-46
Steps to Perform an Advanced Search ..... 5-47
Advanced Find Dialog Box ..... 5-47
**Debugging Mappings** ..... 5-49
General Restrictions in the Mapping Debugger ..... 5-49
Starting a Debug Session ..... 5-50
Debug Panels of the Design Center ..... 5-50
Info Panel ..... 5-50
Data Panel ..... 5-50
Defining Test Data ..... 5-51
Creating New Tables to Use as Test Data ..... 5-51
Editing the Test Data ..... 5-52
Cleaning Up Debug Objects in the Runtime Schema ..... 5-52
Setting Breakpoints ..... 5-52
Setting Watches ..... 5-53
Running the Mapping ..... 5-53
Selecting the First Source and Path to Debug ..... 5-54
Debugging Mappings with Correlated Commit ..... 5-54
Setting a Starting Point ..... 5-55
Debugging Pluggable Submap Operators ..... 5-55
ReInitializing a Debug Session ..... 5-55
Scalability ..... 5-56

## 6 Performing ETL Using Dimensional Objects

**Using Dimensions in ETL Mappings** ..... 6-1
Loading Data Into Dimensions ..... 6-1
Loading Data into Type 1 Dimensions ..... 6-1
Loading Data into Type 2 Slowly Changing Dimensions (SCDs) ..... 6-3
Loading Data into Type 3 Slowly Changing Dimensions (SCDs) ..... 6-5
Example: Loading Data Into Type 2 Slowly Changing Dimensions ..... 6-6
Extracting Data Stored in Dimensions ..... 6-8
Extracting Data from Dimensions ..... 6-8
Extracting Data from Type 2 Slowly Changing Dimensions (SCDs) ..... 6-8

Extracting Data from Type 3 Slowly Changing Dimensions (SCDs) ..... 6-9
Removing Data from Dimensions ..... 6-10
Example: Removing Data from Dimensions ..... 6-11
**Using Cubes in ETL Mappings** ..... 6-12
Loading Data Into Cubes ..... 6-13

## 7 Using SQL*Loader, SAP, and Code Template Mappings

**Creating SQL*Loader Mappings to Extract Data from Flat Files** ..... 7-1
Extracting Data from Flat Files ..... 7-2
Loading Data into a Flat File ..... 7-3
Creating a New Flat File Target ..... 7-4
**Creating SAP Extraction Mappings** ..... 7-4
Defining an SAP Extraction Mapping ..... 7-4
Adding SAP Tables to the Mapping ..... 7-5
Setting the Loading Type ..... 7-5
Setting Configuration Properties for the Mapping ..... 7-6
Setting the Join Rank ..... 7-7
**Retrieving Data from the SAP System** ..... 7-7
Automated System ..... 7-8
Semiautomated System ..... 7-9
Manual System ..... 7-11
**Creating Code Template (CT) Mappings** ..... 7-12
About Prebuilt Code Templates Shipped with Warehouse Builder ..... 7-13
Limitations of Using Certain Prebuilt Code Templates ..... 7-16
Mapping Operators Only Supported in Oracle Target CT Execution Units ..... 7-16
Steps to Perform ETL Using Code Template Mappings ..... 7-17
Creating Template Mapping Modules ..... 7-18
Creating Mappings Using Code Templates ..... 7-19
Defining Execution Units ..... 7-19
Execution View Menu and Toolbars ..... 7-20
Creating Execution Units ..... 7-20
Adding Operators to an Execution Unit ..... 7-21
Adding Operators to Multiple Execution Units ..... 7-21
Removing Operators from an Execution Unit ..... 7-21
Removing Execution Units ..... 7-21
Creating Default Execution Units ..... 7-22
Default Code Template for An Execution Unit ..... 7-22
How Warehouse Builder Displays Code Templates that Can be Associated with Execution Units 7-23
Starting the Control Center Agent (CCA) ..... 7-23
Validating Code Template Mappings ..... 7-23
Generating Code Template Mappings ..... 7-24
Sample Code Generated for CT Mappings ..... 7-25
Deploying Code Template Mappings ..... 7-26
Executing Code Template Mappings ..... 7-27
Viewing Execution Results for Code Template Mappings ..... 7-27
Viewing Execution Results by Using the Results Tab ..... 7-27

Viewing Execution Results by Using the Audit Information Panel ..... 7-28
**Setting Options for Code Templates in Code Template Mappings** ..... 7-28
Setting Properties for Bound Operators in CT Mappings ..... 7-29
Setting Configuration Properties for Code Template Mappings ..... 7-31
**Auditing the Execution of Code Template Mappings** ..... 7-31
Steps to Audit the Execution of Code Template Mappings ..... 7-33
**Using Code Template Mappings to Perform Change Data Capture (CDC)** ..... 7-33
Types of Change Data Capture (CDC) ..... 7-33
Change Data Capture Commands ..... 7-34
Example: Performing Change Data Capture Using Code Templates ..... 7-35
Steps to Perform Change Data Capture Using CDC CTs ..... 7-35
Selecting the Objects for Change Data Capture ..... 7-35
Creating the Mapping that Loads Changes ..... 7-36
Deploying the Change Data Capture Solution ..... 7-37
Starting the Change Data Capture Process ..... 7-37
Adding a Subscriber to the Change Data Capture Process ..... 7-37
Testing the Change Data Capture Process ..... 7-38
Performing Change Data Capture Actions in Warehouse Builder ..... 7-38
**Using Control Code Templates** ..... 7-40
Example: Checking Data Constraints Using Control CTs ..... 7-41
Steps to Log Constraint Violations While Loading Data Into a Target Table ..... 7-41
Creating the Source Module and Importing Source Objects ..... 7-41
Creating the Code Template Mapping that Extracts Data, Checks Data Integrity, and Loads Data into an Oracle Target 7-42
**Using Oracle Target CTs in Code Template Mappings** ..... 7-43
Example: Using Oracle Target Code Templates ..... 7-43
Creating the Source Module and Importing Source Objects ..... 7-43
Creating the Target Module and Target Table ..... 7-43
Creating the CT Mapping that Transforms Source Data Using Oracle Target CTs ..... 7-44
**Moving Data from Heterogeneous Databases to Oracle Database** ..... 7-45
Example: Moving Data from IBM DB2 to Oracle Database Using Integration CTs and Load CTs 7-45
Steps to Extract Data from IBM DB2, Transform Data, and Load it into an Oracle Database 7-46
Create the Source Module ..... 7-46
Create the Target Module and Target Table ..... 7-46
Create the CT Mapping that Extracts, Transforms, and Loads Data ..... 7-47

## 8 Designing Process Flows

**Overview of Process Flows** ..... 8-1
About Process Flow Modules and Packages ..... 8-2
**Example: Creating a Basic Process Flow** ..... 8-2
**Steps for Defining Process Flows** ..... 8-5
Creating Oracle Workflow Locations ..... 8-5
Creating Process Flow Modules ..... 8-6
Creating User Folders Within a Process Flow Module ..... 8-7
Creating Process Flow Packages ..... 8-7

Creating Process Flows ..... 8-8
**Adding Activities to Process Flows** ..... 8-8
About Activities ..... 8-8
Adding Activities ..... 8-9
Parameters for Activities ..... 8-10
**Creating and Using Activity Templates** ..... 8-11
Name and Description Page ..... 8-11
Parameters Page ..... 8-12
Using Activity Templates ..... 8-12
**About Transitions** ..... 8-13
Rules for Valid Transitions ..... 8-13
Connecting Activities ..... 8-13
Configuring Activities ..... 8-14
Using Parameters and Variables ..... 8-14
Using a Namespace ..... 8-15
Using Bindings ..... 8-15
**About Expressions** ..... 8-16
Global Expression Values ..... 8-16
**Defining Transition Conditions** ..... 8-17
**Example: Using Process Flows to Access Flat Files with Variable Names** ..... 8-18
Creating the Process Flow ..... 8-19
Setting Parameters for the User Defined Activity ..... 8-19
**Method 1: Write a script Within Warehouse Builder** ..... 8-20
**Method 2: Call a script maintained outside of Warehouse Builder** ..... 8-20
Configuring the User Defined Activity ..... 8-21
Designing the Mapping ..... 8-21
Deploying and Executing ..... 8-22
Subsequent Steps ..... 8-22
**Example: Using Process Flows to Transfer Remote Files** ..... 8-22
Defining Locations ..... 8-23
Creating the Process Flow ..... 8-23
Setting Parameters for the FTP Activity ..... 8-24
Example: Writing a Script in Warehouse Builder for the FTP Activity ..... 8-24
Using Substitution Variables ..... 8-25
Configuring the FTP Activity ..... 8-26
Registering the Process Flow for Deployment ..... 8-26

## 9 Defining Custom Transformations

**About Transforming Data Using Warehouse Builder** ..... 9-1
Benefits of Using Warehouse Builder for Transforming Data ..... 9-2
**Defining Custom Transformations** ..... 9-2
Defining Functions and Procedures ..... 9-3
Naming the Custom Transformation ..... 9-3
Defining the Parameters ..... 9-4
Specifying the Implementation ..... 9-4
Defining Table Functions ..... 9-4
Naming the Table Function ..... 9-5

Specifying the Return Type ..... 9-5
Specifying Table Function Input and Output Parameters ..... 9-5
Specifying Parallelism Options ..... 9-6
Specifying Data Streaming Options ..... 9-6
Specifying the Table Function Implementation ..... 9-7
Defining PL/SQL Types ..... 9-7
About PL/SQL Types ..... 9-7
Usage Scenario for PL/SQL Types ..... 9-8
Creating PL/SQL Types ..... 9-9
Name and Description Page ..... 9-9
Attributes Page ..... 9-10
Return Type Page ..... 9-10
Summary Page ..... 9-11
**Editing Custom Transformations** ..... 9-11
Editing Function or Procedure Definitions ..... 9-11
Editing PL/SQL Types ..... 9-12
Name Tab ..... 9-12
Attributes Tab ..... 9-12
Return Type Tab ..... 9-12
Editing Table Functions ..... 9-12
**Importing Transformations** ..... 9-13
Restrictions on Using Imported PL/SQL ..... 9-14
**Example: Reusing Existing PL/SQL Code** ..... 9-14
**Using Functions In Non-Oracle Platforms** ..... 9-19
Creating IBM DB2 and SQL Server Functions ..... 9-19
Defining IBM DB2 and SQL Server Functions ..... 9-19
Importing a Function ..... 9-20
Predefined Generic Heterogeneous Functions ..... 9-20
Using the Functions in Mappings ..... 9-21
**Configuring Functions** ..... 9-21
Configuring Oracle Functions ..... 9-21
AUTHID ..... 9-21
Deterministic ..... 9-21
Parallel Enable ..... 9-22
Pragma Autonomous Transaction ..... 9-22

## 10 Understanding Performance and Advanced ETL Concepts

**Best Practices for Designing PL/SQL Mappings** ..... 10-1
Set-Based Versus Row-Based Operating Modes ..... 10-4
Set-Based Mode ..... 10-5
Row-Based Mode ..... 10-5
Row-Based (Target Only) Mode ..... 10-6
About Committing Data in Warehouse Builder ..... 10-7
Committing Data Based on Mapping Design ..... 10-7
Committing Data from a Single Source to Multiple Targets ..... 10-7
Automatic Commit versus Automatic Correlated Commit ..... 10-8
Embedding Commit Logic into the Mapping ..... 10-9

Committing Data Independently of Mapping Design ..... 10-10
Running Multiple Mappings Before Committing Data ..... 10-10
Committing Data at Runtime ..... 10-11
Committing Mappings through the Process Flow Editor ..... 10-12
Ensuring Referential Integrity in PL/SQL Mappings ..... 10-13
**Best Practices for Designing SQL*Loader Mappings** ..... 10-13
Using Conventional Loading to Ensure Referential Integrity in SQL*Loader Mappings.. 10-13
Maintaining Relationships Between Master and Detail Records ..... 10-14
Extracting and Loading Master-Detail Records ..... 10-15
Error Handling Suggestions ..... 10-17
Subsequent Operations ..... 10-18
Using Direct Path Loading to Ensure Referential Integrity in SQL*Loader Mappings ..... 10-18
**Improved Performance through Partition Exchange Loading** ..... 10-21
About Partition Exchange Loading ..... 10-22
Configuring a Mapping for PEL ..... 10-22
Direct and Indirect PEL ..... 10-23
Using Indirect PEL ..... 10-23
Example: Using Direct PEL to Publish Fact Tables ..... 10-24
Using PEL Effectively ..... 10-24
Configuring Targets in a Mapping ..... 10-25
Step 1: Create All Partitions ..... 10-25
Step 2: Create All Indexes Using the LOCAL Option ..... 10-26
Step 3: Primary/Unique Keys Use "USING INDEX" Option ..... 10-26
Restrictions for Using PEL in Warehouse Builder ..... 10-26
**High Performance Data Extraction from Remote Sources** ..... 10-26

## 11 Scheduling ETL Jobs

**Overview of Schedules** ..... 11-1
**Defining Schedules** ..... 11-2
Editing Schedules ..... 11-3
Start and End Dates and Times ..... 11-4
Defining Schedules To Repeat ..... 11-4
By Month ..... 11-6
By Week Number ..... 11-6
By Year Day ..... 11-6
By Month Day ..... 11-7
By Day ..... 11-7
By Hour ..... 11-7
By Minute ..... 11-7
By Second ..... 11-7
By Set Position ..... 11-7
Example Schedules ..... 11-8
**Applying Schedules to ETL Objects** ..... 11-8
**Scheduling ETL Jobs in Oracle Enterprise Manager** ..... 11-9
The SQLPLUS_EXEC_TEMPLATE SQL Script ..... 11-9
The WB_RT_API_EXEC.RUN_TASK Function ..... 11-10

## 12 Deploying to Target Schemas and Executing ETL Logic


**Overview of Deployment and Execution in Warehouse Builder** ..... 12-1
About Deployment ..... 12-1
About Deployment Actions ..... 12-2
About Deployment Status ..... 12-3
About Deploying Dimensional Objects ..... 12-3
About Deploying Mappings and Process Flows ..... 12-3
About Deploying Code Template (CT) Mappings and Web Services ..... 12-3
About Deploying Schedules ..... 12-3
About Execution ..... 12-4
About Configurations ..... 12-4
About Viewing and Setting Configuration Properties for Different Configurations ..... 12-4
**Steps in the Deployment and Execution Process** ..... 12-5
**Deploying Objects** ..... 12-6
Deploying Objects Using the Control Center Manager ..... 12-6
Deploying Objects Using the Projects Navigator ..... 12-7
Deploying Target Systems to a Remote System ..... 12-8
Reviewing Deployment Results ..... 12-8
**Starting ETL Jobs** ..... 12-9
Viewing Execution Results for ETL Jobs ..... 12-10
Viewing the Data ..... 12-11
Scheduling ETL Jobs ..... 12-11
**Starting ETL Jobs in SQL*Plus** ..... 12-12
Managing Jobs Using SQL Scripts ..... 12-12
**Example: Updating a Target Schema** ..... 12-12


## 13 Auditing Deployments and Executions


**About Auditing Deployment and Executions** ..... 13-1
About the Repository Browser ..... 13-2
About the Heterogeneous Repository Browser (HRAB) ..... 13-2
Differences Between Repository Browser and Heterogeneous Repository Browser ..... 13-2
Installing the Heterogeneous Repository Browser on Heterogeneous Databases and OC4J Servers 13-3
Creating Data Stores ..... 13-3
Types of Auditing ..... 13-3
List of Heterogeneous Repository Browser Reports ..... 13-4
Viewing Audit Reports ..... 13-4
**Opening the Repository Browser** ..... 13-5
Managing the Repository Browser Listener ..... 13-5
Accessing the Repository Browser ..... 13-6
Logging in to a Workspace ..... 13-6
Connecting to an Oracle Database ..... 13-7
Connecting to a Heterogeneous Database or OC4J Server ..... 13-7
**Design Reports** ..... 13-7
Repository Navigator ..... 13-7
Object Properties ..... 13-8

Object Reports ..... 13-9
Summary Reports ..... 13-9
Detailed Reports ..... 13-10
Implementation Reports ..... 13-11
Impact Analysis Reports ..... 13-11
Object Lineage ..... 13-11
Object Impact ..... 13-12
**Control Center Reports** ..... 13-12
Deployment Reports ..... 13-13
Deployment Schedule Report ..... 13-13
Locations Report ..... 13-14
Object Summary Report ..... 13-15
Location Object Summary Report ..... 13-15
Deployment Report ..... 13-15
Deployment Error Detail Report ..... 13-16
Execution Reports ..... 13-16
Execution Schedule Report ..... 13-16
Execution Summary Report ..... 13-17
Execution Report ..... 13-17
Error Table Execution Report ..... 13-17
Execution Job Report ..... 13-17
Trace Report ..... 13-18
Job File Report ..... 13-18
Job Start Report ..... 13-18
Job Error Diagnostic Report ..... 13-19
Management Reports ..... 13-19
Service Node Report ..... 13-19
Location Validation Report ..... 13-20
**Common Repository Browser Tasks** ..... 13-20
Identifying Recently-Run Processes ..... 13-20
Identifying Why a Process Run Failed ..... 13-20
Comparing Process Runs ..... 13-21
Discovering Why a Map Run Gave Unexpected Results ..... 13-21
Identifying Recently-Made Deployments ..... 13-21
Identifying the Data Objects That Are Deployed to a Specific Location ..... 13-22
Identifying the Map Runs that Use a Specific Deployed Data Object ..... 13-22
Discovering the Default Deployment Time Settings of a Deployed Process ..... 13-22
Rerunning a Process ..... 13-22
Monitoring a Process Run ..... 13-22
Terminating a Process Run ..... 13-23
Removing the Execution Audit Details for a Process ..... 13-23
Removing Old Deployment Audit details ..... 13-23
Viewing Error Tables Created as a Result of Data Auditor Execution ..... 13-23
Unregistering a Location ..... 13-24
Updating Location Connection Details for a Changed Database Environment ..... 13-24
Updating Service Node Details in a Changing Oracle RAC Environment ..... 13-24

## 14 Managing Metadata Dependencies


**About the Metadata Dependency Manager** ..... 14-1
Example: Lineage and Impact Analysis (LIA) ..... 14-1
About Lineage and Impact Analysis and Metadata Dependency Diagrams ..... 14-3
**Opening an LIA Diagram** ..... 14-4
**Managing and Exploring Objects in an LIA Diagram** ..... 14-4
Exploring Object Lineage and Impact in an LIA Diagram ..... 14-4
Using Find to Search for Objects in an LIA Diagram ..... 14-5
Using Groups in an LIA Diagram ..... 14-5
Managing Groups in an LIA Diagram ..... 14-6
Displaying an Object's Attributes ..... 14-6
Exporting and Printing LIA Diagrams ..... 14-7
**Making Changes to Design Metadata Using Automatic Change Propagation** ..... 14-7
Automated Change Propagation in the Dependency Manager ..... 14-8


## 15 Troubleshooting and Error Handling for ETL Designs


**Inspecting Error Logs in Oracle Warehouse Builder** ..... 15-1
Troubleshooting Validation Errors ..... 15-1
Troubleshooting Generation Errors ..... 15-2
Troubleshooting Deployment and Execution Errors ..... 15-3
Determining the Operators that Caused Errors in Mappings ..... 15-3
Troubleshooting Name and Address Server Errors ..... 15-4
**Using DML Error Logging** ..... 15-4
About DML Error Tables ..... 15-4
Enabling DML Error Logging ..... 15-5
DML Error Logging and ETL ..... 15-5
DML Error Logging Limitations ..... 15-6
**Troubleshooting the ETL Process** ..... 15-6
ORA-04063 While Running Hybrid Maps ..... 15-6
Agent Log Files ..... 15-6
Error Starting the Control Center Agent (CCA) ..... 15-7
Error Executing Web Services from the Secure Web Site ..... 15-7
REP-01012 While Deploying Mappings to a Target Schema ..... 15-7
Unable to Delete a Location ..... 15-8


## 16 Creating and Consuming Web Services in Warehouse Builder


**Introduction to Web Services** ..... 16-1
Advantages of Web Services ..... 16-2
About Web Services in Oracle Warehouse Builder ..... 16-2
About Defining Web Services ..... 16-3
About Publishing Web Services ..... 16-3
About Consuming Web Services ..... 16-3
About Public Web Services ..... 16-4
**Publishing Warehouse Builder Objects as Web Services** ..... 16-4
Creating Web Service Packages ..... 16-6
Creating Web Services Based on Warehouse Builder Objects ..... 16-6

Naming the Web Service ..... 16-7
Defining the Web Service Implementation ..... 16-7
Validating Web Services ..... 16-8
Generating Web Services ..... 16-8
Deploying Web Services ..... 16-9
Deploying Web Services Using the Control Center Manager ..... 16-9
Deploying Web Services Using the Design Center ..... 16-9
**Creating Web Services Based on a URL** ..... 16-10
Naming and Describing a Public Web Service ..... 16-11
**Executing Web Services** ..... 16-11
Using the Control Center Manager to Execute Web Services ..... 16-11
Using a Browser to Execute Web Services ..... 16-12
Performing Operations on Web Services Using a Browser ..... 16-13
Determining If a Web Service or Application Was Deployed to an OC4J Server ..... 16-14
Executing a Control Center Job ..... 16-14
Terminating an Execution Job ..... 16-15
Running Deployed Applications ..... 16-15
**Using Web Services as Activities in Process Flows** ..... 16-16
Rules for Using Web Services in Process Flows ..... 16-16
Steps to Use Web Services in Process Flows ..... 16-16
Synchronizing Web Service Activities with Their Referenced Web Services ..... 16-17
**Using Web Services in Mappings** ..... 16-17
**Using Secure Sockets Layer (SSL) to Access Web Services Securely** ..... 16-19
J2EE Roles for Control Center Agent Security ..... 16-19
Setting Up Secure Access on External OC4J Servers ..... 16-19
Updating the Key Store Password ..... 16-21
**Case Study: Using Web Services for Data Integration** ..... 16-21
Example: Publishing Mappings as Web Services ..... 16-21
Example: Consuming Web Services in Process Flows ..... 16-22
Modify the LOAD_TOT_SALES_CT_MAP Code Template (CT) Mapping ..... 16-22
Import the Currency Converter Web Service ..... 16-22
Create a Process Flow That Consumes the Currency Converter Web Service ..... 16-23
Example: Integrating Warehouse Builder Web Services with Oracle BPEL Process Manager ..... 16-23

## 17 Moving Large Volumes of Data Using Transportable Modules

**About Transportable Modules** ..... 17-1
About Transportable Modules and Oracle Database Technology ..... 17-4
**Benefits of Using Transportable Modules** ..... 17-4
**Instructions for Using Transportable Modules** ..... 17-5
Verifying the Requirements for Using Transportable Modules ..... 17-6
Specifying Locations for Transportable Modules ..... 17-7
Transportable Module Source Location Information ..... 17-7
Creating a Transportable Module ..... 17-8
Describing the Transportable Module ..... 17-8
Selecting the Source Location ..... 17-8
Selecting the Target Location ..... 17-9

Selecting Tablespaces and Schema Objects to Import ..... 17-9
Available Database Objects ..... 17-9
Finding Objects in the Available Database Object List: ..... 17-10
Filtering the Available Database Objects List: ..... 17-10
Objects Not Available for Inclusion in Transportable Modules ..... 17-11
Reviewing the Transportable Module Definitions ..... 17-11
Configuring a Transportable Module ..... 17-12
Transportable Module Configuration Properties ..... 17-12
Schema Configuration Properties ..... 17-14
Target DataFile Configuration Properties ..... 17-15
Tablespace Configuration Properties ..... 17-15
Generating and Deploying a Transportable Module ..... 17-15
Designing Mappings that Access Data through Transportable Modules ..... 17-17
**Editing Transportable Modules** ..... 17-17
Name ..... 17-17
Source Location ..... 17-17
Tablespaces ..... 17-17
Target Locations ..... 17-18
Viewing Tablespace Properties ..... 17-18
Reimporting Metadata into a Transportable Module ..... 17-18

## 18 Performing Data Profiling

**Overview of Data Profiling** ..... 18-1
Sources Supported by Warehouse Builder for Data Profiling ..... 18-1
Using Warehouse Builder Data Profiling with Warehouse Builder ETL ..... 18-2
Using Warehouse Builder Data Profiling with Other ETL Solutions ..... 18-2
About the Data Profile Editor ..... 18-3
**Performing Data Profiling** ..... 18-4
Data Profiling Restrictions ..... 18-5
Prerequisites for Data Profiling ..... 18-5
Steps to Perform Data Profiling ..... 18-6
Creating Data Profiles ..... 18-6
Configuring Data Profiles ..... 18-7
Steps to Configure Data Profiles ..... 18-8
Load Configuration Parameters ..... 18-8
Aggregation Configuration Parameters ..... 18-8
Pattern Discovery Configuration Parameters ..... 18-9
Domain Discovery Configuration Parameters ..... 18-9
Relationship Attribute Count Configuration Parameters ..... 18-9
Unique Key Discovery Configuration Parameters ..... 18-9
Functional Dependency Discovery Configuration Parameters ..... 18-9
Row Relationship Discovery Configuration Parameters ..... 18-10
Redundant Column Discovery Configuration Parameters ..... 18-10
Performance Configuration ..... 18-10
Data Rule Profiling Configuration Parameters ..... 18-10
Profiling Data ..... 18-10
Steps to Profile Data ..... 18-10

Viewing Profile Results ..... 18-11
Data Profile ..... 18-12
Profile Object ..... 18-12
Aggregation ..... 18-12
Data Type ..... 18-13
Domain ..... 18-14
Pattern ..... 18-15
Unique Key ..... 18-16
Functional Dependency ..... 18-17
Referential ..... 18-18
Data Rule ..... 18-20
Using Attribute Sets to Profile a Subset of Columns from a Data Object ..... 18-20
Defining Attribute Sets ..... 18-21
Creating a Data Profile That Contains the Attribute Set ..... 18-21
Editing Data Profiles ..... 18-22
Adding Data Objects to a Data Profile ..... 18-22
**Tuning the Data Profiling Process for Better Profiling Performance** ..... 18-22
Tuning the Data Profile for Better Data Profiling Performance ..... 18-23
Tuning the Oracle Database for Better Data Profiling Performance ..... 18-23
Multiple Processors ..... 18-23
Memory ..... 18-23
I/O System ..... 18-24
**Data Watch and Repair (DWR) for Oracle Master Data Management (MDM)** ..... 18-24
Overview of Data Watch and Repair (DWR) for MDM ..... 18-24
Predefined Data Rules for MDM ..... 18-25
Prerequisites for Performing Data Watch and Repair (DWR) ..... 18-26
Steps to Perform Data Watch and Repair (DWR) Using Warehouse Builder ..... 18-26
Importing MDM Data Rules ..... 18-27
Writing Corrected Data and Metadata to the MDM Application ..... 18-27

## 19 Designing and Deriving Data Rules

**Overview of Data Rules** ..... 19-1
Types of Data Rules ..... 19-2
Data Rules as Objects and Binding Data Rules ..... 19-3
**Using Data Rules** ..... 19-3
Managing Data Rules in Folders ..... 19-4
Deriving Data Rules From Data Profiling Results ..... 19-4
Steps to Derive Data Rules ..... 19-4
Creating Data Rules Using the Create Data Rule Wizard ..... 19-5
Defining the Data Rule ..... 19-6
Editing Data Rules ..... 19-6
Applying Data Rules to Data Objects ..... 19-7
Applying Data Rules within ETL Mappings Manually ..... 19-8

## 20 Monitoring Quality with Data Auditors and Data Rules

**Overview of Data Auditors** ..... 20-1
**Monitoring Data Quality Using Data Auditors** ..... 20-2

Creating Data Auditors ..... 20-3
Specifying Actions for Data That Violates Defined Data Rules ..... 20-3
Editing Data Auditors ..... 20-4
Configuring Data Auditors ..... 20-4
Run Time Parameters ..... 20-5
Data Auditor Parameters ..... 20-5
Code Generation Options ..... 20-6
Auditing Data Objects Using Data Auditors ..... 20-6
Manually Running Data Auditors ..... 20-6
Scheduling a Data Auditor to Run ..... 20-7
Data Auditor Execution Results ..... 20-7
Viewing Data Auditor Error Tables ..... 20-8
Granting Privileges on Error Tables ..... 20-9

## 21 Data Cleansing and Correction with Data Rules

**Overview of Data Rules in ETL and Automatic Data Correction** ..... 21-1
**Generating Correction Mappings from Data Profiling Results** ..... 21-2
Prerequisites for Creating Corrections ..... 21-2
Steps to Create Correction Objects ..... 21-2
Selecting the Data Rules and Data Types for Corrected Schema Objects ..... 21-3
Selecting the Objects to Be Corrected ..... 21-4
Choosing Data Correction and Cleansing Actions ..... 21-5
Choosing Data Correction Actions ..... 21-6
Specifying the Cleansing Strategy ..... 21-6
Viewing the Correction Tables and Mappings ..... 21-7
**Cleansing and Transforming Source Data Based on Data Profiling Results** ..... 21-8
Deploying Schema Corrections ..... 21-8
Deploying Correction Mappings ..... 21-8

## 22 Name and Address Cleansing

**About Name and Address Cleansing in Warehouse Builder** ..... 22-1
Types of Name and Address Cleansing Available in Warehouse Builder ..... 22-2
Example: Correcting Address Information ..... 22-3
Example Input ..... 22-3
Example Steps ..... 22-3
Example Output ..... 22-4
About Postal Reporting ..... 22-5
United States Postal Service CASS Certification ..... 22-5
Canada Post SERP Certification ..... 22-5
Australia Post AMAS Certification ..... 22-6
Input Role Descriptions ..... 22-6
Descriptions of Output Components ..... 22-8
Pass Through ..... 22-8
Name ..... 22-9
Address ..... 22-10
Extra Vendor ..... 22-13

Error Status ..... 22-13
Country-Specific ..... 22-17
Handling Errors in Name and Address Data ..... 22-18
**Using the Name and Address Operator to Cleanse and Correct Name and Address Data** ..... 22-19
Creating a Mapping with a Name and Address Operator ..... 22-19
Specifying Source Data Details and Setting Parsing Type ..... 22-21
Parsing Type ..... 22-21
Primary Country ..... 22-21
Dual Address Assignment ..... 22-21
Specifying Postal Report Details ..... 22-22
**Managing the Name and Address Server** ..... 22-23
Configuring the Name and Address Server ..... 22-23
Starting and Stopping the Name and Address Server ..... 22-24

## 23 Matching, Merging, and Deduplication

**About Matching and Merging in Warehouse Builder** ..... 23-1
Example: A Basic Mapping with a Match Merge Operator ..... 23-2
Overview of the Matching and Merging Process ..... 23-3
Elements of Matching and Merging Records ..... 23-3
Process for Matching and Merging Records ..... 23-4
Constructing Match Bins ..... 23-4
Constructing Match Record Sets ..... 23-4
Constructing Merge Records ..... 23-5
Match Rules ..... 23-5
Conditional Match Rules ..... 23-5
Comparison Algorithms ..... 23-6
Creating Conditional Match Rules ..... 23-8
Match Rules: Basic Example ..... 23-8
Example: Matching and Merging Customer Data ..... 23-8
Example: How Multiple Match Rules Combine ..... 23-9
Example of Transitive Matching ..... 23-10
Weight Match Rules ..... 23-10
Example of Weight Match Rules ..... 23-11
Creating Weight Match Rules ..... 23-12
Person Match Rules ..... 23-12
Person Roles ..... 23-12
Person Details ..... 23-13
Creating Person Match Rules ..... 23-14
Firm Match Rules ..... 23-14
Firm Roles ..... 23-14
Firm Details ..... 23-15
Creating Firm Match Rules ..... 23-15
Address Match Rules ..... 23-16
Address Roles ..... 23-16
Address Details ..... 23-17
Creating Address Match Rules ..... 23-17
Custom Match Rules ..... 23-18

Creating Custom Match Rules ..... 23-18
Merge Rules ..... 23-19
Match ID Merge Rule ..... 23-20
Rank and Rank Record Merge Rules ..... 23-20
Sequence Merge Rule ..... 23-20
Min Max and Min Max Record Merge Rules ..... 23-21
Copy Merge Rule ..... 23-21
Custom and Custom Record Merge Rules ..... 23-21
**Using the Match Merge Operator to Eliminate Duplicate Source Records** ..... 23-22
Steps to Use a Match Merge Operator ..... 23-22
Considerations When Designing Mappings Containing Match Merge Operators ..... 23-24
Restrictions on Using the Match Merge Operator ..... 23-24
Example: Using Two Match Merge Operators for Householding ..... 23-24

## 24 Mappings and Process Flows Reference

**Configuring ETL Objects** ..... 24-1
**Configuring Mappings Reference** ..... 24-1
Runtime Parameters ..... 24-1
Analyze Table Sample Percentage ..... 24-2
Bulk Size ..... 24-2
Chunk Execute Resume Task ..... 24-2
Chunk Force Resume ..... 24-2
Chunk Number of Times to Retry ..... 24-2
Chunk Parallel Level ..... 24-3
Commit Frequency ..... 24-3
Default Audit Level ..... 24-3
Default Operating Mode ..... 24-3
Default Purge Group ..... 24-4
Maximum Number of Errors ..... 24-4
Number of Threads to Process Chunks ..... 24-4
Code Generation Options ..... 24-4
ANSI SQL Syntax ..... 24-5
Commit Control ..... 24-5
Analyze Table Statements ..... 24-5
Enable Parallel DML ..... 24-5
Optimized Code ..... 24-6
Authid ..... 24-6
Use Target Load Ordering ..... 24-6
ERROR TRIGGER ..... 24-6
Bulk Processing Code ..... 24-6
Generation Mode ..... 24-6
Chunking Options ..... 24-7
Chunking Method ..... 24-7
Chunk Table ..... 24-7
Chunk Column ..... 24-7
Chunk Size ..... 24-7
Chunk Type ..... 24-7

SQL Statement ..... 24-8
SQL Statement Chunk Type ..... 24-8
SCD Updates ..... 24-8
Chunking Strategy ..... 24-8
Sources and Targets Reference ..... 24-8
Use LCR APIs ..... 24-8
Database Link ..... 24-8
Location ..... 24-9
Conflict Resolution ..... 24-9
Schema ..... 24-9
Partition Exchange Loading ..... 24-9
Hints ..... 24-9
Constraint Management ..... 24-9
SQL*Loader Parameters ..... 24-11
Configuring Code Template (CT) Mappings ..... 24-12
Use Enclosure Character ..... 24-12
SQL Loader Data Files ..... 24-13
Configuring Flat File Operators ..... 24-13
Flat File Operators as a Target ..... 24-13
Flat File Operator as a Source ..... 24-14
**Configuring Process Flows Reference** ..... 24-15

## 25 Source and Target Operators

**List of Source and Target Operators** ..... 25-1
**Using Oracle Source and Target Operators** ..... 25-2
Setting Properties for Oracle Source and Target Operators ..... 25-2
Capture Consistency ..... 25-2
Change Data Capture Filter ..... 25-2
Enabled ..... 25-3
Trigger Based Capture ..... 25-3
Primary Source ..... 25-3
Loading Types for Oracle Target Operators ..... 25-3
Loading Types for Flat File Targets ..... 25-4
Target Load Order ..... 25-4
Target Filter for Update ..... 25-4
Target Filter for Delete ..... 25-5
Match By Constraint ..... 25-5
Reverting Constraints to Default Values ..... 25-5
Bound Name ..... 25-6
Key Name ..... 25-6
Key Columns ..... 25-6
Key Type ..... 25-6
Referenced Keys ..... 25-6
Error Table Name ..... 25-6
Roll up Errors ..... 25-6
Select Only Errors from this Operator ..... 25-7
Setting Attribute Properties ..... 25-7

Bound Name ..... 25-7
Data Type ..... 25-7
Precision ..... 25-7
Scale ..... 25-7
Length ..... 25-7
Fractional Seconds Precision ..... 25-7
Load Column When Inserting Row ..... 25-8
Load Column When Updating Row ..... 25-8
Match Column When Updating Row ..... 25-8
Update: Operation ..... 25-8
Match Column When Deleting Row ..... 25-8
Chunking Number Column ..... 25-9
Constant Operator ..... 25-9
Construct Object Operator ..... 25-9
Cube Operator ..... 25-10
Cube Operator Properties ..... 25-11
Cube Attribute Properties ..... 25-12
Data Generator Operator ..... 25-12
Setting a Column to the Data File Record Number ..... 25-13
Setting a Column to the Current Date ..... 25-13
Setting a Column to a Unique Sequence Number ..... 25-14
Dimension Operator ..... 25-14
Dimension Operator Properties ..... 25-15
AW Properties ..... 25-15
Dimension Properties ..... 25-16
Error Table ..... 25-16
History Logging Properties ..... 25-17
Orphan Management Policies ..... 25-17
Expand Object Operator ..... 25-18
External Table Operator ..... 25-19
Mapping Input Parameter Operator ..... 25-20
Mapping Output Parameter Operator ..... 25-21
Materialized View Operator ..... 25-22
Queue Operator ..... 25-23
Using a Queue Operator ..... 25-23
Selecting the Queue ..... 25-23
Selecting the Source Type for a Queue Operator ..... 25-24
Selecting the User-Defined or Primary Type for a Queue Operator ..... 25-24
Selecting the Source Object ..... 25-25
Specifying the Source Changes to Process ..... 25-25
Sequence Operator ..... 25-25
Table Operator ..... 25-26
Merge Optimization for Table Operators ..... 25-27
Chunking for Table Operators ..... 25-27
Creating Temporary Tables While Performing ETL ..... 25-28
Is Temp Stage Table ..... 25-28
Extra DDL Clauses ..... 25-28

Temp Stage Table ID ..... 25-28
DML Error Logging ..... 25-28
Data Rules and Loading Tables ..... 25-28
Varray Iterator Operator ..... 25-29
View Operator ..... 25-30
Using the View Operator for Inline Views ..... 25-30
**Using Remote and non-Oracle Source and Target Operators** ..... 25-30
Limitations of Using Non-Oracle or Remote Targets ..... 25-30
Warehouse Builder Workarounds for Non-Oracle and Remote Targets ..... 25-31
**Using Flat File Source and Target Operators** ..... 25-31
Flat File Operator ..... 25-32
Flat File Source Operators ..... 25-32
Flat File Target Operators ..... 25-33
Setting Properties for Flat File Source and Target Operators ..... 25-33
Loading Types for Flat Files ..... 25-33
Field Names in the First Row ..... 25-33

## 26 Data Flow Operators

**List of Data Flow Operators** ..... 26-1
**About Operator Wizards** ..... 26-2
Operator Wizard General Page ..... 26-2
Operator Wizard Groups Page ..... 26-2
Operator Wizard Input and Output Pages ..... 26-3
Operator Wizard Input Connections ..... 26-3
**About the Expression Builder** ..... 26-3
Opening the Expression Builder ..... 26-3
The Expression Builder User Interface ..... 26-4
**Aggregator Operator** ..... 26-5
Group By Clause ..... 26-6
Having Clause ..... 26-7
Aggregate Function Expression ..... 26-8
**Anydata Cast Operator** ..... 26-9
**Deduplicator Operator** ..... 26-10
**Expression Operator** ..... 26-11
**Filter Operator** ..... 26-12
Adding Self Joins in a Mapping ..... 26-13
**Joiner Operator** ..... 26-13
Join Conditions ..... 26-14
Join Conditions and Cartesian Products ..... 26-14
Default Join Conditions and Foreign Keys ..... 26-14
Using Join Conditions for Outer Joins ..... 26-15
Using Join Conditions for Full Outer Joins ..... 26-15
Grouping Join Conditions ..... 26-16
Code Template Mappings and Grouping Join Conditions ..... 26-18
Join Conditions and Join Order ..... 26-18
Filters in Join Conditions ..... 26-19
Join Conditions and SAP Sources ..... 26-19

Specifying Inner and Outer Join Types Using Join Input Roles ..... 26-19
Outer Join Input Roles with More than Two Input Groups ..... 26-20
Steps to Use a Joiner Operator in a Mapping ..... 26-21
**LCR Cast Operator** ..... 26-22
**LCR Splitter Operator** ..... 26-23
**Lookup Operator** ..... 26-23
Using the Lookup Operator ..... 26-24
Name ..... 26-25
Groups ..... 26-25
Lookup Tables ..... 26-25
Input Attributes ..... 26-25
Output Attributes ..... 26-25
Lookup Conditions ..... 26-26
Multiple Match Rows ..... 26-26
No-match Rows ..... 26-28
Type 2 History Lookup ..... 26-28
**Pivot Operator** ..... 26-28
Example: Pivoting Sales Data ..... 26-28
The Row Locator ..... 26-30
Using the Pivot Operator ..... 26-30
General ..... 26-31
Groups ..... 26-31
Input Connections ..... 26-31
Input Attributes ..... 26-32
Output Attributes ..... 26-32
Pivot Transform ..... 26-33
**Post-Mapping Process Operator** ..... 26-34
**Pre-Mapping Process Operator** ..... 26-35
**Set Operation Operator** ..... 26-36
Synchronizing the Attributes in a Set Operation Operator ..... 26-37
**Sorter Operator** ..... 26-37
Order By Clause ..... 26-38
**Splitter Operator** ..... 26-39
Example: Creating Mappings with Multiple Targets ..... 26-40
**Subquery Filter Operator** ..... 26-41
**Table Function Operator** ..... 26-43
Prerequisites for Using the Table Function Operator ..... 26-45
Input ..... 26-45
Output ..... 26-45
Table Function Operator Properties ..... 26-45
Table Function Operator Properties ..... 26-45
Input Parameter Properties ..... 26-45
Output Parameter Group Properties ..... 26-46
Output Parameter ..... 26-46
**Transformation Operator** ..... 26-46
**Unpivot Operator** ..... 26-47
Example: Unpivoting Sales Data ..... 26-47

The Row Locator ..... 26-48
Using the Unpivot Operator ..... 26-48
General ..... 26-48
Groups ..... 26-48
Input Connections ..... 26-49
Input Attributes ..... 26-49
Row Locator ..... 26-49
Output Attributes ..... 26-50
Unpivot Transform ..... 26-51

## 27 Activities in Process Flows

**Using Activities in Process Flows** ..... 27-1
Activities That Represent Objects ..... 27-1
Utility Activities ..... 27-2
Control Activities ..... 27-3
OS Activities ..... 27-3
Setting a Security Constraint ..... 27-4
Setting a Proxy Command and Parameters ..... 27-4
**AND** ..... 27-5
**Assign** ..... 27-6
**Data Auditor Monitor** ..... 27-6
**Enterprise Java Bean** ..... 27-7
Example: Using an Enterprise Java Bean Activity to Leverage Existing Business Logic from EJBs 27-7
Example: Using an Enterprise Java Bean Activity to Load Data From one DB2 Table to Another 27-8
Restrictions on Using an Enterprise Java Bean Activity ..... 27-9
**Email** ..... 27-10
**End** ..... 27-11
**End Loop** ..... 27-12
**File Exists** ..... 27-12
**FORK** ..... 27-13
**For Loop** ..... 27-14
**FTP** ..... 27-14
Writing a Script Within Warehouse Builder ..... 27-14
Using Substitution Variables ..... 27-16
Calling a Script Outside of Warehouse Builder ..... 27-17
**Java Class** ..... 27-17
Example of Using a Java Class Activity in a Process Flow ..... 27-18
Example of Customizing the Java Class Activity Executable ..... 27-18
**Manual** ..... 27-19
**Mapping** ..... 27-19
**Notification** ..... 27-21
Notification Message Substitution ..... 27-21
**OMBPlus** ..... 27-22
**OR** ..... 27-23
**Route** ..... 27-23

**Set Status** ... 27-24
**SQL*PLUS** ... 27-24
Using SQL*PLUS Activities in Process Flows ... 27-24
Using Substitution Variables ... 27-25
SQL *Plus Command ... 27-25
**Start** ... 27-26
**Subprocess** ... 27-26
**Transform** ... 27-27
**User Defined** ... 27-27
**Wait** ... 27-29
**While Loop** ... 27-29
**Web Service** ... 27-29

## 28 Warehouse Builder Transformations Reference

**Predefined Transformations in the Public Oracle Predefined Library** ... 28-1
Administrative Transformations ... 28-1
WB_ABORT ... 28-2
WB_COMPILE_PLSQL ... 28-2
WB_DISABLE_ALL_CONSTRAINTS ... 28-3
WB_DISABLE_ALL_TRIGGERS ... 28-3
WB_DISABLE_CONSTRAINT ... 28-4
WB_DISABLE_TRIGGER ... 28-5
WB_ENABLE_ALL_CONSTRAINTS ... 28-6
WB_ENABLE_ALL_TRIGGERS ... 28-6
WB_ENABLE_CONSTRAINT ... 28-7
WB_ENABLE_TRIGGER ... 28-8
WB_TRUNCATE_TABLE ... 28-9
Character Transformations ... 28-9
WB_LOOKUP_CHAR (number) ... 28-10
WB_LOOKUP_CHAR (varchar2) ... 28-11
WB_IS_SPACE ... 28-11
Control Center Transformations ... 28-12
WB_RT_GET_ELAPSED_TIME ... 28-12
WB_RT_GET_JOB_METRICS ... 28-13
WB_RT_GET_LAST_EXECUTION_TIME ... 28-14
WB_RT_GET_MAP_RUN_AUDIT ... 28-14
WB_RT_GET_NUMBER_OF_ERRORS ... 28-15
WB_RT_GET_NUMBER_OF_WARNINGS ... 28-15
WB_RT_GET_PARENT_AUDIT_ID ... 28-16
WB_RT_GET_RETURN_CODE ... 28-16
WB_RT_GET_START_TIME ... 28-17
Conversion Transformations ... 28-17
Date Transformations ... 28-18
WB_CAL_MONTH_NAME ... 28-19
WB_CAL_MONTH_OF_YEAR ... 28-20
WB_CAL_MONTH_SHORT_NAME ... 28-20
WB_CAL_QTR ... 28-21

WB_CAL_WEEK_OF_YEAR ..... 28-21
WB_CAL_YEAR ..... 28-22
WB_CAL_YEAR_NAME ..... 28-22
WB_DATE_FROM_JULIAN ..... 28-23
WB_DAY_NAME ..... 28-23
WB_DAY_OF_MONTH ..... 28-24
WB_DAY_OF_WEEK ..... 28-24
WB_DAY_OF_YEAR ..... 28-25
WB_DAY_SHORT_NAME ..... 28-25
WB_DECADE ..... 28-26
WB_HOUR12 ..... 28-26
WB_HOUR12MI_SS ..... 28-27
WB_HOUR24 ..... 28-27
WB_HOUR24MI_SS ..... 28-28
WB_IS_DATE ..... 28-28
WB_JULIAN_FROM_DATE ..... 28-29
WB_MI_SS ..... 28-29
WB_WEEK_OF_MONTH ..... 28-30
Number Transformations ..... 28-30
WB_LOOKUP_NUM (on a number) ..... 28-31
WB_LOOKUP_NUM (on a varchar2) ..... 28-32
WB_IS_NUMBER ..... 28-33
OLAP Transformations ..... 28-33
WB_OLAP_AW_PRECOMPUTE ..... 28-34
WB_OLAP_LOAD_CUBE ..... 28-34
WB_OLAP_LOAD_DIMENSION ..... 28-35
WB_OLAP_LOAD_DIMENSION_GENUK ..... 28-35
Other Transformations ..... 28-36
Spatial Transformations ..... 28-37
Streams Transformations ..... 28-37
REPLICATE ..... 28-37
XML Transformations ..... 28-38
WB_XML_LOAD ..... 28-39
WB_XML_LOAD_F ..... 28-39

## 29 Warehouse Builder Code Template Tools and Substitution Methods Reference

**Code Template Tools Supported in Warehouse Builder** ..... 29-1
**Code Template Tools Not Supported in Oracle Warehouse Builder** ..... 29-4
**Code Template Substitution Methods Supported in Warehouse Builder** ..... 29-5
**Code Template Substitution Methods Not Supported in Warehouse Builder** ..... 29-15

## Index

# Preface

This preface contains the following topics:


- Audience
- Documentation Accessibility
- Conventions
- Getting Help
- Related Documents


## Audience

This manual is written for Oracle Database administrators and others who create warehouses using Oracle Warehouse Builder.

## Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at `http://www.oracle.com/accessibility/`.

### Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

### Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

### Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/support/contact.html or visit http://www.oracle.com/accessibility/support.html if you are hearing impaired.

## Conventions

In this manual, Windows refers to the Windows NT, Windows 2000, and Windows XP operating systems. The SQL*Plus interface to Oracle Database may be referred to as SQL.

In the examples, an implied carriage return occurs at the end of each line, unless otherwise noted. You must press the Return key at the end of a line of input.

The following table lists the conventions used in this manual:

| Convention | Meaning |
|---|---|
| .<br>.<br>. | Vertical ellipsis points in an example mean that information not directly related to the example has been omitted. |
| ... | Horizontal ellipsis points in statements or commands mean that parts of the statement or command not directly related to the example have been omitted. |
| **boldface text** | Boldface type in text refers to interface buttons and links. Boldface type also serves as emphasis to set apart main ideas. |
| *italicized text* | Italicized text applies to new terms introduced for the first time. Italicized text also serves as an emphasis on key concepts. |
| `unicode text` | Unicode text denotes exact code, file directories and names, and literal commands. |
| *`italicized unicode text`* | Italicized unicode text refers to parameters whose value is specified by the user. |
| [] | Brackets enclose optional clauses from which you can choose one or none. |

## Getting Help

Help is readily available throughout Warehouse Builder:

- **Menus**: Menu bars throughout Warehouse Builder contain a Help menu. For context-sensitive information, choose **Topic** from the Help menu.
- **Wizards and Dialog Boxes**: Detailed instructions are provided on the pages of the wizards, which take you step-by-step through the process of creating an object. Click the **Help** button for additional information about completing a specific dialog box or a page of a wizard.
- **Tools**: You can identify the tools on a toolbar by the tooltips that appear when you rest the mouse over the icon.

  Some toolbars include a Help icon, which displays the Contents page of the Help system.
- **Lists**: For items presented in lists, a description of the selected item displays beneath the list.

- **Pop-Up menus**: Click the arrow icon on the right side of the title bar for a window. Then choose **Help** from the pop-up menu for context-sensitive information.

You may also want to follow the Oracle By Example tutorials at

http://www.oracle.com/technology/products/warehouse/selfserv_edu/self_service_education.html

## Related Documents

The Oracle Warehouse Builder documentation set includes these manuals:

- *Oracle Warehouse Builder Sources and Targets Guide*
- *Oracle Warehouse Builder Concepts*
- *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide*
- *Oracle Warehouse Builder Installation and Administration Guide for Windows and UNIX*
- *Oracle Warehouse Builder API and Scripting Reference*
- *Oracle Warehouse Builder OMB*Plus Command Reference*
- *Oracle Warehouse Builder Release Notes*

In addition to the Warehouse Builder documentation, you can refer to *Oracle Database Data Warehousing Guide*.

# Part I

## Data Modeling

Oracle Warehouse Builder enables you to design your data warehouses. This part describes how to design your data warehouse and create the data objects that store data.

This part contains the following chapters:


- Chapter 1, "Designing Source and Target Schemas"
- Chapter 2, "Creating Relational Data Objects"
- Chapter 3, "Defining Dimensional Objects"

# 1

# Designing Source and Target Schemas

This chapter provides an overview of designing target schemas. The data in your data warehouse is stored in target schemas. This data is in the form of data objects such as tables, views, dimensions, and cubes. In a traditional data warehousing implementation, there is typically only one target schema, which is the data warehouse target. You can design both relational and dimensional target schemas.

This chapter contains the following topics:

- Designing Target Schemas
- Configuring Data Objects
- Validating Data Objects
- Generating Data Objects

## Designing Target Schemas

A target schema contains the data objects that contain your data warehouse data. To design a target schema, you create any of the dimensional or relational objects listed in Table 2–1 on page 2-2.

You can design a relational target schema or a dimensional target schema. In this section, the term *dimensions* refers to both regular dimensions and Slowly Changing Dimensions (SCDs).

**To design your target schema:**

1. Create the target module that will contain the data objects for your data warehouse.

   See "Creating Target Modules" on page 1-1 for details about how to create a target module.

2. Define the target schema. The schema can be a relational target schema or a dimensional target schema.

   For more details, see "Designing Relational Target Schemas" on page 1-2 or "Designing Dimensional Target Schemas" on page 1-3.

### Creating Target Modules

A target module is a container that holds the metadata definitions of all your data warehouse objects. Each target module corresponds to a target location that represents the physical location where the objects are stored.

> **See Also:** *Oracle Warehouse Builder Sources and Targets Guide* for more information about the targets supported by Oracle Warehouse Builder

**To create a target module:**

1. Expand the project node under which you want to create the target module and then expand the node representing the type of target that you want to create.

   A separate node is displayed for each type of target that you can create. To create a target schema in the Oracle Database, expand the Databases node and then the Oracle node. To create a flat file target, expand the Files node.

2. Right-click the type of target that you want to create and select **New *type* Module**.

   For example, to create an Oracle module, expand the Databases node, right-click the Oracle node, and then select **New Oracle Module**. To create a flat file module, right-click the Files node and select **New Flat File Module**.

3. On the Welcome page of the Create Module Wizard click **Next**.

4. On the Name and Description page, enter information for the following fields and then click **Next**.

   **Name:** Enter a name for the target module.

   **Description:** Enter an optional description for the target module.

   The name and description must conform to the naming standards specified in "Naming Conventions for Data Objects" on page 2-8.

5. On the Connection Information page, enter the details of the physical location where your data warehouse objects will be stored and click **Finish**.

   If you have already created a location that corresponds to the physical location where the data objects will be stored, then select this location from the Location list.

The target module is created and added to the Projects Navigator. You can now create your data objects in this target module.

You can create user folders to organize all or some objects in your module based on specific object characteristics. For example, you create user folders to group tables based on their functionality (sales, marketing, administration).

> **See Also:** *Oracle Warehouse Builder Sources and Targets Guide* for more information about creating user folders within modules

## Designing Relational Target Schemas

A relational target schema is one that contains relational data objects such as tables, views, materialized views, and sequences. All the warehouse data is stored in these objects.

**To design a relational target schema:**

1. If you have not already done so, then create an Oracle module that will contain the objects for your target schema. Ensure that this module is associated with the location created to store the target objects.

2. Define the relational data objects that are part of the target schema.

   Relational data objects include tables, views, materialized views, and sequences. You may have already imported some existing target objects. To create additional

data objects, see "Creating Relational Data Objects" on page 2-1. You can define additional structures pertaining to relational objects such as constraints, indexes, and partitions.

Note that this step only creates the definitions of the objects in the workspace. To create the objects in the target schema, you must deploy these objects.

3. Configure the data objects.

   In this step, you set the physical properties of the data objects. For example, you specify the name of the tablespace in which a table should be created. Each data object has a set of default configuration parameters. You can modify these default values.

   See "Configuring Data Objects" on page 1-4.

4. Validate the data objects. You can only validate Oracle data objects, not data objects in other databases such as DB2 or SQL Server.

   Validation verifies the metadata definitions and configuration parameters of data objects. Correct any errors that you encounter during the validation.

   See "Validating Data Objects" on page 1-4.

5. Generate code that will create these data objects in the target schema. You can generate code only for Oracle data objects.

   Generation produces code that is required to create the data objects created in Step 2 in the target schema.

   See "Generating Data Objects" on page 1-6.

## Designing Dimensional Target Schemas

A dimensional target schema uses dimensional objects to store the data warehouse data. Dimensional objects include dimensions and cubes. Dimensional objects transform the visualization of the target schema from a table-oriented environment to a more business-focused environment. This helps you obtain answers to complex analytical queries quickly and efficiently.

You can create a dimensional target schema only in an Oracle module.

**To design a dimensional target schema:**

1. If you have not already done so, then create the Oracle module that will contain your dimensional objects. Ensure that the location associated with this module refers to the target schema.

2. Define the dimensions required in your target schema as described in "Creating Dimensions" on page 3-14.

   Note that this step only creates the definitions of the dimensions in the workspace. To create the objects in the target schema, you must deploy these dimensions.

3. Define the time dimensions required in your target schema as defined in "Creating Time Dimensions" on page 3-57.

   Data warehouses use time dimensions extensively to store temporal data.

4. Define the cubes required for your target schema as described in "Creating Cubes" on page 3-39.

5. Configure the dimensions and cubes.

Configure the dimensional objects that you created in Steps 2, 3, and 4 to set physical properties for these objects. You can accept the default properties or modify them.

**See Also:**

- "Configuring Dimensions" on page 3-37 for information about configuring dimensions
- "Configuring Cubes" on page 3-55 for information about configuring cubes

6. Validate the dimensions and cubes.

   In this step, you verify the metadata definitions and configuration parameters of the dimensional objects created in Steps 2, 3, and 4. Correct any errors resulting from the validation.

   See "Validating Data Objects" on page 1-4.

7. Generate code that will create these dimensions and cubes in the target schema.

   See "Generating Data Objects" on page 1-6.

## Configuring Data Objects

Configuration defines the physical characteristics of data objects. For example, you can define a tablespace and set performance parameters in the configuration of a table. Or you can specify the type of implementation for dimensional objects. You can change the configuration of an object any time prior to deployment.

You can define multiple configurations for the same set of objects. This feature is useful when deploying to multiple environments, such as test and production.

**See Also:** *Oracle Warehouse Builder Installation and Administration Guide for Windows and UNIX* for more information about creating multiple configurations

All objects have a Deployable parameter, which is set to True by default. To prevent an object from being deployed, set this parameter to False.

You configure objects by using the Projects Navigator. Right-click the object in the Projects Navigator and select **Configure**. A new tab is displayed containing the configuration parameters of the selected object. Specify values for the required configuration parameters.

**See Also:**

- "Configuring Relational Data Objects" on page 2-46
- "Configuring Dimensions" on page 3-37
- "Configuring Cubes" on page 3-55

## Validating Data Objects

Validation is the process of verifying metadata definitions and configuration parameters. These definitions must be valid before you generate and deploy scripts.

Oracle Warehouse Builder runs a series of validation tests to ensure that data object definitions are complete and that scripts can be generated and deployed. When these

tests are complete, the results are displayed. Warehouse Builder enables you to open object editors and correct any invalid objects before continuing. In addition to being a standalone operation, validation also occurs implicitly when you generate or deploy objects.

To detect possible problems and deal with them as they arise, validate in two stages: after creating data object definitions, and after configuring objects for deployment. Validating objects after configuration is more extensive than validating objects as they are defined.

> **Tip:** Validate objects as you create and configure them to resolve problems as they arise. The same error-checking processes are run whether you are validating the design or the configuration.

When you validate an object after it has been defined, the metadata definitions for the objects that you have designed are checked for errors. For example, if you create a table, Warehouse Builder requires that columns be defined. When this object is validated, Warehouse Builder verifies that all components of the table have been defined. If these components are missing, then validation messages are displayed in the Log window.

If you validate an object after it has been configured, then the metadata definitions are rechecked for errors and configuration parameters are checked to ensure that the object will be generated and deployed without any problems. You can then edit invalid objects.

You can validate a single object or multiple objects at one time. You can also validate objects that contain objects, such as modules and projects. In this case, all data objects contained by that object are validated. Use the Projects Navigator to validate data objects.

#### Validating Data Objects by Using the Projects Navigator

In the Projects Navigator, select the data object and click **Validate**. Or select **Validate** from the File menu. You can select multiple objects by holding down the **Ctrl** key while selecting objects.

*or*

In the Projects Navigator, select the data object or data objects. To select multiple objects, hold down the **Ctrl** key while selecting objects. Right-click the data object and select **Validate**. If you selected multiple objects, then ensure that the **Ctrl** key is pressed when you right-click.

Note that you can only select multiple objects of the same type. For example, you can select eight tables, but you cannot select five tables and three views.

## Viewing Validation Results

When you validate data objects, the validation results are displayed in the Log window. A new tab named Results is displayed for each validation. This tab contains the validation results of the selected objects.

You can view validation results for objects even after you close the Results tab. Select the object in the Projects Navigator and from the View menu, select **Validation Messages**. The Log window displays the validation results for the selected object. You can also select a container object, such as a module node or the Tables node, to view

validation results. This action displays validation messages for the all the contained objects.

### Editing Invalid Objects

The results of validating data objects are displayed in the Log window. From this window, you can access the editor for an object and rectify any errors in its definition.

**To edit an invalid definition:**

In the Log window, click the Results tab corresponding to the object you want to edit. Select the node representing the object and click **Go To Source** on the Log window toolbar.

*or*

In the Log window, click the Results tab displaying the validation results of the object that you want to edit. Expand the Validation node under the object node to display the validation messages. Double-click this validation message to open the editor that edits the object. Or, right-click the message and select **Go to Source**.

After you edit the object to correct problems, save the changes by clicking **Save All** on the toolbar, and then revalidate the object.

## Generating Data Objects

When you generate data objects, Warehouse Builder produces the code required to create the data objects in the target schema. Usually, data objects generate a SQL script. The SQL script may contain a mixture of DDL statements and PL/SQL blocks for creating objects using APIs (such as dimension and cube generation). You can view the generated scripts and also store them to a file system.

When you generate code for a data object, Warehouse Builder first validates the object and then generates code. You may skip the validation step and directly generate code for your data objects. However, it is recommended that you validate objects before you generate them. This enables you to discover and correct any errors in data object definitions before the code is generated.

**To generate a single data object:**

In the Projects Navigator, select the data object and click **Generate**.

*or*

In the Projects Navigator, select the data object and then select **Generate** from the File menu.

*or*

In the Projects Navigator, right-click the data object and select **Generate**.

**To generate code for multiple data objects:**

In the Projects Navigator, select the data objects by holding down the **Ctrl** key and click **Generate**.

*or*

In the Projects Navigator, select the data objects and, while continuing to hold down the Ctrl key, right-click and select **Generate**.

You can select and generate code simultaneously for multiple objects only if all the data objects are of the same type. For example, you can generate code simultaneously

for a set of 12 tables. However, you cannot generate code for three tables and two dimensions.

Use collections to generate code for multiple data objects which are of different types and belong to different modules. You can create a collection, add all the objects that you want to generate simultaneously to the collection, and then generate the collection.

### Regenerating Data Objects

If you make any changes to your data object definitions, then you can generate modified scripts for your data object. Select the data object in the Projects Navigator and click **Generate**.

Before you view the modified scripts, close any open tabs containing previous generation scripts. Opening the same script twice without closing the original editor window will appear to give incorrect results.

## Viewing Generation Results and Generated Scripts

When you generate objects, the generation results are displayed in the Log window, if the validation completed successfully. The results of each generation operation are displayed in a separate Results tab.

**To view the generated scripts:**

1. In the Log window, click the Results tab that contains the generation results for the required object.
2. Expand the object node and then the Validation node to view the validation results.

   After the data object is validated successfully, Warehouse Builder generates scripts.
3. Expand the Scripts node under the object node.

   A list of the scripts generated for the selected data object is displayed.
4. Select a specific script and, in the Log window toolbar, click **View Script**. Or right-click a specific script and select **Go to Source**.

   The selected script is displayed in a read-only code viewer.

## Saving Generated Scripts to a File

**To save generated scripts:**

1. In the Log window, click the Results tab that contains the generation results for the required object.
2. Expand the object node and then the Scripts node.

   A list of the generated scripts is displayed, under the Scripts node, for the object that you selected.
3. Select a specific script and, on the toolbar, click **Save Script As**.

   The Save dialog box opens and you can select a location where you want to save the script file.

# 2

# Creating Relational Data Objects

After you finish designing your data warehouse or data mart, you are ready to design your target system. This chapter shows you how to create relational data objects.

This chapter contains the following topics:


- Overview of Data Objects
- Defining Tables
- Defining Views
- Defining Materialized Views
- Defining Constraints
- Defining Indexes
- Defining Partitions
- Defining Attribute Sets
- Defining Sequences
- Defining User-Defined Types
- Defining Queues
- Configuring Relational Data Objects
- Creating Relational Data Objects in Microsoft SQL Server and IBM DB2 UDB


## Overview of Data Objects

Oracle Warehouse Builder supports relational and dimensional data objects. Relational objects, like relational databases, rely on tables and table-derived objects to store and link all of their data. The relational objects you define are physical containers in the database that are used to store data. It is from these relational objects that you run queries after the warehouse has been created. Relational objects include tables, views, materialized views, sequences, user-defined types, and queues. You can also create optional structures associated with relational objects such as constraints, indexes, partitions, and attribute sets.

Dimensional objects contain additional metadata to identify and categorize your data. When you define dimensional objects, you describe the logical relationships that help store the data in a more structured format. Dimensional objects include dimensions and cubes.

The Oracle module contains nodes for each type of relational and dimensional data object that you can define in Warehouse Builder. In the Projects Navigator, expand the module to view nodes for all the supported data objects.

In addition to relational and dimensional objects, Warehouse Builder supports intelligence objects. Intelligence objects are not part of Oracle modules. They are displayed under the Business Intelligence node in the Projects Navigator. Intelligence objects enable you to store definitions of business views. You can deploy these definitions to analytical tools such as Oracle Discoverer and perform ad hoc queries on the warehouse.

> **See Also:** *Oracle Warehouse Builder Sources and Targets Guide* for more information about creating business definitions for Oracle BI Discoverer

### List of Warehouse Builder Data Objects

Table 2–1 describes the types of data objects that you can use in Warehouse Builder.

***Table 2–1 Data Objects in Oracle Warehouse Builder***

| Data Object | Type | Description |
|---|---|---|
| Tables | Relational | The basic unit of storage in a relational database management system. Once a table is created, valid rows of data can be inserted into it. Table information can then be queried, deleted, or updated. To enforce defined business rules on its data, define integrity constraints for a table.<br>See "Defining Tables" on page 2-10 for more information. |
| External Tables | Relational | External tables are tables that represent data from non-relational flat files in a relational format. Use an external table as an alternative to using a Flat File operator and SQL*Loader.<br>See *Oracle Warehouse Builder Sources and Targets Guide* for more information about external tables. |
| Views | Relational | A view is a custom-tailored presentation of data in one or more tables. Views do not actually contain or store data; they derive their data from the tables on which they are based. Like tables, views can be queried, updated, inserted into, and deleted from, with some restrictions. All operations performed on a view affect the base tables of the view. Use views to simplify the presentation of data or to restrict access to data.<br>See "Defining Views" on page 2-15 for more information. |
| Materialized Views | Relational | Materialized views are precomputed tables comprising aggregated or joined data from fact and possibly dimension tables. Also known as a summary or aggregate table. Use materialized views to improve query performance.<br>See "Defining Materialized Views" on page 2-18 for more information. |
| Sequences | Relational | Sequences are database objects that generate lists of unique numbers. You can use sequences to generate unique surrogate key values.<br>See "Defining Sequences" on page 2-35 for more information. |

*Table 2–1 (Cont.) Data Objects in Oracle Warehouse Builder*

| Data Object | Type | Description |
|---|---|---|
| Dimensions | Dimensional | A general term for any characteristic that is used to specify the members of a data set. The three most common dimensions in sales-oriented data warehouses are time, geography, and product. Most dimensions have hierarchies.<br>See "Overview of Dimensions" on page 3-2 for more information. |
| Cubes | Dimensional | Cubes contain measures and links to one or more dimension tables. They are also known as facts.<br>See "Overview of Cubes" on page 3-7 for more information. |
| Advanced Queues | Relational | Advanced queues enable message management and communication required for application integration.<br>See "Creating Advanced Queue Definitions" on page 2-43 for more information. |
| Queue Tables | Relational | Queue tables are tables that store queues. Each queue table contains a payload, whose data type is specified at the time of creating the queue table.<br>See "Creating Queue Table Definitions" on page 2-42 for more information. |
| Object Types | Relational | An object type is composed of one or more user-defined types or scalar types.<br>See "About Object Types" on page 2-36 for more information. |
| Varrays | Relational | A Varray is an ordered collection of elements.<br>See "About Varrays" on page 2-38 for more information. |
| Nested Tables | Relational | A nested table complements the functionality of the Varray data type. A nested table permits a row to have multiple "mini-rows" of related data contained within one object.<br>See "About Nested Tables" on page 2-40 for more information. |

## Supported Data Types

The metadata for the data objects that you create is stored in the repository. The metadata consists of details such as the attribute or column names, data types, and level names.

Table 2–2 displays the data types that you can use to define columns or attributes.

*Table 2–2 Oracle Warehouse Builder Supported Data Types*

| Data Type | Description |
|---|---|
| `BINARY_DOUBLE` | Stores double-precision IEEE 754-format single-precision floating-point numbers. Used primarily for high-speed scientific computation. Literals of this type end with `d`. For example, 3.0235`d`. |
| `BINARY_FLOAT` | Stores single-precision IEEE 754-format single-precision floating-point numbers. Used primarily for high-speed scientific computation. Literals of this type end with `f`. For example, 2.07`f`. |
| `BLOB` | Stores large binary objects in the database, in-line or out-of-line. Every `BLOB` variable stores a locator, which points to a large binary object. The size of a `BLOB` cannot exceed 4 gigabytes. |

*Table 2–2 (Cont.) Oracle Warehouse Builder Supported Data Types*

| Data Type | Description |
|---|---|
| CHAR | Stores fixed-length character data to a maximum size of 4,000 characters. How the data is represented internally depends on the database character set. You can specify the size in terms of bytes or characters, where each character contains one or more bytes, depending on the character set encoding. |
| CLOB | Stores large blocks of character data in the database, in-line or out-of-line. Both fixed-width and variable-width character sets are supported. Every CLOB variable stores a locator that points to a large block of character data. The size of a CLOB cannot exceed four gigabytes. |
| DATE | Stores fixed-length date times, which include the time of day in seconds since midnight. The date defaults to the first day of the current month; the time defaults to midnight. The date function SYSDATE returns the current date and time. |
| FLOAT | Stores a single-precision, floating-point number. FLOAT can be loaded with correct results only between systems where the representation of a FLOAT is compatible and of the same length. |
| INTEGER | A NUMBER subtype that stores integer values with a maximum precision of 38 decimal digits. |
| INTERVAL DAY TO SECOND | Stores intervals of days, hours, minutes, and seconds. |
| INTERVAL YEAR TO MONTH | Stores intervals of years and months. |
| LONG | Stores fixed-length character strings. The LONG data type is like the VARCHAR2 data type, except that the maximum length of a LONG value is 2147483647 bytes (2 gigabytes). |
| LONG RAW | Stores binary data or byte strings. Use this data type to store graphics, sounds, documents, or arrays of binary data. |
| MDSYS.SDOAGGRTYPE | Stores the geometric description of a spatial object and the tolerance. Tolerance is used to determine when two points are close enough to be considered as the same point. |
| MDSYS.SDO_DIM_ARRAY | Stores an array of type MDSYS.SDO_DIM_ELEMENT. |
| MDSYS.SDO_DIM_ELEMENT | Stores the dimension name, lower boundary, upper boundary, and tolerance. |
| MDSYS.SDO_ELEM_INFO_ARRAY | Stores an array of type MDSYS.SDO_ORDINATE_ARRAY. |
| MDSYS.SDO_GEOMETRY | Stores Geographical Information System (GIS) or spatial data in the database. For more information, see *Oracle Spatial Developer's Guide*. |
| MDSYS.SDO_ORDINATE_ARRAY | Stores the list of all vertices that define the geometry. |
| MDSYS.SDO_POINT_TYPE | Stores two-dimensional and three-dimensional points. |

*Table 2–2 (Cont.) Oracle Warehouse Builder Supported Data Types*

| Data Type | Description |
|---|---|
| NCHAR | Stores fixed-length (blank-padded, if necessary) national character data. Because this type can always accommodate multibyte characters, you can use it to hold any Unicode character data. How the data is represented internally depends on the national character set specified when the database was created, which might use a variable-width encoding (UTF8) or a fixed-width encoding (AL16UTF16). |
| NCLOB | Stores large blocks of NCHAR data in the database, in-line or out-of-line. |
| NUMBER | Stores real numbers in a fixed-point or floating-point format. Numbers using this data type are guaranteed to be portable among different Oracle platforms, and offer up to 38 decimal digits of precision. You can store positive and negative numbers, as well as zero, in a NUMBER column. |
| NVARCHAR2 | Stores variable-length Unicode character data. Because this type can always accommodate multibyte characters, you can use it to hold any Unicode character data. How the data is represented internally depends on the national character set specified when the database was created, which might use a variable-width encoding (UTF8) or a fixed-width encoding (AL16UTF16). |
| RAW | Stores binary data or byte strings. For example, a RAW variable might store a sequence of graphics characters or a digitized picture. Raw data is like VARCHAR2 data, except that PL/SQL does not interpret raw data. |
| ROWID | Base 64 string representing the unique address of a row in its table. This data type is primarily for values returned by the ROWID pseudocolumn. |
| SYS.ANYDATA | An Oracle-supplied type that can contain an instance of a given type, with data, plus a description of the type. ANYDATA can be used as a table column data type and lets you store heterogeneous values in a single column. The values can be of SQL built-in types as well as user-defined types. |
| SYS.AQ$_JMS_BYTES_MESSAGE | A type that is the ADT (Abstract Data Type) used to store a BytesMessage in an Oracle Streams AQ queue.<br>A BytesMessage object is used to send a message containing a stream of uninterrupted bytes.<br>For more information about this data type, see:<br>■ *Oracle Database PL/SQL Packages and Types Reference*<br>■ *Oracle Streams Advanced Queuing User's Guide* |
| SYS.AQ$_JMS_MAP_MESSAGE | A type that is the ADT used to store a MapMessage in an Oracle Streams AQ queue.<br>A MapMessage object is used to send a set of name-value pairs where the names are String types, and the values are Java primitive types.<br>For more information about this data type, see:<br>■ *Oracle Database PL/SQL Packages and Types Reference*<br>■ *Oracle Streams Advanced Queuing User's Guide* |

*Table 2–2 (Cont.) Oracle Warehouse Builder Supported Data Types*

| Data Type | Description |
|---|---|
| SYS.AQ$_JMS_MESSAGE | An ADT type that can represent any of five different JMS message types: text message, bytes message, stream message, map message, or object message. Queues created using this ADT can therefore store all five types of JMS messages.<br>For more information about this data type, see *Oracle Database PL/SQL Packages and Types Reference*: |
| SYS.AQ$_JMS_STREAM_ MESSAGE | A type that is the ADT used to store a StreamMessage in an Oracle Streams AQ queue. A StreamMessage object is used to send a stream of Java primitives. It is filled and read sequentially.<br>For more information about this data type, see:<br>■ *Oracle Database PL/SQL Packages and Types Reference*<br>■ *Oracle Streams Advanced Queuing User's Guide* |
| SYS.AQ$_JMS_TEXT_MESSAGE | A type that is the ADT used to store a TextMessage in an Oracle Streams AQ queue. A TextMessage object is used to send a message containing a java.lang.StringBuffer.<br>For more information about this data type, see:<br>■ *Oracle Database PL/SQL Packages and Types Reference*<br>■ *Oracle Streams Advanced Queuing User's Guide* |
| SYS.LCR$_ROW_RECORD | A type that represents a data manipulation language (DML) change to a row in a table. This type uses the LCR$_ROW_LIST type. |
| TIMESTAMP | Extends the DATE data type and stores the year, month, day, hour, minute, and second. The default timestamp format is set by the Oracle Database initialization parameter NLS_TIMESTAMP_FORMAT. |
| TIMESTAMP WITH LOCAL TIME ZONE | Extends the TIMESTAMP data type and includes a time-zone displacement. The time-zone displacement is the difference (in hours and minutes) between local time and Coordinated Universal Time (UTC)—formerly Greenwich Mean Time. You can also use named time zones, as with TIMESTAMP WITH TIME ZONE. |
| TIMESTAMP WITH TIME ZONE | Extends the TIMESTAMP data type and includes a time-zone displacement. The time-zone displacement is the difference (in hours and minutes) between local time and Coordinated Universal Time (UTC)—formerly Greenwich Mean Time. |
| UROWID | Represents the address of certain rows in relational tables that are not physical or are not generated by Oracle Database. For example, row address of an index-organized table and row IDs of non-Oracle foreign tables (such as DB2 accessed using a gateway). The maximum size is 4,000 bytes. |
| VARCHAR | Stores a length-value data type consisting of a binary length subfield followed by a character string of the specified length. The length is in bytes, unless character-length semantics are used for the data file. In that case, the length is in characters. |

*Table 2–2 (Cont.) Oracle Warehouse Builder Supported Data Types*

| Data Type | Description |
|---|---|
| VARCHAR2 | Stores variable-length character data. How the data is represented internally depends on the database character set. The VARCHAR2 data type takes a required parameter that specifies a maximum size up to 4,000 characters. |
| XMLFORMAT | An object type that is used to specify formatting arguments for SYS_XMLGEN() and SYS_XMLAGG() functions. |
| XMLTYPE | An Oracle-supplied type that can be used to store and query XML data in the database. It has member functions that you can use to access, extract, and query the XML data by using XPath expressions. XPath is a standard developed by the W3C committee to traverse XML documents. |
| User-defined Types | Use Oracle built-in data types and other user-defined data types as the building blocks of object types that model the structure and behavior of data in applications.<br>User-defined types include Object Types, Varrays, and Nested tables. |

## About Object Class Definition

Oracle Warehouse Builder architecture comprises several classes of objects, such as First Class Objects, Second Class Objects, and Third Class Objects. This section describes these classes of objects.

### About First Class Objects (FCOs)

A First Class Object (FCO) represents a component in the metadata repository that can be manipulated through the Warehouse Builder interface. FCOs often, but not always, own other objects. For example, a TABLE is an FCO that may own the following Second Class Objects: TABLE_COLUMN, UNIQUE_KEY, FOREIGN_KEY, and CHECK_CONSTRAINT.

For those accessing Warehouse Builder using the Design Center, FCOs generally appear on the navigation tree. Similarly, users who access Warehouse Builder through OMB*Plus can generalize FCOs as objects of OMBCREATE, OMBALTER, OMBRETRIEVE, and OMBDROP commands.

### About Second Class Objects (SCOs)

A Second Class Object (SCO) represents a dependent object component. An SCO is always owned by another object, and can, in turn, own objects itself. For example, the FCO called MAPPING contains the SCO MAPPING_OPERATOR, which in turn contains ATTRIBUTES.

For those accessing Warehouse Builder through the Design Center, SCOs can only be manipulated through an FCO. Similarly, users who access Warehouse Builder through OMB*Plus can only manipulate SCO definitions through a command against an FCO.

### About Third Class and Fourth Class Objects

Third Class and Fourth Class objects are relative rankings of objects owned by other objects. These refer only to objects whose ownership spans several layers. For example, INDEX_COLUMN is an SCO in the scenario where a DIMENSION_TABLE (which is a FCO) owns INDEX_COLUMN. However, INDEX_COLUMN becomes a Third Class

Object in the scenario where the FCO CUBE_TABLE owns the SCO INDEX, which in turn owns INDEX_COLUMN.

## Naming Conventions for Data Objects

The rules for naming data objects depend on the naming mode that you set for Warehouse Builder. Warehouse Builder maintains a business and a physical name for each object stored in a workspace. The business name for an object is its descriptive logical name and the physical name is the name used when Warehouse Builder generates code.

You set the naming mode using the Naming Preferences section of the Preferences dialog box.

> **See Also:** *Oracle Warehouse Builder Concepts* for more information about naming preferences

When you name or rename data objects, use the following naming conventions.

### Naming Oracle Data Objects

In the physical naming mode, the name for an Oracle data object can be between 1 and 30 alphanumeric characters. The name must be unique across the object category that owns the object. Blank spaces are not allowed. Data object names cannot begin with OWB$.

> **See Also:** For information about the length of physical names on other platforms, see:
>
> - "Rules for Naming Objects in IBM DB2 UDB" on page 2-57
> - "Rules for Naming Objects in Microsoft SQL Server" on page 2-58

In the business naming mode, the limit is 200 characters. The name must be unique across the object category that owns the object. For example, because all tables belong to a module, table names must be unique across the module to which they belong. Similarly, module names must be unique across the project to which they belong.

### Describing Data Objects

Edit the description of the data object as necessary. The description can be between 1 and 4,000 alphanumeric characters and can contain blank spaces. Specifying a description for a data object is optional.

### Best Practices for Naming Data Objects

Data object names or FCO names should be unique across the object category that owns the FCO. Additionally, it is a good practice to ensure that SCO names are unique across the object category that owns the FCO and are different from those of the FCO containing the SCO.

For example, a table contains constraints. The table is an FCO and constraints are SCOs. When you define the table, provide a table name that is unique across the module in which it is defined. Additionally, Oracle recommends that you provide constraint names that are unique across all the FCOs and SCOs in that module.

## Using the Data Viewer to View Data Stored in Data Objects

The Data Viewer enables you to view the data stored in relational and dimensional data objects. For example, the data viewer for a table enables you to view the table data. Similarly, the data viewer for a cube enables you to view data stored in a cube.

To access the Data Viewer for a data object, from the Projects Navigator, right-click the data object and select **Data**. The Data Viewer containing the data stored in the data object is displayed in a separate tab.

The Data Viewer tab contains the following buttons: Execute Query, Get More, Where Clause, and More. The More button is displayed at the bottom of the tab.

Click **Execute Query** to execute a query on the data object and fetch its data.

By default, the Data Viewer displays the first hundred rows of data. To retrieve the next set of rows, click **Get More**. Alternatively, you can click **More** to perform the same action.

Click **Where Clause** to specify a condition that is used to restrict the data displayed by the Data Viewer. Clicking this button displays the Where Clause dialog box. Use this dialog box to specify the condition used to filter data. You can use this option for tables and views only.

The columns and column names displayed in the Data Viewer are taken directly from the location in which the actual table is deployed.

---

**Note:** For columns with the data type `LONG`, the Data Viewer displays "<UNSUPPORTED DATATYPE>" instead of the actual value stored in the column.

---

## About Error Tables

Warehouse Builder enables you to create error tables to store logical errors that may occur while loading data into Oracle data objects such as tables, views, materialized view, dimensions, and cubes.

Use error tables to:

- Capture logical errors when data rules are applied to tables, views, or materialized views.
- Capture physical errors using DML error logging.
- Store errors caused by orphan records when an orphan management policy is enabled for dimensional objects.

> **See Also:** "Orphan Management for Dimensional Objects" on page 3-7 for more information about orphan management

### Defining Error Tables for Data Objects

An error table is created for a data object only if you set the Error Table Name configuration parameter for the data object. If you do not specify an error table name for a data object, then logical errors are not logged for that object. However, when a data object has data rules associated with it, even if you do not specify an error table name for the object, Warehouse Builder creates an error table using a default name. For example, if the name of the table for which you specified data rules is `EMP`, then the error table is called `EMP_ERR`.

**To create an error table for a data object:**

1. In the Projects Navigator, right-click the data object for which you want to create an error table, and select **Configure**.

   The Configuration tab containing the configuration parameters for the data object is displayed.

2. In the Configuration tab, expand the Error Tables node.
3. Set the value of the Error Table Name parameter to the name of the error table for the data object.

If you modify the value of the Error Table Name parameter after the data object is deployed, then you must drop the data object and redeploy it. If this data object was used in mappings, then ensure that you synchronize all operators that are bound to this data object and then redeploy the mappings.

> **Note:** The Error Table Name and Truncate Error Table configuration parameters of the Table, View, or Materialized View operators are not used for row-based code.

### Error Table Columns

In addition to the columns contained in the data object, error tables for a data object contain the columns listed in Table 2–3.

***Table 2–3 Error Table Columns***

| Column Name | Description |
|---|---|
| `ORA_ERR_NUMBER$` | Oracle Database error number |
| `ORA_ERR_MESG$` | Oracle Database error message text |
| `ORA_ERR_ROWID$` | Row ID of the row in error (for update and delete) |
| `ORA_ERR_OPTYPE$` | Type of operation: insert (I), update (U), delete (D) |
| `ORA_ERR_TAG$` | Step or detail audit ID from the runtime audit data. This is the `STEP_ID` column in the runtime view `ALL_RT_AUDIT_STEP_RUNS`. |

For scalar data types, if no data rules are applied to the data object, then the columns in the error table are of data type `VARCHAR2(4000)`. This allows physical data errors such as ORA-12899: value too large for column, to be captured. If data rules are applied, then the columns in the error table are of the same data type as the columns in the data object.

For example, the table `TEST` has two columns `C1`, of data type `NUMBER`, and `C2`, of data type `VARCHAR2(10)`. The error table generated for `TEST` will contain the DML error columns `C1` and `C2`. If no data rules are applied to `TEST`, then the data type for both `C1` and `C2` will be `VARCHAR2(4000)`. If data rules are applied to `TEST`, then `C1` will be `NUMBER` and `C2` will be of data type `VARCHAR2(10)`.

## Defining Tables

Tables are metadata representations of relational storage objects. They can be tables from a database system such as Oracle Database or even an SAP system.

The following sections provide information about defining tables:

- Creating Table Definitions on page 2-11
- Editing Table Definitions on page 2-13


## Creating Table Definitions

Tables capture the metadata used to model your target schema. Table definitions specify the table constraints, indexes, partitions, attribute sets, and metadata about the columns and data types used in the table. This information is stored in the workspace. You can later use these definitions to generate .ddl scripts that can be deployed to create physical tables in your target database. These tables can then be loaded with data from chosen source tables.

### Before You Begin

Ensure that you create the target schema that will contain your table as described in "Designing Target Schemas" on page 1-1.

**To create a table in an Oracle module:**

1. From the Projects Navigator, expand the Databases node and then the Oracle node.
2. Expand the module where you want to create the table, right-click Tables, and select **New Table**.

   *or*

   Right-click the module where you want to create a table and select **New**. The New Gallery dialog box is displayed. From the Items section, select **Table** and click **OK**.
3. In the Create Table dialog box, enter a name and an optional description for the table and click **OK**. Alternatively, you can accept the autogenerated unique name for the table and click **OK**.

   The Table Editor is displayed. Define the table using the following tabs:

   - Columns Tab on page 2-12
   - Keys Tab on page 2-13
   - Indexes Tab on page 2-13
   - Partitions Tab on page 2-13
   - Attribute Sets Tab on page 2-13
   - Data Rules Tab on page 2-13

   After you define the table using these tabs, the table definitions are created and stored in the workspace. The new table name is also added to the Projects Navigator. At this stage, only the metadata for the table is created in the workspace. To create the table in your target schema, you must deploy the table.

   **Note:** You can also create a table from the Graphical Navigator.

### Name Tab

Use the Name tab to specify the name and description of a table. This tab contains the following fields:

- **Name:** Represents the name for the table. The name should be unique within the module in which the table is defined.

- **Description:** Specify an optional description for the table.

Follow the rules in "Naming Conventions for Data Objects" on page 2-8 to specify a name and an optional description.

### Columns Tab

Use the Columns tab to define the columns in the table. This tab displays a table that you use to define columns. Each row in the table corresponds to the definition of one table column. Warehouse Builder generates the column position in the order in which you type in the columns. To reorder columns, see "Reordering Columns in a Table" on page 2-15.

Enter the following details for each column:

- **Name:** Enter the name of the column. The column name should be unique within the table.
- **Data Type:** Select the data type of the column from the Data Type list. A default data type is assigned based on the column name. For example, if you create a column named start_date, then the data type assigned is `DATE`. You can change the default assignment if it does not suit your data requirement.

  For a list of supported Oracle Database data types, see "Supported Data Types" on page 2-3.
- **Length:** Specify the length of the column. Length is applicable to character data types only.
- **Precision:** Specify the total number of digits allowed for the column. Precision is applicable for numeric data types only.
- **Scale:** Specify the total number of digits to the right of the decimal point. Scale is applicable to numeric data types only.
- **Seconds Precision:** Used for `TIMESTAMP`, `TIMESTAMP WITH TIME ZONE`, and `TIMESTAMP WITH LOCAL TIME ZONE` data types only. Specify the number of digits in the fractional part of the datetime field.
- **Not Null:** Select this field to specify that the column should not contain null values. By default, all columns in a table allow nulls.
- **Default Value:** Specify the default value for this column. If no value is entered for this column while you are loading data into the table, then the default value is used. If you specify a value for this column while loading data, then the default value is overridden and the specified value is stored in the column.
- **Virtual:** Select this option to indicate that the column is a virtual column.

  Virtual columns are not stored in the database. They are computed using the expression specified in the Expression field. You can refer to virtual columns just like any other column in the table, except that you cannot explicitly write to a virtual column.
- **Expression:** Specify the expression that is used to compute the value of the virtual column. The expression can include columns from the same table, constants, SQL functions, and user-defined PL/SQL functions.
- **Description:** Enter an optional description for the column.

### Keys Tab

Use the Keys tab to create constraints on the table columns. You can create primary keys, foreign keys, unique keys, and check constraints. For more information about creating constraints, see "Creating Constraints" on page 2-21.

### Indexes Tab

Use the Indexes tab to create indexes on the table. Indexes enable faster retrieval of data stored in your data warehouse. You can create the following types of indexes: unique, nonunique, bitmap, function-based, composite, and reverse. For more information about creating these indexes, see "Creating Indexes" on page 2-24.

### Partitions Tab

Use the Partitions tab to create partitions for the table. Partitions enable better manageability of larger tables. They also improve query and load performance. You can create the following types of partitions: range, hash, hash by quantity, list, range-hash, range-hash by quantity and range-list. For more information about partitions and how to create each type of partition, see "Defining Partitions" on page 2-25.

### Attribute Sets Tab

Use the Attribute Sets tab to create attribute sets for the table. For more information about creating attribute sets for a table, see "Defining Attribute Sets" on page 2-33.

### Data Rules Tab

Use the Data Rules tab to apply data rules to a table. Data rules enable you to determine legal data within a table and legal relationships between tables. When you apply a data rule to a table, Warehouse Builder ensures that the data in the table conforms to the specified data rule.

Warehouse Builder provides a set of predefined data rules that are listed in the DERIVED_DATA_RULES node under the Data Rules node of the Projects Navigator. You can define your own data rules by creating data rules under the Data Rules node.

Click **Apply Rule** to apply a data rule to a table. The Apply Data Rule Wizard is displayed. Use this wizard to select the data rule and the column to which the data rule should be applied.

After you apply a data rule to a table, it is listed on the Data rules tab. For the data rule to be applied to a table, ensure that the check box to the left of the data rule name is selected. Deselect this option if you do not want the data rule to be applied to the table.

**See Also:**

- "Overview of Data Rules" on page 19-1
- "Applying Data Rules to Data Objects" on page 19-7
- "Creating Data Rules Using the Create Data Rule Wizard" on page 19-5

## Editing Table Definitions

Use the Table Editor to edit table definitions. To open the editor, right-click the name of the table in the Projects Navigator and select **Open**. Alternatively, you can

double-click the name of the table in the Projects Navigator. This section describes the table definitions that you can edit.

## Renaming Tables

Use one of the following methods to rename a table:

- On the Name tab of the table editor, click the **Name** field and enter the new name for the table. You can also modify the description stored in the **Description** field.

  Alternatively, select the table in the Projects Navigator to display the properties of the table in the Property Inspector. Edit the values of the Physical Name and Description properties.
- In the Projects Navigator, select the table you want to rename and press the **F2** key. The table name is highlighted. Enter the new table name and press the **Enter** key.
- In the Projects Navigator, right-click the table name and select **Rename**. The table name is highlighted. Enter the new name for the table and press the **Enter** key.

## Adding, Modifying, and Deleting Table Columns

Use the Columns tab of the Table Editor to add, modify, and remove table columns.

### Adding Columns

Navigate to the Columns tab. Click the **Name** field in an empty row and enter the details that define the new column. For more information, see "Columns Tab" on page 2-12.

### Modifying Columns

Use the Columns tab of the Table Editor to modify column definitions. You can modify any of the attributes of the column definition either by entering the new value or selecting the new value from a list. For more information, see "Columns Tab" on page 2-12.

### Deleting Columns

Navigate to the Columns tab. Right-click the gray cell to the left of the column name that you want to remove and select **Delete**.

## Adding, Modifying, and Deleting Table Constraints

Navigate to the Keys tab of the Table Editor.

For details on adding and editing constraints, see "Creating Constraints" on page 2-21 and "Editing Constraints" on page 2-23 respectively.

To delete a constraint, select the row that represents the constraint by clicking the gray cell to the left of the column name. Click **Delete** at the bottom of the tab.

## Adding, Modifying, and Deleting Attribute Sets

Use the Attribute Sets tab of the Table Editor to add, modify, or delete attribute sets in a table.

For details about adding attribute sets, see "Creating Attribute Sets" on page 2-34. See "Editing Attribute Sets" on page 2-34 for instructions on how to edit an attribute set.

To delete an attribute set, navigate to the Attribute Sets tab. Right-click the cell to the left of the attribute set that you want to remove and select **Delete**.

### Reordering Columns in a Table

By default, columns in a table are displayed in the order in which they are created. This order is also propagated to the DDL script generated to create the table. If this default ordering does not suit your application needs, or if you want to further optimize query performance, then you can reorder the columns.

**To change the position of a column:**

1. If the Table Editor is not already open for the table, then open the editor.

   You can do this by double-clicking the name of the table in the Projects Navigator. Alternatively, you can right-click the name of the table in the Projects Navigator and select **Open**.

2. On the Columns tab, select the gray square located to the left of the column name.

   The entire row is highlighted.

3. Use the buttons on the left of the Columns tab to move the column to the required position.

   The position of the column is now updated.

4. Close the Table Editor**.**

You must deploy the changed table definition in order for the new column position in the table to be saved in the workspace.

## Defining Views

You can define views and materialized views in Warehouse Builder. This section describes views. Views are used to simplify the presentation of data or restrict access to data. Often the data that users are interested in is stored across multiple tables with many columns. When you create a view, you create a stored query to retrieve only the relevant data or only data that the user has permission to access.

> **See Also:** "Defining Materialized Views" on page 2-18 for information about materialized views

A view can be defined to model a query on your target data. This query information is stored in the workspace. You can later use these definitions to generate .ddl scripts that can be deployed to create views in your target system.

This section contains the following topics:


- Creating View Definitions on page 2-15
- Editing View Definitions on page 2-17


## Creating View Definitions

A view definition specifies the query used to create the view, constraints, attribute sets, data rules, and metadata about the columns and data types used in the view. This information is stored in the workspace. You can generate the view definition to create .ddl scripts. These scripts can be deployed to create the physical views in your database.

### Before You Begin

Ensure that you create the target schema that will contain your view as described in "Creating Target Modules" on page 1-1.

**To define a view:**

1. From the Projects Navigator, expand the Databases node and then the Oracle node.
2. Expand the target module where you want to create the view, right-click **Views**, and select **New View**.

   *or*

   Right-click the target module where you want to create the view and select **New**. In the New Gallery dialog box, select **View** and click **OK**.
3. In the Create View dialog box, enter a name and an optional description for the view and click **OK**.

   The View Editor is displayed.

   > **Note:** The name and description must follow the naming conventions listed in "Naming Conventions for Data Objects" on page 2-8.

4. Provide information on the following tabs of the View Editor:
   - Columns Tab on page 2-16
   - Query Tab on page 2-16
   - Keys Tab on page 2-17
   - Attribute Sets Tab on page 2-17
   - Data Rules Tab on page 2-17

   Warehouse Builder creates a definition for the view, stores this definition in the workspace, and adds this view name in the Projects Navigator.

   > **Note:** You can also define a view from the Graphical Navigator.

### Name Tab

Use the Name tab to modify the name and description that you provided in the Create View dialog box. Ensure that the name and description follow the rules listed in "Naming Conventions for Data Objects" on page 2-8.

### Columns Tab

Use the Columns tab to define the columns in the view. For each view column, enter the following details: Name, Data Type, Length, Precision, Scale, Seconds Precision, Not Null, Default Value, Virtual, Expression, and Description. For more information about each detail, see "Columns Tab" on page 2-12.

### Query Tab

Use the Query tab to define the query used to create the view. A view can contain data from tables that belongs to a different module than the one to which the view belongs. You can also combine data from more than one table using joins. Ensure that the query statement you type is valid. Warehouse Builder does not validate the text in the Query tab and will attempt to deploy a view even if the syntax is invalid.

### Keys Tab

Use the Keys tab to define logical constraints for a view. Although these constraints are not used when enumerating DDL for the view, they can be useful when the view serves as a data source in a mapping. The Mapping Editor can use the logical foreign key constraints to include the referenced dimensions as secondary sources in the mapping. For more information about creating constraints, see "Creating Constraints" on page 2-21.

**Note:** You cannot create check constraints for views.

### Attribute Sets Tab

Use the Attribute Sets tab to define attribute sets for the view. For more information about attribute sets and how to create them, see "Defining Attribute Sets" on page 2-33.

### Data Rules Tab

Use the Data Rules tab to specify the data rules that are applied to the view. Data rules help to ensure data quality by defining the legal data within a table, or legal relationships between tables. For more information about the Data Rules tab, see "Data Rules Tab" on page 2-13.

## Editing View Definitions

Use the View Editor to edit view definitions. To open the View Editor, right-click the view in the Projects Navigator and select **Open**. The following sections describe the view definitions that you can edit.

### Renaming Views

Use the Name tab of the View Editor to rename views. Click the **Name** field and enter the new name for the view. You can also modify the description stored in the **Description** field. Enter the new name over the highlighted object name. Alternatively, select the view in the Projects Navigator to display the view properties in the Property Inspector. Edit the values of the Physical Name and Description properties.

### Adding, Modifying, and Deleting View Columns

Use the Columns tab to add, modify, or delete view columns. For more information, see "Columns Tab" on page 2-16.

**Adding columns:** On the Columns tab, click the **Name** field in an empty row and enter the details that define a column.

**Editing columns:** Use the Columns tab of the Table Editor to modify column definitions. You can modify any of the attributes of the column definition.

**Removing columns:** On the Columns tab, right-click the gray cell to the left of the column name that you want to remove and select **Delete**.

### Adding, Modifying, and Deleting View Constraints

Use the Keys tab of the View Editor to add, modify, or delete view constraints. For details on adding and editing constraints, see "Creating Constraints" on page 2-21 and "Editing Constraints" on page 2-23 respectively.

To delete constraints, on the Keys tab, select the row that represents the constraint. Click **Delete** at the bottom of the tab.

### Adding, Modifying, and Deleting Attribute Sets

Use the Attribute Sets tab of the View Editor to add, modify, or delete attribute sets. For details about adding attribute sets, see "Creating Attribute Sets" on page 2-34. See "Editing Attribute Sets" on page 2-34 for instructions on how to edit an attribute set.

To delete an attribute set, navigate to the Attribute Sets tab. Right-click the cell to the left of the attribute set that you want to remove and select **Delete**.

# Defining Materialized Views

Materialized views improve query performance. When you create a materialized view, you create a set of query commands that aggregate or join data from multiple tables. Materialized views provide precalculated data that can be reused or replicated to remote data marts. For example, data about company sales is widely sought throughout an organization.

When you create a materialized view, you can configure it to take advantage of the query rewrite and fast refresh features available in Oracle Database. For information about query rewrite and fast refresh, see "Fast Refresh for Materialized Views" on page 2-53.

## Creating Materialized View Definitions

A materialized view definition specifies the query used to create the materialized view, constraints, indexes, partitions, attribute sets, data rules, and metadata about the columns and data types used in the materialized view. You can generate the view definition to obtain .ddl scripts that are used to deploy the materialized view.

### Before You Begin

Create the target schema that will contain your materialized view, as described in "Creating Target Modules" on page 1-1.

**To define materialized views:**

1. From the Projects Navigator, expand the Databases node and then the Oracle node.
2. Expand the target module where you want to create the materialized view, right-click **Materialized Views**, and select **New Materialized View**.

   *or*

   Right-click the target module where you want to create the view and select **New**. In the New Gallery dialog box, select **Materialized View** and click **OK**.
3. In the Create Materialized View dialog box, enter a name and an optional description for the materialized view and click **OK**.

   The name and description must follow the naming conventions listed in "Naming Conventions for Data Objects" on page 2-8.
4. Provide information on the following tabs of the Materialized View Editor:
   - Columns Tab on page 2-19
   - Query Tab on page 2-19
   - Keys Tab on page 2-19
   - Indexes Tab on page 2-19
   - Partitions Tab on page 2-19

- Attribute Sets Tab on page 2-19
- Data Rules Tab on page 2-20

Warehouse Builder creates a definition for the materialized view, stores this definition in the workspace, and inserts its name in the Projects Navigator.

**Note:** You can also define a materialized view from the Graphical Navigator.

### Columns Tab

Use the Columns tab to define the materialized view columns. For each column, specify the following details: Name, Data Type, Length, Precision, Scale, Seconds Precision, Not Null, Default Value, and Description. For more information about the details to be provided for each materialized view column, see "Columns Tab" on page 2-12.

### Query Tab

Use the Query tab to define the query used to create the materialized view. Ensure that you type a valid query in the **Select Statement** field. For column names, use the same names that you specified on the Columns Tab. If you change a column name on the columns page, then you must manually change the name in the Query tab. Warehouse Builder does not validate the text in the Query tab and will attempt to deploy a materialized view even if the syntax is invalid.

### Keys Tab

Use the Keys tab to define constraints for the materialized view. Defining constraints is optional. These constraints are for logical design purposes only and are not used when enumerating DDL for the materialized view. You can create primary keys, foreign keys, and unique keys. For information about creating constraints, see "Creating Constraints" on page 2-21.

**Note:** You cannot create check constraints for materialized views.

### Indexes Tab

Use the Indexes tab to define indexes on the materialized view. Defining indexes is optional. You can create the following types of indexes: Unique, non-Unique, Bitmap, Function-based, Composite, and Reverse. For information about creating indexes, see "Creating Indexes" on page 2-24.

### Partitions Tab

Use the Partitions tab to define partitions on the materialized view. Partitioning a materialized view is optional. You can perform Index Partitioning, Range Partitioning, Hash Partitioning, Hash by Quantity Partitioning, List Partitioning, or Composite Partitioning.

### Attribute Sets Tab

Use the Attribute Sets tab to define attribute sets for the materialized view. Defining attribute sets is optional. For information about how to define attribute sets, see "Creating Attribute Sets" on page 2-34.

### Data Rules Tab

Use the Data Rules tab to specify data rules that must be applied to the materialized view data. For more information, see "Data Rules Tab" on page 2-13.

## Editing Materialized View Definitions

Use the Materialized View Editor to edit a materialized view definition. To open the Materialized View Editor, right-click the materialized view and select **Open**. The following sections describe the type of editing operations that you can perform on a materialized view.

### Renaming Materialized Views

Double-click the **Name** field on the Name tab of the editor. This selects the name. Type the new name.

Alternatively, select the materialized view in the Projects Navigator to display the materialized view properties in the Property Inspector. Edit the values of the Physical Name and Description properties.

### Adding, Modifying, and Deleting Materialized View Columns

Use the Columns tab to add, modify, or delete materialized view columns.

**Adding columns:** On the Columns tab, click the Name field in an empty row and enter the details for the column. For more information about these details, see "Columns Tab" on page 2-16.

**Removing columns:** On the Columns tab, right-click the gray cell to the left of the column name that you want to remove and select **Delete**.

### Adding, Modifying, and Deleting Materialized View Constraints

Use the Keys tab of the Materialized View Editor to add, modify, or delete materialized view constraints. For details on adding and editing constraints, see "Creating Constraints" on page 2-21 and "Editing Constraints" on page 2-23 respectively.

To delete a constraint, on the Keys tab, select the row that represents the constraint. Click **Delete** at the bottom of the tab.

### Adding, Modifying, and Deleting Attribute Sets

Use the Attribute Sets tab to add, modify, or delete attribute sets in a materialized view. For details about adding attribute sets, see "Creating Attribute Sets" on page 2-34. See "Editing Attribute Sets" on page 2-34 for instructions on how to edit an attribute set.

To delete an attribute set, on the Attribute Sets tab, right-click the cell to the left of the attribute set that you want to remove and select **Delete**.

# Defining Constraints

You can optionally create constraints on relational data objects such as tables, views, and materialized views.

## About Constraints

Constraints are used to enforce the business rules that you want to associate with the information in a database. Constraints prevent the entry of invalid data into tables.

Business rules specify conditions and relationships that must always be true, or must always be false.

For example, if you define a constraint for the `salary` column of the `employees` table as Salary < 10000, then this constraint enforces the rule that no row in this table can contain a numeric value greater than 10000 in this column. If an `INSERT` or `UPDATE` statement attempts to violate this integrity constraint, then an error message is displayed. Remember that constraints slow down load performance.

You can define the following constraints for tables, views, and materialized views:

- **Unique Key (UK):** A UK constraint requires that every value in a column or set of columns (key) be unique. No two rows of a table can have duplicate values in a specified column or set of columns. A UK column can also contain a null value.
- **Primary Key (PK):** A value defined on a key (column or set of columns) specifying that each row in the table can be uniquely identified by the values in the key (column or set of columns). No two rows of a table can have duplicate values in the specified column or set of columns. Each table in the database can have only one PK constraint. A PK column cannot contain a null value.
- **Foreign Key (FK):** A rule defined on a key (column or set of columns) in one table that guarantees that the values in that key match the values in a PK or UK key (column or set of columns) of a referenced table.
- **Check Constraint:** A user-defined rule for a column (or set of columns) that restricts inserts and updates of a row based on the value it contains for the column (or set of columns). A Check condition must be a Boolean expression that is evaluated using the values in the row being inserted or updated.

  For example, the condition *Order Date < Ship Date* checks that the value of the Order Date column is always less than that of the Ship Date column. If not, there is an error when the table is loaded and the record is rejected. A check condition cannot contain subqueries and sequences or `SYSDATE`, `UID`, `USER`, or `USERENV` SQL functions. Although check constraints are useful for data validation, they slow load performance.

## Creating Constraints

Use the Keys tab of the object editors to create constraints. You can create the following types of constraints: primary key, foreign key, unique key, and check constraints.

**To create constraints on a table, view, or materialized view:**

1. Open the editor for the data object to which you want to add constraints.

   In the Projects Navigator, double-click the data object on which you want to define a constraint. Alternatively, you can right-click the data object in the Projects Navigator and select **Open**.

2. Navigate to the **Keys** tab.
3. Depending on the type of constraint that you want to define, see one of the following sections:
   - Defining Primary Key Constraints on page 2-22
   - Defining Foreign Key Constraints on page 2-22
   - Defining Unique Key Constraints on page 2-22
   - Defining Check Constraints on page 2-23

### Defining Primary Key Constraints

**To define a primary key constraint:**

1. On the Keys tab, click the **Add Constraint** button.

   A blank row is displayed with the cursor positioned in the Name column.

2. Enter the name of the constraint in the **Name** column.

   Constraint names must be unique within the module that contains the data object on which the constraint is defined.

3. In the **Type** column, select **Primary Key**.

   Press the Tab key to exit from the Type column or use the mouse and click the empty space in the Keys tab.

4. Click **Add Local Column**.

   A new row is added below the current row that contains the constraint name and constraint type. This new row displays a list in the Local Columns column.

5. In the **Local Columns** list of the new row, select the name of the column that represents the primary key.

6. (Optional) To create a composite primary key, repeat Steps 4 and 5 for each column that you want to add to the primary key.

### Defining Foreign Key Constraints

**To define a foreign key constraint:**

1. On the Keys tab, click the **Add Constraint** button.

   A blank row is displayed with the cursor positioned in the Name field.

2. Enter the name of the constraint in the **Name** column.

   Constraint names must be unique within the module that contains the data object on which the constraint is defined.

3. In the **Type** column, select **Foreign Key**.

   Press the tab key to navigate out of the Type column or use the mouse and click the empty space in the Keys tab.

4. In the References column, click the Ellipsis button.

   The Key Selector dialog box is displayed.

5. In the Key Selector dialog box, select the primary key constraint that the foreign key being created references.

   For example, the `DEPARTMENTS` table has a primary key called `DEPT_PK` defined on the `department_id` column. To specify that the column `department_id` of the `EMPLOYEES` table is a foreign key that references the primary key `DEPT_FK`, select `DEPT_FK` under the node that represents the `DEPARTMETNS` table in the Key Selector dialog box.

6. Click **OK**.

### Defining Unique Key Constraints

**To define a unique key constraint:**

1. On the Keys tab, click the **Add Constraint** button.

A blank row is displayed with the cursor positioned in the Name field.

2. Enter the name of the constraint in the **Name** column and press the **Enter** key.

   You can also press the Tab key or click any other location in the editor.

   Constraint names must be unique within the module that contains the data object on which the constraint is defined.

3. In the **Type** column, select **Unique Key**.

   Press the tab key to navigate out of the Type column or use the mouse and click the empty space in the Keys tab.

4. Click **Add Local Column**.

   A new row is added below the current row that contains the constraint name and constraint type. This new row displays a list in the Local Columns column.

5. In the **Local Columns** list of the new row, select the name of the column on which a unique key should be created.

6. (Optional) To create a composite unique key, repeat steps 4 and 5 for each column that you want to add to the unique key.

### Defining Check Constraints

**To define a check constraint:**

1. On the Keys tab, click the **Add Constraint** button.

   A blank row is displayed with the cursor positioned in the Name field.

2. Enter the name of the constraint in the **Name** column and press the **Enter** key.

   You can also press the Tab key or click any other location in the editor.

   Constraint names must be unique within the module that contains the data object on which the constraint is defined.

3. In the **Type** column, select **Check Constraint**.

   Press the tab key to exit from the Type column or use the mouse and click the empty space in the Keys tab.

4. In the Check Condition column, enter the condition to be applied for the check constraint. For example, salary > 2000. If you leave this field blank, then an error is generated during validation and you cannot generate valid code for this constraint.

   The column name referenced in the check condition must exactly match the physical column name defined in the table. Warehouse Builder does not check the syntax of the condition during validation. This may result in errors during deployment. If this happens, then check the Repository Browser for details.

## Editing Constraints

You can edit constraints using the Keys tab of the object editors to accomplish the following tasks:

- Rename a constraint
- Change the constraint type
- Modify the check condition
- Modify the referenced column for a foreign key constraint

- Modify the primary key column for a primary key

After editing constraint definitions, ensure that you regenerate and redeploy the data object containing the modified constraints.

# Defining Indexes

Use indexes to enhance query performance of your data warehouse. In Warehouse Builder, you can define indexes for tables and materialized views. In the following sections, the word *table* refers to all objects for which you can define indexes.

Indexes are important for speeding queries by quickly accessing data processed in a warehouse. You can create indexes on one or more columns of a table to speed SQL statement execution on that table. Indexes have the following characteristics:

- Indexes provide pointers to the rows in a table that contain a given key value.
- Index column values are stored presorted.
- Because the database stores indexes in a separate area of the database, you can create and drop indexes at any time without affecting the underlying table.
- Indexes are independent of the data in the table. When you delete, add, or update data, the indexes are maintained automatically.

> **See Also:** *Oracle Database Data Warehousing Guide* for more information about indexing strategies

## Creating Indexes

You create indexes by using the Indexes tab in the editor. To start the editor, navigate to the table or other data object on the Projects Navigator and double-click it, or right-click and select **Open.** When you select an index type, Warehouse Builder displays the appropriate template enabling you to create the index. Index names must be unique within the module that contains the data object on which the indexes are defined.

For all types of indexes except bitmap indexes, you can determine whether the index inherits the partitioning method of the underlying table. An index that inherits its partitioning method is known as a *local index* whereas an index with its own partitioning method is known as a *global index.* For additional information, see "Index Partitioning" on page 2-32.

You can create the following types of indexes in Warehouse Builder:

- **Unique:** These indexes ensure that no two rows of a table have duplicate values in the key column or composite key columns.
- **Non-Unique:** These are B-tree type indexes that do not impose restrictions against duplicate values in the key column or composite key columns.
- **Bitmap:** These indexes are primarily used for data warehousing applications to enable the querying of large amounts of data. These indexes use bitmaps as key values instead of a list of row IDs. Bitmaps are effective when the values for the index key comprise a small list. For example, `AGE_GROUP` could be a good index key but `AGE` would not.

  Bitmaps enable star query transformations, which are cost-based query transformations aimed at efficiently executing star queries. A prerequisite of the star transformation is that a bitmap index must be built on each of the foreign key columns of the cube or cubes.

When you define a bitmap index in Warehouse Builder, set its scope to LOCAL and partitioning to NONE.

- **Function-based:** These indexes compute and store the value of a function or expression that you define on one or more columns in a table. The function can be an arithmetic expression that contains a PL/SQL function, package function, C callout, or SQL function.
- **Composite:** Also known as concatenated indexes, these are indexes on multiple columns. The columns can be in any order in the table and need not be adjacent to each other.

  To define a composite index in Warehouse Builder, create the index as you would any other index and assign between 2 and 32 index columns.
- **Reverse:** For each indexed column except for the row ID column, this index reverses the bytes in the columns. Because the row ID is not reversed, this index maintains the column order.

  To define a reverse index in Warehouse Builder, create the index as you would any other index and then go to the Configurations tab of the data object and set the Index Sorting parameter listed under the Performance Parameters to REVERSE.

## Defining Partitions

Partitions enable you to efficiently manage very large tables and indexes by dividing them into smaller, more manageable parts. Partitions improve query and load performance because operations work on subsets of data. Use partitions to enhance data access and improve overall application performance, especially for applications that access tables and indexes with millions of rows and many gigabytes of data.

In Warehouse Builder, you can define partitions for tables, indexes, materialized views, and MOLAP cubes. For brevity, in the following sections, the word *table* is used to refer to all objects for which you can define partitions. The following sections discuss partitioning for all the objects previously listed except partitioning MOLAP cubes, which is described separately.

You define partitions for these objects by using the Partitions tab in the object editors. Depending on the type of partition that you create, you may also need to configure tablespaces for the partitions in the Configuration tab.

You can perform the following types of partitioning:

- **Range Partitioning**: Use range partitioning to create partitions based on a range of values in a column. When you use range partitioning with a date column as the partition key, you can design mappings that instantly update target tables, as described in "Improved Performance through Partition Exchange Loading" on page 10-21.
- **Hash Partitioning**: Use hash partitioning to direct Oracle Database to evenly divide the data across a recommended even number of partitions. This type of partitioning is useful when data is not historical and there is no obvious column or column list.
- **Hash by Quantity Partitioning**: To quickly define hash partitioning, use Hash by quantity partitioning. This is the same as hash partitioning except that you specify only a partition key and the number of partitions. The partitions are created and named automatically. You can then configure the partitions to share the same tablespace list.

- **List Partitioning**: Use list partitioning to explicitly assign rows to partitions based on a partition key that you select. This enables you to organize the data in a structure not available in the table.
- **Composite Partitioning**: You can use Warehouse Builder to specify a composite of either range-hash, range-hash by quantity, or range-list partitioning. Oracle Database first performs the range partitioning and then further divides the data using the second partitioning that you select. For example, in range-list partitioning, you can base partitions on the sales transaction date and then further divide the data based on lists of states where transactions occurred.
- **Index Partitioning**: You can define an index that inherits the partitioning method of its underlying table. Or, you can partition an index with its own partitioning strategy.

Depending on the partition type that you create, the Partitions tab is dynamically altered to display only the relevant information. Columns or rows on this tab that are not required for a particular partition type are hidden.

For example, when you create a range or list partition, since you cannot create subpartitions, the rows Subpartition Key and Subpartition Template are hidden. When you create a range-hash partition, the Subpartition Template row contains an entry for Hash Subpartition Quantity with the condition "=".

The conditions that define the upper bound for subpartitions depend on the type of partitioning method used. For example, for a range-list partition, the condition allowed for determining the upper bound for a partition must be based on equality. Thus, the column that contains the condition (the column between the Partition and Value columns) contains "=" and is disabled. However, the condition for determining the upper bound for the subpartition is displayed as "<" and you cannot edit this field.

## Range Partitioning

Range partitioning is the most common type of partitioning and is often used to partition data based on date ranges. For example, you can partition sales data into monthly partitions.

To use range partitioning, go to the Partitions tab in the editor to specify a partition key and assign a name and value range for each partition you want to create.

**To partition data by range:**

1. On the Partitions tab in the object editor, click the cell under **Type** and select **Range.**

   If necessary, click the plus sign to the left of Type to expand the template for the range partition.

2. Select a partition key under the Partition Key node.

   Warehouse Builder lists all the columns for the object you selected under Key Columns. You can select a column of any data type; however, `DATE` is the most common partition key for range partitioning.

   You can base the partition key on multiple key columns. To add another key column, select the partition key node and click **Add**.

3. Define the partitions under the Partitions node.

   To assist you in defining the partitions, the template offers two partitions that you can edit but not delete. P1 represents the first partition and PDEFAULT represents

the last partition. If you want to partition data based on month, then you can rename P1 to *Jan* and PDEFAULT to *Dec.*

The last partition is set to the keyword MAXVALUE, which represents a virtual infinite value that sorts higher than any other value for the data type, including the null value.

To add more partitions between the first and last partitions, click the Partitions node and select **Add.**

In **Values,** specify the greatest value for the first range and all the additional ranges that you create. These values are the less than values.

### Example of Range Partitioning

Figure 2–1 shows how to define a partition for each quarter of a calendar year.

***Figure 2–1 Example Table with Range Partitioning***

| | Type | Key Columns | Partitions | | Values |
|---|---|---|---|---|---|
| ⊟ | Range | | | | |
| ⊟ | Partition Key | | | | |
| | | SALE_DATE | | | |
| ⊟ | Partition Interval | | | | |
| | | | Partition Interval | = | |
| ⊟ | Partitions | | | | |
| | | | QTR1 | < | DATE '2005-04-01' |
| | | | QTR2 | < | DATE '2005-07-01' |
| | | | QTR3 | < | DATE '2005-10-01' |
| | | | QTR4 | < | MAXVALUE |

You can also partition data for each month or week. When you design mappings using such a table, consider enabling Partition Exchange Loading (PEL). PEL is a data definition language (DDL) operation that swaps existing partitions on the target table with new partitions. Because it is not a data manipulation language (DML) operation, the exchange of partitions occurs instantaneously.

## Hash Partitioning

Hash partitioning assigns data to partitions based on a hashing algorithm that Oracle Database applies to a partitioning key you identify. The hashing algorithm evenly distributes rows among partitions, giving partitions approximately the same size. Hash partitioning is a good and easy-to-use alternative to range partitioning when data is not historical and there is no obvious column or column list where logical range partition pruning can be advantageous.

**To partition data based on the hash algorithm:**

1. On the Partitions tab in the object editor, click the cell below **Type** and select **Hash.**

   If necessary, click the plus sign to the left of Type to expand the template for defining hash partitions.

2. Select a partition key in the Key Columns column under the Partition Key node.

   Warehouse Builder lists all the columns for the object you selected. You can select a column of any data type.

3. Define the partitions under the Partitions node.

Warehouse Builder provides two partitions that you can rename. Click the Partitions node and select **Add** to add as many partitions as necessary.

Oracle Database uses a linear hashing algorithm. To prevent data from clustering within specific partitions, you should define the number of partitions by a power of two (for example, 2, 4, 8).

## Hash by Quantity Partitioning

Use hash by quantity partitioning to quickly define hash partitioning. When you define a partition key and specify the number of partitions, the partitions are automatically created and named. You can then configure the partitions to share the same tablespace list.

**To partition data based on the hash by quantity algorithm:**

1. On the Partitions tab in the object editor, click the cell below **Type** and select **Hash by Quantity.**

   If necessary, click the plus sign to the left of Type to expand the template for defining hash by quantity partitions.

2. Define the partition key using the Partition Key column under the Partition Key node.

3. Define the number of partitions in the Values column under the Partitions node. The default value is two partitions.

   Oracle Database uses a linear hashing algorithm and, to prevent data from clustering within specific partitions, you should define the number of partitions by a power of two (for example, 2, 4, 8).

4. In the Configuration tab, define the Partition Tablespace List and Overflow Tablespace List.

   To display the Configuration tab for a data object, right-click the data object and select **Configure**.

## List Partitioning

List partitioning enables you to explicitly assign rows to partitions. You can achieve this by specifying a list of discrete values for each partition. The advantage of list partitioning is that you can group and organize unordered and unrelated sets of data in a natural way.

Figure 2–2 shows an example of a table partitioned into list partitions based on the instructions described below.

**To partition data based on a list of values:**

1. On the Partitions tab in the object editor, click the cell below **Type** and select **List.**

   If necessary, click the plus sign to the left of Type to expand the template for defining list partitions.

2. Select a partition key using the Key Columns column under the Partition Key node.

   Warehouse Builder lists all the columns for the object you selected. You can select a column of any data type.

3. Define the partitions under the Partitions node.

PDEFAULT is set to the keyword DEFAULT and includes all rows not assigned to any other partition. A partition that captures all unassigned rows is essential for maintaining the integrity of the data.

To assist you in defining the partitions, the template offers two partitions that you can edit but not delete. P1 represents the first partition and PDEFAULT represents the last partition.

To add more partitions between the first and last partitions, click the Partitions node and select **Add.**

In **Values,** enter a comma-separated list of values for each partition that corresponds to data in the partition key you previously selected. For example, if the partition key is `COUNTRY_ID`, then you can create partitions for Asia, Eastern Europe, Western Europe, and so on. Then, for the values for each partition, list the corresponding COUNTRY_IDs for each country in the region.

### Example of List Partitioning

Figure 2–2 shows a table with data partitioned into different regions by using list partitioning based on the `COUNTRY_ID` column. Each partition has a single comma-separated list.

***Figure 2–2 List Partitioning Based on a Single Key Column***

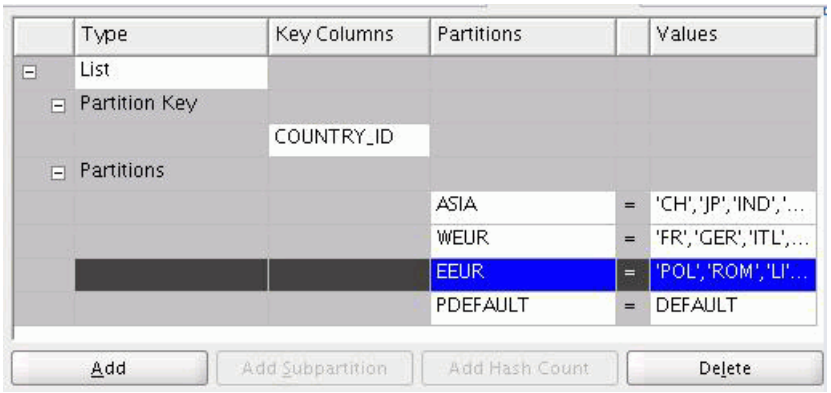
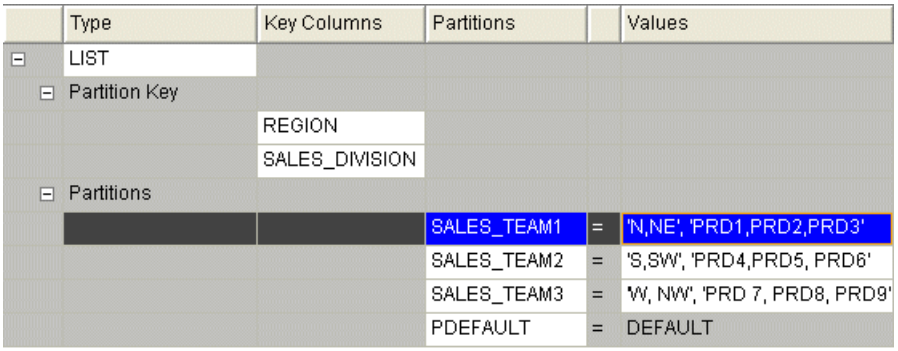



Figure 2–3 shows a table with data partitioned based on key columns REGION and SALES_DIVISION. Each partition includes two comma-separated lists enclosed by single quotation marks. In this example, N, NE, S, SW, W, and NW correspond to REGION while PRD1, PRD2, PRD3, and so on correspond to SALES_DIVISION.

***Figure 2–3 List Partitioning Based on Multiple Key Columns***



## Composite Partitioning

Composite partitioning methods include range-hash, range-hash by quantity, and range-list partitioning. Oracle Database first performs the range partitioning and then further divides the data using the second partitioning that you select.

The steps for defining composite partition methods are similar to those used to define simple partition methods such as (range, hash, and list) but include additional options.

**To partition data based on range and then subpartition based on list, hash, or hash by quantity:**

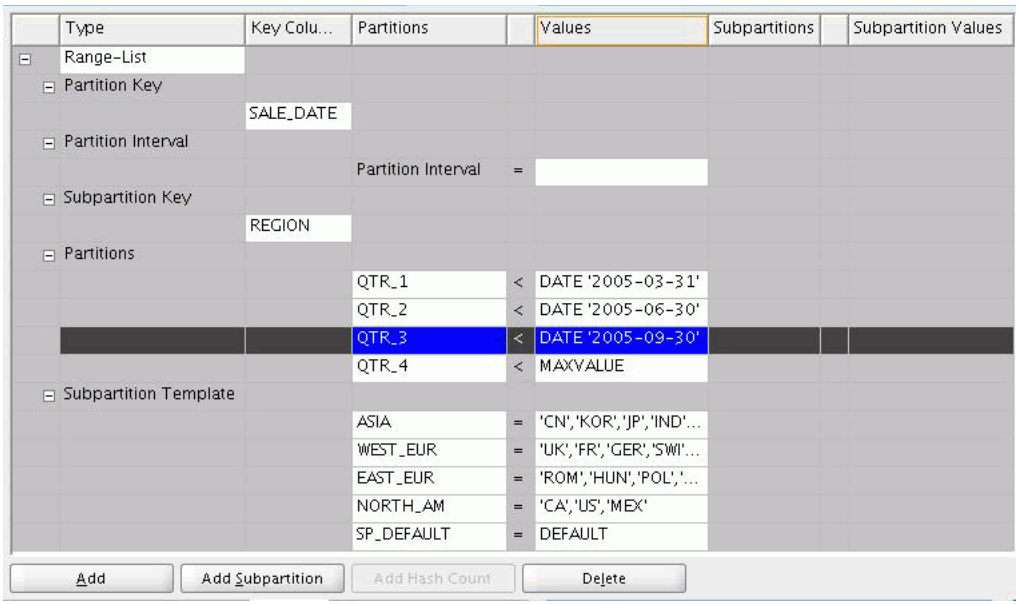
1. On the Partitions tab in the object editor, click the cell below **Type** and select one of the composite partitioning methods.

   If necessary, click the plus sign to the left of Type to expand the template.

2. Select a partition key and define partitions as described in "Range Partitioning" on page 2-26.

   In Figure 2–4, the partition key is `SALE_DATE` and its associated range partitions are QTR_1, QTR_2, QTR_3, and QTR_4.

***Figure 2–4 Range-List Partitioning with List Defined Under a Subpartition Template***




3. Select a column for the subpartition key in the Key Columns list under the Subpartition Key node.

4. Under the Subpartition Template node, define the values for the second partitioning method as described in "About the Subpartition Template" on page 2-31.

5. (Optional) Define custom subpartitions.

   For range-list partitions, you can specify custom subpartitions that override the defaults you defined under the subpartition node. For details, see "Creating Custom Subpartitions" on page 2-31.

6. Configure the Partition Tablespace List and Overflow Tablespace List in the Configuration tab.

To display the Configuration tab for a data object, right-click the data object and select **Configure**.

### About the Subpartition Template

Use the subpartition template to specify the second partitioning method in composite partitioning. The steps you take depend on the type of composite partition you select.

For range-hash by quantity, enter the number of subpartitions only.

For range-hash, the subpartition template enables you to enter names for the subpartitions only.

For range-list, name the lists and enter comma-separated values. Be sure to preserve the last subpartition as set to DEFAULT.

Figure 2–4 shows a list subpartition based on the `REGION` key column and subpartitions for groups of countries. Warehouse Builder divides each partition (such as QTR_1 and QTR_2) into subpartitions (such as ASIA and WEST_EUR).

### Creating Custom Subpartitions

Using the subpartition template is the most convenient and likely the most common way to define subpartitions. Entries that you specify under the Subpartition Template node apply uniformly to all the partitions under the partition node. However, in some cases, you may want to override the subpartition template.

For range-hash by quantity, select a partition and then click **Add Hash Count.** Warehouse Builder expands the partition node to enable you to specify the number of hash subpartitions that uniquely apply to that partition.

For range-hash, select a partition and then click **Add Subpartition.** Warehouse Builder expands the partition node and you can name subpartitions for that partition only.

For range-list, select a partition and then click **Add Subpartition.** Warehouse Builder expands the partition node to enable you to specify list subpartitions for that partition only. Be sure to preserve the last subpartition as set to DEFAULT.

Figure 2–5 shows that partition QTR_1 is subpartitioned into lists for UK, EUR, and ALL_OTHERS whereas the other quarters are partitioned according to the subpartition template.

*Figure 2–5 Subpartitions Overriding Subpartition Template*

| Type | Key Colu... | Partitions | | Values | Subpartitions | | Subpartition Va... |
|---|---|---|---|---|---|---|---|
| Range-List | | | | | | | |
| Partition Key | | | | | | | |
| | SALE_DATE | | | | | | |
| Partition Interval | | | | | | | |
| | | Partition Interval | = | | | | |
| Subpartition Key | | | | | | | |
| | REGION | | | | | | |
| Partitions | | | | | | | |
| | | QTR_1 | < | DATE '2005-03-31' | | | |
| | | | | | UK | = | 'GB','IRE','SCOT' |
| | | | | | EUR | = | 'FR','GR','SP','POR' |
| | | | | | ALL_OTHERS | = | DEFAULT |
| | | QTR_2 | < | DATE '2005-06-30' | | | |
| | | QTR_3 | < | DATE '2005-09-30' | | | |
| | | QTR_4 | < | MAXVALUE | | | |
| Subpartition Template | | | | | | | |
| | | ASIA | = | 'CN','KOR','JP','IND'... | | | |
| | | WEST_EUR | = | 'UK','FR','GER','SWI'... | | | |
| | | EAST_EUR | = | 'ROM','HUN','POL','... | | | |
| | | NORTH_AM | = | 'CA','US','MEX' | | | |
| | | SP_DEFAULT | = | DEFAULT | | | |

Add | Add Subpartition | Add Hash Count | Delete

## Index Partitioning

For all types of indexes except bitmap indexes, you can determine whether the index inherits the partitioning method of the underlying table. An index that inherits the partitioning method of the underlying table is known as a *local index*. An index with its own partitioning method is known as a *global index*.

### Local Index

Local indexes are partitioned on the same columns and have the same definitions for partitions and subpartitions as specified on the underlying table. Furthermore, local indexes share the same tablespaces as the table.

For example, if you used range-list partitioning to partition a table of sales data by quarter and then by region, then a local index is also partitioned by quarter and then by region.

Bitmap indexes can only be defined as local indexes to facilitate the best performance for querying large amounts of data.

To define an index as local in Warehouse Builder set the **Scope** to LOCAL and **Partitioning** to NONE.

### Global Index

A global index is one in which you can partition the index independently of the partition strategy applied to the underlying table. You can choose between range or hash partitioning. The global index option is available for all indexes except bitmap indexes.

In releases before Oracle Database 10*g*, Oracle recommended that you not use global indexes for data warehouse applications because deleting partitions on the table during partition maintenance would invalidate the entire index and result in having to

rebuild the index. Beginning with Oracle Database 10*g*, this is no longer a limitation, as global indexes are no longer negatively affected by partitioning maintenance.

Nonetheless, local indexes are likely to be the preferred choice for data warehousing applications due to ease in managing partitions and the ability to parallelize query operations.

A global index is useful when you want to specify an index partition key other than any of the table partition keys. For a global index, ensure that there are no duplicate rows in the index key column and select unique for the index type.

### Index Performance Considerations

As you decide which type of index to use, consider that indexes rank in performance in the following order:

1. Unique and local index
2. Unique and global index
3. All other non-unique indexes (normal, bitmap, function-based) and local index.

## Configuring Partitions

For some but not all partitioning methods, you must configure partition tablespaces.

You can access the parameters for partitions using the Projects Navigator. Right-click the table and select **Configure** to display the Configuration tab for the table. Scroll down to view the Partition Parameters node.

### Partition Tablespace List

Enter a comma-separated list of tablespaces when you partition by any of the following methods: hash by quantity, range-list, range-hash, or range-hash by quantity.

If you neglect to specify partition tablespaces, then the default tablespaces associated with the table are used and the performance advantage for defining partitions is not realized.

### Overflow Tablespace List

Enter a comma-separated list of tablespaces when you partition by the method hash by quantity. If you provide a list of tablespaces less than the number of partitions, then the Oracle Database cycles through those tablespaces.

If you neglect to specify overflow tablespaces, then the default tablespaces associated with the table are used and the performance advantage for defining partitions is not realized when the limits for the partition tablespaces are exceeded.

# Defining Attribute Sets

An attribute set contains a chosen set of columns. Attribute sets are useful while defining a mapping or during data import and export. Warehouse Builder enables you to define attribute sets for tables, views, and materialized views. For brevity, in the following sections, the word *table* is used to refer to all objects for which you can define attribute sets

## Creating Attribute Sets

Use the Attribute Sets tab of the object editors to create attribute sets. You can define attribute sets for tables, views, and materialized views.

**To define an attribute set:**

1. From the Projects Navigator, right-click the object name in which the attribute set is to be defined and select **Open.**

   The object editor is displayed.

2. Select the Attribute Sets tab.

   This tab contains two sections: *Attribute sets* and *Attributes of the selected attribute set*.

   The *Attribute sets* section displays the attribute sets defined for the table. It contains two columns that define each attribute set: Name and Description.

   The *Attributes of the selected attribute set* section lists all the attributes in the table. The attributes that are selected using the Include column are the ones that are included in the attribute set that is selected in the *Attribute sets* section.

3. In the *Attribute sets* section, click the **Name** field of an empty row and enter a name for the attribute set.

   In physical mode, you must enter a name between 1 and 30 valid characters. Spaces are not allowed. In logical mode, you can enter up to 200 valid characters. The attribute set name must be unique within the object.

   Notice that all the table attributes are displayed in the *Attributes of the selected attribute set* section.

4. In the *Attributes of the selected attribute set* section, select **Include** for each attribute you want to include in the attribute set. The order in which you select the columns determines their initial order in the attribute set.

   Click **Select All** to select all the displayed columns in the attribute set. Click **Deselect All** to exclude all the columns from the attribute set. To remove a column from the attribute set, deselect **Include**.

## Editing Attribute Sets

Use the Attribute Sets tab of the object editor to edit attribute sets. Before you edit an attribute set, ensure that the editor is open for the object that contains the attribute set. Also, navigate to the Attribute Sets tab of the editor.

You can perform the following actions when you edit an attribute set:

- Rename the attribute set

  Click the name of the attribute set in the **Name** column of the **Attribute sets of the entity** section and enter the new name.

- Add or remove attributes from the attribute set

  **Adding attributes to an attribute set:** Select the attribute set to which you want to add attributes by clicking the gray cell to the left of the attribute set name in the *Attribute sets* section. In the *Attributes of the selected attribute set* section, select **Include** for each attribute that you want to include in the attribute set.

  **Removing attributes from an attribute set:** Select the attribute set from which you want to remove attributes by clicking the gray cell to the left of the attribute set. In

the *Attributes of the selected attribute set* section, unselect **Include** for the attributes that you want to remove from the attribute set.

- Delete the attribute set

  In the *Attribute Sets* section, right-click the gray cell to the left of the attribute set name and select **Delete**.

# Defining Sequences

A sequence is a database object that generates a serial list of unique numbers. You can use sequences to generate unique primary key values and to coordinate keys across multiple rows or tables. Sequence values are guaranteed to be unique. When you create a sequence, you are creating sequence definitions that are saved in the workspace. Sequence definitions can be used in mappings to generate unique numbers while transforming and moving data to your target system.

The following sections provide information about using sequences:


- "About Sequences" on page 2-35
- "Creating Sequence Definitions" on page 2-35
- "Editing Sequence Definitions" on page 2-35


## About Sequences

A sequence is referenced in SQL statements with the NEXTVAL and CURRVAL pseudocolumns. Each new sequence number is incremented by a reference to the pseudocolumn NEXTVAL, whereas the current sequence number is referenced using the pseudocolumn CURRVAL. These attributes are created when you define a sequence.

You can also import sequence definitions from existing source systems using the Metadata Import Wizard.

## Creating Sequence Definitions

**To create a sequence:**

1. From the Projects Navigator, expand the Databases node and then the target module node.
2. Right-click Sequences and select **New Sequence** from the menu.

   Warehouse Builder displays the Create Sequence dialog box.
3. Use the Name field to specify a name and the Description field to specify an optional description for the sequence.

   In addition to the rules listed in "Naming Conventions for Data Objects" on page 2-8, the name must be unique across the module.
4. Click **OK.**

   Warehouse Builder stores the definition for the sequence and inserts its name in the Projects Navigator.

## Editing Sequence Definitions

Use the Edit Sequence dialog box to edit a sequence definition. You can edit the name, description, and column notes of a sequence.

To edit sequence properties, right-click the name of the sequence from the Projects Navigator and select **Open.** Or double-click the name of the sequence. The Edit Sequence dialog box is displayed. This dialog box contains two tabs: Name Tab and Columns Tab. Use these tabs to rename a sequence or edit sequence columns.

### Name Tab

Rename a sequence by typing the new name in the Name field. You can also rename a sequence by right-clicking the sequence name in the Projects Navigator and selecting **Rename**.

Follow the rules in "Naming Conventions for Data Objects" on page 2-8 to specify the name and description.

To modify a sequence description, enter the new description in the Description field.

### Columns Tab

The Columns tab displays the sequence columns CURRVAL and NEXTVAL.

## Defining User-Defined Types

User-defined data types use Oracle built-in data types and other user-defined data types as the building blocks of object types that model the structure and behavior of data in applications. The built-in data types are mostly scalar types and do not provide the same flexibility that modeling an application-specific data structure does.

User-defined data types extend the modeling capabilities of native data types by specifying both the underlying persistent data (attributes) and the related behaviors (methods). With user-defined types, you can create better models of complex entities in the real world by binding data attributes to semantic behavior.

Consider a simple example of a `Customers` table. The Customer address information is usually modeled as four or five separate fields, each with an appropriate scalar type. User-defined types allow for a definition of "address" as a composite type and also to define validation on that type.

Warehouse Builder enables you to define the following user-defined data types:

- Object types
- Varrays
- Nested tables

### About Object Types

Object types are abstractions of the real-world entities, such as purchase orders, that application programs deal with. An object type is a heterogeneous user-defined type. It is composed of one or more user-defined types or scalar types.

An object type is a schema object with the following components:

- **Name:** A name identifies the object type uniquely within that schema.
- **Attributes:** An attribute is used to create the structure and state of the real-world entity that is represented by an object. Attributes can be built-in types or other user-defined types.
- **Methods:** A method contains functions or procedures that are written in PL/SQL or Java and stored in the database, or written in a language such as C and stored

externally. Methods are code-based representations of the operations that an application can perform on the real-world entity.

**Note:** Methods are currently not supported.

For example, the `ADDRESS` type definition can be defined as follows:

```
CREATE TYPE ADDRESS AS OBJECT ( street_name varchar2(240), door_
no varchar2(30), po_box_no number, city varchar2(35), state
varchar2(30), country varchar2(30));
```

Once the type has been defined it can be used across the schema for any table that requires the type definition "address" as one of its fields.

## Defining Object Types

**To define an object type:**

1. In the Projects Navigator, expand the Databases node and then the Oracle node.
2. Expand the target module in which you want to create the object type.
3. Expand the User Defined Types node.
4. Right-click Object Types and select **New Object Type**.

   The Create Object Type dialog box is displayed.
5. Enter a name and an optional description for the object type and click **OK**.

   The Object Type Editor is displayed. Use the following tabs on the editor to define the object type:

   - Name Tab
   - Columns Tab

### Name Tab

Use the Name field to enter a name for the object type. Use the Description field to enter an optional description for the object type. To rename an object type, select the name and enter a new name.

Follow the rules in "Naming Conventions for Data Objects" on page 2-8 to specify a name and description.

### Columns Tab

Use the Columns tab to define the attributes in the object type. This tab displays a table that you can use to define attributes. Each row in the table corresponds to the definition of one object type attribute.

Specify the following details for each attribute:

- **Name:** Enter the name of the attribute. The attribute name must be unique within the object type.
- **Data Type:** Select the data type of the attribute from the Data Type list. Warehouse Builder assigns a default data type for the attribute based on the name. For example, if you create an attribute named `start_date`, then the data type assigned is `DATE`. You can change the default assignment if it does not suit your data requirement.

> **See Also:** "Supported Data Types" on page 2-3 for a list of supported Oracle Database data types

- **Length:** Specify the length of the attribute. Length is applicable to character data types only.
- **Precision:** Specify the total number of digits allowed for the attribute. Precision is applicable for to data types only.
- **Scale:** Specify the total number of digits to the right of the decimal point. Scale is applicable to numeric data types only.
- **Seconds Precision:** Specify the number of digits in the fractional part of the datetime field. Seconds precision is used for `TIMESTAMP`, `TIMESTAMP WITH TIME ZONE`, and `TIMESTAMP WITH LOCAL TIME ZONE` data types.
- **Not Null:** Select this field to specify that the attribute should not contain NULL values. By default, all columns in a table allow nulls. This column is not applicable to object types.
- **Default Value:** Specify the default value for this attribute. If no value is entered for this column while data is stored in the table, then the default value is used. If you specify a value for this column while loading data, then the default value is overridden and the specified value is stored in the column. This column is not applicable for object types.
- **Virtual:** Select this option to indicate that the attribute behaves like a virtual column.

  Virtual columns are not stored in the database. They are computed using the expression specified in the Expression field. You can refer to virtual columns just like any other column in the table, except that you cannot explicitly write to a virtual column.
- **Expression:** Specify the expression that is used to compute the value of the virtual attribute. The expression can include columns from the same table, constants, SQL functions, and user-defined PL/SQL functions.
- **Description:** Type an optional description for the attribute.

## Editing Object Types

**To edit an object type:**

1. In the Projects Navigator, expand the Databases node and then the Oracle node.
2. Expand the module that contains the object type.
3. Expand the User Defined Types node and then the Object Types node.
4. Right-click the object type that you want to edit and select **Open**.

The Object Type Editor is displayed. Use the Name and Columns tabs as defined in "Defining Object Types" on page 2-37 to edit the definition of the object type.

## About Varrays

A Varray is an ordered collection of data elements. The position of each element in a Varray is stored as an index number. You can use this number to access particular elements. When you define a Varray, you specify the maximum number of elements it can contain. You can change this number later. Varrays are stored as opaque objects (such as `RAW` or `BLOB`).

If the customer has more than one address, for example three addresses, then you can create another type, a table type, that holds three addresses. The following example creates a table of `ADDRESS` type:

```
TYPE address_store is VARRAY(3) of address;
```

The first address in the list is considered as the primary address and the remaining addresses are considered as the secondary addresses.

## Defining Varrays

**To define a Varray:**

1. From the Projects Navigator, expand the Databases node and then the Oracle node.
2. Expand the target module in which you want to create the Varray.
3. Expand the User Defined Types node.
4. Right-click **Varrays** and select **New Varray**.

   The Create Varray dialog box is displayed.
5. Enter a name and an optional description for the Varray and click **OK**.

   The Varray Editor is displayed. Use the following tabs on the editor to define the object type:

   - Name Tab
   - Details Tab

### Name Tab

Use the Name field to enter a name for the Varray. Use the Description field to enter an optional description for the Varray. To rename a Varray, select the name and enter a new name.

Follow the rules in "Naming Conventions for Data Objects" on page 2-8 to specify a name and an optional description.

### Details Tab

Use the Details tab to specify the value for the following fields:

- **Data Type:** Select the data type of the attribute from the Data Type list.

  **See Also:** "Supported Data Types" for a list of supported Oracle Database data types

- **Length:** Specify the length of the Varray element. Length is applicable for character data types only.
- **Precision:** Specify the total number of digits allowed for the Varray element. Precision is applicable to numeric data types only.
- **Scale:** Specify the total number of digits to the right of the decimal point. Scale is applicable to numeric data types only.
- **Seconds Precision:** Specify the number of digits in the fractional part of the datetime field. Seconds precision is used for `TIMESTAMP`, `TIMESTAMP WITH TIME ZONE`, and `TIMESTAMP WITH LOCAL TIME ZONE` data types only.
- **Size:** Specify the size of the Varray.

## Editing Varrays

**To edit a Varray:**

1. From the Projects Navigator, expand the Databases node and then the Oracle node.
2. Expand the module that contains the Varray type.
3. Expand the User Defined Types node and then the Varrays node.
4. Right-click the Varray that you want to edit and select **Open**.

The Varray Editor is displayed. Use the Name and Details tabs of the Varray Editor, as described in "Defining Varrays" on page 2-39, to edit the definition of the Varray.

## About Nested Tables

A nested table is an unordered collection of data elements. Nested tables enable you to have any number of elements. There is no maximum number of elements specified in the definition of the table. The order of the elements is not preserved. All the operations, such as SELECT, INSERT, and DELETE that you perform on ordinary tables can be performed on nested tables. Elements of a nested table are stored in a separate storage table containing a column that identifies the parent table row or object to which each element belongs. The elements may be built-in types or user-defined types. You can view a nested table as a single-column table, or if the nested table is an object type, as a multicolumn table, with a column for each attribute of the object type.

Nested tables are used to store an unordered set of elements that do not have a predefined size, such as customer references.

## Defining Nested Tables

**To define a nested table:**

1. From the Projects Navigator, expand the Databases node and then the Oracle node.
2. Expand the target module where you want to create the nested table.
3. Expand the User Defined Types node.
4. Right-click **Nested Tables** and select **New Nested Table**.

   The Create Nested Table dialog box is displayed.
5. Enter the name and an optional description for the nested table and click **OK**.

   The Nested Table Editor is displayed. Use the following tabs on the editor to define the nested table.

   - Name Tab
   - Details Tab

### Name Tab

Use the Name field to enter a name for the nested table. Use the Description field to enter an optional description for the nested table. To rename a nested table, select the name and enter a new name.

Follow the rules in "Naming Conventions for Data Objects" on page 2-8 to specify a name and an optional description.

### Details Tab

Use the Details tab to specify the value for the following fields:

- **Data Type:** Select the data type of the attribute from the Data Type list.

  **See Also:** "Supported Data Types" on page 2-3 for a list of supported data types

- **Length:** Specify the length of the nested table element. Length is applicable for character data types only.
- **Precision:** Specify the total number of digits allowed for the nested table element. Precision is applicable to numeric data types only.
- **Scale:** Specify the total number of digits to the right of the decimal point. Scale is applicable to numeric data types only.
- **Seconds Precision:** Specify the number of digits in the fractional part of the datetime field. Seconds precision is used for `TIMESTAMP`, `TIMESTAMP WITH TIME ZONE`, and `TIMESTAMP WITH LOCAL TIME ZONE` data types only.

## Editing Nested Tables

**To edit a nested table:**

1. From the Projects Navigator, expand the Databases node and then the Oracle node.
2. Expand the module that contains the nested table.
3. Expand the User Defined Types node and then the Nested Tables node.
4. Right-click the nested table that you want to edit and select **Open**.

The Nested Table Editor is displayed. Use the Name and Details tabs, as defined in "Defining Nested Tables" on page 2-40, to edit the definition of the nested table.

# Defining Queues

Queues enable asynchronous information sharing using messages. Use queues to implement incremental data warehousing or replication solutions, both within a database or from one database to another.

Before you can share data in the form of messages, you must create a data object that stores and manages multiple messages. This object is the Advanced Queue (AQ). You can propagate messages between different queues by defining queue propagations. The following sections describe how to define and use queues.

Queues provide the following advantages:

- Creating applications that communicate with each other in a consistent, reliable, secure, and autonomous manner
- Storing messages in database tables, bringing the reliability and recoverability of the database to your messaging infrastructure
- Retaining messages in the database automatically for auditing and business intelligence

## Creating Queue Table Definitions

Queues are stored in queue tables. Each queue table is a database table and contains one or more queues. Creating a queue table creates a database table with approximately 25 columns. These columns store Oracle AQ metadata and the user-defined payload.

You can create queue tables that store any type of messages by using the `SYS.ANYDATA` data type to define the type of data stored in the queue table.

**To create queue tables:**

1. In the Projects Navigator, expand the Databases node, the Oracle node, and then the module node under which you want to define a queue table.
2. Expand the Queues node, right-click Queue Tables and select **New Queue Table**.

   The Create Queue Table dialog box is displayed.
3. Specify the details required to define queue tables and click **OK**.

   For more details, see "Defining the Payload Type of Queue Tables" on page 2-42.

The metadata for the queue table is created in the workspace and the queue table is added to the Projects Navigator under the Queues node.

### Defining the Payload Type of Queue Tables

Use the Create Queue Table dialog box or the Edit Queue Table dialog box to provide additional details about the queue table such as the payload type and order in which the messages in the queue should be sorted. The following section describes the details to be provided for a queue table.

#### Name

The Name field represents the name of the queue table. The name should be unique within the Oracle module containing that queue table. For more details, see "Naming Conventions for Data Objects" on page 2-8.

#### Description

Use the Description field to provide an optional description for the queue table.

#### Payload Type

Each queue contains a payload that stores the actual data that is to be transferred. The Payload Type represents the type of data that is permitted in the queue table.

You can select one of the following options for the payload type: Object Type, `SYS.ANYDATA`, `RAW`, `SYS.AQ$_JMS_BYTES_MESSAGE`, `SYS.AQ$_JMS_MAP_MESSAGE`, `SYS.AQ$_JMS_MESSAGE`, `SYS.AQ$_JMS_STREAM_MESSAGE`, `SYS.AQ$_JMS_TEXT_MESSAGE`, and `XMLTYPE`.

#### Search For

Use the Search For field to search for an object. This field is enabled only when you select Object Type as the Payload Type.

The area below the Search For field displays the object types in your workspace. Object types are listed under the module to which they belong. To search for an object with a specific name, enter the name of the object in the Search For field and click **Go**.

### Enable Transactional Property for Messages

Select **Enable Transactional Property for Messages** to enable message grouping. This option is enabled for all object-typed queues except `SYS.ANYDATA` queues.

Message grouping enables messages belonging to one queue to be grouped so that they form a set that can only be consumed by one user at a time. All messages belonging to a group must be created in the same transaction, and all messages created in one transaction belong to the same group.

This feature enables users to segment complex messages into simple messages. It is also useful if the message payload contains complex large objects such as images and video that can be segmented into smaller objects.

### Secured Queue Table

Select **Secured Queue Table** when you want to create a secure queue. This option is enabled for all object-typed queues except `SYS.ANYDATA` queues.

Secure queues are queues for which Oracle Streams Advanced Queuing (AQ) agents must be associated explicitly with one or more database users who can perform queue operations, such as enqueue and dequeue. The owner of a secure queue can perform all queue operations on the queue, but other users cannot perform queue operations on a secure queue, unless they are configured as secure queue users.

### Sort Messages By

Use the Sort Messages By list to specify the order in which the messages contained in the payload should be sorted. The options you can use to sort messages are:

- **Enqueue_time:** Sorts by the arrival time of the message.
- **Priority:** Sorts by message priority.
- **Enqueue_time, priority:** Sorts by the arrival time of the message and, for messages with the same arrival time, sorts by message priority.
- **Priority, enqueue_time:** Sorts by message priority and, for messages with the same priority, sorts by arrival time of the message.

## Editing Queue Tables

To edit queue tables, right-click the queue table and select **Open**. The Edit Queue Table dialog box is displayed. Use this to modify your queue table definition.

You can modify queue tables and change the name, description, or payload type. However, if you modify the payload type of a queue table, then the queue table and any queues based on this queue table will dropped and recreated. Thus, all the existing data in the queue will be lost. Once deployed, it is recommend to not to modify the payload type.

## Creating Advanced Queue Definitions

Advanced Queues (AQs) provide database-integrated message queuing functionality. They optimize the functions of Oracle Database so that messages can be stored persistently, propagated between queues on different computers and databases, and transmitted using Oracle Net Services, HTTP, and HTTPS.

Use advanced queues to propagate and manage data either within an Oracle database or between different databases. Every advanced queue is based on a queue table that stores the actual queue data.

> **See Also:** *Oracle Streams Concepts and Administration* for more information about advanced queues

**To create an advanced queue definition:**

1. In the Projects Navigator, expand the Databases node, the Oracle node, and then the Oracle module node under which you want to define an advanced queue.
2. Right-click the Queues node and select **New Advanced Queue**.

   The Create Advanced Queue dialog box is displayed.
3. Provide details such as the advanced queue name and the queue table on which the advanced queue is based. Click **OK**.

   For more information, see "Specifying the Queue Table on which the AQ is Based" on page 2-44.

### Specifying the Queue Table on which the AQ is Based

Use the Create Advanced Queue dialog box to specify details such as the advanced queue name and the queue table on which the AQ is based. You can choose an existing queue table or create a new one. Use the Edit Advanced Queue dialog box to modify the name, description, or the queue table on which the AQ is based.

#### Name

The Name field represents the name of the advanced queue. The name should be unique within the Oracle module containing that advanced queue. For more details, see "Naming Conventions for Data Objects" on page 2-8.

#### Description

Use the Description field to provide an optional description for the advanced queue.

#### Select a Queue Table

Use the Select a Queue Table list to select the queue table that stores messages contained in the advanced queue.

#### Create New Queue Table

Select **Create New Queue Table** to create a new queue table that will contain the advanced queue data. The Create Queue Table dialog box that guides you through the process of defining a queue table is displayed. For more information about defining queue tables, see "Creating Queue Table Definitions" on page 2-42.

## Editing Advanced Queue Definitions

You can edit advanced queues and modify the properties that you specified while creating the advanced queues. This includes the name, description, and queue table that stores queue data.

To edit an advanced queue, in the Projects Navigator, right-click the advanced queue and select **Open**. The Edit Advanced Queue dialog box is displayed. Use this to edit the advanced queue.

For more information about the options on this dialog box, see "Creating Advanced Queue Definitions" on page 2-43.

After you edit an advanced queue definition, ensure that you synchronize any mappings that use this advanced queue.

## Creating Queue Propagations

Queue propagations enable you to propagate messages between different queues. For example, you have two queues SRC_QUE and TGT_QUE. In SRC_QUE, define a queue propagation with the target queue as TGT_QUE to propagate messages from SRC_QUE to TGT_QUE.

Queue propagations are typically used in replication, when you have two databases located in different location and you want to replicate the source database to the target location.

**To create a queue propagation:**

1. In the Projects Navigator, expand the Databases node, the Oracle node, and then the Oracle module node under which you want to define a queue propagation.
2. Expand the Queues node, right-click the advanced queue under which you want to create a propagation, and select **New**.

   The New Gallery dialog box is displayed.
3. On the New Gallery dialog box, select **Queue Propagations** and click **OK**.

   The Create Queue Propagation dialog box is displayed.
4. Use the Create Propagation dialog box to define the target queue and click **OK**.

The metadata for the queue propagation is created in the workspace and the queue propagation is added under the advanced queue in the Projects Navigator.

### Selecting a Target Queue for Propagation

Use the Create Queue Propagation dialog box or the Edit Queue Propagation dialog box to specify the target queue for propagation. The following sections describe the fields contained in this page.

#### Name

The Name field represents the name of the queue propagation. The name should be unique within the advanced queue under which the queue propagation is defined.

To rename a queue propagation, select the name and enter the new name.

#### Description

The Description field represents the description of the queue propagation. Providing a description is optional.

#### Select a Target Queue for Propagation

Use the Select a Target Queue for Propagation section to define the target queue. The area below this section displays a node tree containing the advanced queues defined in the current project. Select the advanced queue which will be used as a target. Messages from the AQ under which you define the queue propagation can then be propagated to the AQ defined as the target queue.

Use the Search For field to search for a particular object using the object name. Enter the name of the object in this field and click **Go**.

## Editing Queue Propagations

You can edit queue propagations and modify the selections you made which defining the queue propagation. The options you can modify include the name, description, and target queue.

To edit a queue propagation, in the Projects Navigator, right-click the queue propagation and select **Open**. The Edit Queue Propagation dialog box is displayed. For more information about the options on this dialog box, see "Creating Queue Propagations" on page 2-45.

# Configuring Relational Data Objects

Earlier in the design phase, you defined a logical model for your target system using Warehouse Builder design objects. In the configuration phase, you assign physical deployment properties to the object definitions by configuring parameters such as tablespaces, partitions, and other identification parameters. You also configure runtime parameters such as job names, and runtime directories.

Set these physical properties using the Configuration tab of the data object. The following sections show you how to assign physical properties to your logical design model:


- Configuring Target Modules on page 2-46
- Configuring Tables on page 2-48
- Configuring Materialized Views on page 2-51
- Configuring Views on page 2-54
- Configuring Sequences on page 2-54
- Configuring Advanced Queues on page 2-54
- Configuring Queue Tables on page 2-55
- Configuring Queue Propagations on page 2-55


## Configuring Target Modules

Each target module provides top-level configuration options for all the objects contained in that module.

**To configure an Oracle module:**

1. From the Projects Navigator, expand **Databases,** expand **Oracle,** and right-click a target module name, and select **Configure.**

   Warehouse Builder displays the Configuration tab.

2. Choose the parameters that you want to configure and click the space to the right of the parameter name to edit its value.

   For each parameter, you can either select an option from a list, type a value, or click the Ellipsis button to display another properties dialog box.

3. Configure the parameters listed in the following sections.

### Deployment System Type

**PL/SQL Generation Mode:** Defines the target database type. The options you can select are: Default, Oracle 10g, Oracle10gR2, Oracle11gR1, Oracle11gR2, Oracle8i, and Oracle9i. Code generation is based on your choice in this field. For example, select Oracle 9i to ensure the use of Oracle 9*i* code constructs. If you select **Oracle8i**, then row-based code is generated.

Each release introduces new functionality, some of which you may use only in conjunction with the latest version of the Oracle Database. For example, if you select

**Oracle8i** as the PL/SQL Generation Mode, then you cannot access some Oracle 9*i* Warehouse Builder components such as table functions and external tables.

### Generation Preferences

**End of Line:** Defines the end of line markers for flat files. This depends on the platform to which you are deploying your warehouse. For UNIX, use `\n`, and for Windows NT, use `\r\n`.

### Generation Target Directories

**ABAP Extension:** File name extension for ABAP scripts. The default is `.abap`.

**ABAP Run Parameter File:** Suffix for the parameter script in an ABAP job. The default is _run.ini.

**ABAP Spool Directory:** The location where ABAP scripts are buffered during script generation processing.

**DDL Directory:** Enter a location for the scripts that create database objects in the target schema. The default is `ddl\`.

**DDL Extension:** Enter a file name extension for DDL scripts. The default is .ddl

**DDL Spool Directory:** Enter a buffer location for DDL scripts during the script generation processing. The default is `ddl\log`.

**LIB Directory:** Enter a location for the scripts that generate Oracle functions and procedures. The default is `lib\`.

**LIB Extension:** Enter a suffix to be appended to a mapping name. The default is `.lib`.

**LIB Spool Directory:** Enter a location for the scripts that generate user-defined functions and procedures. The default is `lib\log\`.

**LOADER Directory:** Enter a location for the control files. The default is `ctl\`.

**LOADER Extension:** Enter a suffix for the loader scripts. The default is `.ctl`.

**LOADER Run Parameter File:** Enter a suffix for the parameter initialization file. The default is `_run.ini`.

**PL/SQL Directory:** Enter a location for the PL/SQL scripts. The default is `pls\`.

**PL/SQL Run Parameter File:** Enter a suffix for the parameter script in a PL/SQL job. The default is `_run.ini`.

**PL/SQL Spool Directory:** Enter a buffer location for PL/SQL scripts during the script generation processing. The default is `pls\log\`.

**PL/SQL Extension:** Enter a file name extension for PL/SQL scripts. The default is `.pls`.

**SQLPlus Directory:** Enter a location for the PL/SQL scripts. The default is `sql\`.

**SQLPlus Extension:** Enter a file name extension for PL/SQL scripts. The default is `.sql`.

**SQLPLus Run Parameter File:** Enter a suffix for the parameter script in a PL/SQL job. The default is `_run.ini`.

**Staging File Directory:** For all ABAP configuration related to SAP tables. The default is `abap\`.

**Tcl Directory:** Enter a location for the tcl scripts. The default is `tcl\`.

> **See Also:** *Oracle Warehouse Builder Sources and Targets Guide* for more information about using SAP tables

### Identification

**Application Short Name:** This parameter is obsolete and is no longer used.

**Deployable:** Select this option to indicate that the objects contained in the module can be deployed.

**Location:** Represents the location with which the module is associated. If the module is a source module, then this value represents the location from which the data is sourced. If the module is a target module, then this value represents the location to which the generated code and object data are deployed.

**Main Application Short Name:** This parameter is obsolete and is no longer used.

**Top Directory:** Represents the name of the directory to which the generated code is stored. The default value for this parameter is `..\..\codegen\`. You can change this value to any directory in which you want to store generated code.

### Run Time Directories

**Archive Directory:** Not currently used. The default is `archive\`.

**Input Directory:** Not currently used. The default is `input\`.

**Invalid Directory:** Directory for SQL*Loader error and rejected records. The default is `invalid\`.

**Log Directory:** Log directory for the SQL*Loader. The default is `log\`.

**Receive Directory:** Not currently used. The default is `receive\`.

**Sort Directory:** Not currently used. The default is `sort\`.

**Work Directory:** Not currently used. The default is `work\`.

### Tablespace Defaults

**Default Index Tablespace:** Defines the name of each tablespace where indexes are created. The default is null. If you configure an index tablespace at the target module level and not at the object level, then the tablespace value configured at the target module level is used during code generation. If you configure a tablespace for each index at the object level, then the tablespace value configured at the target module level is overwritten.

**Default Object Tablespace:** Defines the name of each tablespace where objects are created (for example, tables, views, or materialized views). The default is null. If you configure object tablespace at the target module level and not at the individual object level, then the value configured at the target module level is used during code generation. If you configure a tablespace for each individual object, then the tablespace value configured at the target module level is overwritten.

## Configuring Tables

Warehouse Builder generates DDL scripts for each table defined in a target module.

**To configure the physical properties for a table:**

1. In the Projects Navigator, right-click the name of a table and select **Configure.**

   The Configuration tab for the table is displayed.

2. Set the configuration parameters listed under the following nodes: Error Table, Foreign Keys, Identification, Parallel, Performance Parameters, Partition Parameters, Storage Space, and Change Data Capture.

### Error Table

**Error Table Name:** Indicates the name of the error table that stores the rows that were not loaded into the table during a load operation.

**Tablespace:** Indicates the name of the tablespace in which the error table is stored.

### Foreign Keys

The Foreign Keys node is displayed if your table contains a foreign key definition. A separate node is displayed, under the Foreign Keys node, for each foreign key in the table. Under this node, parameters are listed under the following categories: Creation Method and Identification.

The Creation Method category contains the following parameters:

- **Constraint Checking:** Indicates whether the checking of this constraint can be deferred until after the transaction is committed. Set this parameter to DEFERABLE to indicate that, in subsequent transactions, you can set the SET CONSTRAINTS clause to defer checking this constraint until after the transaction is committed. Set this parameter to NOT DEFERABLE to indicate that you cannot use the SET CONSTRAINTS clause ti defer checking this constraint until after the transaction is committed. The default is NOT DEFERABLE.
- **Constraint State:** Indicates whether the constraint should be enabled. Select ENABLE to apply the constraint to the table data. Select DISABLE to disable the integrity constraint. The default is ENABLE.
- **Constraint Validation:** The options you can set are NOVALIDATE or VALIDATE. The effect of setting this parameter is different based on whether the constraint is enabled or disabled.
- **EXCEPTIONS INTO:** Indicates the name of the exceptions table. You cannot use this parameters when you set NOVALIDATE as the Constraint Validation.
- **INITIALLY:** The options you can set for this parameter are IMMEDIATE or DEFERRED. The default setting is IMMEDIATE. IMMEDIATE indicates that a deferrable constraint must be checked at the end of each SQL statement. DEFERRED indicates that a deferrable constraint must be checked at the end of subsequent transactions.
- **NOVALIDATE mode:** The options you can set for this parameter are NORELY and RELY, with NORELY being the default setting. Setting this parameter to RELY activates a constraint in NOVALIDATE mode for query rewrite.
- **ON DELETE:** During a delete operation, this parameter indicates how to handle foreign keys that depend on the record being deleted. The options you can set for this parameter are CASCADE or SET NULL. CASCADE deletes dependent foreign key values. NOT NULL sets the dependent foreign key values to null.

### Identification

**Deployable:** Select this option to indicate that you want to deploy this table. Scripts are generated only for tables marked deployable.

**Error Table Only:** Select this option to perform generation or deployment actions only on the error table associated with the table. Use this option to add an error table to an

existing table. This setting only controls the actions of the Deployable parameter, but does not override it.

Deselect this option to deploy the error table along with the table.

### Parallel

**Parallel Access Mode:** Enables parallel processing when the table has been created. The default is `PARALLEL`.

**Parallel Degree:** Indicates the degree of parallelism. This is the number of parallel threads used in the parallel operation.

### Performance Parameters

**Buffer Cache:** Indicates how Oracle Database should store rows in the buffer cache.

**Data Segment Compression:** Indicates whether data segments should be compressed. Compressing reduces disk use. The default is NOCOMPRESS.

**Logging Mode:** Indicates whether the DML actions are logged in the redo log file. To improve performance, set this parameter to `NOLOGGING`. The default is `LOGGING`.

**Row-level Dependency:** Indicates whether row-level dependency tracking.

**Row Movement:** Indicates if Oracle Database can move a table row.

**Statistics Collection:** Indicates if statistics should be collected for the table. Specify MONITORING if you want modification statistics to be collected on this table.

### Partition Parameters

**Partition Tablespace List:** Specify a comma-separated list of tablespaces. For simple partitioned objects, it is used for a HASH BY QUANTITY partition tablespace. For composite partitioned tables, it is used for subpartition template to store the list of tablespaces.

**Overflow Tablespace List:** Specify a comma separated list of tablespaces for overflow data. For simple partitioned objects, it is used for HASH BY QUANTITY partition overflow tablespaces. The number of tablespaces does not have to equal the number of partitions. If the number of partitions is greater than the number of tablespaces, then Oracle Database cycles through the names of the tablespaces.

### Storage Space

Storage parameters enable you to define how the table is stored in the database. This category contains parameters such as BUFFER_POOL, FREELIST GROUPS, FREELISTS, INITIAL, MINEXTENTS, MAXEXTENTS, NEXT, and PCTINCREASE.

The **Tablespace** parameter defines the name of each tablespace where the table is created. The default value is null. If you accept the default value of null, then the table is generated based on the tablespace value set in the configuration parameters of the target module. If you configure the tablespace for individual objects, then the tablespace value configured for the target module is overwritten.

### Change Data Capture

**Enable:** Indicates if change data capture is enabled for the table. Select True for this parameter to enable change data capture for the table.

**Table Position:** Indicates the position of the table in the change data capture.

## Configuring Materialized Views

**To configure the physical properties for a materialized view:**

1. From the Projects Navigator, right-click a materialized view name and select **Configure.**

   The Configuration tab for the materialized view is displayed.

2. Follow the configuration guidelines listed for tables. For more information, see "Configuring Tables" on page 2-48.

3. Configure the Materialized View Parameters listed in the following section.

4. Configure the Materialized View Log parameters as described in "Materialized View Log Parameters" on page 2-53.

### Materialized View Parameters

The following are parameters for materialized views:

- **Base Tables:** Specify a comma-separated list of base tables referenced by the materialized view. Separate each table name with a comma. If a table name is not in uppercase, then enclose the name in double quotation marks.

- **BUILD:** Indicates when the materialized view is populated. The options are Immediate (default), Deferred, and Prebuilt.

  **Immediate:** Populates the materialized view when it is created.

  **Deferred:** Delays the population of the materialized view until the next refresh operation. You can select this option when you are designing a materialized view and the metadata for the base tables is correct but the data is not.

  **Prebuilt:** Indicates that the materialized view is prebuilt.

- **Default Rollback Segment:** The options are DEFAULT, DEFAULT MASTER, DEFAULT LOCAL, and NONE. The default setting is DEFAULT LOCAL. Specify DEFAULT to indicate that Oracle Database should choose which rollback segment to use. Specify DEFAULT MASTER for the remote rollback segment to be used at the remote site. Specify DEFAULT LOCAL for the remote rollback segment to be used for the local refresh group that contains the materialized view. Specify NONE to name both master and local segments.

- **FOR UPDATE:** Select Yes to allow a subquery, primary key, row ID, or object materialized view to be updated. The default setting is No.

- **Local Rollback Segment:** Specify a named remote rollback segment to be used for the local refresh group of the materialized view. The default is null.

- **Master Rollback Segment:** Indicates the name of the remote rollback segment to be used at the remote master site for the materialized view.

- **NEXT (date):** Indicates the interval between automatic refreshes. Specify a datetime value for this parameter.

- **Query Rewrite:** Indicates if the materialized view is eligible for query rewrite. The options are ENABLE and DISABLE. The default is DISABLE.

  **Enable:** Enables query rewrite. For other query rewrite requirements, see "Fast Refresh for Materialized Views" on page 2-53.

  **Disable:** Disables query rewrite. You can disable query rewrite when you know that the data in the materialized view is stale or when you want to make changes to the query statement.

- **REFRESH:** Indicates the refresh method. The options are Complete, Fast, Force, and Never. The default setting is Force.

  **Complete:** Oracle Database truncates the materialized view and reexecutes the query upon refresh.

  **Fast:** Uses materialized views to only apply changes to the base table data. There are many requirements for fast refresh to operate properly. For more information, see "Fast Refresh for Materialized Views" on page 2-53.

  **Force:** Oracle Database attempts to refresh using the fast mode. If unable to refresh in fast mode, the Oracle server reexecutes the query upon refresh.

  **Never:** Prevents the materialized view from being refreshed.

- **Refresh On:** The options are COMMIT and DEMAND. Specify COMMIT to indicate that a fast refresh is to occur whenever the database commits a transaction that operates on a master table of the materialized view. Specify DEMAND to indicate that the materialized view should be refreshed on demand. You can do this by using one of the refresh procedures of the DBMS_MVIEW package. The default setting is DEMAND.
- **Start With:** Indicates the first automatic refresh time. Specify a datetime value for this parameter.
- **Using Constraints:** The options that you can select for this parameter are TRUSTED or ENFORCED. Select TRUSTED to allow Oracle Database to use dimension and constraint information that has been declared trustworthy by the DBA but has not been validated by Oracle Database. ENFORCED is the default setting.
- **WITH:** Select PRIMARY_KEY to create a primary key materialized view. Select ROWID to create a ROWID materialized view. The default setting is PRIMARY_KEY.

#### Performance Parameters

**Buffer Cache:** Indicates how the blocks retrieved for this table are placed in the buffer cache. The options you can select are CACHE or NOCACHE. When you select CACHE, the blocks are placed at the most recently used end of the least recently used (LRU) list in the buffer cache when a full table scan is performed. Setting the Buffer Cache parameter to CACHE is useful for frequently accessed tables, such as small lookup tables. When you select NOCACHE, the blocks are placed at the least recently used end of the LRU list in the buffer cache.

**Data Segment Compression:** Indicates if segments should be compressed on disk to reduce space usage. The options you can set for this parameter are COMPRESS, COMPRESS ALL, or NOCOMPRESS. The default is NOCOMPRESS. Set this parameter to COMPRESS to compress data only during a direct path INSERT, when it is productive to do so. Set this parameter to COMPRESS ALL compresses data during all DML operations on the table.

**Logging Mode:** Indicates whether the DML actions are logged in the redo log file. To improve performance, set this parameter to NOLOGGING. The default is LOGGING.

#### Error Table

- **Error Table Name:** Indicates the name of the error table that stores the rows that were not loaded into the table during a load operation.
- **Tablespace:** Indicates the name of the tablespace in which the error table is stored.

#### Parallel

- **Parallel Access Mode:** Enables parallel processing when the table has been created. The default is PARALLEL.
- **Parallel Degree:** Indicates the degree of parallelism. This is the number of parallel threads used in the parallel operation.

#### Identification

- **Deployable:** Select TRUE to indicate if you want to deploy this materialized view. Warehouse Builder generates scripts only for materialized views marked deployable.
- **Error Table Only:** Select this option to perform generation or deployment actions only on the error table associated with the materialized view. Use this option to add an error table to an existing materialized view. This setting controls the actions of the Deployable parameter, but does not override it.

  Deselect this option to deploy the error table along with the materialized view.

#### Hash Partition Parameters

- **Hash Partition Tablespace List:** Indicates the tablespace that stores the partition or subpartition data. To specify multiple tablespaces, use a comma-separated list.

### Materialized View Log Parameters

You can configure the following materialized view log parameters.

**Record Primary Key**: Select PRIMARY KEY to indicate that the primary key of all rows changed should be recorded in the materialized view log.

**Record ROWID**: Select ROWID to indicate that the row ID of all rows changed should be recorded in the materialized view log.

**Record SEQUENCE**: Select SEQUENCE to indicate that a sequence value providing additional ordering information should be recorded in the materialized view log.

Sequence numbers are necessary to support fast refresh after some update scenarios.

**COLUMNS**: Specify the columns whose values you want to be recorded in the materialized view log for all rows that are changed. Typically these columns are filter columns and join columns.

**Generate Materialized View Log**: Select YES to generate DDL for materialized view log. The default is YES.

**New Values**: Specify INCLUDING to save both new and old values in the log. Specify EXCLUDING to disable the recording of new values in the log. EXCLUDING is the default.

### Fast Refresh for Materialized Views

You can configure a materialized view to refresh incrementally. When you update the base tables for a materialized view, the database stores updated record pointers in the materialized view log. Changes in the log tables are used to refresh the associated materialized views.

To ensure incremental refresh of materialized views, verify the following conditions:

- The Refresh parameter must be set to Fast and the Base Tables parameter must list all base tables.

- Each base table must have a PK constraint defined. Warehouse Builder generates a create statement based on the PK constraint and utilizes that log to refresh the dependent materialized views.
- The materialized view must not contain references to nonrepeating expressions such as SYSDATE, ROWNUM, and nonrepeatable PL/SQL functions.
- The materialized view must not contain references to `RAW` and `LONG RAW` data types.
- There are additional restrictions for materialized views with statements for joins, aggregations, and unions. For information about additional restrictions, see *Oracle Database Data Warehousing Guide*.

## Configuring Views

Warehouse Builder generates a script for each view defined in a target module. You can configure the parameters listed in the following categories.

### Identification

**Deployable:** Set to TRUE to deploy this view.

**Error Table Only:** Select this option to perform generation or deployment actions only on the error table associated with the view. Use this option to add an error table to an existing view. This setting only controls the actions of the Deployable parameter, but does not override it. Deselect this option to deploy the error table along with the view.

### Error Table

**Error Table Name:** Indicates the name of the error table that stores the rows that were not loaded into the view during a load operation.

**Tablespace:** Indicates the name of the tablespace in which the error table is stored.

## Configuring Sequences

A script is generated for each sequence object. A sequence object has a Start With parameter and an Increment By parameter. Both parameters are numeric.

**To configure the physical properties for a sequence:**

1. Right-click the name of a sequence and select **Configure.**

   The Configuration tab for the sequence is displayed.

2. Configure the following Sequence parameters:

   **Increment By:** The number by which you want to increment the sequence.

   **Start With:** The number at which you want the sequence to start.

3. Configure the following Identification parameter:

   **Deployable:** Select this option to indicate that you want to deploy this sequence. Warehouse Builder only generates scripts for sequences marked deployable.

## Configuring Advanced Queues

Use the following steps to configure advanced queues.

1. From the Projects Navigator, expand the Databases node and then the Oracle node.

2. Expand the Oracle module containing the advanced queue, then the Queues node, and right-click the advanced queue name and select **Configure.**

   Warehouse Builder displays the Configuration tab that contains configuration parameters for the advanced queue.

3. Choose the parameters you want to configure and click the space to the right of the parameter name to edit its value.

   For each parameter, you can either select an option from a list, type a value, or click the Ellipsis button to display another properties dialog box.

Following are the parameters that you can configure.

**Dequeue Enabled** Set this parameter to true to enable dequeuing for the advanced queue.

**Enqueue Enabled** Set this parameter to true to enable enqueuing for the advanced queue.

**Max Retries** Represents the number of times a dequeue can be attempted on a message. The maximum value of max_retries is 2**31 -1.

**Retention Time** Represents the number of seconds for which a message is retained in the queue table after being dequeued from the queue.

**Retry Delay** Represents the delay time, in seconds, before this message is scheduled for processing again after an application rollback. The default value of this parameter is 0, which means the message can be retried as soon as possible. This parameter has no effect if Max Retires parameter is set to 0.

## Configuring Queue Tables

Use the following steps to configure queue tables.

1. From the Projects Navigator, expand the Databases node and then the Oracle node that contains the queue table.
2. Expand the Queues node and then the Queue Tables node.
3. Right-click the name of the queue table to be configured and select **Configure.**

   Warehouse Builder displays the Configuration tab that contains configuration parameters for the queue table.

4. Choose the parameters you want to configure and click the space to the right of the parameter name to edit its value.

   For each parameter, you can either select an option from a list, type a value, or click the Ellipsis button to display another properties dialog box.

The Generation Options node contains the Generate Queue Table parameter. Set this parameter to True to generate code to create the queue table that will persist the messages of this advanced queue. If the queue table exists in the database, then you need not create it and you can set Generate Queue Table to False.

## Configuring Queue Propagations

Use the following steps to configure queue propagations.

1. From the Projects Navigator, expand the Databases node and then the Oracle node that contains the queue table.
2. Expand the Queues node and then the advanced queue that contains the queue propagation.
3. Right-click the name of the queue propagation to be configured and select **Configure.**

   Warehouse Builder displays the Configuration tab that contains configuration parameters for the queue propagation.
4. Choose the parameters you want to configure and click the space to the right of the parameter name to edit its value.

   For each parameter, you can either select an option from a list, type a value, or click the Ellipsis button to display another properties dialog box.

The configuration parameters are described in the following sections.

**Rule Condition**: Represents a rule condition to check whether the message can be propagated to the subscriber. This parameter is applicable only for non-streams queues.

**Transformation:** Represents the transformation that is applied before propagation to the target queue. This parameter is applicable only for non-streams queues.

#### Generation Options

- **Generate Database Link:** Set this parameter to True to generate a script that creates the database link used for propagation.
- **Generate Queue Propagation:** Set this parameter to True to generate code that creates the queue propagation.
- **Generate Ruleset and Rule for Replication:** Set this parameter to true to generate the code for RULE and RULESET for replication purposes. This parameter is applicable only for streams queues.
- **Generate Schedule Propagation:** Set this parameter to true to generate code for scheduling the queue propagation. This parameter is applicable only for non-streams queues.

#### Replication Options (Only for Streams Queues)

- **Not Permitted Tag Values:** List of comma separated Tag values (in Hexadecimal numbers) that are not allowed for propagation.
- **Permitted Tag Values:** List of comma separated Tag values (in Hexadecimal numbers) that are allowed for propagation.

#### Scheduling Options

- **Duration:** Represents the duration of propagation to be performed. The default value is null. This parameter is applicable only for non-streams queue.
- **Latency:** Represents the latency for the queue propagation. By default the value is 60. This parameter is applicable only for non-streams queue.
- **Next Time:** Represents the next time when the propagation is performed. The default value is null. This parameter is applicable only for non-streams queue.
- **Start Time:** Represents the start time for the propagation. The default value is `SYSDATE`. This parameter is applicable only for non-streams queue.

# Creating Relational Data Objects in Microsoft SQL Server and IBM DB2 UDB

When you create a Microsoft SQL Server module or an IBM DB2 module, you can define data objects such as tables, views, and sequences in this module. Use the editors to define tables and views.

**To define a table, view, or sequence in Microsoft SQL Server or IBM DB2 UDB module:**

1. Expand the project node under which you want to create data objects and then expand the Databases node.
2. For SQL Server, expand the SQL Server module node and then the module in which the data objects are to be created.

   For IBM DB2 UDB, expand the DB2 module node and then the module in which the data objects are to be created.
3. Right-click the node representing the type of object that you want to create and select **New <type of object>**. The editor for the object is displayed.

   For example, to create a table, right-click the Tables node and select **New Table**.
4. Based on the type of object being created, follow the instructions mentioned in one of the following sections:
   - "Creating Table Definitions" on page 2-11
   - "Creating View Definitions" on page 2-15
   - "Creating Sequence Definitions" on page 2-35

### Differences in the Object Editors for Heterogeneous Databases

The following differences exist when you define tables, views, and sequences in the SQL Server or DB2 module:

- The Table Editor and View Editor do not contain the Indexes and Partitions tabs.
- The types of constraints supported for SQL Server and DB2 tables are Primary Key, Foreign Key, Check Constraints, and Unique Keys.
- On the Columns tab, the field Seconds Precision is not available.
- On the Columns tab, the Datatypes list contains the data types that you can use. If you use native access, then the Datatypes list contains the data types native to that platform. If you use Gateways to access the heterogeneous database, then the Datatypes list contains Oracle data types.
- You can use the following attributes in DB2 sequences: START_WITH, INCREMENT_BY, MINVALUE, MAXVALUE, CYCLE, and CACHE.

## Rules for Naming Objects in IBM DB2 UDB

When you import data objects from DB2, the case used in object names is preserved. However, for all objects created using Warehouse Builder, the names are automatically converted to uppercase.

Following are the rules for naming objects in a DB2 module:

- Object names and column names must be unique.
- The maximum length for the object name is 128 characters.

- The maximum length for each column name is 30 characters.
- The following characters are illegal in names: `, *, +, |, [, ], :, ;, ", ', &, <, >, ?, /, and Space
- Names cannot begin with a space, a digit, or with any of the following characters: _, `, &, *, +, |, [, ], :, ;, ", ', <, >, ?, and /.

> **Note:** Enclosed illegal characters are only allowed in names when you import objects. You cannot use illegal characters within Warehouse Builder.

## Rules for Naming Objects in Microsoft SQL Server

When you import data objects from SQL Server, the case used in object names is preserved. However, for all objects created using Warehouse Builder, the names are automatically converted to uppercase.

Following are the rules for naming objects in Microsoft SQL Server:

- Object names and column names must be unique.
- The maximum length for the object name is 128 characters.
- The following characters are illegal in names: ~, `, !, %, ^, &, ;, *, (, ), {, }, [, ], |, \, :, ", /, ?, >, and <.
- Names cannot contain spaces, periods, or mathematical symbols.
- Names cannot begin with a space or with any of the following characters: ~, `, !, %, ^, &, *,( ,) {, }, [, ], |, \, :, ;, ", ', /, ?, <, >, and $.
- Column names cannot begin with mathematical symbols or periods.

> **Note:** Enclosed illegal characters are only allowed in names when you import objects. You cannot use illegal characters within Warehouse Builder.

# 3

# Defining Dimensional Objects

Warehouse Builder enables you to define, deploy, and load dimensional objects. You can deploy dimensional objects either to a relational schema or to an analytical workspace in the database.

This chapter contains the following topics:


- Overview of Dimensional Objects
- Overview of Implementing Dimensional Objects
- Creating Dimensions
- Creating Slowly Changing Dimensions
- Editing Dimension Definitions
- Configuring Dimensions
- Creating Cubes
- Editing Cube Definitions
- Configuring Cubes
- Creating Time Dimensions
- Populating Time Dimensions


## Overview of Dimensional Objects

Objects that contain additional metadata to identify and categorize data are called dimensional objects. Warehouse Builder enables you to design, deploy, and load two types of dimensional objects: dimensions and cubes. In this chapter, *dimensional object* refers to both dimensions and cubes. Most analytic queries require the use of a time dimension. Warehouse Builder provides tools that enable you to easily create and populate time dimensions by answering simple questions.

### Steps to Create Dimensional Objects

Creating dimensional objects consists of following high-level tasks.

1. Define dimensional objects

   Defining dimensional objects consists of specifying the logical relationships that help store data in a more structured format. For example, to define a dimension, you describe its attributes, levels, and hierarchies. To define a cube, you define its measures and dimensions.

You can use wizards or editors to define dimensional objects. For more details, see:

- "Creating Dimensions" on page 3-14
- "Creating Slowly Changing Dimensions" on page 3-31
- "Creating Cubes" on page 3-39
- "Creating Time Dimensions" on page 3-57

2. Implement dimensional objects

   See "Overview of Implementing Dimensional Objects" on page 3-9

3. Deploy dimensional objects

   **See Also:**

   - "About Deploying Dimensional Objects" on page 12-3
   - "Deploying Objects" on page 12-6

4. Load dimensional objects

   To load data into dimensional objects, create a mapping that defines the data flow and transformations from the source objects to the dimensional object. You then deploy and execute this mapping.

   **See Also:**

   - "Using Dimensions in ETL Mappings" on page 6-1
   - "Using Cubes in ETL Mappings" on page 6-12

## Overview of Dimensions

A dimension is a structure that organizes data. Examples of commonly used dimensions are Customers, Time, and Products.

For relational dimensions, using dimensions improves query performance because users often analyze data by drilling down on known hierarchies. An example of a hierarchy is the Time hierarchy of year, quarter, month, day. The Oracle Database uses these defined hierarchies by rewriting queries that retrieve data from materialized views rather than detail tables.

A dimension consists of a set of levels and a set of hierarchies defined over these levels. To create a dimension, you must define the following:

- Dimension attributes
- Levels
- Level attributes

  This includes surrogate and business identifiers for levels.

- Hierarchies

  **See Also:** *Oracle Warehouse Builder Concepts* for more information about defining dimension attributes, levels, level attributes, and hierarchies

### Overview of Surrogate Identifiers

A surrogate identifier uniquely identifies each level record across all the levels of the dimension. It must be composed of a single attribute. Surrogate identifiers enable you to hook facts to any dimension level as opposed to the lowest dimension level only. For a dimension that has a relational or ROLAP implementation, the surrogate identifier should be of the data type `NUMBER`.

You need to use a surrogate key if:

- your dimension is a Type 2 or Type 3 SCD. In these cases, we can have multiple dimension records loaded for each business key value, so we need an extra unique key to track these records.
- your dimension contains more that one level and is implemented using a star schema. Thus, any cube that references such a dimension will reference more than one dimension level.

  If no surrogate key is defined, then only the leaf-level dimension records are saved in the dimension table, the parent level information is stored in extra columns in the leaf-level records. But there is no unique way to reference the upper level in that case.

You do not need a surrogate key for any Type 1 dimensions, implemented by star or snowflake, where only the leaf level(s) are referenced by a cube. Dimensions with multiple hierarchies will still work with no surrogate key, as long as only the leaf levels are referenced by the cube.

## Overview of Slowly Changing Dimensions

A Slowly Changing Dimension (SCD) is a dimension that stores and manages both current and historical data over time in a data warehouse. In data warehousing, there are three commonly recognized types of SCDs. describes the types of SCDs, as described in Table 3–1.

***Table 3–1 Types of Slowly Changing Dimensions***

| Type | Description |
|---|---|
| Type 1 | Stores only one version of the dimension record. When a change is made, the record is overwritten and no historic data is stored. |
| Type 2 | Stores multiple versions of the same dimension record. When the dimension record is modified, new versions are created while the old ones are retained. |
| Type 3 | Stores one version of the dimension record. This record stores the previous value and current value of selected attributes. |

Use Type 2 and Type 3 SCDs to store and manage both current and historical data over time in a data warehouse. Type 1 dimensions, referred to as dimensions, do not preserve historical data.

### Additional Attributes for Slowly Changing Dimensions (SCDs)

To create a Type 2 SCD or a Type 3 SCD, in addition to the regular dimension attributes, you need additional attributes that perform the following roles:

- **Triggering Attributes:** These are attributes for which historical values must be stored. For example, in the PRODUCTS dimension, the attribute PACKAGE_TYPE of the Product level can be a triggering attribute. This means that when the value of this attribute changes, the old value needs to be stored.
- **Effective Date:** This attribute stores the start date of the record's life span.

- **Expiration Date:** This attribute stores the end date of the record's life span.
- **Previous Attribute:** For Type 3 SCDs only, this attribute stores the previous value of a versioned attribute.

An attribute can play only one of the above roles. For example, an attribute cannot be a regular attribute and an effective date attribute. When you use the wizard to create a Type 2 SCD or a Type 3 SCD, Warehouse Builder creates the required additional attributes.

### Overview of Defining Type 2 Slowly Changing Dimensions

A Type 2 SCD retains the full history of values. When the value of a triggering attribute changes, the current record is closed. A new record is created with the changed data values and this new record becomes the current record. Each record contains the effective date and expiration date to identify the time period for which the record was active. Warehouse Builder also enables you to set a specific non-null date value as the expiration date. The current record is the one with a null or the previously specified value in the expiration date.

All the levels in a dimension need not store historical data. Typically, only the lowest levels is versioned.

To define a Type 2 Slowly Changing Dimension (SCD), you must identify the following:

- For the level that stores historical data, specify the attributes used as the effective date and the expiration date.
- Choose the level attribute(s) that will trigger a version of history to be created.

  You cannot choose the surrogate identifier, effective date attribute, or expiration date attribute as the triggering attribute.

Each version of a record is assigned a different surrogate identifier. The business identifier connects the different versions together in a logical sense. Typically, if there is a business need, Type 2 SCDs are used.

#### Type 2 SCD Example

Consider the `Customers` Type 2 SCD that contains two levels, Household and Customer. Customer is the leaf level and Household is the non-leaf level. Table 3–2 lists dimension attributes of the `Customers` Type 2 SCD.

***Table 3–2 Dimension Attributes of the Customers Type 2***

| Attribute Name | Identifier |
|---|---|
| `ID` | Surrogate identifier |
| `BUSN_ID` | Business identifier |
| `ADDRESS` | |
| `ZIP` | |
| `MARITAL_STATUS` | |
| `HOME_PHONE` | |
| `EFFECTIVE_DATE` | Effective Date |
| `EXPIRATION_DATE` | Expiration Date |

The Household level implements the following attributes: ID, BUSN_ID, ADDRESS, ZIP, EFFECTIVE_DATE, and EXPIRATION_DATE. The Customer level implements the following attributes: ID, BUSN_ID, MARITAL_STATUS, HOME_PHONE, EFFECTIVE_DATE, and EXPIRATION_DATE.

The Customers_tab table implements the Customers Type 2 SCD (for a relational or ROLAP implementation). Table 3–3 lists the columns in the Customers_tab table, along with details about the dimension level and the attribute that each column implements.

***Table 3–3 Columns that Implement the Customers Type 2 SCD Level Attributes***

| Column Name in the Customers_tab table | Level Name | Dimension Attribute Name |
|---|---|---|
| DIMENSION_KEY | | |
| H_ID | Household | ID |
| H_BUSN_ID | Household | BUSN_ID |
| H_ADDRESS | Household | ADDRESS |
| H_ZIP | Household | ZIP |
| H_EFFECTIVE_DATE | Household | EFFECTIVE_DATE |
| H_EXPIRATION_DATE | Household | EXPIRATION_DATE |
| C_ID | Customer | ID |
| C_BUSN_ID | Customer | BUSN_ID |
| C_MARITAL_STATUS | Customer | MARITAL_STATUS |
| C_HOME_PHONE | Customer | HOME_PHONE |
| C_EFFECTIVE_DATE | Customer | EFFECTIVE_DATE |
| C_EXPIRATION_DATE | Customer | EXPIRATION_DATE |

**To create the Customers Type 2 SCD:**

- Specify that the ZIP attribute of the Household level and the MARITAL_STATUS attribute of the Customer level are the triggering attributes.
- Use two additional attributes to store the effective date and the expiration date of the level records. When you use the Create Dimension wizard, Warehouse Builder creates these additional attributes for the lowest level only. If you use the Dimension Editor, then you must explicitly create these attributes and apply them to the required levels.

## Overview of Hierarchy Versioning

For Type 2 SCDs, when the non-leaf level of a dimension contains versioned attributes, the versioning of this non-leaf level results in the versioning of its corresponding child records, if they have effective date and expiration date attributes. For example, in the Customers Type 2 SCD described in "Type 2 SCD Example" on page 3-4, when the value of the H_ZIP is updated in a particular Household level record, the child records corresponding to this Household level are automatically versioned.

Hierarchy versioning is not enabled by default for Type 2 SCDs. Thus, when you create a Type 2 SCD using the Create Dimension Wizard, hierarchy versioning is disabled. Use the Dimension Editor to enable hierarchy versioning.

### Steps to Enable Hierarchy Versioning

1. Right-click the Type 2 SCD in the Projects Navigator and select **Open**.

   The Dimension Editor is displayed.

2. Navigate to the SCD tab.

3. Click **Settings** to the right of the Type 2: Store the Complete change history option.

   The Type 2 Slowly Changing Dimension dialog box is displayed. The attributes of each level are displayed under the level node.

4. In the child level that should be versioned when its parent attribute changes, for the attribute that represents the parent attribute of this child level, select **Trigger History** in the Record History column.

   For example, you create the `Customers` Type 2 SCD using the Create Dimension Wizard. Then open the editor for this Type 2 SCD and navigate to the Type 2 Slowly changing Dimension dialog box. The `Customer` level has an attribute called `HOUSEHOLD_ID`. This attribute represents the parent attribute of each `Customer` record. For the `HOUSEHOLD_ID` attribute, select **Trigger History** in the Record History column.

## Overview of Defining Type 3 Slowly Changing Dimensions (SCDs)

A Type 3 Slowly Changing Dimension (SCD) stores two versions of values for certain selected level attributes. Each record stores the previous value and the current value of the versioned attributes. When the value of any of the versioned attributes changes, the current value is stored as the old value and the new value becomes the current value. Each record stores the effective date that identifies the date from which the current value is active. This doubles the number of columns for the versioned attributes and is used rarely.

Before you define a Type 3 SCD, identify the following:

1. For each level, specify which attributes should be versioned. That is, identify which attributes should store the previous value as well as the current value.

2. For each versioned attribute, specify the attribute that stores the previous value.

   The following restrictions apply to attributes that can have a previous value.

   - An attribute specified as a previous value cannot have further previous values.
   - The surrogate identifier cannot have previous values.

3. For each level that is versioned, specify the attribute that stores the effective date.

Warehouse Builder recommends that you do not include previous value attributes in the business identifier of a Type 3 SCD.

### Type 3 SCD Example

The `PRODUCTS` dimension described in "Dimension Example" on page 3-14 can be created as a Type 3 SCD. The attributes `PACKAGE_TYPE` and `PACKAGE_SIZE` of the Product level should be versioned. You define two additional attributes to store the previous values, say `PREV_PACK_SIZE` and `PREV_PACK_TYPE` in the Product level. Suppose the value of the `PACKAGE_TYPE` attribute changes, Warehouse Builder stores the current value of this attribute in `PREV_PACK_TYPE` and stores the new value in the `PACKAGE_TYPE` attribute. The effective date attribute can be set to the current system date or to any other specified date.

## Overview of Cubes

Cubes contain measures and link to one or more dimensions. The axes of a cube contain dimension members and the body of the cube contains measure values. Most measures are additive. For example, sales data can be organized into a cube whose edges contain values for Time, Products, and Promotions dimensions and whose body contains values from the measures Value sales, and Dollar sales.

A cube is linked to dimension tables over foreign key constraints. Since data integrity is vital, these constraints are critical in a data warehousing environment. The constraints enforce referential integrity during the daily operations of the data warehouse.

Data analysis applications typically aggregate data across many dimensions. This enables them to look for anomalies or unusual patterns in the data. Using cubes is the most efficient way of performing these type of operations. In a relational implementation, when you design dimensions with warehouse keys, the cube row length is usually reduced. This is because warehouse keys are shorter than their natural counterparts. This results is lesser amount of storage space needed for the cube data. For a MOLAP implementation, OLAP uses VARCHAR2 keys.

A typical cube contains:

- A primary key defined on a set of foreign key reference columns or, in the case of a data list, on an artificial key or a set of warehouse key columns. When the cube is a data list, the foreign key reference columns do not uniquely identify each row in the cube.
- A set of foreign key reference columns that link the table with its dimensions.

To create cubes, you must define the cube measures and the cube dimensionality. For more information about measures and cube dimensionality, see *Oracle Warehouse Builder Concepts*.

## Orphan Management for Dimensional Objects

Warehouse Builder's orphan management policy enables you to manage orphan records in dimensional objects (dimensions and cubes). An orphan record is one that does not have a corresponding existing parent record. Orphan management automates the process of handling source rows that do not meet the requirements necessary to form a valid dimension or cube record.

Orphan records can occur when:

- a record that is loaded into a dimensional object does not have a corresponding parent record.

  A dimension record is considered an orphan if one or more of its level references is null or nonexistant. A cube record is considered an orphan if one or more dimension records that it references is either nonexistent or null.
- a record is deleted from a dimensional object. This could result in the child records of the deleted record not having an existing parent record.

Warehouse Builder enables you to specify different orphan management policies for loading dimensional object data and for removing dimensional object data.

---

**Note:** Orphan management is not supported for MOLAP dimensions and cubes.

---

### Orphan Management While Loading Data Into Dimensional Objects

An orphan record is created while loading data into a dimensional object if you insert a record that does not have an existing parent record. For example, you load data into the City level of the Geography dimension. The value of the State attribute in this record does not exist in the State level. This record is an orphan record. Or you load data into the SALES cube, but the value for the Customer ID does not exist in the Customers dimension.

Warehouse Builder enables you to specify the integrity policy used while loading orphan records into a dimensional object. You can specify different actions for records that have a null parent record and records that have an invalid parent record.

The orphan management policy options that you can set for loading are:

- **Reject Load**: The record is not inserted.
- **Default Parent:** You can specify a default parent record. This default record is used as the parent record for any record that does not have an existing parent record. If the default parent record does not exist, then Warehouse Builder creates the default parent record.

  You specify the attribute values of the default parent record at the time of defining the dimensional object. If any ancestor of the default parent does not exist, then Warehouse Builder also creates this record.
- **No Maintenance:** This is the default behavior. Warehouse Builder does not actively detect, reject, or fix orphan records.

  **See Also:**

  - "Orphan Tab" on page 3-28 for details about setting an orphan management policy for dimensions
  - "Orphan Tab" on page 3-51 for details about setting an orphan management policy for cubes

### Orphan Management While Removing Data From Dimensional Objects

Orphan records can be created while removing data if the record that is being removed has corresponding child records. For example, you remove a record in the State level of the Geography dimension. This state has city records that refer to it. All the city records that refer to the deleted state record will become orphan records.

While removing data from a dimension, you can select one of the following orphan management policies:

- **Reject Removal**: Warehouse Builder does not allow you to delete the record if it has existing child records.
- **No Maintenance:** This is the default behavior. Warehouse Builder does not actively detect, reject, or fix orphan records.

### Error Tables

Error tables store any records that are detected as anomalous, by Orphan Management, during a load or remove operation on a dimension. Error tables are created when you deploy a dimension for the first time if you select the Deploy Error Tables option on the Orphan tab of the dimension editor.

Following are the records that appear in error tables:

- Records that are not inserted during a load operation

- Records whose parents are defaulted during the load operation
- Records that could not be deleted during a remove operation

Warehouse Builder creates one error table for each implementation object. For example, if a dimension is implemented using a snowflake schema, then multiple error tables are created. If the dimension is implemented using a star schema, then one error table is created. The name of the error table is the same as the implementation object suffixed with an _ERR. If the implementation table is called `CITY`, then the error table is called `CITY_ERR`.

> **Note:** Since orphan management is not supported for MOLAP dimensional objects, error tables are created for dimensions and cubes that have a relational or ROLAP implementation only.

## Overview of Implementing Dimensional Objects

To implement a dimensional object is to create the physical structure of the dimensional object. Warehouse Builder provides the following implementations for dimensional objects:

- Relational Implementation of Dimensional Objects
- ROLAP Implementation of Dimensional Objects
- MOLAP Implementation of Dimensional Objects

> **Note:** To use a MOLAP implementation, you must have the following:
>
> - Oracle Database 10*g* Enterprise Edition with the OLAP option
> - Oracle Database 11*g* Enterprise Edition with the OLAP option
> - OLAP 10.1.0.4 or higher

The implementation is set using the Storage page of the Wizard that is used to create the dimensional object or the Storage tab of the object editor. You can further refine the implementation deployment options using the Deployment Option configuration parameter. For more information about setting this parameter, see "Configuring Dimensions" on page 3-37 and "Configuring Cubes" on page 3-55.

### Relational Implementation of Dimensional Objects

A relational implementation stores the dimensional object and its data in a relational form in the database. The dimensional object data is stored in implementation objects that are typically tables. Any queries that are executed on the dimensional object obtain data from these tables. Warehouse Builder creates the DDL scripts that create the dimensional object. You can then deploy these scripts to the database using the Control Center. For relational dimensions, Warehouse Builder can use a star schema, a snowflake schema, or a manual schema to store the implementation objects.

> **See Also:** *Oracle Warehouse Builder Concepts* for more information about how the star schema and snowflake schema store dimension data

When you use the wizard to define dimensional objects, Warehouse Builder creates the database tables that store the dimensional object data. It also defines the association between the dimension object attributes and the implementation tables that defines the table columns that store the dimensional object data.

When you define a dimensional object using the editors, you can decide whether you want Warehouse Builder to create the implementation tables or you want to store the dimensional object data in your own tables and views. If you want Warehouse Builder to create implementation objects, then perform auto binding for the dimensional object. To use your own implementation tables to store the dimensional object data, perform manual binding.

---

**Note:** For a relational implementation, you cannot view the data stored in the dimensional object using the Data Viewer. However, you can view the data stored in the implementation tables of the dimensional object using the Data Viewer.

---

## Binding

Binding is the process of connecting the attributes of the dimensional object to the columns in the table or view that store their data. You perform binding only for dimensional objects that have a relational or ROLAP implementation. For multidimensional objects, binding is implicit and is resolved in the analytic workspace.

For dimensions, you connect the level attributes and level relationships to the columns in the implementation objects. For cubes, you connect the measures and dimension references to implementation table columns.

Warehouse Builder provides two methods of binding: Auto Binding and Manual Binding.

### When to Perform Binding

- When you create a dimensional object using the wizard, the object will be bound for you. If you make any changes to the dimensional object using the editor, then you must re-bind the object before you deploy them.
- When you create a dimensional object using the editor, you must bind the dimensional object to its implementation objects before deployment.
- When you make any change to a dimensional object definition using the editors, you must rebind the dimensional object to its implementation objects.

## Auto Binding

Auto binding refers to the creation of the implementation tables, if they do not already exist. The attributes and relationships of the dimensional object are then bound to the columns that store their data. In the case of a dimension, the number of tables used to store the dimension data depends on the options you select for the storage. You can perform auto binding using both the wizards and the editors. When you use the editors to create dimensional objects, you can perform both auto binding and manual binding.

**To perform auto binding:**

1. In the Projects Navigator, right-click the dimensional object and select **Open**.

   The editor for this dimensional object is displayed.

2. On the Physical Bindings tab, select node that represents the dimensional object.
3. From the File menu, select **Bind**.

   If the Bind option is not enabled, then verify if the dimensional object uses a relational or ROLAP implementation. In the case of dimensions, ensure that the Manual option is not set in the Implementation section of the Storage tab.

Alternatively, you can perform auto binding by right-clicking the dimensional object in the Projects Navigator and selecting **Bind**.

### Manual Binding

In manual binding, you must explicitly bind the attributes of the dimensional objects to the database columns that store their data. You use manual binding when you want to bind a dimensional object to existing tables or views. If a dimensional object is already bound to certain implementation objects (as shown on the Physical Bindings tab of the Dimension Editor), then unbind the dimensional object and then perform manual binding. For details about unbinding dimensional objects, see "Unbinding" on page 3-12.

**To perform manual binding for a dimensional object:**

1. Create the implementation objects (tables or views) that you will use to store the dimensional object data.

   In the case of relational or ROLAP dimensions, create the sequence used to load the surrogate identifier of the dimension. You can choose to use an existing sequence.
2. In the Projects Navigator, right-click the dimensional and select **Open**.

   The editor for the dimensional object is displayed.
3. On the Physical Bindings tab, right-click a blank area, select **Add** and then the type of object that represents the implementation object.

   Warehouse Builder displays the Add a New or Existing <Object> dialog box. For example, if the dimension data is stored in a table, then right-click a blank area on the Physical Bindings tab, select **Add** and then **Table**. The Add a New or Existing Table dialog box is displayed.
4. Choose the **Select an existing <Object>** option and then select the data object from the list of objects displayed in the selection tree.
5. Click **OK**.

   A node representing the object that you just added is displayed on the canvas.
6. For dimensions, if more than one data object is used to store the dimension data, perform steps 3 to 5 for each data implementation object.
7. For dimensions, map the attributes in each level of the dimension to the columns that store their data. Also map the level relationships to the database column that store their data.

   For cubes, map the measures and dimension references to the columns that store the cube data.

   To map to the implementation object columns, hold down your mouse on the dimension or cube attribute, drag, and then drop on the column that stores the attribute value.

   For example, for the `PRODUCTS` dimension described in "Dimension Example" on page 3-14, the attribute `NAME` in the Groups level of the `PRODUCTS` dimension is

stored in the GROUP_NAME attribute of the PRODUCTS_TAB table. Hold down the mouse on the NAME attribute, drag, and drop on the GROUP_NAME attribute of the PRODUCTS_TAB table.

### Unbinding

Warehouse Builder also enables you to unbind a dimensional object. Unbinding removes the connections between the dimensional object and the tables that store its data.

To unbind a dimensional object from its current implementation, select the dimensional object in the Projects Navigator and, from the File menu, select **Unbind**. Unbinding removes the bindings between the dimensional object and its implementation objects. However, it does not delete the implementation objects.

## ROLAP Implementation of Dimensional Objects

A ROLAP implementation, like a relational implementation, stores the dimensional object and its data in a relational form in the database. Additionally, depending on the type of ROLAP implementation, it either creates CWM2 metadata in the OLAP catalog or OLAP cube materialized views.

This section describes how ROLAP implementations are classified.

### ROLAP Implementation

The dimensional object and its data are stored in a relational form in the database and the CWM2 metadata for the dimensional object is stored in the OLAP catalog. This enables you to query the dimensional object from Discoverer (for OLAP).

### ROLAP with MVs Implementation

The dimensional object and its data are stored in a relational form in the database. Additionally, cube-organized materialized views are created in an analytic workspace.

---

**Note:** In Oracle Warehouse Builder 11*g* Release 2 (11.2), only star schema tables are supported for the ROLAP with MVs implementation.

---

### About OLAP Catalog

The OLAP catalog is the metadata repository provided for the OLAP option in the Oracle Database. This metadata describes the data stored in relational tables.

When you deploy a dimensional object using Warehouse Builder, you can specify if the dimensional object metadata should be stored in the OLAP catalog.

OLAP metadata is dynamically projected through a series of views called the active catalog views (views whose names begin with ALL_CWM2_AW).

In Oracle Database 10*g*, the OLAP catalog metadata is used by OLAP tools and applications to access data stored in relational star and snowflake schemas. External application such as Discoverer use the OLAP catalog to query relational and multidimensional data. The application does not need to be aware of whether the data is located in relational tables or in analytic workspaces, nor does it need to know the mechanism for accessing it.

The OLAP catalog uses the metadata it stores to access data stored in relational tables or views. The OLAP catalog defines logical multidimensional objects and maps them

to the physical data sources. The logical objects are dimensions and cubes. The physical data sources are columns of a relational table or view.

## MOLAP Implementation of Dimensional Objects

In a MOLAP implementation, the dimensional object data is stored in an analytic workspace in Oracle Database 10*g* or Oracle Database 11*g*. This analytic workspace, in turn, is stored in the database.

If the Oracle location of the computer containing the AW uses Oracle Database 10*g*, then the OLAP 10*g* form analytic workspaces are generated. If the location used Oracle Database 11*g*, then the OLAP 11*g* form analytic workspaces are generated.

### Analytic Workspace

An analytic workspace is a container within the Oracle Database that stores data in a multidimensional format. Analytic workspaces provide the best support to OLAP processing. An analytic workspace can contain a variety of objects such as dimensions and variables.

An analytic workspace is stored in a relational database table, which can be partitioned across multiple disk drives like any other table. You can create many analytic workspaces within a single schema to share among users. An analytic workspace is owned by a particular user and other users can be granted access to it. The name of a dimensional object must be unique within the owner's schema. For more information about analytic workspaces, see *Oracle OLAP User's Guide*.

## Deployment Options for Dimensional Objects

After you define dimensional objects, you must deploy them to instantiate them in the database. To specify the type of implementation for dimensional objects, you set the configuration parameter Deployment Option.

Warehouse Builder provides the following deployment options for dimensions: Deploy All, Deploy Data Objects Only, Deploy to Catalog, and Deploy Aggregation.

**Deploy All** For a relational or ROLAP implementation, the dimension is deployed to the database and a CWM definition to the OLAP catalog. For a ROLAP with MVs implementation, the dimension is deployed to the database and cube-organized materialized views are created in an analytic workspace. For a MOLAP implementation, the dimension is deployed to the analytic workspace.

**Deploy Data Objects Only** Deploys the dimension only to the database. You can select this option only for dimensions that use a relational or a ROLAP implementation.

**Deploy to Catalog** Deploys the CWM definition to the OLAP catalog only. Use this option if you want applications such as Discoverer for OLAP to access the dimension data after you deploy data only. You can also use this option if you previously deployed with "Data Objects Only" and now want to deploy the CWM Catalog definitions without redeploying the data objects again.

**Deploy Aggregation** Deploys the aggregations defined on the cube measures. This option is available only for cubes.

# Creating Dimensions

To create dimensions, use one of the following methods:

- Creating Dimensions Using the Create Dimension Wizard

  The wizard enables you to create a fully functional dimension object quickly. When you use the wizard, many settings are defaulted to the most commonly used values. You can modify these settings later using the Dimension Editor. If you choose a relational implementation for the dimension, then the implementation tables and the dimension bindings are also created in the workspace.

  For more information about the defaults used by the Dimension wizard, see "Defaults Used By the Create Dimension Wizard" on page 3-20.

- Creating Dimensions Using the Dimension Editor

  The Dimension Editor gives you full control over all aspects of the dimension definition and implementation. This provides maximum flexibility. Use the editor to create a dimension from scratch or to edit a previously created dimension.

- Using the Time Dimension wizard

  The Time Dimension wizard enables you to create and populate time dimensions. For more information about the Time Dimension wizard, see "Creating Time Dimensions" on page 3-57.

## Dimension Example

An example of a dimension is the Products dimension that you use to organize product data. Table 3–4 lists the levels in the `PRODUCTS` dimension and the surrogate identifier and business identifier for each of the levels in the dimension.

***Table 3–4 Products Dimension Level Details***

| Level | Attribute Name | Identifier |
|---|---|---|
| Total | ID | Surrogate |
| | Name | Business |
| | Description | |
| Groups | ID | Surrogate |
| | Name | Business |
| | Description | |
| Product | ID | Surrogate |
| | UPC | Business |
| | Name | |
| | Description | |
| | Package Type | |
| | Package Size | |

The `PRODUCTS` dimension contains the following hierarchy:

Hierarchy 1: Total > Groups > Product

## Creating Dimensions Using the Create Dimension Wizard

**To create a dimension using the Create Dimension wizard:**

1. From the Projects Navigator expand the Databases node and then the Oracle node.
2. Expand the module where you want to create the dimension.
3. Right-click the Dimensions node and select **New Dimension**.

   Warehouse Builder displays the Welcome page of the Create Dimension wizard. Click **Next** to proceed. The wizard guides you through the following pages:

   - Name and Description Page on page 3-15
   - Storage Type Page on page 3-15
   - Dimension Attributes Page on page 3-17
   - Levels Page on page 3-18
   - Level Attributes Page on page 3-18
   - (Relational and ROLAP dimensions only) Slowly Changing Dimension Page on page 3-19
   - Pre Create Settings Page on page 3-19
   - Dimension Creation Progress Page on page 3-20
   - Summary Page on page 3-20

### Name and Description Page

Use the Name and Description page to describe your dimension. Enter the following information on this page:

- **Name:** This is the name used to refer to the dimension. The dimension name must be unique within a module.
- **Description:** You can type an optional description for the dimension.

### Storage Type Page

Use the Storage Type page to specify the type of storage for the dimension. The storage type determines how the dimension data is physically stored in the database. The options you can select for storage type are:

- ROLAP: Relational storage
- ROLAP: with MVs
- MOLAP: Multidimensional storage

You select the storage type based on the volume of data stored at the lowest level of the entire cube and the refresh rate required.

**ROLAP: Relational storage** Warehouse Builder stores the dimension definition and its data in a relational form in the database. Select this option to create a dimension that uses a relational or ROLAP implementation.

Relational storage is preferable if you want to store detailed, high volume data or you have high refresh rates combined with high volumes of data. Use relational storage if you want to perform one of the following:

- Store detailed information such as call detail records, point of sales (POS) records and other such transaction oriented data.

- Refresh high volumes of data at short intervals.
- Detailed reporting such as lists of order details.
- Ad hoc queries in which changing needs require more flexibility in the data model.

Operational data stores and enterprise data warehouses are typically implemented using relational storage. You can then derive multi-dimensional implementations from this relational implementation to perform different analysis types.

If the database containing the target schema has the OLAP option installed, then you can also deploy the dimensions to the OLAP catalog.

When you choose a relational implementation for a dimension, the implementation tables used to store the dimension data are created. The default implementation of the dimension is using a star schema. This means that the data for all the levels in the dimension is stored in a single database table.

**ROLAP: with MVs** Warehouse Builder stores the dimension definition and its data in a relational form in the database. Additionally, cube-organized MVs are created in the analytic workspace. Select this option to create a dimension that uses a relational implementation and stores summaries in the analytic workspace. Using this option provides summary management based on cube-organized MVs in Oracle 11*g* Database. Query performance is greatly improved, without the need to make any modification to your queries.

When you choose a ROLAP with MVs implementation:

- the implementation tables used to store the dimension data are created. The default implementation of the dimension is using a star schema.
- the dimension is stored in an analytic workspace that uses the same name as the Oracle module to which the dimension belongs. The tablespace that is used to store the analytic workspace is the tablespace that is defined as the users tablespace for the schema that contains the dimension metadata.

**MOLAP: Multidimensional storage** Warehouse Builder stores the dimension definition and dimension data in an analytic workspace in the database. Select this option to create a dimension that uses a MOLAP implementation.

Multidimensional storage is preferable when you want to store aggregated data for analysis. The refresh intervals for a multidimensional storage are usually longer than relational storage as data needs to be pre-calculated and pre-aggregated. Also, the data volumes are typically smaller due to higher aggregation levels. Use multidimensional storage to perform the following:

- Advanced analysis such as trend analysis, what-if analysis, or to forecast and allocate data.
- Constant analysis using a well-defined consistent data model with fixed query patterns.

When you choose a MOLAP implementation, the dimension is stored in an analytic workspace that uses the same name as the Oracle module to which the dimension belongs. The tablespace that is used to store the analytic workspace is the tablespace that is defined as the users tablespace for the schema that contains the dimension metadata.

> **Note:** For information about certain limitations of deploying dimensions to the OLAP catalog, see "Limitations of Deploying Dimensions to the OLAP Catalog" on page 3-30.

### Dimension Attributes Page

Use the Dimension Attributes page to define the dimension attributes. A dimension attribute is applicable to one or more levels in the dimension. By default, the following attributes are created for each dimension: ID, Name, and Description. You can rename the ID attribute or delete it.

Specify the following details for each dimension attribute:

- **Name:** This is the name of the dimension attribute. The name must be unique within the dimension.
- **Description:** Type an optional description for the dimension attribute.
- **Identifier:** Select the type of dimension attribute. Select one of the following options:

  **Surrogate:** Indicates that the attribute is the surrogate identifier of the dimension. Specifying a surrogate identifier for a dimension is optional.

  **Business:** Indicates that the attribute is the business identifier of the dimension

  **Parent:** Since you can create values-based hierarchies only using the Dimension Editor, this option is displayed only in the Attributes tab of the Dimension Editor. In a value-based hierarchy, selecting Parent indicates that the attribute stores the parent value of an attribute. If the attribute is a regular dimension attribute, then leave this field blank.

  > **Note:** You can create value-based hierarchies only when you choose a MOLAP implementation for the dimension.

  The options displayed in the Identifier list depend on the type of dimension. When you create a dimension with a relational or ROLAP implementation, only the Surrogate and Business options are displayed. For MOLAP dimensions, only the Business and Parent options are displayed.
- **Data Type:** Select the data type of the dimension attribute from the list.

  > **Note:** The following data types are not supported for MOLAP implementations: `BLOB`, `INTERVAL DAY TO SECOND`, `INTERVAL YEAR TO MONTH`, `RAW`, `TIMESTAMP WITH TIME ZONE`, `TIMESTAMP WITH LOCAL TIME ZONE`.

- **Length:** For character data types, specify the length of the attribute.
- **Precision:** For numeric data types, define the total number of digits allowed for the column.
- **Scale:** For numeric data types, define the total number of digits to the right of the decimal point.
- **Seconds Precision:** Represents the number of digits in the fractional part of the datetime field. It can be a number between 0 and 9. The seconds precision is used

only for `TIMESTAMP`, `TIMESTAMP WITH TIME ZONE`, and `TIMESTAMP WITH LOCAL TIME ZONE` data types.

- **Descriptor:** Select the type of descriptor. The options are: Short Description, Long Description, End date, Time span, Prior period, and Year Ago Period.

  Descriptors are very important for MOLAP implementations. For example, in a custom time dimension, you must have Time Span and End Date to allow time series analysis.

### Levels Page

The Levels page defines the levels of aggregation in the dimension. A dimension must contain at least one level. The only exception is value-based hierarchies that contain no levels. You can create a value-based hierarchy using the Dimension Editor only.

Enter the following details on the Levels page:

- **Name:** This is the name of the level. The level name must be unique within the dimension.
- **Description:** Type an optional description for the level.

List the levels in the dimension such that the parent levels appear above the child levels. Use the arrow keys to move levels so that they appear in this order.

Warehouse Builder creates a default hierarchy called STANDARD that contains the levels in the same order that you listed them on the Levels page. The attributes used to store the parent key references of each level are also created. For a relational or ROLAP dimension, two attributes are created, one for the surrogate identifier and one for the business identifier, that correspond to the parent level of each level. For a MOLAP dimension, for each level, one attribute that corresponds to the business identifier of the parent level is created.

For example, the Products dimension contains the following levels: Total, Groups, and Product. Two level relationships are created in the dimension, one each under the Product and Groups levels. For relational or ROLAP dimensions, these level relationships reference the surrogate identifier of the parent level. Level relationships are only displayed in the Physical Bindings Tab of the Dimension Editor.

---

**Note:** To create additional hierarchies, use the Hierarchies tab of the Dimension Editor as described in Hierarchies Tab on page 3-26.

---

### Level Attributes Page

The Level Attributes page defines the level attributes of each dimension level. You define level attributes by selecting the dimension attributes that apply to the level. The dimension attributes are defined on the Dimension Attributes page of the Create Dimension wizard.

The Level Attributes page contains two sections: Levels and Level Attributes.

**Levels** The Levels section lists all the levels defined in the Levels page of the Create Dimension wizard. Select a level in this section to specify the dimension attributes that this level implements. You select a level by clicking the level name.

**Level Attributes** The Level Attributes section lists all the dimension attributes defined in the Dimension Attributes page. For each level, choose the dimension attributes that the level implements. To indicate that a dimension attribute is implemented by a level, select the **Applicable** option for the dimension attribute. The

name of the level attribute can be different from that of the dimension attribute. Use the **Level Attribute Name** field to specify the name of the level attribute.

For example, to specify that the dimension attributes ID, Name, Description, and Budget are implemented by the State level:

1. Select the State level in the Levels section.
2. In the Level Attributes section, select the **Applicable** option for the attributes ID, Name, Description, and Budget.

By default, the following defaults are used:

- The attributes ID, Name, and Description are applicable to all levels.
- All dimension attributes are applicable to the lowest level in the dimension.

### Slowly Changing Dimension Page

Use of this functionality requires the Warehouse Builder Enterprise ETL Option.

The Slowly Changing Dimension page enables you to define the type of slowly changing policy used by the dimension. This page is displayed only if you had chosen **Relational storage (ROLAP)** as the storage type on the Storage Type Page.

For more information about Slowly Changing Dimensions concepts, see *Oracle Warehouse Builder Concepts*.

Select one of the following options for the slowly changing policy:

- **Type 1: Do not store history:** This is the default selection. Warehouse Builder creates a dimension that stores no history. This is a normal dimension.
- **Type 2: Store the complete change history:** Select this option to create a Type 2 Slowly Changing Dimension. Warehouse Builder creates the following two additional dimension attributes and makes them applicable for the lowest level in the Type 2 SCD:
  - Effective date
  - Expiration date

  All the attributes of the lowest level in the Type 2 SCD, except the surrogate and business identifier, are defined as the triggering attributes.

  > **Note:** You cannot create a Type 2 or Type 3 Slowly Changing Dimension if the type of storage is MOLAP.

- **Type 3: Store only the previous value:** Select this option to create a Type 3 Slowly Changing Dimension. Warehouse Builder assumes that all the level attributes at the lowest level, excluding the surrogate ID and business ID, should be versioned. For each level attribute that is versioned, an additional attribute is created to store the previous value of the attribute.

### Pre Create Settings Page

The Pre Create Settings page displays a summary of the options selected on the previous pages of the Create Dimension wizard. This includes the attributes, levels, hierarchies, storage type, and the slowly changing policy used for the dimension. Warehouse Builder uses these settings to create the dimension definition and the database tables that implement the dimension. It also binds the dimension attributes to the table columns that store the attribute data.

Click **Next** to proceed with the implementation of the dimension. To change any of the options you previously selected, click **Back**.

> **Note:** Review this page carefully as it summarizes the implementation and its objects.

### Dimension Creation Progress Page

The Dimension Creation Progress page displays the progress of the dimension implementation that was started on the Pre-Create Settings page. The Message Log section on this page provides information about the individual tasks completed during the dimension implementation. Click **Next** to proceed.

### Summary Page

The Summary page provides a brief summary of the options that you selected using the Create Dimension wizard. Use the Summary page to review the selected options. Click **Finish** to create the dimension. You now have a fully functional dimension. This dimension is displayed under the Dimensions node of the Projects Navigator.

Warehouse Builder creates the metadata for the following in the workspace:

- The dimension object.
- The objects that store the dimension data.

  For a relational implementation, a database table that stores the dimension data is created. Warehouse Builder binds the attributes in the dimension to the database columns used to store their values.

  For a MOLAP implementation, the analytic workspace that stores the dimension data is created.
- (Relational and ROLAP dimensions only) The database sequence used to generate the surrogate identifier for all the dimension levels.

Warehouse Builder creates the definitions of these objects in the workspace and not the objects themselves.

**Deploying Dimensions** To create the dimension in the target schema, you must deploy the dimension. For a ROLAP dimension, ensure that you deploy the sequence and the implementation tables before you deploy the dimension. Alternatively, you can deploy all these objects at the same time. For more information see "ROLAP Implementation of Dimensional Objects" on page 3-12.

> **Note:** When you delete a dimension, the associated objects such as sequence, database tables, or AWs are not deleted. You must explicitly delete these objects.

## Defaults Used By the Create Dimension Wizard

When you create a dimension using the Create Dimension wizard, default values are set for some of the attributes that are used to create the dimension. The following sections describe the defaults used.

### Storage

For a relational storage, the star schema is used as the default implementation method.

When you choose multidimensional storage, the dimension is stored in an analytic workspace that has the same name as the Oracle module in which the dimension is defined. If the analytic workspace does not exist, then it is created. The analytic workspace is stored in the users tablespace of the schema that owns the Oracle module.

### Dimension Attributes

Warehouse Builder creates default dimension attributes with the properties specified in Table 3–5.

***Table 3–5 Default Dimension Attributes***

| Dimension Attribute Name | Identifier | Data Type |
|---|---|---|
| ID | Surrogate | NUMBER |
| Name | Business | VARCHAR2 |
| Description | | VARCHAR2 |

You can add additional attributes. For your dimension to be valid, you must define the surrogate and business identifiers.

### Hierarchies

Warehouse Builder creates a default hierarchy called STANDARD that contains all the levels listed on the Levels page of the Create Dimension wizard. The hierarchy uses the levels in the same order that they are listed on the Levels page.

### Level Attributes

The ID, Name, and Description attributes are applicable to each level defined in the dimension. All the dimension attributes are applicable to the lowest level in the dimension. The lowest level is the level that is defined last on the Levels page.

### Slowly Changing Dimensions

When you create a Type 2 SCD, all the attributes of the lowest level, except the surrogate identifier and the business identifier, are versioned. Two additional attributes are created to store the effective date and the expiration date of each record. For example, if you create the `Products` dimension described in "Dimension Example" as a Type 2 SCD, then the attributes `UPC`, `Package_type`, and `Package_size` are versioned. Warehouse Builder creates two additional attributes called `EXPIRATION_DATE` and `EFFECTIVE_DATE`, of data type `DATE`, to store the effective date and expiration date of versioned records.

For a Type 3 SCD, all level attributes of the lowest level, except the surrogate identifier and the primary identifier, are versioned. Warehouse Builder creates additional attributes to store the previous value of each versioned attribute. Additionally, an attribute to store the effective date is created. For example, if you create the Products dimension described in "Dimension Example" as a Type 3 SCD, then additional attributes called `PREV_DESCRIPTION`, `PREV_PACKAGE_TYPE`, `PREV_PACKAGE_SIZE`, and `PREV_UPC` are created to store the previous values of the versioned attributes. These data type for these attributes are the same the ones used to store the current value of the attribute. Warehouse Builder also creates an attribute EFFECTIVE_TIME to store the effective time of versioned records. This attribute uses the `DATE` data type.

### Orphan Management Policy

For relational and ROLAP dimensions, the default orphan management policy for loading data into and removing data from dimensions is No Maintenance.

The Deploy Error Tables option is deselected.

### Implementation Objects

For each dimension, in addition to the dimension object, certain implementation objects are created. The number and type of implementation objects depends on the storage type of the dimension.

For time dimensions, irrespective of the storage type, a map that loads the time dimension is created. The name of the map is the dimension name followed by '_MAP'. For example, the map that loads a time dimension called TIMES will be called TIMES_MAP.

#### ROLAP: Relational Storage

For a relational storage, the following implementation objects are created:

**Table:** A table with the same name as the dimension is created to store the dimension data. A unique key is created on the dimension key column. For example, when you define a dimension called `CHANNELS`, a table called `CHANNELS_TAB` is created to store the dimension data. Also, a unique key called `CHANNELS_DIMENSION_KEY_PK` is created on the dimension key column.

**Sequence:** For a dimension that uses a relational storage, a sequence that loads the dimension key values is created. For example, for the dimension called `CHANNELS`, a sequence called `CHANNELS_SEQ` is created.

#### ROLAP: with MVs

For a ROLAP with MVs implementation, the implementation table and the sequence that loads the surrogate identifier, as described in "ROLAP: Relational Storage" on page 3-22, are created. Additionally, an analytic workspace with the same name as the Oracle module containing the dimension is created.

#### MOLAP: Multidimensional Storage

For a multidimensional storage, if it does not already exist, an analytic workspace with the same name as the Oracle module that contains the dimension is created. For example, if you create a dimension called `PRODUCTS` in the `SALES_WH` module, then the dimension is stored in an analytic workspace called `SALES_WH`. If an analytic workspace with this name does not already exist, it is first created and then the dimension is stored in this analytic workspace.

## Creating Dimensions Using the Dimension Editor

The Dimension Editor enables advanced users to create dimensions according to their requirements. You can also edit a dimension using the Dimension Editor.

Use the Dimension Editor to create a dimension if you want to perform one of the following tasks:

- Use the snowflake implementation methods.
- Create value-based hierarchies.
- Create dimension roles.
- Skip levels in a hierarchy.

- Use existing database tables or views to store the dimension data. This is referred to as manual binding.
- Specify an orphan management policy.
- Create more than one hierarchies in a dimension.

**To define a dimension using the Dimension Editor:**

1. From the Projects Navigator expand the Databases node and then the Oracle node.
2. Expand the target module where you want to create the dimension.
3. Right-click **Dimensions** and select **New**.

   The New Gallery dialog box is displayed.
4. Select **Dimension without using Wizard** and click **OK**.

   Warehouse Builder displays the Create Dimension dialog box.
5. Specify a name and an optional description for the dimension and click **OK**.

   The Dimension Editor is displayed with the Name tab containing the name and description you provided.

   To define the dimension, provide information about the following tabs:

   - Name Tab on page 3-23
   - Storage Tab on page 3-24
   - Attributes Tab on page 3-25
   - Levels Tab on page 3-25
   - Hierarchies Tab on page 3-26
   - SCD Tab on page 3-27
   - Orphan Tab on page 3-28
   - Physical Bindings Tab on page 3-29

   **Note:** When you use the Dimension Editor to create a dimension that has a relational implementation, the physical structures that store the dimension data are not automatically created. You must create these structures either manually or using the Bind option in the File menu.

6. For dimensions that have a relational or ROLAP with MVs implementation, bind the attributes in the dimension to the database columns that store their data, see "Physical Bindings Tab" on page 3-29.

### Name Tab

Use the Name tab to describe your dimension. You also specify the type of dimension and the dimension roles on this tab.

The **Name** field represents the name of the dimension. The dimension name must be unique within the module. Use the **Description** field to enter an optional description for the dimension.

**Dimension Roles** Use the Dimension Roles section to define dimension roles. You define the following for each dimension role:

- **Name:** Represents the name of the dimension role.
- **Description:** Specify an optional description for the dimension role.

> **See Also:** *Oracle Warehouse Builder Concepts* for more information about dimension roles

### Storage Tab

Use the Storage tab to specify the type of storage for the dimension. The storage options you can select are described in the following sections.

**ROLAP: Relational Storage** Select the Relational option to store the dimension and its data in a relational form in the database. Use this option to create a dimension that uses a relational or ROLAP implementation.

For a relational storage, you can select one of the following methods to implement the dimension:

- **Star schema:** Implements the dimension using a star schema. This means that the dimension data is stored in a single database table or view.
- **Snowflake schema:** Implements the dimension using a snowflake schema. This dimension data is stored in more than one database table or view.
- **Manual:** You must explicitly bind the attributes from the dimension to the database object that stores their data. When you select this option, you are assigned write access to the Physical Binding tab in the Dimension Editor and the auto binding feature is disabled so that you do not accidentally remove the bindings that you manually created.

When you perform auto binding, these storage settings are used to perform auto binding.

Click **Create composite unique key** to create a composite unique key on the business identifiers of all levels. For example, if your dimension contains three levels, when you create a composite unique key, a unique key that includes the business identifiers of all three levels is created. Creating a composite unique key enforces uniqueness of a dimension record across the dimension at the database level.

If the database containing the target schema has the OLAP option installed, then you can also deploy the dimensions to the OLAP catalog by setting the configuration parameter as described in "Specifying How Dimensions are Deployed" on page 3-38.

**ROLAP: with Cube MVs** Warehouse Builder stores the dimension definition and its data in a relational form in the database. Additionally, materialized view summaries are created for the implementation tables in the analytic workspace. Select this option to create a dimension that uses a ROLAP implementation and stores summaries in the analytic workspace.

When you choose a ROLAP with MVs implementation, specify the name of the analytic workspace that should store the summary data using the AW Name field in the MOLAP: Multidimensional storage section.

**MOLAP: Multidimensional storage** Select the MOLAP option to store the dimension and its data in a multidimensional form in the database. Use this option to create a dimension that uses a MOLAP implementation. The dimension data is stored in an analytic workspace.

Enter values for the following fields:

- **AW Name:** Enter the name of the analytic workspace that stores the dimension data. Alternatively, you can click the Select button to display a list of MOLAP objects in the current project. Warehouse Builder displays a node for each module in the project. Expand a module to view the list of dimensional objects in the module. Selecting an object from list stores the dimension in the same analytic workspace as the selected object.
- **AW Tablespace Name:** Enter the name of the tablespace in which the analytic workspace is stored.

Dimensions with multiple hierarchies can sometimes use the same source column for aggregate levels (that is, any level above the base). In such cases, you select the **Generate surrogate keys in the analytic workspace** option. During a load operation, the level name is added as a prefix to each value. It is recommended that you select this option unless you know that every dimension member is unique.

If you are sure that dimension members are unique across levels, then you can use the exact same names in the analytic workspace as the source. For example, if your relational schema uses numeric surrogate keys to assure uniqueness, then you need not create new surrogate keys in the analytic workspace. The **Use natural keys from data source** option enables you to use the same natural keys from the source in the analytic workspace.

---

**Note:** If you edit a dimension and change the Storage type from ROLAP to MOLAP, then the data type of the surrogate identifier is changed to `VARCHAR2`.

---

### Attributes Tab

Use the Attributes tab to define the dimension attributes. The Attributes tab contains two sections: Sequence and Dimension Attributes.

**Sequence** The Sequence attribute is required only for dimensions that have a relational implementation and that have a surrogate identifier defined. Use the Sequence field to specify the name of the database sequence that populates the dimension key column. Click **Select** to the right of this field to display the Available Sequences dialog box. This dialog box contains a node for each module in the project. Expand a module node to view the sequences contained in the module. Select a sequence from the displayed list.

**Dimension Attributes** Use the Dimension Attributes section to define the details of the dimension attributes as described in "Dimension Attributes Page" on page 3-17.

### Levels Tab

Use the Levels tab to define the dimension levels and the attributes for each level in the dimension. You also use this tab to create value-based hierarchies.

Before you define level attributes, ensure that the dimension attributes are defined on the Dimension Attributes tab. To define the level attributes for a level, you must select the dimension attributes that the level implements. The Levels tab contains two sections: Levels and Level Attributes.

**Levels** The Levels section displays the levels in the dimension. Provide the following details for each level:

- **Name:** Enter the name of the dimension level. The name must be unique within the dimension.

- **Description:** Enter an optional description for the level.

**Level Attributes** The Level Attributes section lists all the dimension attributes defined on the Attributes tab. The values that you specify in this section are applicable to the level selected in the Levels section. The Level Attributes section contains the following:

- **Dimension Attribute Name:** Represents the name of the dimension attribute.
- **Applicable:** Select the Applicable option if the level selected in the Levels section implements this dimension attribute.
- **Level Attribute Name:** Represents the name of the level attribute. Use this field to specify a name for the level attribute, a name that is different from that of the dimension attribute. This is an optional field. If you do not specify a name, then the level attribute will have the same name as the dimension attribute.
- **Description:** Specify an optional description for the level attribute.
- **Default Value:** Specify the default value of the level attribute.

For example, to specify that the Groups level implements the dimension attributes ID, Name, and Description:

- Select the Groups level in the **Levels** section.
- In the Level Attributes section, select the **Applicable** option for the ID, Name, and Description attributes.

### Hierarchies Tab

Use the Hierarchies tab to create dimension hierarchies. The Hierarchies tab contains two sections: Hierarchies and Levels.

**Hierarchies** Use the Hierarchies section to define the hierarchies in the dimension. For each hierarchy, define the following:

- **Hierarchy:** Represents the name of the hierarchy. To create a new hierarchy, enter the name of the hierarchy in this field.
- **Value-based:** Select this option to create a value-based hierarchy. A value-based hierarchy contains no levels. It must have an attribute identified as the parent identifier. Since you can create value-based hierarchies only for MOLAP dimensions, this option is displayed only if you select **MOLAP: Multidimensional storage** on the Storage tab.

  **See Also:** *Oracle Warehouse Builder Concepts* for information about value-based hierarchies

- **Description:** Enter an optional description for the hierarchy.
- **Default:** Select the Default option if the hierarchy is the default hierarchy for the dimension. When a dimension has more than one hierarchy, query tools show the default hierarchy. It is recommended that you set the most commonly used hierarchy as the default hierarchy.

To delete a hierarchy, right-click the cell to the left of the **Hierarchy** field and select **Delete**. Alternatively, you can select the hierarchy by clicking the cell to the left of the Hierarchy field and press the Delete button.

When you create a hierarchy, ensure that you create the attributes that store the parent level references for each level. For a relational or ROLAP dimension, create two

attributes to store the surrogate identifier reference and business identifier reference of each level. For a MOLAP dimension, create one attribute to store the reference to the business identifier of the parent level of each level.

**Levels** The Levels section lists all the levels defined on the Levels tab of the Dimension Editor. Use this section to specify the levels used in each hierarchy. The Levels section contains the following:

- **Level:** Represents the name of the level. Click the list to display all the levels defined in the dimension.
- **Skip to Level:** Represents the parent level of the level indicated by the Level field. Use this field to define skip-level hierarchies.

  For example, the Products dimension contains the following hierarchy:

  Total > Product

  This hierarchy does not include the Groups level. Thus the Product level must skip the Groups level and use the Total level as a parent. To create this hierarchy, select the Product level in the **Level** field and select Total from the **Skip to Level** list.
- **Summary Level:** Represents the dimension level used to load summaries in the analytic workspace. This option is displayed only if you select **ROLAP: with Cube MVs** on the Storage tab.

Use the arrows to the left of the Levels section to change the order in which the levels appear in the section.

### SCD Tab

Use this tab to specify the type of slowly changing policy that the dimension implements. Since you can create a Slowly Changing Dimension only for dimensions that use a relational implementation, the options on this tab are enabled only if you select **ROLAP: Relational Storage** or **ROLAP: with Cube MVs** on the Storage tab.

> **Note:** If you choose a MOLAP implementation on the Storage Tab, then the options on this tab are disabled.

The options that you can select for slowly changing policy are:

- **Type 1: Do not keep history:** Creates a normal dimension that stores no history.
- **Type 2: Store the complete change history:** Select this option to create a Type 2 SCD. Click **Settings** to specify the additional details such as triggering attribute, effective date and expiration date for each level.as described in "Creating Type 2 Slowly Changing Dimensions Using the Dimension Editor" on page 3-32.
- **Type 3: Store only the previous value:** Select this option to create a Type 3 SCD. Click **Settings** to specify the additional details such as effective date and the attributes used to store the previous value of versioned attributes as described in "Creating Type 3 Slowly Changing Dimensions Using the Dimension Editor" on page 3-35.

> **Note:** You cannot create a Type 2 or Type 3 Slowly Changing Dimension if you have specified the type of storage as MOLAP.

When you create a Type 2 or Type 3 SCD using the Dimension Editor, you must create the dimension attributes that store the effective data and expiration date and apply them to the required levels.

### Orphan Tab

The Orphan tab defines the orphan management policy used while loading data into the dimension or removing data from the dimension. This tab contains two sections: Orphan Management for Removal and Orphan Management for Loading.

**Orphan Management for Loading** Use this section to specify the orphan management policy for loading data into a dimension. You can specify different orphan management policies for records that have null parent records and records that have invalid parent records. Use the options under the heading Null parent key values to specify the orphan management policy for records that have a null parent. Use the options under the heading Invalid parent key values to specify the orphan management policy for records that have an invalid parent record.

For records with a null parent and records with an invalid parent, select one of the following orphan management policies:

- **No Maintenance:** Warehouse Builder does not actively detect, reject, or fix orphan rows.
- **Default Parent:** Warehouse Builder assigns a default parent row for any row that does not have an existing parent row at the time of loading data. You use the Settings button to define the default parent row. For more information about assigning a default parent row, refer "Specifying the Default Parent for Orphan Rows" on page 3-28.
- **Reject Orphan:** Warehouse Builder does not insert the row if it does not have an existing parent row.

**Orphan Management for Removal** You use this section to specify the orphan management policy for removing data from a dimension. Select one of the following options:

- **No maintenance:** Warehouse Builder does not actively detect, reject, or fix orphan rows.
- **Reject Removal:** Warehouse Builder does not remove a row if the row has existing child rows.

#### Deployment Options

Select **Deploy Error Table(s)** to generate and deploy the error tables related to orphan management along with the dimension.

### Specifying the Default Parent for Orphan Rows

Use the Default Parent dialog to specify the default parent record of an orphan row. You can specify a default parent record for all the dimension levels.

The Default Parent dialog contains a row for each dimension level. displays a table that contains the following four columns:

- **Levels:** The Levels column displays a node for each level in the dimension. Expand a level node to display all the attributes in the level.
- **Identifying Attribute:** Represents the name of the level attribute.
- **Data Type:** Displays the data type of the attribute.

- **Default Value:** Specify a default value for the level attribute.

### Physical Bindings Tab

Use the Physical Bindings tab to bind the dimension to its implementation objects. Binding is the process of specifying the database columns that will store the data of each attribute and level relationship in the dimension. When you use the Create Dimension wizard to create a dimension, binding is automatically performed. When you use the editor to create a dimension, you must specify the details of the database tables or views that store the dimension data.

Choose one of the following options to bind dimension attributes to the database columns that store their data:

- Auto binding
- Manual binding

**Auto Binding** When you perform auto binding, Warehouse Builder maps the attributes in the dimension to the database columns that store their data. When you perform auto binding for the first time, Warehouse Builder also creates the tables that are used to store the dimension data.

To perform auto binding, select the dimension in the Projects Navigator or on the Physical Bindings tab. From the file menu, select **Bind**. Alternatively, right-click the dimension in the Projects Navigator and select **Bind**. For more information about the auto binding rules, see "Auto Binding" on page 3-10.

When you perform auto binding on a dimension that is already bound, Warehouse Builder uses the following rules:

- If the implementation method of the dimension remains the same, then Warehouse Builder rebinds the dimensional object to the existing implementation objects. The implementation method can be either Star or Snowflake.

  For example, you create a Products dimension using the star schema implementation method and perform auto binding. The dimension data is stored in a table called Products. You modify the dimension definition at a later date but retain the implementation method as star. When you now auto bind the Products dimension, Warehouse Builder rebinds the Products dimension attributes to the same implementation tables.

- If the implementation method of a dimension is changed, then Warehouse Builder deletes the old implementation objects and creates a new set of implementation tables. If you want to retain the old implementation objects, then you must first unbind the dimensional object and then perform auto binding.

  For example, you create a Products dimension using the star schema implementation method and bind it to the implementation table. You now edit this dimension and change its implementation method to snowflake. When you now perform auto binding for the modified Products dimension, Warehouse Builder deletes the table that stores the dimension data, creates new implementation tables, and binds the dimension attributes and relationships to the new implementation tables.

**Manual Binding** In manual binding, you must explicitly bind the attributes in each level of the dimension to the database columns that store their data. You can either bind to existing tables or create new tables and bind to them. You would typically use manual binding to bind existing tables to a dimension. Use manual binding if no auto binding or rebinding is required.

**To perform manual binding:**

1. In the Projects Navigator, right-click the dimension and select **Open**.

   Warehouse Builder displays the editor for this dimension.

2. On the Physical Bindings tab, right-click a blank area, select **Add** and then select the type of database object that stores the dimension data.

   For example, if the dimension data is stored in a table, then right-click a blank area on the Physical Bindings tab, select **Add** and then **Table**. Warehouse Builder displays the Add a new or existing Table dialog box. To store the dimension data, you either select an existing table or create a new table.

3. Repeat Step 2 as many times as the number of database objects that are used to store the dimension data. For example, if the dimension data is stored in three database tables, then perform Step 2 thrice.

4. Bind each attribute in the dimension to the database column that stores its data.

After you define a dimension and perform binding (for ROLAP dimensions only), you must deploy the dimension and its associated objects. For more information about deploying dimensions, see "Deploying Dimensions" on page 3-20.

## Limitations of Deploying Dimensions to the OLAP Catalog

For dimensions with a ROLAP implementation, there are implications and limitations related to the various dimension structures when either reporting on the underlying tables or deploying to the OLAP catalog. Although the dimension may be successfully deployed, errors could occur when other applications, such as Oracle Discoverer access the OLAP catalog.

The following are items that are affected by this limitation:

- No reporting tool has metadata about all aspects of dimensional metadata we capture, so this must be incorporated into the query/reports. Otherwise you will see odd information because of the way the data is populated in the implementation tables.

  The dimension and cube implementation tables store solved rows which contain negative key values. You can filter out these rows in your queries or reports. When you create a query or report, use the view that is associated with a dimension instead of the dimension itself. Each dimension has a view that is associated with it. The view name is specified in the configuration parameter View Name of the dimension or cube.

- Skip-level hierarchies and ragged hierarchy metadata is not deployed to the OLAP catalog.

  If you create a dimension that contains skip-level or ragged hierarchies, then the metadata for these is stored in the Warehouse Builder repository but is not deployed to the OLAP catalog.

- Dimensions with multiple hierarchies must have all dimension attributes mapped along all the hierarchies.

## Using Control Rows

Control rows enable you to link fact data to a dimension at any level. For example, you may want to reuse a Time dimension in two different cubes to record the budget data at the month level and the actual data at the day level. Because of the way dimensions

are loaded with control rows, you can perform this without any additional definitions. Each member in a dimension hierarchy is represented using a single record.

Warehouse Builder creates control rows when you load data into the dimension. All control rows have negative dimension key values starting from -2. For each level value of higher levels, a row is generated that can act as a unique linking row to the fact table. All the lower levels in this linking or control rows are nulled out.

Consider the Products dimension described in "Dimension Example" on page 3-14. You load data into this dimension from a table that contains four categories of products. Warehouse Builder inserts control rows in the dimension as shown in Table 3–6. These rows enable you to link to a cube at any dimension level. Note that the table does not contain all the dimension attribute values.

***Table 3–6 Control Rows Created for the Products Dimension***

| Dimension Key | Total Name | Categories Name | Product Name |
|---|---|---|---|
| -3 | TOTAL | | |
| -9 | TOTAL | Hardware | |
| -10 | TOTAL | Software | |
| -11 | TOTAL | Electronics | |
| -12 | TOTAL | Peripherals | |

### Determining the Number of Rows in a Dimension

To obtain the real number of rows in a dimension, count the number of rows by including a `WHERE` clause that excludes the NULL rows. For example, to obtain a count on Products, count the number of rows including a `WHERE` clause to exclude NULL rows in Product.

## Creating Slowly Changing Dimensions

You can create an SCD either using the Create Dimension Wizard or the Dimension Editor.

To create an SCD using the Create Dimension Wizard, use the Slowly Changing Dimension page of the Create Dimension Wizard. You only specify the type of SCD that you want to create on this page. Warehouse Builder assumes default values for all other required parameters. For more information about the Slowly Changing Dimension page, see "Slowly Changing Dimension Page" on page 3-19.

> **Note:** Type 1 does not require additional licensing; however, Type 2 and Type 3 SCDs require the Warehouse Builder Enterprise ETL Option.

To create a Type 2 SCD or a Type 3 SCD, in addition to the regular dimension attributes, you need additional attributes that perform the following roles.

### Triggering Attribute

These are attributes for which historical values must be stored. For example, in the `PRODUCTS` dimension, the attribute `PACKAGE_TYPE` of the Product level can be a triggering attribute. This means that when the value of this attribute changes, the old value needs to be stored.

#### Effective Date

This attribute stores the start date of the record's life span.

#### Expiration Date

This attribute stores the end date of the record's life span.

An attribute can play only one of the above roles. For example, an attribute cannot be a regular attribute and an effective date attribute. When you use the wizard to create a Type 2 SCD or a Type 3 SCD, Warehouse Builder creates the required additional attributes.

## Creating Type 2 Slowly Changing Dimensions Using the Dimension Editor

A Type 2 SCD stores the full history of values for each attribute and level relationship.

To create a Type 2 SCD using the Dimension Editor, define the following:

- The attributes that trigger history saving.
- The attributes that store the effective date and the expiration date.

> **Note:** You can create a Type 2 SCD only for dimensions that have a relational implementation.

**To create a Type 2 SCD using the Dimension Editor:**

1. From the Projects Navigator, expand the Databases node and then the Oracle node.
2. Expand the target module where you want to create the Type 2 SCD.
3. Right-click **Dimensions,** select **New**, then **Dimension without using Wizard**.
4. Provide information about the Name tab of the Dimension Editor as described in the "Name Tab" on page 3-23.
5. On the Attributes tab, for each level, create two additional attributes to store the effective date and the expiration date. For more information about creating attributes, see "Attributes Tab" on page 3-25.
6. Provide information about the following tabs of the Dimension Editor:
   - Levels Tab on page 3-25
   - Hierarchies Tab on page 3-26
7. On the Slowly Changing tab, select the **Type 2: Store the complete change history** option.
8. Click **Settings** to the right of this option.

   Warehouse Builder displays the Type 2 Slowly Changing Policy dialog box. Specify the details of the Type 2 SCD as described in "Type 2 Slowly Changing Dimension Dialog Box" on page 3-32.
9. Provide information about the Storage Tab of the Dimension Editor.

### Type 2 Slowly Changing Dimension Dialog Box

Use the Type 2 Slowly Changing Dimension dialog box to specify the effective date attribute, expiration date attribute, and the versioned attribute. This dialog box

displays a table that contains the following columns: Levels, Identifying Attribute, Data Type, and Record History.

- **Levels:** Represents the levels in the dimension. Expand a level node to view its level attributes.
- **Identifying Attribute:** Represents the level attribute.
- **Data Type:** Represents the data type of the level attribute.
- **Record History:** Use this list to indicate that an attribute is versioned or that it stores the effective date or expiration date of the level record.
  - **Trigger History:** Select this option for an attribute if the attribute should be versioned.
  - **Effective Date:** Select this option for an attribute if it stores the value of the effective date of the level record.
  - **Expiration Date:** Select this option for an attribute id it stores the expiration date of the level record.

  The surrogate ID and the business ID of a level cannot be versioned.

For example, in the `PRODUCTS` Type 2 SCD, the attributes that store the effective date and expiration date are `EFFECTIVE_TIME` and `EXPIRATION_TIME` respectively. You must create these dimension attributes and apply them to the `Product` level. The attribute `PACKAGE_TYPE` should be versioned. Thus, for this attribute, you select **Trigger history** under the Record History column. When the value of the `PACKAGE_TYPE` attribute changes, the existing record is closed and a new record is created using the latest values.

## Updating Type 2 Slowly Changing Dimensions

All the levels in a dimension need not store historical data. Typically, only the lowest level, also called the leaf level, stores historical data. However, you can also store historical data for other dimension levels.

When a record in a Type 2 SCD is versioned, the old record is marked as closed and a new record is created with the updated values. The expiration date of the record is set to indicate that it is closed. The new record is referred to as the current record and, by default, has a default expiration of NULL. While loading data into the Type 2 SCD, you can set the expiration date by using the configuration parameters for the Dimension operator. For more information, see "Dimension Operator" on page 25-14.

You can update the following in a Type 2 SCD:

- Leaf level attribute
- Leaf level versioned attribute
- Non-leaf level attribute
- Non-leaf level versioned attribute
- Leaf level parent attribute

The following sections describe the Warehouse Builder functionality for these update operations.

### Updating a Leaf Level Attribute

When you update a leaf level attribute, the value of this attribute is updated in the corresponding record.

For example, if you update the value of C_HOME_PHONE in a Customer level record, then the record is updated with the changed phone number.

#### Updating a Leaf Level Versioned Attribute

When you update a leaf level versioned attribute, the current record is marked as closed. A new record is created with the updated value of the versioned attribute.

For example, if you update the marital status of a customer, then the current record is marked as closed. A new record with the updated marital status is created for that customer.

#### Updating a non-leaf Level Attribute

When you update an attribute in a non-leaf level, the open records of the non-leaf level and the child records corresponding to this non-leaf level are updated with the new value.

For example, when you update the H_ADDRESS attribute in a Household level record, the current open record for that household is updated. All open child records corresponding to that particular household are also updated.

#### Updating a non-leaf Level Versioned Attribute

The update functionality depends on whether hierarchy versioning is enabled or disabled.

**Hierarchy Versioning Disabled**

The non-leaf level record corresponding to the versioned attribute is closed and a new record is created with the updated value. The child records of this non-leaf level record are updated with the changed value of the non-leaf level versioned attribute.

For example, when the value of H_ZIP in a Household level record is updated, the current open record for that household is closed. A new record with the updated value of H_ZIP is created. The value of H_ZIP is updated in all the child records corresponding to the updated household record.

**Hierarchy Versioning Enabled**

The non-leaf level record corresponding to the versioned attribute is closed and a new record is created with the updated value. Child records corresponding to this non-leaf level record are also closed and new child records are created with the updated value.

For example, when the value of H_ZIP in a Household level record is updated, the current open record for that household and its corresponding child records are closed. New records are created, with the updated value, for the household and for the child records corresponding to this household.

#### Updating the Leaf Level Parent Attribute

In addition to updating the level attributes in a Type 2 SCD, you can also update the parent attribute of a child record. In the Customers Type 2 SCD, the attribute H_BUSN_ID in a Customer record stores the parent attribute of that customer. The update functionality for the leaf level parent attribute depends on whether hierarchy versioning is enabled or disabled.

**Hierarchy Versioning Disabled**

The child record is updated with the new parent attribute value.

For example, when you update the value of the `H_BUSN_ID` attribute representing the parent record of a `Customer` record, the `Customer` record is updated with the new values.

**Hierarchy Versioning Enabled**

The child record is closed and a new record with the changed parent attribute value is created.

For example, when you update the `H_BUSN_ID` attribute of a customer record, the current customer record is closed. A new customer record with the updated `H_BUSN_ID` is created.

## Creating Type 3 Slowly Changing Dimensions Using the Dimension Editor

A Type 3 SCD stores two versions of values for certain selected attributes. You can create a Type 3 SCD only for dimensions that have a relational implementation. Specify the following:

- The attributes that should be versioned.
- The attributes that will store the previous value of each versioned attribute.

  For each versioned attribute, you must create an additional attribute to store the previous value of the attribute. For example, if you want to version the Population attribute, then you create an additional attribute to store the previous value of population.

**To create a Type 3 SCD:**

1. From the Projects Navigator, expand the Database node and then the Oracle node.
2. Expand the target module where you want to create the Type 3 SCD.
3. Right-click **Dimensions**, select **New**.

   The New Gallery dialog box is displayed.
4. Select **Dimension without Using Wizard** and click **OK**.
5. Provide information about the Name tab of the Dimension Editor as described in "Name Tab" on page 3-23.
6. On the Attributes tab, for each level, create an additional attribute to store the expiration date of the attributes in the level as described in "Attributes Tab" on page 3-25.

   Consider an example where you want to store previous values for the package_type and package_size attributes of the Products dimension. In this case, create two new attributes prev_package_type and prev_package_size to store the previous values of these attributes.
7. Provide information about the following tabs of the Dimension Editor:
   - Levels Tab on page 3-25
   - Hierarchies Tab on page 3-26
8. On the Slowly Changing tab, select the **Type 3: Store only the previous value** option. Click **Settings** to the right of this option.

   Warehouse Builder displays the Type 3 Slowly Changing Policy dialog box. Specify the details of the Type 2 SCD using this dialog box as described in "Type 3 Slowly Changing Dimension Dialog Box" on page 3-36.
9. Provide information about the Storage Tab of the Dimension Editor.

### Type 3 Slowly Changing Dimension Dialog Box

Use the Type 3 Slowly Changing Dimension dialog box to specify the implementation details. Use this dialog box to select the attribute that stores effective date, the attributes that should be versioned, and the attributes that store the previous value of the versioned attributes.

This dialog box displays a table that contains four columns: Levels, Identifying Attribute, Previous Attribute, and Record History.

- **Levels:** Displays the levels in the dimension. Expand a level node to view the level attributes.
- **Identifying Attribute:** Represents the level attribute.
- **Previous Attribute:** Represents the attribute that stores the previous value of the versioned attribute. Use the list to select the previous value attribute. Specify a previous value attribute only for versioned attributes. You must explicitly create the attributes that store the previous values of versioned attributes. Again, create these as dimension attributes and apply them to the required level.
- **Effective:** Indicates if an attribute stores the effective date. If the attribute stores the effective date, then select **Effective date** from the **Effective** list.

  The surrogate ID of a level cannot be versioned.

Consider the `PRODUCTS` Type 3 SCD. The `EFFECTIVE_TIME` attribute stores the effective date of the Product level records. The `PACKAGE_TYPE` attribute of the Product level should be versioned. The attribute that stores the previous value of this attribute, represented by the **Previous Attribute** column, is `PREVIOUS_PACKAGE_TYPE`. When the value of the `PACKAGE_TYPE` attribute changes, Warehouse Builder does the following:

- Moves the existing value of the `PACKAGE_TYPE` attribute the `PREVIOUS_PACKAGE_TYPE` attribute.
- Stores the new value of population in the `PACKAGE_TYPE` attribute.

## Editing Dimension Definitions

Use the Dimension Editor to edit the definition of a dimension. When you edit a dimension definition, the changes are made only in the object metadata. To update the physical object definition, deploy the modified dimension using the Control Center.

---

**Note:** Once you create a Slowly Changing Dimension, you cannot modify its type using the Dimension Editor.

---

**To edit a dimension or Slowly Changing Dimension definition:**

Right-click the dimension in the Projects Navigator and select **Open**.

*or*

Double-click the dimension in the Projects Navigator.

The Dimension Editor is displayed. Modify the definition using the tabs in the Dimension Editor.

> **Note:** When you modify the implementation (star or snowflake) of a relational or ROLAP dimension, ensure that you first unbind the dimension and then perform binding. This creates the physical bindings according to the modified implementation.

For more information about these tabs, see the following sections:

- Creating Dimensions Using the Dimension Editor on page 3-22
- Type 2 Slowly Changing Dimension Dialog Box on page 3-32
- Type 3 Slowly Changing Dimension Dialog Box on page 3-36

## Configuring Dimensions

When you configure a dimension, you configure both the dimension and the underlying table.

**To configure the physical properties for a dimension:**

1. From the Projects Navigator, right-click the dimension name and select **Configure**.

   The Configuration tab is displayed.

2. Configure the dimension parameters listed under the following categories.

   For a dimension that uses a relational or ROLAP implementation, you can also configure the implementation tables. For more information, see "Configuring Tables" on page 2-48.

### Identification

**Deployable:** Select TRUE to indicate that you want to deploy this dimension. Warehouse Builder generates scripts only for table constraints marked deployable.

**Deployment Options:** Use this parameter to specify the type of implementation for the dimension. Select one of the following options: Deploy All, Deploy Data Objects Only, Deploy to Catalog Only.

For more information about deployment options, see "Specifying How Dimensions are Deployed" on page 3-38.

**View Name:** Specify the name of the view that is created to hide the control rows in the implementation table that stores the dimension data. This is applicable for relational or ROLAP dimensions that use a star schema. The default view name, if you do not explicitly specify one, is the dimension name suffixed with "_v".

**Visible:** This parameter is not used in code generation.

### Summary Management

The parameters in this section need to be set only if the dimension uses a ROLAP with MVs implementation.

- **Enable MV Refresh:** Enables the materialized views that store the summary data to be refreshed. The default value of this parameter is False.
- **MV Constraints:** Set either TRUSTED or ENFORCED for this parameter.

  Trusted constraints result in a more efficient refresh operation. Setting this parameter to Trusted allows use of non-validated RELY constraints and rewrite

against materialized views during refresh. However, if the trusted constraint information is invalid, then the refresh may corrupt the materialized view.

Setting this parameter to Enforced allows the use of only validated, enforced constraints and rewrite against materialized views.

- **Refresh Mode:** Select either Fast, Complete, or Force for this parameter.

  For Complete refresh, the materialized view's defining query is recalculated.

  For Fast refresh, only changed rows are inserted in the cube and the affected areas of the cube are re-aggregated.

  Force refresh first applies Fast refresh. If this is not possible, then it applies Complete refresh.

- **Refresh Next Date:** Represents the next data on which the materialized views should be refreshed.

- **Refresh On:** Set to DEMAND or ONDATE.

  Setting this parameter to Demand causes the materialized views to be updated on demand. Setting to ONDATE refreshes the materialized views on the date specified in the Refresh Next Date parameter.

- **Refresh Start Date:** Represents the date on which to begin refreshing the materialized views.

## Specifying How Dimensions are Deployed

You can specify the form in which dimensions are deployed to the target schema by setting the Deployment Option configuration parameter. The values you can set for deployment option of dimensions are: Deploy All, Deploy Data Objects Only, and Deploy to Catalog. For steps on setting the Configuration Options parameter, see "Configuring Dimensions" on page 3-37.

In addition to the Deployment Option configuration parameter, the form in which dimensions are deployed also depends on the generation mode you specify. The PL/SQL Generation Mode parameter of the Oracle module containing the dimension represents the Oracle Database to which objects in the module are deployed. You can set the PL/SQL Generation Mode to one of the following options: Default, Oracle10g, Oracle10gR2, Oracle11gR1, Oracle11gR2, Oracle8i, and Oracle9i. For more information, see "Configuring Target Modules" on page 2-46.

Table 3–7 describes how dimensions with ROLAP implementations are deployed on different Oracle Database versions.

***Table 3–7 Deployment Options for ROLAP Dimensions***

| Deployment Option | Target Schema: Oracle Database 10*g*, ROLAP Implementation | Target Schema: Oracle Database 11*g*, ROLAP Implementation | Target Schema: Oracle Database 11*g*, ROLAP with MVs Implementation |
|---|---|---|---|
| Deploy Data Objects | relational dimension DDL | relational dimension DDL | relational dimension DDL |
| Deploy to Catalog | CWM2 | CWM2 | 11g form AW+ |
| Deploy All | relational dimension DDL and CWM2 | relational dimension DDL and CWM2 | relational dimension DDL and 11g form AW+ |

Table 3–8 describes how dimensions with a MOLAP implementation are deployed on different Oracle Database versions.

*Table 3–8 Deployment Options for Dimensions with a MOLAP Implementation*

| Deployment Option | Target Schema: Oracle 10*g* Database | Target Schema: Oracle 11*g* Database |
|---|---|---|
| Deploy Data Objects | 10g form AW | 11g form AW |
| Deploy to Catalog | 10g form AW | 11g form AW |
| Deploy All | n/a | n/a |

# Creating Cubes

Warehouse Builder provides the following two methods of creating a cube:

- Using the Create Cube Wizard to Create Cubes

  Use the Create Cube wizard to create a basic cube quickly. Warehouse Builder assumes default values for most of the parameters and creates the database structures that store the cube data.

- Using the Cube Editor to Create Cubes

  Use the Cube Editor to create a cube when you want to specify certain advanced options such as aggregation methods and solve dependency order. These options are not available when you use the Create Cube wizard.

Alternatively, you can use the Create Cube wizard to quickly create a basic cube object. Then use the Cube Editor to specify the other options.

## About Calculated Measures in Cubes

While defining measures in a cube, you can also create calculated measures. A calculated measure is a measure whose data is not stored. Its value is calculated when required using the expression defined for the measure.

Calculated measures can be classified into the following two types:

- Standard Calculation
- Custom Expression

### Standard Calculation

Standard calculations are based on the templates. Warehouse Builder enables you to define the following standard calculations: Basic Arithmetic, Advanced Arithmetic, Prior/Future Comparison, and Time Frame.

#### Basic Arithmetic

This type enables you to perform basic arithmetic calculations such as the following. Table 3–9 lists the basic arithmetic calculations.

*Table 3–9 List of Basic Calculated Measures*

| Calculation Name | Description |
|---|---|
| Addition | Use this calculation to add either two measures or a measure and a number. |
| Subtraction | Use this calculation to subtract two measures or a measure and a number. |
| Multiplication | Use this calculation to multiply two measures or a measure and a number. |

*Table 3–9 (Cont.) List of Basic Calculated Measures*

| Calculation Name | Description |
|---|---|
| Division | Use this calculation to divide two measures or a measure and a number. |
| Ratio | |

### Advanced Arithmetic

This type enables you to create the advanced calculations such as the ones defined in Table 3–10.

*Table 3–10 List of Advanced Arithmetic Calculated Measures*

| Calculation Name | Description |
|---|---|
| Cumulative Total | Use this calculation to return the cumulative total of measure data over time periods within each level of a specified dimension.<br>For example, Cumulative Sales for 2001= Sales Q1 + Sales Q2 + Sales Q3 + Sales Q4 |
| Index | Use this calculation to return the ratio of a measure's value as a percentage of a baseline value for the measure. The formula for the calculation is:<br>(Current member / Base Line member)<br>For example, Consumer Price Index (assuming baseline cost of goods is 1967) = (2001 Cost of Goods/1967 Cost of Goods)) * 100 |
| Percent Markup | Use this calculation to return the percentage markup between two measures where the basis for the calculation is the older measure. The formula used for this calculation is: (y-x)/x.<br>For example, the new price is 110 and the old price is 100. The percentage markup is calculated:<br>(110-100)/100 = +10%. |
| Percent Variance | Use this calculation to return the percent difference between a measure and a target for that measure. For example, the percentage variance between sales and quota is:<br>(Sales-Quota)/Quota. |
| Rank | Use this calculation to return the numeric rank value of each dimension member based on the value of the specified measure. For example, the rank of TV sales where DVD is 150, TV is 100, and Radio is 50 would be 2. |
| Share | Use this calculation to return the ratio of a measure's value to the same measure value for another dimension member or level. The formula for this calculation is:<br>(Current member / Specified member). |
| Variance | Use this calculation to calculate the variance between a base measure and a target for that measure. An example of variance is:<br>Sales Variance = Sales - Sales Forecast. |

### Prior/Future Comparison

Use this type to define prior and future value calculations such as the ones described in Table 3–11.

*Table 3–11 List of Prior/Future Comparison Calculated Measures*

| Calculated Measure Name | Description |
|---|---|
| Prior Value | Use this calculation to return the value of a measure from an earlier time period. |
| Difference from Prior Period | Use this calculation to return the difference between the current value of a measure and the value of that measure from a prior period. The formula for this calculation is:<br>(Current Value - Previous Value) |
| Percent Difference from Prior Period | Use this calculation to return the percentage difference between the current value of a measure and the value of that measure from a prior period. The formula for this calculation is:<br>((Current Value - Previous Value) / Previous Value) |
| Future Value | Use this calculation to return the value of an item for a future time period. For example, Sales a Year from Now = Sales from October 2006 if the current time is October 2005. |

### Time Frame

This type enables you to create the time series calculations listed in Table 3–12.

*Table 3–12 List of Time Series Calculated Measures*

| Calculated Measure Name | Description |
|---|---|
| Moving Average | Use this calculation to return the average value for a measure over a rolling number of time periods. And example of this calculation is:<br>Moving average sales for the last 3 months = (Jan Sales + Feb Sales + March Sales)/3 |
| Moving Maximum | Use this calculation to return the maximum value for a measure over a rolling number of time periods. An example of this calculation is:<br>Moving maximum sales for the last 3 months = the largest Sales value for Jan, Feb, and March. |
| Moving Minimum | Use this calculation to return the minimum value for a measure over a rolling number of time periods. An example of this calculation is:<br>Moving minimum sales for the last 3 months = the smallest Sales value for Jan, Feb, and March. |
| Moving Total | Use this calculation to return the total value for a measure over a rolling number of time periods. An example of this calculation is:<br>Moving total sales for the last 3 months = (Jan Sales + Feb Sales + March Sales). |
| Period to Date | Use this calculation to sum measure data over time periods, to create cumulative measure data. An example of this calculation is:<br>Year-to-date sales to March = Jan Sales + Feb Sales + March Sales. |

### Custom Expression

Select the **Custom Expression** option to specify an expression that is used to compute the calculated measure.

## Cube Example

The `Sales` cube stores aggregated sales data. It contains the following two measures: `Value_sales` and `Dollar_sales`.

- `Value_sales`: Stores the amount of the sale in terms of the quantity sold.
- `Dollar_sales`: Stores the amount of the sale.

Table 3–13 describes the dimensionality of the Sales cube. It lists the name of the dimension and the dimension level that the cube references.

***Table 3–13 Dimensionality of the Sales Cube***

| Dimension Name | Level Name |
|---|---|
| Products | Product |
| Customers | Customer |
| Times | Day |

## Using the Create Cube Wizard to Create Cubes

Use the following steps to create a cube using the wizard:

1. From the Projects Navigator expand the Databases node and then the Oracle node.
2. Expand the target module where you want to create the cube.
3. Right-click **Cubes,** select **New Cube**.

   Warehouse Builder displays the Welcome page of the Cube wizard. Click **Next** to proceed. The wizard guides you through the following pages:

   - Name and Description Page on page 3-42
   - Storage Type Page on page 3-42
   - Dimensions Page on page 3-44
   - Measures Page on page 3-44
   - Summary Page on page 3-45

### Name and Description Page

Use the Name and Description page to describe the cube. Enter the following details on this page:

- **Name:** The name of the cube. The cube name must be unique within the module.
- **Description:** Specify an optional description for the cube.

### Storage Type Page

Use the Storage Type page to specify the type of storage for the cube. The storage type determines how the cube data is physically stored in the database. The options you can select for storage type are:

- ROLAP: Relational storage
- ROLAP: with MVs
- MOLAP: Multidimensional storage

You select the storage type based on the volume of data stored at the lowest level of the entire cube and the refresh rate required.

### ROLAP: Relational storage

Warehouse Builder stores the cube definition and its data in a relational form in the database. Use this option to create a cube that has a relational or ROLAP implementation.

Relational storage is preferable if you want to store detailed, high volume data or you have high refresh rates combined with high volumes of data. Use relational storage if you want to perform one of the following:

- Store detailed information such as call detail records, point of sales (POS) records and other such transaction oriented data.
- Refresh high volumes of data at short intervals.
- Detailed reporting such as lists of order details.

Operational data stores and enterprise data warehouses are typically implemented using relational storage. You can then derive MOLAP implementations from this relational implementation to perform different types of analysis.

If the database containing the target schema has the OLAP option installed, then you can also deploy the dimensions to the OLAP catalog.

When you choose a relational implementation for a cube, the implementation table used to store the cube data is created.

### ROLAP: with MVs

Warehouse Builder stores the cube definition and its data in a relational form in the database. Additionally, cube-organized MVs are created in the analytic workspace. Select this option to create a cube that uses a ROLAP implementation and stores summaries in the analytic workspace.

Using this option provides summary management based on cube-organized MVs in Oracle 11*g* Database. Query performance is greatly improved, without the need to make any modification to your queries.

Cubes created using the ROLAP: with MVs implementation can only store summary data in the cube MV.

When you choose the ROLAP with MVs implementation:

- the implementation table used to store the cube data is created.
- the cube is stored in an analytic workspace that uses the same name as the Oracle module to which the dimension belongs. The tablespace that is used to store the analytic workspace is the tablespace that is defined as the users tablespace for the schema that contains the dimension metadata.

> **Note:** If a cube uses the ROLAP: with Cube MVs implementation, then all dimensions that this cube references must also use the ROLAP: with Cube MVs implementation.

### MOLAP: Multidimensional storage

Warehouse Builder stores the cube definition and the cube data in an analytic workspace in the database. Use this option to create a cube that has a MOLAP implementation.

Multidimensional storage is preferable when you want to store aggregated data for analysis. The refresh intervals for a multidimensional storage are usually longer than relational storage as data needs to be pre-calculated and pre-aggregated. Also, the data

volumes are typically smaller due to higher aggregation levels. Use multidimensional storage to perform the following:

- Advanced analysis such as trend analysis, what-if analysis, or to forecast and allocate data
- Drill and pivot data with instant results

When you choose a MOLAP implementation, the name used to store the cube in the analytic workspace is generated. If no analytic workspace exists, then one is created using the name you specify.

### Dimensions Page

The Dimensions page defines the dimensionality of the cube. A cube must refer to at least one dimension. You define dimensionality by selecting the dimensions that the cube references. You can use the same dimension to define multiple cubes. For example, the dimension TIMES can be used by the SALES cube and the COST cube.

The Dimensions page contains two sections: Available Dimensions and Selected Dimensions.

**Available Dimensions** The Available Dimensions section lists all the dimensions in the workspace. Each module in the project is represented by a separate node. Expand a module node to view all the dimensions in that module.

Warehouse Builder filters the dimensions displayed in the Available Dimensions section based on the implementation type chosen for the dimension. If you select ROLAP as the storage type, then only dimensions that have a relational implementation are listed. If you select MOLAP as the storage type, then only dimensions stored in an analytic workspace are listed.

**Selected Dimensions** The Selected Dimensions section lists the dimensions that you selected in the Available Dimensions section. Use the right arrow to the move a dimension from the Available Dimensions list to the Selected Dimensions list.

### Measures Page

Use the Measures page to define the measures of the cube. For each measure, specify the following details:

- **Name:** The name of the measure. The name of the measure must be unique within the cube.
- **Description:** An optional description for the measure.
- **Data Type:** Select the data type of the measure.

> **Note:** The following data types are not supported for MOLAP implementations: `BLOB`, `INTERVAL DAY TO SECOND`, `INTERVAL YEAR TO MONTH`, `RAW`, `TIMESTAMP WITH TIME ZONE`, `TIMESTAMP WITH LOCAL TIME ZONE`.

- **Length**: Specify length for character data types only.
- **Precision**: Define the total number of digits allowed for the measure. Precision is defined only for numeric data types.
- **Scale:** Define the total number of digits to the right of the decimal point. Scale is defined only for numeric data types.

- **Seconds Precision:** Represents the number of digits in the fractional part of the datetime field. It can be a number between 0 and 9. The seconds precision is used only for `TIMESTAMP`, `TIMESTAMP WITH TIME ZONE`, and `TIMESTAMP WITH LOCAL TIME ZONE` data types.

### Summary Page

Use the Summary page to review the options that you specified using the Cube wizard. Click **Finish** to complete defining the cube. This cube is displayed under the Cubes node of the Projects Navigator.

Warehouse Builder creates the metadata for the following in the workspace:

- The cube object.
- The definition of the table that stores the cube data.

  For a relational or ROLAP implementation, the definition of the database table that stores the cube data is created. Additionally, foreign keys are created in the table that stores the cube data to each data object that stores the data relating to the dimension the cube references.

  For a MOLAP implementation, the analytic workspace that stores the cube data is created. The wizard only creates the definitions for these objects in the workspace. It does not create the objects in the target schema.

**Deploying Cubes** To create the cube and its associated objects in the target schema, you must deploy the cube. Before you deploy a ROLAP cube, ensure that you successfully deploy the database table that stores the cube data. Alternatively, you can deploy both the table and the cube together. For more information, see "MOLAP Implementation of Dimensional Objects" on page 3-13.

---

**Note:** When you delete a cube, the associated objects such as the database table or analytic workspace are not deleted. You must explicitly delete these objects.

---

## Defaults Used by the Create Cube Wizard

When you create a cube using the Create Cube wizard, the following defaults are used:

- **MOLAP Storage:** The cube is stored in an analytic workspace that has the same name as the Oracle module in which the cube is created. The analytic workspace is stored in the users tablespace of the schema that owns the Oracle module.
- **Solve:** By default, the cube is solved on demand.
- **Aggregation Function:** The default aggregation function for all dimensions that the cube references is SUM.

## Using the Cube Editor to Create Cubes

The Cube Editor enables advanced users to create cubes according to their requirements. You can also use the Cube Editor to edit a cube.

Use the Cube Editor to create a cube if you must:

- Specify the dimensions along which the cube is sparse.
- Define aggregation methods for the cube measures.

- Precompute aggregations for a level.

**To create a cube using the Cube Editor:**

1. From the Projects Navigator expand the Databases node and then the Oracle node.
2. Expand the target module where you want to create the cube.
3. Right-click **Cubes**, select **New**.

   The New Gallery dialog Box is displayed.
4. Select **Cube without using Wizard** and click **OK**.

   Warehouse Builder displays the Cube Editor. To define a cube, provide information about the following tabs of the Cube Details panel:

   - Name Tab on page 3-46
   - Storage Tab on page 3-46
   - Dimensions Tab on page 3-47
   - Measures Tab on page 3-49
   - Aggregation Tab on page 3-50
   - Orphan Tab on page 3-51
   - Physical Bindings Tab on page 3-51

   When you use the Cube Editor to create a cube, the physical objects that store the cube data are not automatically created. You must create these objects.
5. To bind the cube measures and the dimension references to the database columns that store their data, see "Physical Bindings Tab" on page 3-51. You perform this step only for cubes that use a ROLAP implementation.

### Name Tab

Use the Name tab to describe the cube. Specify the following details on this tab:

- **Name**: Specify a name for the cube. The cube name must be unique within the module.
- **Description**: Specify an optional description for the cube.

### Storage Tab

The Storage tab specifies how the cube and its data should be stored. You can select either Relational or MOLAP as the storage type.

**ROLAP: Relational Storage** Select the **ROLAP: Relational storage** option to store the cube definition and its data in a relational form in the database. Use this option to create a cube that has a relational or ROLAP implementation. The cube data is stored in a database table or view.

Select the **Create bitmap indexes** option to generate bitmap indexes on all the foreign key columns in the fact table. This is required for a star query. For more information, see *Oracle Database Data Warehousing Guide*.

Select the **Create composite unique key** option to create a unique key on the dimension foreign key columns.

If the database containing the target schema has the OLAP option installed, then you can also deploy the dimensions to the OLAP catalog.

**ROLAP: with Cube MVs** Select the **ROLAP: with Cube MVs** option to store the dimension definition and its data in a relational form in the database and cube materialized view summaries in the analytic workspace.

When you choose a ROLAP with MVs implementation, specify the name of the analytic workspace that should store the summary data using the **AW Name** field in the MOLAP: Multidimensional storage section.

> **Note:** If a cube uses the ROLAP: with Cube MVs implementation, then all dimensions that this cube references must also use the ROLAP: with Cube MVs implementation.

**MOLAP: Multidimensional storage** Select the **MOLAP: Multidimensional storage** option to store the cube data in an analytic workspace. Use this option to create a cube with a MOLAP implementation. Use the Analytic Workspace section to specify the storage details. Enter the following details in this section:

- **AW Name:** This field specifies the name of the analytic workspace that stores the cube definition and cube data. Use the **Select** button to display the Analytic Workspaces dialog box. This dialog box lists the dimensional objects in the current project. Selecting an object from list stores the cube in the same analytic workspace as the selected object.
- **AW Tablespace Name:** Represents the name of the tablespace in which the analytic workspace is stored. If you do not specify a name, then the analytic workspace is stored in the default users tablespace of the owner of the Oracle module.

### Dimensions Tab

Use the Dimensions tab to define the dimensionality of the cube. This tab displays a table that you use to select the dimensions that the cube references and the **Advanced** button. You can change the order of the dimensions listed in this tab by using the arrows on the left of this tab. The Advanced button is enabled only for cubes that use MOLAP implementation or ROLAP with cube MVs implementation.

Use the **Advanced** button to define the sparsity of the dimensions referenced by the cube. Clicking this button displays the Advanced dialog box. Since you can define sparsity only for MOLAP cubes, the Advanced button is enabled only if the Storage type is MOLAP. For more information about the Sparsity dialog box, see "Advanced Dialog Box" on page 3-48.

The table on the Dimensions tab contains the following columns:

- **Dimension:** This field represents the name of the dimension that the cube references. Click the Ellipsis button in this field to display the Available Modules dialog box. This dialog box displays the list of dimensions in the current project. Select a dimension from this list.

  Warehouse Builder filters the dimensions displayed in this list based on the storage type specified for the cube. If you define a relational implementation for the cube, then only those dimensions that use a relational implementation are displayed. If you define a MOLAP implementation for the cube, then only the dimensions that use a MOLAP implementation are displayed.
- **Level:** The **Levels** displays all the levels in the dimension selected in the Dimension field. Select the dimension level that the cube references.

- **Role:** The **Role** list displays the dimension roles, if any, that the selected dimension contains. Select the dimension role that the cube uses. You can specify dimension roles for relational dimensions only.

**Advanced Dialog Box** Use the Advanced dialog box to specify the sparsity of the dimensions that the cube references. Sparsity is applicable for only for MOLAP cubes and ROLAP cubes that are implemented using cube MVs. For more information about sparsity, see *Oracle OLAP User's Guide*.

This dialog box displays a table that contains two columns: Dimensions and Sparsity.

- **Dimensions:** This column displays all the dimensions listed on the Dimension tab of the Cube Editor. The dimensions are listed in the order in which they appear on the Dimensions tab. To change the order in which the dimensions appear on this dialog box, you must change the order in which the dimensions are listed on the Dimensions Tab of the Cube Editor.
- **Sparsity:** Sparsity specifies that the cube data is sparse along a particular dimension. Select **Sparsity** for a dimension reference if the cube data is sparse along that dimension. For example, if the data in the SALES cube is sparse along the Promotions dimension, select Sparsity for the Promotions dimension.

  All the sparse dimensions in a cube must be grouped together starting from the least sparse to the most sparse. For example, the Sales cube references the dimensions Times, Products, Promotions, and Channels. This is the order in which the dimensions are listed in the Advanced dialog box. The cube data is sparse along the dimensions Promotions and Channels, with Promotions being the most sparse. Then all these dimensions should appear as a group in the following order: Times, Products, Channels, and Promotions. You cannot have any other dimension listed in between these dimensions.

  Use the following guidelines to order dimensions:

  - List the time dimension first to expedite data loading and time-based analysis. Time is often a dense dimension, although it may be sparse if the base level is Day or the cube has many dimensions.
  - List the sparse dimensions in order from the one with the most members to the one with the least. For a compressed cube, list the sparse dimensions in order from the one with the least members to the one with the most.

  Defining sparsity for a cube provides the following benefits:

  - Improves data retrieval speed.
  - Reduces the storage space used by the cube.

**Compress Cube** Select this option to compress the cube data and then store it. Compressed storage uses less space and results in faster aggregation than a normal space storage. For more details on compressing cubes, see *Oracle OLAP User's Guide*.

Compressed storage is normally used for extremely sparse cubes. A cube is said to be extremely sparse if the dimension hierarchies contain levels with little change to the number of dimension members from one level to the next. Thus many parents have only one descendent for several contiguous levels. Since the aggregated data values do not change from one level to the next, the aggregate data can be stored once instead of repeatedly.

For compressed composites, you can only choose SUM and non-additive aggregation operators.

**Partition Cube** Select this option to partition the cube along one of its dimensions. Partitioning a cube improves the performance of large measures.

Use the table below the Partition Cube option to specify the dimension along which the cube is partitioned. The specified dimension must have at least one level-based hierarchy and its members must be distributed evenly, such that every parent at a particular level has roughly the same number of children. Use the **Dimension** column to select the dimension along which the cube is partitioned. Use the **Hierarchy** and **Level** columns to select the dimension hierarchy and level.

Time is usually the best choice to partition a cube because it meets the required criteria. In addition, data is loaded and rolled off by time period, so that new partitions can be created and old partitions dropped as part of the data refresh process.

**Use a Global Index** Select this option to create a global partitioned index.

### Measures Tab

Use the Measures tab to define the cube measures. Specify the following details for each measure:

- **Name:** The name of the measure. The measure name must be unique within the cube.
- **Description:** An optional description for the measure.
- **Data Type:** The data type of the measure.
- **Length:** The maximum number of bytes for the measure. Length is specified only for character data.
- **Precision:** Define the total number of digits allowed for the column. Precision is defined only for numeric data types.
- **Scale:** Define the total number of digits to the right of the decimal point. Scale is defined only for numeric data types.
- **Seconds Precision:** Represents the number of digits in the fractional part of the datetime field. It can be a number between 0 and 9. The seconds precision is used only for `TIMESTAMP`, `TIMESTAMP WITH TIME ZONE`, and `TIMESTAMP WITH LOCAL TIME ZONE` data types.
- **Expression:** Use this field to define a calculated measure. A calculated measure is a measure whose data is not stored. Its value is calculated when required using the expression defined. Click the Ellipsis button to display the Calculated Measure wizard. For more information about the Calculated Measure wizard, see "Calculated Measure Wizard" on page 3-50.

  You can use any other measures defined in the cube to create an expression for a measure. The expression defined can be validated only at deploy time.

  > **Note:** You can create calculated measures for MOLAP dimensions only.

Click the **Generate Calculated Measures** button to generate a series of standard calculations for a base measure. This is a time-saver operation for creating share, rank and time based calculations. Any calculated measure that you create using this option can also be created manually using the Calculated Measure wizard.

### Calculated Measure Wizard

Use the Calculated Measure wizard to create calculated measures in a cube that uses a MOLAP implementation. These calculated measures, just like the other measures defined on the cube, are deployed to an analytic workspace. The wizard enables you create certain extra calculations that are not created when you click Generate Calculated Measures.

**Define Calculated Measure Details** Use this page to define the details of the calculated measure. The contents of this page depend on the type of calculation you chose on the Select Calculated Measure Type page. For example, if you choose addition as the calculated measure type, this page displays the two lists that enable you to select the measure s that should be added.

If you chose Custom Expression on the Select Calculated Measure Type page, the Expression Builder interface is displayed. Use this interface to define a custom measure. For more information about the Expression Builder, see "About the Expression Builder" on page 26-3.

**Reviewing the Summary Information** Use the Finish page to review the information defined for the calculated measure. Click **Back** to change any of the selected values. Click **Finish** to complete the calculated measure definition.

### Aggregation Tab

Use the Aggregation tab to define the aggregations that must be performed for each dimension that the cube references. You select the aggregate function that is used to aggregate data. You can also precompute measures along each dimension that the cube references. By default, aggregation is performed for every alternate level starting from the lowest level. The default aggregate function is SUM. For more details on the strategies for summarizing data, see the chapter about summarizing data in the *Oracle OLAP User's Guide*.

Specify the following options on the Aggregations tab:

- **Cube Aggregation Method:** Select the aggregate function used to aggregate the cube data. The default selection is SUM.
- **Summary Refresh Method:** Select the data refresh method. The options you can select are On Demand and On Commit.

**Summary Strategy for Cube** Use this section to define levels along which data should be precomputed for each dimension. The **Dimension** column lists the dimensions that the cube references. To select the levels in a dimension for which data should be precomputed, click the Ellipsis button in the **PreCompute** column to the right of the dimension name. The PreCompute dialog box is displayed. Use this dialog box to select the levels in the dimension along which the measure data is precomputed. You can specify the levels to be precomputed for each dimension hierarchy. By default, alternate levels, starting from the lowest level, are precomputed.

> **Note:** You cannot define aggregations for pure relational cubes (cubes implemented in a relational schema in the database only and not in OLAP catalog).

**Precomputing ROLAP Cubes** For ROLAP cubes, aggregation is implemented by creating materialized views that store aggregated data. These materialized views improve query performance. For MOLAP implementations, the aggregate data is generated

and stored in the analytic workspace along with the base-level data. Some of the aggregate data is generated during deployment and the rest is aggregated on the fly in response to a query, following the rules defined in the Aggregation tab.

> **Note:** The materialized views created to implement ROLAP aggregation are not displayed under the Materialized Views node in the Projects Navigator.

### Orphan Tab

Use the Orphan tab to specify the orphan management policy to use while loading data into the cube or deleting it. The Orphan tab contains two sections: Null Dimension key values and Invalid dimension key values, that you use to specify the action to be taken for cube records with null dimension key values and cube records with invalid dimension key values respectively.

Select one of the following options to specify the orphan management policy for cube records with null and invalid dimension key values:

- **No Maintenance:** Warehouse Builder does not actively detect, reject, or fix orphan rows.
- **Default Dimension Record:** Warehouse Builder assigns a default dimension record for any row that has an invalid or null dimension key value. Use the Settings button to define the default parent row.
- **Reject Orphan:** Warehouse Builder does not insert the row if it does not have an existing dimension record.

Select **Deploy Error Table(s)** to generate and deploy the error tables related to orphan management along with the dimension.

### Physical Bindings Tab

After you define the cube structure, you must specify the details of the database tables or views that store the cube data. The Physical Bindings tab enables you to define the implementation objects for cubes. Choose one of the following options to bind the cube to the database object that stores its data:

- Auto binding
- Manual binding

**Auto Binding** When you perform auto binding, the measures and dimension references of the cube are automatically mapped to the database columns that store their data.

To perform auto binding, select the cube in the Projects Navigator. From the File menu, click **Bind**. Warehouse Builder maps the measures and dimension references in the cube to the table that stores the cube data.

> **See Also:** *Oracle Warehouse Builder Concepts* for information about auto binding rules

**Manual Binding** In manual binding, you must explicitly map the measures and dimension references in the cube to the database objects that store their data. You can either store the cube data in existing tables or create new tables.

To perform manual binding:

1. Open the Cube Editor for the cube and navigate to the Physical Bindings tab.
2. Right-click a blank area, select **Add** and then select the type of objects that will store the cube data. For example, if the cube data will be stored in a table, right-click a blank area, select **Add** and then **Table**.

   Warehouse Builder displays the Add a new or Existing table dialog box. You either select an existing table or create a new table to store the cube data.
3. Map each attribute in the dimension to the database column that stores its data.

After you define the cube using the Data Object and perform binding (for ROLAP cubes only), you must deploy the cube. For more information about deploying cubes, see "Deploying Cubes" on page 3-45.

## Cubes Stored in Analytic Workspaces

Cubes that use a MOLAP implementation are stored in analytic workspaces. The analytic workspace engine in Oracle Database 10*g* provides APIs called AWXML. These APIs enable both client/server usage (as in Analytic Workspace Manager) and batch-like usage with java stored procedures. This section describes implementation details for MOLAP cubes.

### Ragged Cube Data

If you select **Use natural keys from data source** on the Storage tab of a dimension, mapping code (AWXML mapping code) that can handle ragged fact data for any cube that uses this dimension is generated. The source column for the cube dimension level is actually mapped to every parent level also. This enables ragged fact data to be loaded.

If you select **Generate surrogate keys in the analytic workspace** on the Storage tab of a dimension, when you create a mapping that loads data at the level of this dimension, you will be loading cube dimension members for this level only.

### Defining Aggregations

Warehouse Builder enables you to reuse existing dimensions without the need of defining additional hierarchies. Aggregations are generated based on the cube dimension level references you define. Only hierarchies where the cube dimension level is a member will be included in the aggregation. If the cube dimension level referenced is a non-leaf level of the hierarchy, then levels lower in the hierarchy will be excluded when the cube or measures are solved. For example, if you have two cubes, BUDGET and SALES, they can share the same dimension definitions without additional dimension hierarchy definitions.

### Auto Solving MOLAP Cubes

An important attribute of the OLAP AWXML engine is its ability to auto-solve cubes that are stored in analytic workspaces. You can auto-solve both compressed and non-compressed cubes. A compressed cube is one for which the **Compress Cube** option on the Advanced Dialog Box is selected.

A cube is auto-solved if any of the following conditions are satisfied:

- The cube is compressed
- The cube is not compressed, and the following additional conditions are true:
  - The solve property for all the measures is set to Yes.

  - The dimension levels that the cube references are at the leaf level of all hierarchies the level is a member of.
- Mapping that contains the cube is executed

Incremental Aggregation of cube is dependent on auto-solve (load and aggregate in one operation). Incremental aggregation is a property of the cube operator in the mapping and applies only to auto-solved cubes.

Warehouse Builder can generate cubes that are not auto-solved cubes if any of the following conditions are true:

- The cube is solved by the mapping that loads the cube
- Warehouse Builder transformations are used to solve the cube
- The cube is non-compressed and any of the following conditions are true:
  - Some of the measures have the Solve property set to No.
  - The dimension levels that the cube references are non-leaf levels of a hierarchy the level is a member of.

### Solving Cube Measures

You can choose to solve only one cube measure for both compressed and non-compressed cubes. A compressed cube is one for which the Compress Cube option on the Advanced Dialog Box is selected.

To solve only one measure in a compressed cube, use the following steps:

1. Open the Cube Editor for the cube and navigate to the Aggregation tab.

   You can open the Cube Editor by double-clicking the cube name in the Projects Navigator.
2. Select the measure that you want to solve on the **Measures** section of the Aggregation tab.
3. The **Aggregation for measure** section displays a row for each dimension that the cube references. In the row that represents the dimension along which you want to solve the cube, select **NOAGG** in the **Aggregation Function** column.

To solve only one measure in a non-compressed cube, you will need the latest database patch 10.2.0.2. If you have Oracle Database 10g Release 1 (10.1), then refer to bug 4550247 for details about a patch. The options defined on cube measures for solve indicate which measures will be included in the primary solve. The solve indicator on the cube operator in the map however indicates whether this solve will be executed or not. So the map can just load data or load and solve the data.

### Solving Cubes Independent of Loading

You can solve cubes independent of loading using the predefined transformation WB_OLAP_AW_PRECOMPUTE. This function also enables you to solve measures independently of each other. This transformation function is available in the Globals Navigator under the Public Transformations node in the OLAP category of the Predefined node.

The following example solves the measure `SALES` in the `SALES_CUBE`:

```
declare
    rslt VARCHAR2(4000);
begin
    rslt:=WB_OLAP_AW_PRECOMPUTE('MART','SALES_CUBE','SALES');
end;
```

/

This function contains parameters for parallel solve and maximum number of job queues. If the cube is being solved in parallel, then an asynchronous solve job is launched and the master job ID is returned through the return value of the function.

**Calculation Plans Generated** The following calculation plans are generated:

- Calculation plan for the cube
- Calculation plan for each stored measure

This enables measures to be solved individually after a data load, or entire cubes to be solved. The actual calculation plan can also exclude levels based on the metadata.

### Parallel Solving of Cubes

You can enable parallel solving of cubes by configuring the mapping that loads the cube. The cube operator has a property called **Allow Parallel Solve** and also a property for the **Max Job Queues Allocated**. These two properties determine if parallel solving is performed and also the size of the job pool. The default is to let the AWXML engine determine this value.

### Output of a MOLAP Cube Mapping

When you execute a mapping that loads a cube, one of the output parameters is AW_EXECUTE_RESULT. When the map is executed using parallel solve, this output parameter will contain the job ID. You can then use the following data dictionary views to determine when the job is complete and what to do next:

- ALL_SCHEDULER_JOBS
- ALL_SCHEDULER_JOB_RUN_DETAILS
- ALL_SCHEDULER_RUNNING_JOBS

If the mapping is not executed using parallel solve, the AW_EXECUTE_RESULT output parameter will return the 'Successful' tag or an error. For more information about the error, see the OLAPSYS.XML_LOAD_LOG table.

## Restriction on Referencing a Non-Leaf Level

A cube cannot reference a non-leaf-level when a dimension is a star implementation that is defined without surrogate keys, because non-leaf records are not included in the dimension table. While the parent level information is included in the child records, there is no way to uniquely identify it. While the cube editor allows you to pick a non-leaf level for the dimension reference in the cube, you will obtain incorrect results when you load the cube.

A deployment error is possible on the fact table, if the data types of the parent level business key and the leaf-level business key do not match. In that case, the foreign key definition will be incorrect, because the foreign key column will have the data type of the parent level, but the referenced primary key column will have the data type of the leaf level.

For example, create a simple dimension with two levels, `L1` and `L2`. Assign them business key attributes with these data types: `L1 NUMBER` and `L2 VARCHAR2(20)`. Then, create a cube to reference this dimension and pick `L1` (the parent) as the referenced dimension. When creating the surrogate key attribute role for the ID, change the dimension attribute to null, so that the dimension does not have surrogate

keys. Rebind the cube after picking the parent level as the reference. If you then attempt to deploy the fact table, then OWB returns an ORA-XXXXX error.

## Editing Cube Definitions

You can edit a cube and alter its definition using the Cube Editor. When you edit a dimension definition, the changes are made only in the object metadata. To update the physical object definition, deploy the modified dimension using the Control Center.

**To edit a cube definition:**

Right-click the cube in the Projects Navigator and select **Open**.

*or*

Double-click the cube in the Projects Navigator.

The Cube Editor is displayed. Edit the cube definition using these tabs. For more information about the tabs in the Cube Editor, see Using the Cube Editor to Create Cubes on page 3-45.

## Configuring Cubes

When you configure a cube, you configure both the cube and the underlying table.

**To configure the physical properties for a cube:**

1. From the Projects Navigator, right-click the cube name and select **Configure**.

   The Configuration tab for the cube is displayed.

2. Configure the cube parameters listed in the following categories.

   In addition to these parameters, use the following are some guidelines for configuring a cube.

   - Foreign Key constraints exist for every dimension.
   - Bitmap indexes have been generated for every foreign key column to its referenced dimension.

### Identification

**Deployable:** Select TRUE to indicate if you want to deploy this cube. Warehouse Builder generates scripts only for table constraints marked deployable.

**Deployment Options:** Use this parameter to specify the type of implementation for the cube. The options are:

- **Deploy All:** For a relational or ROLAP implementation, the cube is deployed to the database and a CWM definition to the OLAP catalog. For a MOLAP implementation, the cube is deployed to the analytic workspace.
- **Deploy Data Objects only:** Deploys the cube only to the database. You can select this option only for cubes that have a relational implementation.
- **Deploy to Catalog only:** Deploys the CWM definition to the OLAP catalog only. Use this option if you want applications such as Discoverer for OLAP to access the cube data after you deploy data only. Use this option if you previously deployed with "Data Objects Only" and now want to deploy the CWM Catalog definitions without redeploying the data objects again.
- **Deploy Aggregation:** Deploys the aggregations defined on the cube measures.

**Materialized View Index Tablespace:** The name of the tablespace that stores the materialized view indexes.

**Materialized View Tablespace:** The name of the tablespace that stores the materialized view created for the cube.

**Visible:** This parameter is not used in code generation.

#### Summary Management

- **Cost Based Aggregation:** This parameter is applicable to MOLAP cubes and ROLAP cubes with OLAP summaries (materialized views). Represents the percentage of preaggregation for cubes.

  Setting a value of 0 for this parameter does not create any aggregate values. The aggregations are computed at runtime. Subsequently, this value results in the fastest maintenance and the least storage space. However, it also results in the slowest query response time.

  Setting a value of 100 for this parameter creates all the aggregate values. These values just need to be fetched when a query is executed. This value results in the fastest query response time. However, the maintenance is slow and a lot of storage space is used.

- **Enable Query Rewrite:** Set this parameter to ENABLE to enable query rewrite. The query rewrite mechanism in the Oracle server automatically rewrites SQL queries to use existing materialized views. This improves query performance. Set this parameter only if the cube uses a ROLAP with MVs implementation.

For information about the parameters Enable MV Refresh, MV constraints, Refresh Mode, Refresh Next Date, Refresh On, and Refresh Start Date, see "Summary Management" on page 3-37.

## Specifying How Cubes are Deployed

You can specify the form in which cubes are deployed to the target schema by setting the Deployment Option configuration parameter. The values you can set for deployment option of cubes are: Deploy All, Deploy Data Objects Only, Deploy to Catalog, and Deploy Aggregation. For steps on setting the Configuration Options parameter, see "Configuring Cubes" on page 3-55.

In addition to the Deployment Option configuration parameter, the form in which cubes are deployed also depends on the generation mode you specify. The PL/SQL Generation Mode parameter of the Oracle module containing the cube represents the Oracle Database to which objects in the module are deployed. You can set the PL/SQL Generation Mode to one of the following options: Default, Oracle10g, Oracle10gR2, Oracle11gR1, Oracle11gR2, Oracle8i, and Oracle9i. For more information, see "Configuring Target Modules" on page 2-46.

Table 3–14 describes how cubes with ROLAP implementations are deployed on different Oracle Database versions.

*Table 3–14 Deployment Options for ROLAP Cubes*

| Deployment Option | Target Schema: Oracle Database 10*g*, ROLAP Implementation | Target Schema: Oracle Database 11*g*, ROLAP Implementation | Target Schema: Oracle Database 11*g*, ROLAP with MVs Implementation |
|---|---|---|---|
| Deploy Data Objects | n/a | n/a | n/a |
| Deploy to Catalog | CWM2 | CWM2 | 11g form AW+ |

Table 3–15 describes how cubes with a MOLAP implementation are deployed on different Oracle Database versions.

*Table 3–15 Deployment Options for Cubes with a MOLAP Implementation*

| Deployment Option | Target Schema: Oracle Database 10*g* | Target Schema: Oracle Database 11*g* |
|---|---|---|
| Deploy Data Objects | 10g form AW | 11g form AW |
| Deploy to Catalog | 10g form AW | 11g form AW |
| Deploy All | n/a | n/a |

# Creating Time Dimensions

Warehouse Builder provides the Create Time Dimension wizard that enables you to create a fully functional time dimension quickly. The mapping that populates the time dimension is also created automatically. When you choose a relational implementation for a time dimension, the implementation objects that store the time dimension data are also created.

You can also use the Dimension Editor to define a time dimension with your own specifications. In this case, you must create the implementation objects and the map that loads the time dimension.

## Creating a Time Dimension Using the Time Dimension Wizard

Use the following steps to create a time dimension using the Create Time Dimension wizard:

1. From the Projects Navigator expand the **Databases** node and then the **Oracle** node.
2. Expand the target module where you want to create a time dimension.
3. Right-click **Dimensions**, select **New**.

   The New Gallery dialog box is displayed.
4. Select **Time Dimension** and click **OK**.

   Warehouse Builder displays the Welcome page of the Create Time Dimension wizard. Click **Next** to proceed. The wizard guides you through the following pages:

   - Name and Description Page on page 3-58
   - Storage Page on page 3-58
   - Data Generation Page on page 3-58
   - Levels Page (Calendar Time Dimension Only) on page 3-59

- Levels Page (Fiscal Time Dimension Only) on page 3-59
- Pre Create Settings Page on page 3-60
- Time Dimension Progress Page on page 3-60
- Summary Page on page 3-60


### Name and Description Page

The Name page describes the time dimension. Provide the following details on the Name page:

- **Name:** Type the name of the time dimension. The name must be unique within a module.
- **Description:** Type an optional description for the time dimension.

### Storage Page

Use the Storage page to specify how the time dimension data should be stored in the database. You select the storage type based on the volume of data stored at the lowest level of the entire cube and the refresh rate required. The storage type options are:

- **ROLAP: Relational storage:** Stores the time dimension definition in a relational form in the database. Select this option to create a time dimension that uses a relational or ROLAP implementation.

  Warehouse Builder automatically creates the underlying tables required to implement this time dimension. A star schema is used to implement the time dimension.

  If the database containing the target schema has the OLAP option installed, you can also deploy the dimensions to the OLAP catalog.
- **ROLAP with MVs:** Stores the time dimension definition and its data in a relational form in the database. Additionally, cube-organized MVs are created in the analytic workspace. Select this option to create a dimension that uses a relational implementation and stores summaries in the analytic workspace.

  Using this option provides summary management based on cube-organized MVs in Oracle Database 11*g*. Query performance is greatly improved, without the need to make any modification to your queries.
- **MOLAP: Multidimensional storage:** Stores the time dimension definition and data in an analytic workspace. Select this option to create a time dimension that uses a MOLAP implementation.

  Warehouse Builder stores the time dimension in an analytic workspace with same name as the module. The tablespace that is used to store the analytic workspace is the tablespace that is defined as the users tablespace for the schema that contains the dimension metadata.

For more information about these options, see "Storage Type Page" on page 3-15.

### Data Generation Page

Use the Data Generation page to specify additional information about the time dimension such as the type of time dimension and the range of data stored in it. This page contains details about the range of data stored in the time dimension and the type of temporal data.

**Range of Data** The Range of Data section specifies the range of the temporal data stored in the time dimension. To specify the range, define the following:

- **Start year:** The year from which to store data in the time dimension. Click the list to select a starting year.
- **Number of years:** The total number of years, beginning from Start Year, for which the time dimension stores data. Specify the number of years by selecting a value from the list.

**Type of Time Dimension** Use the Type of Time Dimension section to specify the type of time dimension to create. Select one of the following options for type of time dimension:

- **Calendar:** Creates a calendar time dimension.
- **Fiscal:** Creates a fiscal time dimension. Enter the following additional details to create a fiscal time dimension:
  - **Fiscal Convention:** Select the convention that you want to use to represent the fiscal months. The options available are 544 and 445.
  - **Fiscal Year Starting:** Select the date and month from which the fiscal year starts.
  - **Fiscal Week Starting:** Select the day from which the fiscal week starts.

### Levels Page (Calendar Time Dimension Only)

Use the Levels page to select the calendar hierarchy that should be created and the levels that it contains. Since there is no drill-up path from the Calendar Week level to any of the levels above it, the following two options are provided to create a calendar hierarchy:

- Normal Hierarchy
- Week Hierarchy

**Normal Hierarchy** The Normal Hierarchy contains the following levels:

- Calendar year
- Calendar quarter
- Calendar month
- Day

Select the levels to be included in the calendar hierarchy. You must select at least two levels.

**Week Hierarchy** The Week Hierarchy contains two levels: Calendar Week and Day. Use this hierarchy to create a hierarchy that contains the Calendar Week level. When you select the Week Hierarchy option, both these levels are selected by default.

### Levels Page (Fiscal Time Dimension Only)

Use the Levels page to select the levels that should be included in the fiscal hierarchy. The levels you can select are:

- Fiscal year
- Fiscal quarter
- Fiscal month

- Fiscal week
- Day

You must select a minimum of two levels. Warehouse Builder creates the fiscal hierarchy that contains the selected levels. To create additional hierarchies, use the Dimension Editor. For more information about using the Dimension Editor, see "Editing Time Dimension Definitions" on page 3-61.

### Pre Create Settings Page

The Pre Create Settings page displays a summary of the options you selected on the previous pages of the Create Time Dimension wizard. This includes the attributes, levels, hierarchies, and the name of the map that is used to populate the time dimension. Warehouse Builder uses these settings to create the objects that implement the time dimension. Click **Next** to proceed with the implementation of the wizard. Click **Back** to change any options that you selected on the previous wizard pages.

### Time Dimension Progress Page

The Time Dimension Progress page displays the progress of the time dimension implementation. The progress status log on this page lists the activities that are performed by the Time Dimension wizard to implement the time dimension. After the process is completed, click **Next** to proceed.

### Summary Page

The Summary page summarizes the options selected in the wizard pages. Use this page to review the options you selected.

Click **Finish** to complete the creation of the time dimension. You now have a fully functional time dimension. This dimension is displayed under the Dimensions node of the Projects Navigator. The mapping that loads this time dimension is displayed under the Mappings node in the Projects Navigator.

Warehouse Builder creates the following objects:

- The time dimension object.
- The sequence that populates the surrogate ID of the time dimension levels
- The physical structures that store the time dimension data.

  For a relational implementation, the database tables that store the dimension data are created in the workspace. Warehouse Builder also binds the time dimension attributes to the database columns that store their values. For a MOLAP implementation, the analytic workspace that stores the time dimension and its data is created.
- A mapping that populates the time dimension.

> **Note:** When you delete a time dimension, the table, sequence, and the mapping associated with the time dimension are not deleted. You must explicitly delete these objects.

## Defaults Used by the Time Dimension Wizard

When you create a time dimension using the Time Dimension wizard, the following defaults are used:

- **Storage:** The default implementation for the relational storage is the star schema. For a MOLAP implementation, the dimension is stored in an analytic workspace that has the same name as the Oracle module in which the time dimension is created. The analytic workspace is stored in the tablespace that is assigned as the users tablespace for the schema that owns the Oracle module containing the dimension.
- **Hierarchy:** A standard hierarchy that contains all the levels listed on the Levels page of the Create Dimension wizard is created. The hierarchy contains the levels in the same order that they are listed on the Levels page.

## Editing Time Dimension Definitions

**To edit a time dimension:**

1. From the Projects Navigator expand the Databases node then the Oracle node.
2. Expand the target module that contains the time dimension to be edited.
3. Right-click the time dimension that you want to edit and select **Open**. You can also double-click the time dimension. Warehouse Builder displays the Dimension Editor for the time dimension.
4. Edit the information about the following tabs:
   - Name Tab on page 3-61
   - Storage Tab on page 3-62
   - Attributes Tab on page 3-62
   - Levels Tab on page 3-63
   - Hierarchies Tab on page 3-63

When you modify a time dimension, a new population map and new implementation tables are created. You can choose to either delete the existing population map and implementation tables or to retain them.

Use the Mapping Editor to modify the time dimension population map. You must deploy the mapping that populates the time dimension.

If you delete the population map before deploying the map, you cannot populate data into the time dimension. The work around is to run the time dimension wizard again and create another dimension population map.

### Name Tab

Use the Name tab to describe the Time dimension. Enter the following details on the Name tab:

- **Name:** The name of the time dimension. The name must be unique within the module. For more information about naming conventions, see "Naming Conventions for Data Objects" on page 2-8.
- **Description:** An optional description for the time dimension.
- **Range of Data:** Specifies the range of the data stored in the time dimension. To specify the range, define the following:
  - **Starting year:** The year from which data should be stored in the time dimension. Click the list to select a starting year.
  - **Number of years:** The total number of years, beginning from Starting Year, for which the time dimension stores data. Select a value from the list.

### Storage Tab

Use the Storage tab to specify the type of storage for the time dimension. The storage options you can use are Relational or MOLAP.

**Relational** Selecting the **Relational** option stores the time dimension definition in a relational form in the database. Select one of the following options for the relational implementation of the time dimension:

- **Star schema:** The time dimension is implemented using a star schema. This means that the time dimension data is stored in a single database table or view.
- **Snowflake schema:** The time dimension is implemented using a snowflake schema. This means that the time dimension data is stored in multiple tables or views.

If the database containing the target schema has the OLAP option installed, you can also deploy the dimensions to the OLAP catalog.

**MOLAP** Select MOLAP to store the time dimension definition and data in an analytic workspace in the database. This method uses an analytic workspace to store the time dimension data. Provide the following details for a MOLAP implementation:

- **AW Name:** Enter the name of the analytic workspace that stores the time dimension. Click the Ellipsis button to display a list of available AWs. Warehouse Builder displays a node for each module in the current project. Expand a module to view the list of dimensional objects in the module. Selecting an object from list stores the time dimension in the same analytic workspace as the selected object.
- **Tablespace Name:** Enter the name of the tablespace that stores the analytic workspace. If you do not enter a value, then the analytic workspace is stored in the tablespace that is defined as the users tablespace for the schema containing the time dimension metadata.

### Attributes Tab

The Attributes tab defines the dimension attributes and the sequence used to populate the dimension key of the time dimension. The Sequence field represents the name of the sequence that populates the dimension key column of the time dimension. Use the **Select** to the right of this field to select a sequence from the Available Sequences dialog box. This dialog box lists all the sequences that belong to the current project.

**Dimension Attributes** The Dimension Attributes section lists the dimension attributes of the time dimension. You also use this page to create new dimension attributes. For each attribute, you specify the following details:

- **Name:** The name of the dimension attribute. The attribute name must be unique within the dimension.
- **Description:** An optional description for the attribute.
- **Identifier:** Represents the type of identifier of the attribute. The lists displays two options: Surrogate and Business. Select the type of identifier.
- **Data Type:** Select the data type of the attribute.
- **Length**: Specify length only for character data types.
- **Precision**: Define the total number of digits allowed for the column. Precision is defined only for numeric data types.

- **Scale:** Define the total number of digits to the right of the decimal point. Scale is defined only for numeric data types.
- **Seconds Precision:** Represents the number of digits in the fractional part of the datetime field. It can be a number between 0 and 9. The seconds precision is used only for `TIMESTAMP`, `TIMESTAMP WITH TIME ZONE`, and `TIMESTAMP WITH LOCAL TIME ZONE` data types.
- **Descriptor:** Select the type of descriptor. The options are: Short Description, Long Description, Start date, End date, Time span, and Prior period.

### Levels Tab

The Levels tab defines the levels in the time dimension. You can create additional levels by entering the name and an optional description for the level in the Levels section. For more information about the contents of the Levels tab, see "Level Attributes Page" on page 3-18.

### Hierarchies Tab

Use the Hierarchies tab to create additional hierarchies in the time dimension. When you modify the time dimension definition, the map that populates it must reflect these changes. Click **Create Map** to recreate the map that populates the time dimension. For a fiscal time dimension, you can modify the fiscal settings by clicking **Fiscal Settings**. The Fiscal Information Settings dialog box is displayed. Use this dialog box to modify the fiscal convention, fiscal year start, and fiscal week start.

The Hierarchies tab contains two sections: Hierarchies and Levels.

- **Hierarchies:** Use this section to create hierarchies. Warehouse Builder displays any existing hierarchies in the time dimension. You create additional hierarchies by specifying the name of the hierarchy and type of hierarchy. The options for type of hierarchy are None, Fiscal, Calendar Week, and Calendar Year. Use the **Default** property to indicate which of the hierarchies is the default hierarchy.
- **Levels:** The Levels section lists the levels in the time dimension. When you create a new hierarchy, choose the levels that you want to include in your hierarchy by selecting the Applicable option.

## Modifying the Implementation of Time Dimensions

Use the Time Dimension editor to modify the implementation of a time dimension. The implementation details determine if the time dimension is implemented using a star schema or a snowflake schema.

For each time dimension, Warehouse Builder automatically creates a mapping that loads the time dimension. Thus, when you modify a time dimension, the mapping that loads the time dimension must also be modified.

> **Note:** You cannot use the Unbind option to unbind a time dimension from its implementation objects.

**To modify the implementation of a time dimension:**

1. In the Projects Navigator, double-click the time dimension whose implementation you want to modify.

   The editor is opened for the time dimension.

2. On the Storage tab, under the ROLAP: Relational Storage option, select the new implementation for the time dimension.

   To change to a star schema implementation, select **Star**.

   To change to a snowflake implementation, select **Snowflake**.

3. On the Hierarchies tab, click **Create map**.

   This redefines the mapping that loads the time dimension based on the implementation changes made.

The Physical Bindings tab displays the modified bindings for the time dimension.

## Populating Time Dimensions

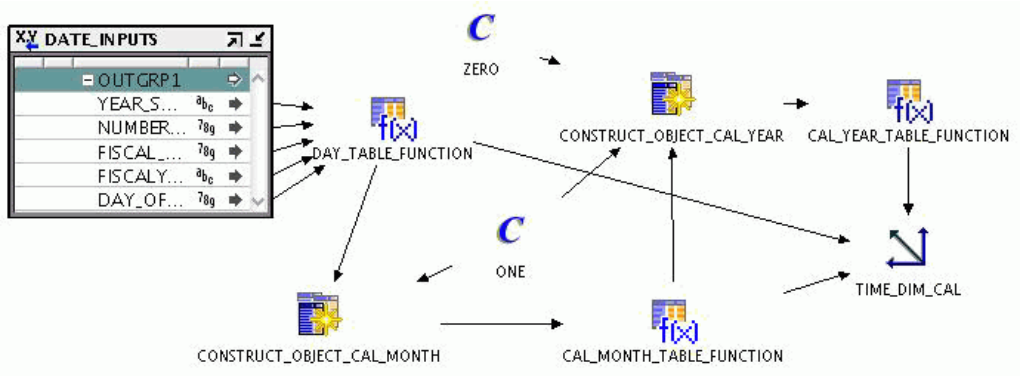
You populate a time dimension by creating a mapping that loads data into the time dimension. When you create a time dimension using the Create Time Dimension wizard, Warehouse Builder creates a mapping that populates the time dimension based on the values of the following parameters:

- Start year of the data
- Number of years of the data
- Start day and month of fiscal year (only for fiscal time dimensions)
- Start day of fiscal week (only for fiscal time dimensions)
- Fiscal type (only for fiscal time dimensions)

The values of these parameters are initialized at the time of creating the time dimension using the Create Time Dimension wizard.

Figure 3–1 displays a mapping to load a calendar time dimension. The Mapping Input operator `DATE_INPUTS` represents the attributes needed to populate the time dimension. The values of the attributes in this operator are set based on the values you provide when you created the time dimension. You can modify these values by double-clicking the DATE_INPUTS operator, clicking the Output Attributes link, and modifying the values of the input parameters. However, when you modify parameter values, you must regenerate the redeploy the mapping that loads the time dimension.

***Figure 3–1 Mapping that Populates a Time Dimension***




## Dynamically Populating Time Dimensions

In certain warehouse scenarios, you may need to dynamically populate your time dimension based on the current requirements. Although the values used to populate

the time dimension are set at the time of creating the time dimension, Warehouse Builder enables you to modify these values and dynamically populate the time dimension when required.

The Deployment Preferences contain a preference called Prompt for Execution Parameters. Setting this parameter to True enables you to provide values for the input parameters at runtime.

**To dynamically populate time dimensions:**

1. Ensure that the Prompt for Execution Parameters preference is set to True.

   In the Tools menu, select **Preferences** to display the Preferences dialog box. In the left panel, expand the OWB node, click Deployment in the left panel and select **Prompt for Execution Parameters**.

2. Execute the mapping that loads the time dimension by right-clicking the mapping and selecting **Start**.

   The Input Parameters dialog box is displayed containing the input parameters that are used to load the time dimension such as start year, number of years.

3. Set values for the parameters YEAR_START_DATE and NUMBER_YEARS and click **OK**. For fiscal time dimensions, also set values for FISCAL_TYPE, DAYS_OF_FISCAL_WEEK, and FISCALYEAR_START_DATE.

## Overlapping Data Populations

You can run a map that populates the time dimension multiple times. During each run you specify the attributes required to populate the time dimension. It is possible that a run of the mapping may overlap with the previous runs, meaning you may attempt to load data that already exists in the time dimension. In such a case, if a record was populated by a previous run, Warehouse Builder does not populate the data again.

For example, in the first run, you populate the time dimension with data from the year 2000 for 5 years. In the second run, you populate the time dimension with data from 2003 for 3 years. Since the records from beginning 2003 to end 2004 already exist in the time dimension, they are not created again.

# Part II

## Performing ETL

Oracle Warehouse Builder enables you to perform Extract, Transform, and Load (ETL) operations. You can choose among different methods of transforming source data before loading it into your data warehouse. This part discusses deploying data and ETL objects and data auditing.

This part contains the following chapters:


- Chapter 4, "Overview of Transforming Data"
- Chapter 5, "Creating PL/SQL Mappings"
- Chapter 6, "Performing ETL Using Dimensional Objects"
- Chapter 7, "Using SQL*Loader, SAP, and Code Template Mappings"
- Chapter 8, "Designing Process Flows"
- Chapter 9, "Defining Custom Transformations"
- Chapter 10, "Understanding Performance and Advanced ETL Concepts"
- Chapter 11, "Scheduling ETL Jobs"
- Chapter 12, "Deploying to Target Schemas and Executing ETL Logic"
- Chapter 13, "Auditing Deployments and Executions"
- Chapter 14, "Managing Metadata Dependencies"
- Chapter 15, "Troubleshooting and Error Handling for ETL Designs"
- Chapter 16, "Creating and Consuming Web Services in Warehouse Builder"
- Chapter 17, "Moving Large Volumes of Data Using Transportable Modules"

# 4

# Overview of Transforming Data

One of the main functions of an Extract, Transform, and Load (ETL) tool is to transform data. Oracle Warehouse Builder provides various methods of transforming data. This chapter provides an overview of data transformation in Warehouse Builder.

This chapter contains the following topics:


- About Data Transformation in Oracle Warehouse Builder
- About Mappings
- About Operators
- About Transformations
- About Transformation Libraries


## About Data Transformation in Oracle Warehouse Builder

After you import your source data and define the target, you can consider how to transform the source data into the output desired for the target. In Warehouse Builder, you specify how to transform the data by designing *mappings* in the Mapping Editor. A mapping is a Warehouse Builder entity that describes the sequence of operations required to extract data from sources, transform the data, and load the data into one or more targets.

The fundamental unit of design for a mapping is the *operator*. You use an operator to represent each distinct operation you want to perform in the mapping. Operations include extracting data, loading data, and transforming data (aggregating, joining, performing a lookup, and so on). To indicate the order of operations, you connect the mappings with data flow connections.

To specify data transformation in a mapping, select from the many prebuilt *transformation operators* or design a new transformation. The prebuilt transformation operators enable commonly performed operations such as filtering, joining, and sorting. Warehouse Builder also includes prebuilt operators for complex operations such as merging data, cleansing data, or profiling data. If none of the prebuilt transformation operators meet your needs, you can design a new one. You can design the new transformation operator based on the Oracle Database library of PL/SQL functions, procedures, package functions, and package procedures.

Extraction and loading operations are represented by any of the numerous *source and target operators*. For example, a Table operator represents a table and a Flat File operator represents a flat file. Whether that operator specifies an extraction or loading operation depends on how you connect the operator relative to other operators in the mapping.

An important distinction to understand is the difference between the operator in the mapping and the object it represents. The operator and the object are separate entities until you *bind* the two together. For example, when you add a table operator to a mapping, you can bind that operator to a specific table in the repository. With the operator bound to the table, you can *synchronize* changing definitions between the two. If the table operator represents a target and you change the operator in the mapping, then you can propagate those changes back to the table in the repository. If the operator represents a source that incurred a change in its metadata definition, then you can reimport the table in the Design Center and then propagate those changes to the table operator in the Mapping Editor.

## About Mappings

Mappings describe a series of operations that extract data from sources, transform it, and load it into targets. They provide a visual representation of the flow of the data and the operations performed on the data. When you design a mapping in Oracle Warehouse Builder, you use the Mapping Editor interface. Alternatively, you can create and define mappings using OMB*Plus, the scripting interface for Warehouse Builder as described in *Oracle Warehouse Builder API and Scripting Reference*.

Based on the ETL logic that you define in a mapping, Warehouse Builder generates the code required to implement your design. Warehouse Builder can generate code for the following languages:

- PL/SQL mappings, see "PL/SQL Mappings" on page 5-3
- SQL*Loader mappings, see "SQL*Loader Mappings" on page 5-3
- SAP ABAP mappings, see "SAP ABAP Mappings" on page 5-3
- Code Template mappings, "Code Template (CT) Mappings" on page 5-3

## About Operators

The basic design element for a mapping is the operator. Use operators to represent sources and targets in the data flow. Also use operators to define how to transform the data from source to target. The operators that you select as sources affect how you design the mapping. Based on the operators that you select, Warehouse Builder assigns the mapping to one of the following mapping generation languages. Each of these code languages require you to adhere to certain rules when designing a mapping.

- SQL

  For mappings that contains code templates, Warehouse Builder generates SQL code.

- PL/SQL

  For all mappings that do not contain either a Flat File operator as a source or a SAP/R3 source, Warehouse Builder generates PL/SQL code. Design considerations for PL/SQL mappings depend upon whether you specify a row-based or set-based operating mode as described in Chapter 10, "Understanding Performance and Advanced ETL Concepts".

- SQL*Loader

  When you define a Flat File operator as a source, Warehouse Builder generates SQL*Loader code. To design a SQL*Loader mapping correctly, follow the guidelines described in "Flat File Source Operators" on page 25-32.

- ABAP

  When you define a SAP/R3 source, Warehouse Builder generates ABAP code. For mapping design considerations for SAP sources, see "Creating SAP Extraction Mappings" on page 7-4.

## Types of Operators

As you design a mapping, you select operators from the Mapping Editor palette and drag them onto the canvas.

This section introduces the types of operators and refers you to other chapters in this manual for detailed information.

- Source and Target Operators: These operators represent Oracle Database objects, flat files, remote sources or targets, and non-Oracle sources and targets.
- Remote and Non-Oracle Source and Target Operators: The special requirements for using these operators are discussed in "Using Remote and non-Oracle Source and Target Operators" on page 25-30.
- Transformation Operators: These operators transform data.
- Pre/Post Processing Operators: These operators call a function or procedure before or after executing a mapping.
- Pluggable Mapping Operators: These are mappings that function as operators in other mappings.
- Real-time Data Warehousing Operators: These operators are used in creating realtime and batch mappings.

### Source and Target Operators

Use source and target operators to represent relational database objects and flat file objects.

Table 4–1 lists each source and target operator alphabetically and gives a brief description. For more information about these operators, see Chapter 25, "Source and Target Operators".

***Table 4–1 Source and Target Operators***

| Icon | Operator Name | Description |
|---|---|---|
| | Constant | Produces a single output group that can contain one or more constant attributes. |
| | Construct Object | Produces object types and collection types. |
| | Cube | Represents a cube that you previously defined. |
| | Data Generator | Provides information such as record number, system date, and sequence values. |
| | Dimension | Represents a dimension that you previously defined. |
| | Expand Object | Expands an object type to obtain the individual attributes that comprise the object type. |

*Table 4–1 (Cont.) Source and Target Operators*

| Icon | Operator Name | Description |
|---|---|---|
| | External Table | Represents an external table that you previously defined or imported. |
| | Flat File | Represents a flat file that you previously defined or imported. |
| | Materialized View | Represents a materialized view that you previously defined |
| | Queue | Represents an advanced queue that you previously defined. |
| | Sequence | Generates sequential numbers that increment for each row. |
| | Table | Represents a table that you previously defined or imported. |
| | Varray Iterator | Iterates through the values in the table type. |
| | View | Represents a view that you previously defined or imported. |

## Transformation Operators

To transform your source data, use data flow operators in your mapping.

Table 4–2 lists each data flow operator alphabetically and gives a brief description. For more information about these operators, see Chapter 26, "Data Flow Operators", Chapter 22, "Name and Address Cleansing", and Chapter 23, "Matching, Merging, and Deduplication".

*Table 4–2 Data Flow Operators*

| Icon | Operator Name | Description |
|---|---|---|
| | Aggregator | Performs data aggregations, such as SUM and AVG, and provides an output row set with aggregated data. |
| | Anydata Cast | Converts an object of type Sys.AnyData to either a primary type or to a user-defined type. |
| | Deduplicator | Removes duplicate data in a source by placing a DISTINCT clause in the select code represented by the mapping. |
| | Expression | Enables you to write SQL expressions that define nonprocedural algorithms for one output parameter of the operator. The expression text can contain combinations of input parameter names, variable names, and library functions. |
| | Filter | Conditionally filters out rows from a row set. |
| | Joiner | Joins multiple row sets from different sources with different cardinalities and produces a single output row set. |
| | Lookup | Performs a lookup of data from a lookup object such as a table, view, materialized view, external table, cube, or dimension. |
| | Match Merge | Data quality operator that identifies matching records and merges them into a single record. |

*Table 4–2 (Cont.) Data Flow Operators*

| Icon | Operator Name | Description |
|---|---|---|
| | Name and Address | Identifies and corrects errors and inconsistencies in name and address source data. |
| | Pivot | Transforms a single row of attributes into multiple rows. Use this operator to transform data that contained across attributes instead of rows. |
| | Set Operation | Performs union, union all, intersect, and minus operations in a mapping. |
| | Sorter | Sorts attributes in ascending or descending order. |
| | Splitter operator | Splits a single input row set into several output row sets using a boolean split condition. |
| | Subquery Filter | filter rows based on the results of a subquery. |
| | Table Function | Enables you to develop custom code to manipulate a set of input rows and return a set of output rows of the same or different cardinality that can be queried like a physical table. |
| | Transformation | Transforms the attribute value data of rows within a row set using a PL/SQL function or procedure. |
| | Unpivot | Converts multiple input rows into one output row. It enables you to extract from a source once and produce one row from a set of source rows that are grouped by attributes in the source data. |

## Pre/Post Processing Operators

Use Pre/Post Processing operators to perform processing before or after executing a mapping. The Mapping parameter operator is used to provide values to and from a mapping.

Table 4–3 lists the Pre/Post Processing operators and the Mapping Parameter operators and gives a brief description. For more details about these operators, see Chapter 26, "Data Flow Operators".

*Table 4–3 Pre/Post Processing Operators*

| Icon | Operator | Description |
|---|---|---|
| | Mapping Input Parameter | Passes parameter values into a mapping. |
| | Mapping Output Parameter | Sends values out of a mapping. |
| | Post-Mapping Process | Calls a function or procedure after executing a mapping. |
| | Pre-Mapping Process | Calls a function or procedure prior to executing a mapping. |

## Pluggable Mapping Operators

A pluggable mapping is a reusable grouping of mapping operators that behaves as a single operator.

Table 4–4 lists the Pluggable Mapping operators and gives a brief description. For more information, see Chapter 26, "Data Flow Operators".

***Table 4–4 Pluggable Mapping Operators***

| Icon | Operator | Description |
|---|---|---|
| | Pluggable Mapping | Represents a reusable mapping. |
| | Pluggable Mapping Input Signature | A combination of input attributes that flow into the pluggable mapping. |
| | Pluggable Mapping Output Signature | A combination of output attributes that flow out of the pluggable mapping. |

### Real-time Data Warehousing Operators

Real-time data warehousing operators enable you to use queues to perform Change Data Capture. You can create mappings that capture changes in source objects and then apply them to target tables.

Table 4–5 lists the real-time data warehousing operators.

***Table 4–5 Pluggable Mapping Operators***

| Icon | Operator | Description |
|---|---|---|
| | LCR Cast | Expands an LCR (Logical Change Record) object into its constituent columns. |
| | LCR Splitter | Directs changes to different tables along data flow paths. |

# About Transformations

Transformations are PL/SQL functions, procedures, packages, and types that enable you to transform data. You use transformations when designing mappings and process flows that define ETL processes. Transformations are stored in the Warehouse Builder workspace and can be used in the project in which they are defined. Transformation packages are deployed at the package level but executed at the transformation level.

## Types of Transformations

In Warehouse Builder, transformations can be categorized as follows:

- Predefined Transformations
- Custom Transformations

### Predefined Transformations

Warehouse Builder provides a set of predefined transformations that enable you to perform common transformation operations. These predefined transformations are part of the public Oracle Predefined library that consists of built-in and seeded functions and procedures. You can directly use these predefined transformations to transform your data.

Predefined transformations are organized into the following categories:

- Administration

- Character
- Control Center
- Conversion
- Date
- Numeric
- OLAP
- Others
- SYS
- Spatial
- Streams
- XML

For more information about the transformations that belong to each category, see Chapter 28, "Warehouse Builder Transformations Reference".

### Custom Transformations

A custom transformation is one that is created by the user. Custom transformations can use predefined transformations as part of their definition.

Custom transformations contain the following categories:

- **Functions:** The Functions category contains standalone functions. This category is available under the Custom node of the Public Transformations node in the Globals Navigator. It is also created automatically under the Transformations node of every Oracle module, DB2 module, and SQL Server module in the Projects Navigator.

  Functions can be defined by the user or imported from a database. A function transformation takes 0 to *n* input parameters and produces a result value.

- **Procedures:** The Procedures category contains any standalone procedures used as transformations. This category is available under the Custom node of the Public Transformations node in the Globals Navigator. It is also automatically created under the Transformations node of each Oracle module in the Globals Navigator.

  Procedures can be defined by the user or imported from a database. A procedure transformation takes 0 to *n* input parameters and produces 0 to *n* output parameters.

- **Table functions:** The Table Functions category contains any standalone table functions you can use as transformations. This category is available under the Custom node of the Public Transformations node in the Globals Navigator. It is also automatically created under the Transformations node of each Oracle module in the Projects Navigator.

  The Table functions category is also listed under Packages. Any table functions created here belong to the package.

- **Packages:** The Packages category contains packages, which in turn contain functions, procedures, table functions, and PL/SQL types. This category is available under the Custom node of the Public Transformations node in the Globals Navigator. It is also automatically created under the Transformations node of each Oracle module in the Globals Navigator.

PL/SQL packages can be created or imported in Warehouse Builder. The package body may be modified. For packages that are imported from database objects, the package header (which is the signature for the function or procedure) cannot be modified. However, for packages that are imported from Warehouse Builder, the package header can be modified. For instructions, see "Importing Transformations" on page 9-13

- **PL/SQL Types:** The PL/SQL Types category contains any standalone PL/SQL types. This includes PL/SQL record types, REF cursor types, and nested table types. The PL/SQL Types category is automatically created in each package that you define under the Packages node in the Transformations node of the Projects Navigator. It is also available under every package that you define in the Globals Navigator. In the Globals Navigator, expand the Public Transformations node, the Oracle node, the Custom node, and then the Packages node to define PL/SQL types under a package.

For further instructions, see "Defining Custom Transformations" on page 9-2.

# About Transformation Libraries

A transformation library consists of a set of reusable transformations. Each time you create a repository, Warehouse Builder creates a Transformation Library containing transformation operations for that repository. This library contains the standard public Oracle Predefined library and an additional library for each Oracle module defined within the project.

Transformation libraries are available under the Public Transformations node of the Globals Navigator in the Design Center.

## Types of Transformation Libraries

Transformation libraries can be categorized as follows:

- **Public Oracle Predefined Library**

  This is a collection of predefined functions from which you can define procedures for your public Oracle Custom library. The public Oracle Predefined library is contained in the Globals Navigator. Expand the Pre-Defined node under the Public Transformations node. Each category of predefined transformations is represented by a separate node. Expand the node for a category to view the predefined transformations in that category. For example, expand the Character node to view the predefined character transformations contained in the public Oracle Predefined library.

- **Public Oracle Custom Library**

  This is a collection of reusable transformations created by the user. These transformations are categorized as functions, procedures, and packages defined within your workspace.

  The transformations in the public Oracle Custom library are available under the Custom node of the Public Transformations node. Any transformation that you create under this node is available across all projects in the workspace. For information about creating transformations in the public Oracle Custom library, see "Defining Custom Transformations" on page 9-2.

  When you deploy a transformation defined in the public Oracle Custom library, the transformation is deployed to the location that is associated with the default control center.

## Accessing Transformation Libraries

Because transformations can be used at different points in the ETL process, Warehouse Builder enables you to access transformation libraries from different points in the Design Center.

You can access the transformation libraries from the following points:

- Expression Builder

  While creating mappings, you may need to create expressions to transform your source data. The Expression Builder interface enables you to create the expressions required to transform data. Because these expressions can include transformations, Warehouse Builder enables you to access transformation libraries from the Expression Builder.

  Transformation libraries are available under the Transformations tab of the Expression Builder. The Private node under TRANSFORMLIBS contains transformations that are available only in the current project. These transformations are created under the Transformations node of the Oracle module. The Public node contains the custom transformations from the public Oracle Custom library and the predefined transformations from the public Oracle Predefined library.

- Add Transformation Operator dialog box

  The Transformation operator in the Mapping Editor enables you to add transformations, both from the public Oracle Predefined library and the public Oracle Custom library, to a mapping. You can use this operator to transform data as part of the mapping.

- Function Editor, Procedure Editor, Edit Function dialog box, or Edit Procedure dialog box

  The Implementation tab of these editors enables you to specify the PL/SQL code that is part of the function or procedure body. You can use transformations in the PL/SQL code.

# 5

# Creating PL/SQL Mappings

After you create data object definitions in Oracle Warehouse Builder, you can design extract, transform, and load (ETL) operations that move data from sources to targets. In Warehouse Builder, you design these operations in a mapping.

You can also use the Mapping Debugger to debug data flows created in mappings.

This chapter contains the following topics:


- Overview of Oracle Warehouse Builder Mappings
- Example: Defining a Simple PL/SQL Mapping
- Steps to Perform Extraction, Transformation, and Loading (ETL) Using Mappings
- Defining Mappings
- Adding Operators to Mappings
- Connecting Operators, Groups, and Attributes
- Editing Operators
- Setting Mapping Properties
- Configuring Mappings
- Synchronizing Operators and Workspace Objects
- Example: Using a Mapping to Load Transaction Data
- Example: Using the Mapping Editor to Create Staging Area Tables
- Using Pluggable Mappings
- Copying Operators Across Mappings and Pluggable Mappings
- Grouping Operators in Mappings and Pluggable Mappings
- Locating Operators, Groups, and Attributes in Mappings and Pluggable Mappings
- Debugging Mappings


## Overview of Oracle Warehouse Builder Mappings

A mapping is a Warehouse Builder object that you use to perform extract, transform, and load (ETL). A mapping defines the data flows for moving data from disparate sources to your data warehouse.

You can extract data from sources that include, but are not limited to, flat files, Oracle databases, SAP, or other heterogeneous databases such as SQL Server and IBM DB2.

Use mapping operators or transformations to transform the data, according to your requirements, and to load the transformed data into the target objects.

**See Also:**

- "Defining Mappings" on page 5-11 for information about defining mappings
- Chapter 26, "Data Flow Operators" for information about mapping transformation operators and how to use them
- Chapter 28, "Warehouse Builder Transformations Reference" for more information about predefined Warehouse Builder transformations

As with other Warehouse Builder objects, after you define a mapping, you must validate, generate, and deploy the mapping. Once you define and deploy a mapping, you can execute the mapping using the Control Center Manager, schedule it for later execution, or incorporate it in a process flow.

**See Also:**

- "Deploying Objects" on page 12-6 for information about deploying mappings
- "Scheduling ETL Jobs" on page 11-1 for information about scheduling ETL objects
- "Designing Process Flows" on page 8-1 for information about defining process flows

## Types of Mappings

Warehouse Builder mappings can be classified according to the supported data extraction technologies used in the mapping. The code generated by Warehouse Builder depends on the sources and targets used in the mapping.

The different types of mappings include:

- PL/SQL Mappings
- SQL*Loader Mappings
- SAP ABAP Mappings
- Code Template (CT) Mappings

Warehouse Builder generates ETL code in different languages for the different mapping types. The generated code is deployed to a target location, where it executes. By selecting the appropriate data extraction technology for each mapping, you can access a wide range of data source types, and satisfy different technical requirements, such as performance and security.

**Note:** Mappings other than PL/SQL mappings do not support all Warehouse Builder data transformation capabilities. If you must use one of the non-PL/SQL mapping types but you still need to perform complex transformations supported only in PL/SQL mappings, use the non-PL/SQL mapping to load a staging table, and then use a PL/SQL mapping to perform the rest of the required transformation.

### PL/SQL Mappings

PL/SQL mappings are the default mapping type in Warehouse Builder and should be used in most situations. For PL/SQL mappings, Warehouse Builder generates PL/SQL code that is deployed to an Oracle Database location, where it executes.

Data extraction from other locations is primarily performed through database links to other Oracle Databases or through Oracle Database gateways for non-Oracle data sources. PL/SQL mappings offer the full range of Warehouse Builder data transformation capabilities.

### SQL*Loader Mappings

For SQL*Loader mappings, Warehouse Builder generates SQL*Loader control files. The control file is deployed to the target database, where SQL*Loader executes, loading the source data into the database.

> **Note:** SQL*Loader mappings support only a subset of transformations available in PL/SQL mappings. For information about the limitations on the transformations available in SQL*Loader mappings, see Chapter 26, "Data Flow Operators".

### SAP ABAP Mappings

ABAP mappings are the only supported method of extracting data from SAP R/3 source systems. For ABAP mappings, Warehouse Builder generates ABAP code. This code can then be deployed to an SAP R/3 instance automatically or manually by an administrator, depending upon security and other administrative requirements specific to the SAP environment. The ABAP code executes, generating a flat file as output, which is then transparently moved to the target database system and loaded into the Oracle target database.

> **Note:** ABAP mappings support only a subset of transformations available in PL/SQL mappings. For information about the limitations on the transformations available in ABAP mappings, see Chapter 26, "Data Flow Operators".

### Code Template (CT) Mappings

Code templates provide a general framework for implementing data extraction, movement, and loading mechanisms. For Code Template (CT) mappings, Warehouse Builder generates data extraction or other mapping code based on the contents of a code template. This code is then deployed to a remote agent on a target system, where it executes. The technology used to load data into the target database depends upon the contents of the code template. Warehouse Builder provides a collection of code templates that implement common data extraction methods. Other code templates use bulk data extraction and loading tools for faster and more flexible data movement.

Choose CT mappings when you need to:

- Access a data source without using Oracle Database gateways or ODBC, or you have a specific need for JDBC connectivity. Warehouse Builder provides code templates that support access to any JDBC data source or target.
- Load data from an XML source file.
- Perform bulk data unloads and loads for maximum data movement performance. Warehouse Builder provides code templates to support bulk data movement from

some common databases. You can write your own code templates for sources not supported by the Warehouse Builder-provided code templates.

- Implement ETL processes where data moves from a non-Oracle database source directly to a non-Oracle database target. For example, you can define an ETL process that extracts from an IBM DB2 source and loads a Microsoft SQL Server target directly.
- Implement new data integration patterns without requiring changes to Warehouse Builder itself. Code templates provide maximum data integration flexibility for business and technical requirements beyond those supported by Warehouse Builder out of the box.

**Note:** CT mappings support only a subset of transformations available in PL/SQL mappings. For information about the limitations on the transformations available in CT mappings, see "Mapping Operators Only Supported in Oracle Target CT Execution Units" on page 7-16.

## About Chunking for PL/SQL Mappings

Chunking enables you to parallelize the processing of large updates by dividing the source data in a mapping into smaller chunks. The chunks are defined by means of user-defined chunking criteria. At runtime, the chunks are executed in parallel and each chunk is loaded separately into the target object.

**Note:** Chunking can be performed for PL/SQL mappings only.

When you define chunking, Warehouse Builder creates a chunking task in the target schema at deployment time. During execution, the DML represented by each chunk is run in parallel.

For a PL/SQL mapping that uses chunking, you can configure the following parameters using the Mapping Configuration Editor:

- Code Generation Parameters

  You can set code generation parameters for chunking either while defining the mapping or later, by editing the mapping. Use the Mapping Configuration Editor to set code generation parameters. These parameters are used to generate the code required to execute the logic defined in the mapping.

- Runtime Parameters

  Runtime chunking parameters can be configured at design-time, by using the Mapping Configuration Editor, or while executing the mapping. When you set runtime chunking parameters both at design time and execution time, the values set at execution time override the ones defined at design time.

**See Also:**

- "Chunking Options" on page 24-7
- "SCD Updates" on page 24-8

### Types of Chunking

Warehouse Builder enables you to perform the following types of chunking:

- Serial chunking

  Serial chunking is useful in scenarios where it is not possible to process all source data in one set. The primary use case for serial chunking is loading Type 2 SCDs, where multiple updates to a single target row must be processed separately.

  To configure serial chunking parameters for a mapping, use the SCD Updates and Runtime Parameters nodes in the Mapping Configuration Editor.

- Parallel chunking

  Parallel chunking enables you to update table data in parallel with incremental commit in two high-level steps. In the first step, the rows in a table are grouped into smaller sized chunks. In the second step, a user-specified statement is executed in parallel on these chunks and a commit is issued after the processing is complete.

  Parallel chunking is used for parallel DML operations. To configure parallel chunking parameters for a mapping, use the Chunking Options and Runtime Parameters nodes in the Mapping Configuration Editor.

### Methods of Performing Parallel Chunking

You can perform parallel chunking in one of the following ways:

- Chunking by number column

  The data in the source table is divided into chunks by defining the size of each chunk.

- Chunking by ROWID

  The data in the source table is divided into chunks based on the `ROWID` value of each row in the table.

- Chunking by SQL statement

  Uses a SQL statement to return two columns that define the range of each chunk, either by `ROWID` or column number.

## Overview of the Mapping Editor

The Mapping Editor is built around the Mapping Editor canvas, on which the operators and connections in a mapping are displayed graphically and can be manipulated. Several other Design Center panels are context-sensitive and display mapping-related items when a mapping is open in the Mapping Editor. These include the following:

- **Component Palette:** Displays operators that you can use in a mapping. Select an object, either from the canvas or Projects Navigator, and Warehouse Builder displays the object properties in the Property Inspector.

  Use the filter at the top of the palette to limit the display of operators to a particular type. For example, select Transformation Operators in the filter to display only operators that transform data.

- **Structure View:** Displays a hierarchical view of the operators, attribute groups, and attributes in the mapping.

- **Bird's Eye View:** Displays the layout of the entire mapping. Enables you to move the view of the canvas with a single mouse dragging operation. You can thus reposition your view of the canvas without using the scroll bars.

The Bird's Eye View displays a miniature version of the entire canvas. It contains a blue box that represents the portion of the canvas that is currently in focus. For mappings that span more than the canvas size, click the blue box and drag it to the portion of the canvas that you want to focus on.

- **Property Inspector:** Displays the properties of the mapping, operators, attribute groups, or attributes currently selected in the Mapping Editor canvas.
- **Mapping Debug Toolbar:** Displays icons for each command used in debugging mappings. When you are debugging mappings, the Debug toolbar is displayed at the top of the Mapping Editor canvas.
- **Diagram Toolbar:** Displays icons for each command used to navigate the canvas and change the magnification of objects on the canvas.

## Mapping Editor Canvas

The Mapping Editor canvas is the area that you use to graphically design your mappings. Mappings define how data extracted from the source is transformed before being loaded into the targets.

The Mapping Editor canvas contains two tabs: Logical View and Execution View.

## Logical View

The Logical view of the Mapping Editor enables you to design the data flows that define your mapping. You first drag and drop operators representing the source objects, the target objects, and the transformations. Next you establish a data flow between these operators that represents the data transformation by drawing data flow connections between operators.

**See Also:**

- "Adding Operators to Mappings" on page 5-13 for information about adding source, target, and transformation operators
- "Connecting Operators, Groups, and Attributes" on page 5-16 for information about connecting operators in the mapping

## Execution View

Use the Execution View of the Mapping Editor to define execution units for Code Template (CT) mappings. The Execution View is available only when you create CT mappings.

An *execution unit* represents the set of related tasks that are to be performed using a code template. A code template contains the logic to perform a particular ETL processing on a particular platform during runtime, such as moving data between two Oracle Database instances, or unloading data from a DB2 database into a flat file. An execution unit can be implemented by a single generated script such as a PL/SQL package or by a code template.

Execution units enable you to break up your mapping execution into smaller related units. Each execution unit may be associated with a code template that contains the template to perform the required data integration task on the specified platform. An execution unit can be implemented by a single generated script such as a PL/SQL package or by a code template.

The Execution View tab of the Mapping Editor displays the operators and data flows from the Logical View in an iconized form. You cannot edit operators or create data flows in the Execution View. You can only perform these tasks using the Logical View.

The contents of the Execution Unit view are based on the selected configuration. Thus, you can use different code templates for different configurations. For example, if you have two configurations, Development and QA, you can use one set of code templates for Development and another for QA.

### Execution View Menu and Toolbars

When you select the Execution View tab, the Execution menu is displayed in the Design Center and an Execution toolbar is displayed at the top of the Mapping Editor canvas. Use the options in the Execution menu or the Execution toolbar to:

- Create and delete execution units
- Define default execution units
- Associate code templates with execution units

### Mapping Editor Display Options

You can control how the editor displays the mappings on the canvas by selecting **Graph** from the menu bar and then selecting **Options**. Warehouse Builder displays the Options dialog box that enables you to set display options for the Mapping Editor canvas.

The Options dialog box contains the following options. You can either select or deselect any of these options.

- **Input Connector:** Select this option to display an arrow icon on the left of attributes that you can use as input attributes.
- **Key Indicator:** Select this option to display a key icon to the left of the attribute that is a foreign key attribute in an operator.
- **Data Type:** Select this option to display the data type of attributes in all operators.
- **Output Connector:** Select this option to display an arrow icon on the right of attributes that you can use as output attributes.
- **Enable Horizontal Scrolling:** Select this option to enable horizontal scrolling for operators.
- **Automatic Layout:** Select this option to use an automatic layout for the mapping.

## Example: Defining a Simple PL/SQL Mapping

This section describes the creation of a basic PL/SQL mapping that loads data from a source table to a target table. The purpose of this example is to illustrate the use of mappings and help you understand the objective achieved by creating mappings. Because the example is very basic, it does not perform any transformation on the source data. However, in a typical data warehousing environment, transformations are an integral part of mappings.

The `SALES` table contains the sales data of an organization. This table is located in the `SRC` schema in an Oracle Database. You need to load this sales data into the target table `SALES_TGT`, located in the `TGT` schema in your data warehouse. Both source and target tables contain the same number of columns with the same column names.

You define a PL/SQL mapping that defines how data from the source table is loaded into the target table.

**To define a mapping that loads data from a source Oracle Database table to a target Oracle Database table:**

1. If you have not already done so, in the Projects Navigator, create an Oracle module corresponding to the source. This module, called `SRC_MOD`, is associated with a location `SRC_LOC` that corresponds to the `SRC` schema. Also, import the `SALES` table into the `SRC_MOD` module.
2. In the Projects Navigator, create the target module whose location corresponds to the data warehouse schema in which your target table is located.

   Create the `WH_TGT` Oracle module, with its associated location `TGT_LOC` corresponding with the `TGT` schema.
3. In the Projects Navigator, expand the `WH_TGT` module.
4. Right-click the Mappings node and select **New Mapping**.

   The Create Mapping dialog box is displayed.
5. Provide the following information about the mapping and click **OK**.
   - **Name:** Enter the name of the mapping.
   - **Description:** Enter an optional description for the mapping.

   The Mapping Editor is displayed. Use this interface to define the data flow between source and target objects.
6. Expand the Tables node under the `SRC_MOD` module.
7. Drag and drop the `SALES` table from the Projects Navigator to the Mapping Editor canvas.

   The operator representing the `SALES` table is added to the canvas. The operator name appears in the upper-left corner. Below the operator name is the name of the group. The name and number of groups depend on the type of operator. Table operators have one group called INOUTGRP1. Below the group, the attribute names and their data types are listed.
8. From the Projects Navigator, drag and drop the `SALES_TGT` table, under the `WH_TGT` module, to the Mapping Editor canvas.

   The operator representing the `SALES_TGT` table is added to the canvas. You can view each attribute name and data type.
9. Connect the attributes of the source table `SALES` to the corresponding attributes in the target table `SALES_TGT`.

   To connect all attributes in the operator, click and hold down your left mouse button on the group INOUTGRP1 of the `SALES` operator, drag, and release the mouse button on the group INOUTGRP1 of the `SALES_TGT` operator.

   The Connect dialog box is displayed.

   > **See Also:** "Connecting Operators, Groups, and Attributes" on page 5-16 for details about connecting attributes

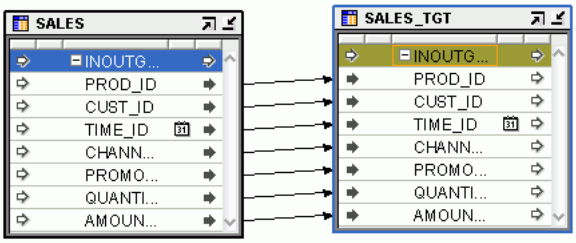
10. In the Connection Options section, select **Match by name of source and target operators** and click **Preview**.

The Connections section displays the association between the source and target attributes. The Source Attribute column lists the source table attributes and the Target Attribute column lists the attributes in the target table to which the source attributes are loaded.

11. Click **OK** to close the Connect dialog box.

    The completed mapping looks as shown in Figure 5–1.

***Figure 5–1 Mapping that Loads Data from Source Table to Target Table***




12. Validate the mapping by selecting the mapping in the Projects Navigator and clicking **Validate**. Or, right-click the mapping in the Projects Navigator and select **Validate**.

    Validation runs tests to verify the metadata definitions and configuration parameters in the mapping. Resolve validation errors, if any.

13. Generate the mapping by selecting the mapping in the Projects Navigator and clicking **Generate**. Or, right-click the mapping in the Projects Navigator and select **Generate**.

    Generation creates the scripts that will be used to create the mapping in the target schema. Resolve generation errors, if any, and regenerate the mapping. Warehouse Builder creates the scripts for this mapping.

You have now defined a mapping that extracts data from a source table called `SALES` and loads it into a target table called `SALES_TGT`. The metadata for this mapping is stored in the repository. And, after successful generation, the scripts to create this mapping in the target schema are ready.

To perform ETL and transfer the data from the source table to the target table, you must first deploy the mapping to the target schema, and then execute the mapping as defined in "Starting ETL Jobs" on page 12-9.

## Steps to Perform Extraction, Transformation, and Loading (ETL) Using Mappings

### Before You Begin

First verify that your project contains a target module or a Template Mappings module with a defined location. You must create your mapping in this module.

Also import any existing data you intend to use as sources or targets in the mapping.

**To define mappings that perform ETL:**

1. In the Projects Navigator, define the mapping that contains the logic for performing ETL.

See "Defining Mappings" on page 5-11 for more information about defining mappings and "Creating Code Template (CT) Mappings" on page 7-12 for information about creating CT mappings.

2. In the Mapping Editor, add the required operators to the mapping. Operators enable you to perform ETL.

   See "Adding Operators to Mappings" on page 5-13 for more information about adding operators.

3. Connect the operators in the mapping to define how data from the source objects should be transformed before loading it into the target objects.

   See "Connecting Operators, Groups, and Attributes" on page 5-16 for information about establishing connections between operators.

4. (Optional) Edit the operators in the mapping to set operator, group, or attribute properties. You may need to edit operators to specify how a certain transformation is to be performed. For example, if you use the Filter operator to restrict the rows loaded into the target, edit the Filter operator and specify the condition that should be used to filter rows.

   See "Editing Operators" on page 5-21 for more information about editing operators in mappings.

5. Configure the mapping.

   For PL/SQL mappings, see "Configuring Mappings" on page 5-27. For CT mappings, see "Setting Options for Code Templates in Code Template Mappings" on page 7-28.

6. Validate the mapping by right-clicking the mapping in the Projects Navigator and selecting **Validate**.

   Validation verifies the metadata definitions and configuration parameters of the mapping to check if they conform to the rules defined by Warehouse Builder for mappings.

7. Generate the mapping by right-clicking the mapping in the Projects Navigator and selecting **Generate**.

   Generation uses the metadata definitions and configuration settings to create the code that is used to create the mapping in the target schema.

   When you generate a mapping, the generated code contains comments that help you identify the operator for which the code is generated. This enables you to debug errors that you may encounter when you deploy the mapping.

8. Deploy the mapping to the target schema to create the PL/SQL code generated for the mapping to the target schema.

   For more information about deployment, see "Deploying Objects" on page 12-6.

9. Execute the mapping to extract data from the source table and load it into the target table.

   For more information about executing ETL objects, see "Starting ETL Jobs" on page 12-9.

### Subsequent Steps

After you design a mapping and generate its code, you can create a process flow or proceed directly with deployment followed by execution.

Use process flows to interrelate mappings. For example, you can design a process flow such that the completion of one mapping triggers an email notification and starts another mapping. For more information, see Chapter 8, "Designing Process Flows".

After you design mappings, generate code for them, and deploy them to their targets, you can:

- Execute the mappings immediately as described in "Starting ETL Jobs" on page 12-9.
- Schedule the mappings for later execution as described in "Scheduling ETL Jobs" on page 12-1.
- Create process flows, to orchestrate the execution of one or more mappings, along with other activities as described in "Designing Process Flows" on page 8-1.

## Defining Mappings

A mapping is a Warehouse Builder object that contains the metadata regarding the transformation performed by the mapping. The metadata includes details of the sources from which data is extracted, the targets into which the transformed data is loaded, and the settings used to perform these operations.

### Steps to Define a Mapping

1. In the Projects Navigator, expand the project, the Databases node, the Oracle node, and then the Oracle module in which you want to define the mapping.
2. Right-click the Mappings node and select **New Mapping.**

   Warehouse Builder opens the Create Mapping dialog box.
3. Enter a name and an optional description for the new mapping.

   For rules on naming and describing mappings, see "Rules for Naming Mappings" on page 5-12.
4. Click **OK.**

   Warehouse Builder stores the definition for the mapping and inserts its name in the Projects Navigator. Warehouse Builder opens a Mapping Editor for the mapping and displays the name of the mapping in the title bar.

### Steps to Open a Previously Created Mapping

1. In the Projects Navigator, expand the project, the Databases node, the Oracle node, and then the Oracle module in which the mapping is defined.
2. Expand the Mappings node.
3. Open the Mapping Editor in one of the following ways:
   - Double-click a mapping.
   - Select a mapping and then from the **File** menu, select **Open.**
   - Select a mapping and press **Ctrl + O**.
   - Right-click a mapping, and select **Open.**

   Warehouse Builder displays the Mapping Editor.

**Note:** When you open a mapping that was created using OMB*Plus, although the mapping has multiple operators, it may appear to contain only one operator. To view all the operators, click **Auto Layout** on the toolbar.

## Rules for Naming Mappings

The rules for naming mappings depend on the naming mode that you select in the Naming Preferences section of the Preferences dialog box. Warehouse Builder maintains a business and a physical name for each object in the workspace. The business name is a unique descriptive name that makes sense to a business-level user of the data. The physical name is the name Warehouse Builder uses when generating code.

When you name objects while working in one naming mode, Warehouse Builder creates a default name for the other mode. Therefore, when working in the business name mode, if you assign a name to a mapping that includes mixed cases, special characters and spaces, Warehouse Builder creates a default physical name for you. For example, if you save a mapping with the business name *My Mapping (refer to doc#12345)*, the default physical name is *MY_MAPPING_REFER_TO_DOC#12345*.

When you name or rename objects in the Mapping Editor, use the following naming rules.

### Naming and Describing Mappings

In the physical naming mode, a mapping name can be from 1 to 30 alphanumeric characters, and blank spaces are not allowed. In the business naming mode, the limit is 200 characters and blank spaces and special characters are allowed. In both naming modes, the name should be unique across the project.

**Note for scheduling mappings:** If you intend to schedule the execution of the mapping, there is an additional consideration. For any ETL object that you want to schedule, the limit is 25 characters for physical names and 1995 characters for business names. Follow this additional restriction to enable Warehouse Builder to append, to the mapping name, the suffix *_job* and other internal characters required for deployment and execution.

After you create the mapping definition, you can view its physical and business name in the Property Inspector.

Edit the description of the mapping as necessary. The description can be up to 4,000 alphanumeric characters and can contain blank spaces.

### Rules for Naming Attributes and Groups

You can rename groups and attributes independent of their sources. Attribute and group names are logical. Although attribute names of the object are often the same as the attribute names of the operator to which they are bound, their properties remain independent of each other.

### Rules for Naming Operators

Business names for the operators must meet the following requirements:

- The length of the operator name can be any string of 200 characters.
- The operator name must be unique within its parent group. The parent group could be either a mapping or its parent pluggable mapping container.

Physical names for operators must meet the following requirements:

- The length of the operator name must be between 1 and 30 characters.
- The operator name must be unique within its parent group. The parent group could be either a mapping or its parent pluggable mapping container.
- The operator name must conform to the syntax rules for basic elements as defined in the *Oracle Database SQL Language Reference*.

In addition to physical and business names, some operators also have bound names. Every operator associated with a workspace object has a bound name. During code generation, Warehouse Builder uses the bound name to reference the operator to its workspace object. Bound names have the following characteristics:

- Bound names need not be unique.
- Bound names must conform to the general Warehouse Builder physical naming rules, except if the object was imported and contains restricted characters such as spaces.
- Typically, you do not change bound names directly. Instead, you change these by synchronizing with a workspace object outside the mapping.
- In physical naming mode, when you modify the physical name of an operator attribute, Warehouse Builder propagates the new physical name as the bound name when you synchronize.

## Adding Operators to Mappings

Operators enable you to perform data transformations. Some operators are bound to workspace objects while others are not. Many operators that represent built-in transformations (such as Joiner, Filter, and Aggregator) do not directly refer to underlying workspace objects, and therefore are not bound. Other operators, such as Table, View, Dimension, and Cube, do refer to objects in the workspace, and therefore can be bound.

As a general rule, when you add a data source or target operator such as a Table operator to a mapping, that operator refers to a table defined in the workspace, although it is not the same object as the table itself. Source and target operators in mappings are said to be *bound* to underlying objects in the workspace. Note that it is possible for multiple operators in a single mapping to be bound to the same underlying object. For example, to use a table `EMP` as a source for two different operations in a mapping, you can add two table operators named EMP_1 and EMP_2 to the mapping and bind them to the same underlying `EMP` table in the workspace.

To distinguish between the two versions of operators, this chapter refers to objects in the workspace either generically as *workspace objects* or specifically as *workspace tables, workspace views,* and so on. This chapter refers to operators in the mapping as *Table operators, View operators,* and so on. Therefore, when you add a dimension to a mapping, refer to the dimension in the mapping as the *Dimension operator* and refer to the dimension in the workspace as the *workspace dimension.*

Warehouse Builder maintains separate workspace objects for some operators. This enables you to work on these objects independently. You can modify the workspace object without affecting the logic of the mapping. After modification, you can decide how to synchronize the discrepancy between the workspace object and its corresponding operator. This provides maximum flexibility during your warehouse design.

For example, when you reimport a new metadata definition for the workspace table, you may want to propagate those changes to the Table operator in the mapping. Conversely, as you make changes to a Table operator in a mapping, you may want to propagate those changes back to its associated workspace table. You can accomplish these tasks by a process known as *synchronizing*.

> **See Also:** "Synchronizing Operators and Workspace Objects" on page 5-28 for more information about synchronizing mapping operators and workspace objects

### Operators that Bind to Workspace Objects

The operators that you can bind to associated objects in the workspace are as follows:

Construct Object
Cube
Dimension
Expand Object
External Table
Flat File
Lookup
Materialized View
Pluggable Mapping
Pre-Mapping Process
Post-Mapping Process
Queue
Sequence
Table Function
Table
Transformation
Varray Iterator
View

**To add an operator to a mapping:**

1. Open the Mapping Editor.
2. From the Component Palette, drag an operator icon and drop it onto the canvas. Alternatively, from the Graph menu, select **Add**, the type of operator you want to add, and then the operator.

   If the Component Palette is not displayed, select **Component Palette** from the View menu.

   If you select an operator that you can bind to a workspace object, the Mapping Editor displays the **Add** ***operator_name*** **Operator** dialog box. For details on how to use this dialog box, see "Using the Add Operator Dialog Box to Add Operators" on page 5-15.

   If you select an operator that you cannot bind to a workspace object, Warehouse Builder may display a wizard or dialog box to help you create the operator.
3. Follow any prompts that are displayed by Warehouse Builder and click **OK.**

   The Mapping Editor displays the operator maximized on the canvas. The operator name appears in the upper-left corner. You can view each attribute name and data type.

If you want to minimize the operator, click the arrow in the upper-right corner and the Mapping Editor displays the operator as an icon on the canvas. To maximize the operator, double-click the operator on the canvas.

## Using the Add Operator Dialog Box to Add Operators

The Add Operator dialog box enables you to add operators to a mapping. When you add an operator that you can bind to a workspace object, the Mapping Editor displays the **Add *operator_name* Operator** dialog box.

Select one of the following options on this dialog box:

- Create Unbound Operator with No Attributes
- Select from Existing Repository Object and Bind

### Create Unbound Operator with No Attributes

Select **Create unbound operator with no attributes** to define a new workspace object that is not bound to a workspace object, such as a new staging area table or a new target table.

In the New Operator Name field, enter a name for the new operator. Warehouse Builder displays the operator on the canvas without any attributes.

You can now add and define attributes for the operator as described in "Editing Operators" on page 5-21. Next, to create the new workspace object in a target module, right-click the operator and select **Create and Bind**.

For an example of how to use this option in a mapping design, see "Example: Using the Mapping Editor to Create Staging Area Tables" on page 5-37.

### Select from Existing Repository Object and Bind

Click **Select from existing repository object and bind** to add an operator based on an existing workspace object. The object may have been previously defined or imported into the workspace.

Either type the prefix to search for the object or select from the displayed list of objects within the selected module.

To select multiple items, press the Ctrl key as you click each item. To select a group of items located in a series, click the first object in your selection range, press the Shift key, and then click the last object.

You can add operators based on workspace objects within the same module as the mapping or from other modules. If you select a workspace object from another module, then the Mapping Editor creates a connector, if one does not already exist. The connector establishes a path for moving data between the mapping location and the location of the workspace object.

## Using Pseudocolumns ROWID and ROWNUM in Mappings

You can use the pseudocolumns `ROWID` and `ROWNUM` in mappings. The `ROWNUM` pseudocolumn returns a number indicating the order in which a row was selected from a table. The `ROWID` pseudocolumn returns the rowid (binary address) of a row in a database table.

You can use the `ROWID` and `ROWNUM` pseudocolumns in Table, View, and Materialized View operators in a mapping. These operators contain an additional column called COLUMN USAGE that is used to identify attributes used as `ROWID` or `ROWNUM`. For

normal attributes, this column defaults to TABLE USAGE. To use an attribute for `ROWID` or `ROWNUM` values, set the COLUMN USAGE to ROWID or ROWNUM respectively.

You can map a ROWID column to any attribute of data type `ROWID`, `UROWID`, or `VARCHAR2`. You can map ROWNUM column to an attribute of data type `NUMBER` or to any other data type that allows implicit conversion from `NUMBER`.

Note that `ROWID` and `ROWNUM` pseudocolumns are not displayed in the object editors since they are not real columns.

## Connecting Operators, Groups, and Attributes

After you select mapping source operators, operators that transform data, and target operators, you are ready to connect them. Data flow connections graphically represent how the data flows from a source, through operators, and to a target. The Mapping Connection dialog box assists you in creating data flows between operators.

You can connect operators by any of the following methods:

- **Connecting Operators**: Define criteria for connecting groups between two operators.
- **Connecting Groups**: Define criteria for connecting all the attributes between two groups.
- **Connecting Attributes**: Connect individual operator attributes to each other, one at a time.
- **Using an Operator Wizard:** For operators such as the Pivot operator and Name and Address operator, you can use the wizard to define data flow connections.
- **Using the Mapping Connection Dialog Box:** Define criteria for connecting operators, groups, or attributes.

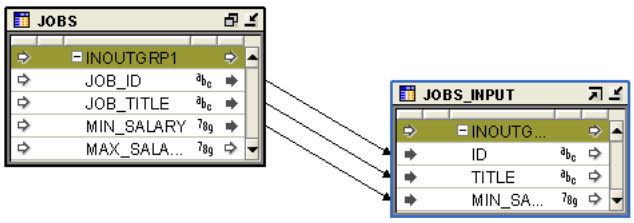
  To display the Mapping Connection dialog box, right-click an operator, group, or attribute, select **Connect To** and then the name of the operator to which you want to establish a connection. The Mapping Connections dialog box is displayed.

  For more information about using this dialog box, see "Using the Mapping Connection Dialog Box" on page 5-18.

After you connect operators, data flow connections are displayed between the connected attributes.

Figure 5–2 displays a mapping with attributes connected.

***Figure 5–2 Connected Operators in a Mapping***



## Connecting Operators

You can connect one operator to another if there are no existing connections between the operators. Both of the operators that you want to connect must be displayed in their icon form.

You can also connect from a group to an operator. Hold down the left-mouse button on the group, drag and then drop on the title of the operator.

**To connect one operator to another:**

1. Select the operator from which you want to establish a connection.
2. Click and hold down the left mouse button while the pointer is positioned over the operator icon.
3. Drag the mouse away from the operator and toward the operator icon to which you want to establish a connection.

   As you drag, a line appears indicating the connection.
4. Release the mouse button over the target operator.

   The Mapping Connection dialog box is displayed. Use this dialog box to specify connections between groups and attributes within these groups as described in "Using the Mapping Connection Dialog Box" on page 5-18.

## Connecting Groups

When you connect groups, the Mapping Editor assists you by either automatically copying the attributes or prompts you for more information as described in "Using the Mapping Connection Dialog Box" on page 5-18.

**To connect one group to another:**

1. Select the group from which you want to establish a connection.
2. Click and hold down the left mouse button while the pointer is positioned over the group.
3. Drag the mouse away from the group and towards the group to which you want to establish a connection.

   As you drag, a line appears indicating the connection.
4. Release the mouse button over the target group.

   If you connect from an operator group to a target group containing attributes, the Mapping Connection Dialog Box is displayed. Use this dialog box to specify connections between attributes as described in "Using the Mapping Connection Dialog Box" on page 5-18.

   If you connect from one operator group to a target group with no existing attributes, the Mapping Editor automatically copies the attributes and connects the attributes. This is useful for designing mappings such as the one shown in "Example: Using the Mapping Editor to Create Staging Area Tables" on page 5-37.

## Connecting Attributes

You can draw a line from a single output attribute of one operator to a single input attribute of another operator.

**To connect attributes:**

1. Click and hold down the left mouse button while the pointer is positioned over an output attribute.
2. Drag the mouse away from the output attribute and toward the input attribute to which you want data to flow.

   As you drag the mouse, a line appears on the Mapping Editor canvas to indicate a connection.
3. Release the mouse over the input attribute.
4. Repeat Steps 1 through 3 until you create all the required data flow connections.

You can also select more than one attribute in Step 1. To select more than one attribute, hold down the Ctrl key and select attributes by clicking them. If you select multiple source attributes, then you can only release the mouse over a group and not over an output attribute. The Mapping Connection dialog box is displayed. Use this dialog box to define the data flow between the source attributes and target attributes.

As you connect attributes, remember the following rules:

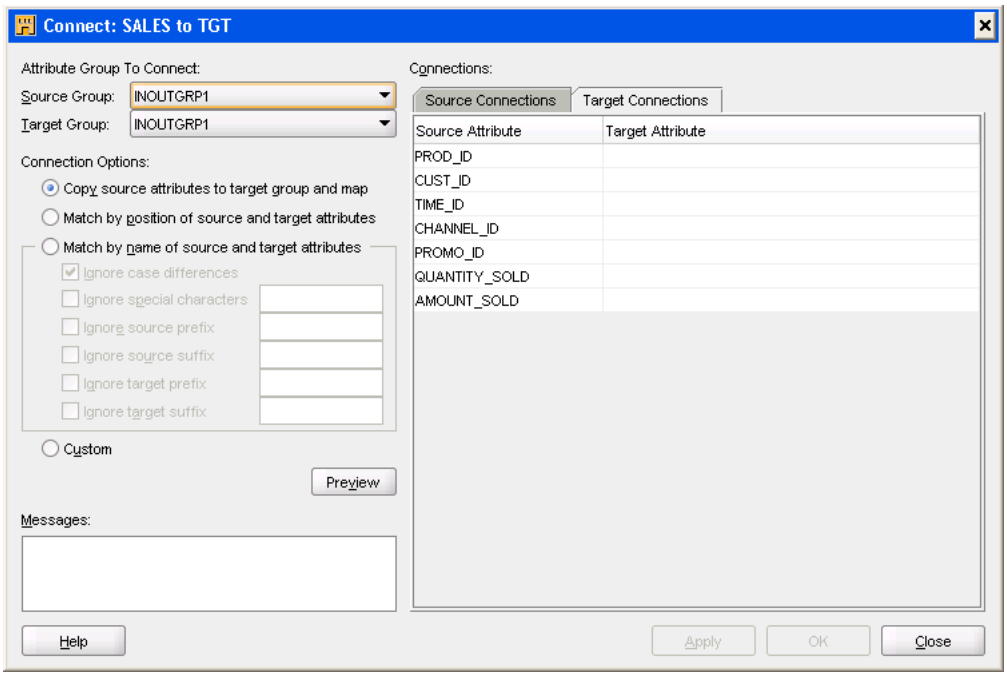
- You cannot connect to the same input or inout attribute twice.
- You cannot connect attributes within the same operator.
- You cannot connect out of an input-only attribute nor can you connect into an output-only attribute.
- You cannot connect operators in such a way as to contradict an established cardinality. Instead, use a Joiner operator.

## Using the Mapping Connection Dialog Box

The Mapping Connection dialog box enables you to define connections between operators in the mapping. Typically, mappings contain numerous operators that implement the complex transformation logic required for your data loading process. The operators that you want to connect may be situated far away from each other on the mapping and thus require scrolling. Warehouse Builder provides an efficient method to connect operators, groups, and attributes by using the Mapping Connection dialog box.

Figure 5–3 displays the Mapping Connection Dialog box.

*Figure 5–3 Mapping Connection Dialog Box*




Complete the following sections to define connections between operators:

- (Optional) Attribute Group to Connect
- Connection Options
- Messages
- Connections

## Attribute Group to Connect

Use this section to select the source and target groups between which you want to establish connections. This section is displayed only if you try to connect a source operator to a target group, a source group to a target operator, or a source operator to a target operator.

**Source Group** The Source Group corresponds to the group on the source operator from which data flows. The Source Group list contains all the output groups and input/output groups in the source operator. Select the group from which you want to create a data flow.

**Target Group** The Target Group corresponds to the group on the operator to which data flows. The Target Group list contains the input groups and input/output groups in the target operator. Select the group to which you want to create a data flow.

Once you select the Source Group and Target Group, you can specify connections between attributes in the source and target groups. Thus, you can establish data flows between all groups of the source and target operators at the same time.

---

**Note:** If you have created any connections for the selected source or target group, when you select a different group, Warehouse Builder displays a warning asking if you want to save the current changes.

---

## Connection Options

The Connection Options section enables you to use different criteria to automatically connect all the source attributes to the target attributes.

Select one of the following options for connecting attributes:

- Copy Source Attributes to Target Group and Match
- Match By Position of Source and Target Attributes
- Match By Name of Source and Target Attributes

After you select the option that you use to connect attributes in the groups, click **Preview** to view the mapping between the source and target attributes in the Connections section. Review the mappings and click **OK** once you are satisfied that the mappings are what you wanted.

### Copy Source Attributes to Target Group and Match

Use this option to copy source attributes to a target group that already contains attributes. The Mapping Editor connects from the source attributes to the new target attributes based on the selections that you make in the Connect Operators dialog box. Warehouse Builder does not perform this operation on target groups that do not accept new input attributes, such as dimension and cube target operators.

### Match By Position of Source and Target Attributes

Use this option to connect existing attributes based on the position of the attributes in their respective groups. The Mapping Editor connects all attributes in order until all attributes of the target are matched. If the source operator contains more attributes than the target, then the remaining source attributes are left unconnected.

### Match By Name of Source and Target Attributes

Use this option to connect attributes with matching names. By selecting from the list of options, you connect between names that do not match exactly. You can combine the following options:

- **Ignore case differences:** Considers the same character in lower-case and upper-case a match. For example, the attributes FIRST_NAME and First_Name match.
- **Ignore special characters:** Specify characters to ignore during the matching process. For example, if you specify a hyphen and underscore, the attributes FIRST_NAME, FIRST-NAME, and FIRSTNAME all match.
- **Ignore source prefix, Ignore source suffix, Ignore target prefix, Ignore target suffix:** Specify prefixes and suffixes to ignore during matching. For example, if you select Ignore source prefix and enter USER_ into the text field, then the source attribute USER_FIRST_NAME matches the target attribute FIRST_NAME.

## Messages

This section displays any informational messages that result from previewing the connection options. Information such as certain source or target attributes not connected due to unresolved conflicts is displayed in this section.

## Connections

The Connections section displays the connections between the source attributes belonging to the Source Group and the target attributes belonging to the Target

Group. Any existing connections between attributes of the Source Group and Target Group are displayed in this section.

This section contains two tabs: Source Connections and Target Connections. Both tabs display a spreadsheet containing the Source Attribute and Target Attribute columns. The Source Attribute column lists all attributes, or the attributes selected on the canvas, for the group selected as the Source Group. The Target Attribute column lists all the attributes of the group selected as the Target Group. Any changes that you make on the Source Connections tab or the Target Connections tab are immediately reflected in the other tab.

#### Source Connections Tab

The Source Connections tab enables you to quickly establish connections from the Source Group. The Target Attribute column on this tab lists the attributes from the Target Group. Use this tab to specify the source attribute from which each target attribute is connected. For each target attribute, map zero or one source attribute. To connect a particular source attribute to the listed target attribute, for each target attribute, enter the name of the source attribute in the corresponding Source Attribute column.

As you begin typing an attribute name, Warehouse Builder displays a list containing the source attributes whose names begin with the letters you type. If you see the source attribute that you want to connect in this list, then select the attribute by clicking it. You can use wild cards such as * and ? to search for the source attributes from which you want to create a data flow. You can also sort the columns listed under Target Attribute column. When the attribute name contains the space or comma characters, use double quotes to quote the name of the source attribute.

#### Target Connections Tab

The Target Connections tab enables you to quickly establish connections to the Target Group. The Source Attribute column displays the list of attributes from the Source Group. Use this tab to specify the source attributes from which each target attribute is connected. For each source attribute, enter the name of one or more target attributes in the corresponding Target Attribute column. To connect a source attribute to more than one target attributes, type the names of the source attributes separated by a comma in the Target Attribute column.

As you begin typing an attribute name, Warehouse Builder displays a list containing the target attributes whose names begin with the letters you type. If the target attribute that you want to connect to is displayed in this list, then select the attribute by clicking it. You can also use wild cards such as * and ? to search for target attributes to which you want to create a data flow. You can sort the columns listed under Source Attribute column.

## Editing Operators

Each operator has an editor associated with it. Use the operator editor to specify general and structural information for operators, groups, and attributes. In the operator editor you can add, remove, or rename groups and attributes. You can also rename an operator.

For attributes that can have an expression associated with them, such as attributes in the output group of a Constant, Expression, or Aggregator operator, you can also edit the expression specified.

Editing operators is different from assigning loading properties and conditional behaviors. To specify loading properties and conditional behaviors, use the properties windows as described in "Configuring Mappings" on page 5-27.

**To edit an operator, group, or attribute:**

1. Select an operator from the Mapping Editor canvas.

   Or select any group or attribute within an operator.

2. Right-click and select **Open Details.**

   The Mapping Editor displays the operator editor with the Name Tab, Groups Tab, and Input and Output Tabs for each type of group in the operator.

   Some operators include additional tabs. For example, the Match Merge operator includes tabs for defining Match rules and Merge rules.

3. Follow the prompts on each tab and click **OK** when you are finished.

## Name Tab

The Name tab displays the operator name and an optional description. You can rename the operator and add a description. Name the operator according to the conventions listed in "Rules for Naming Mappings" on page 5-12.

## Groups Tab

Edit group information on the Groups tab.

Each group has a name, direction, and optional description. You can rename groups for most operators but cannot change group direction for any of the operators. A group can have one of these directions: Input, Output, Input/Output.

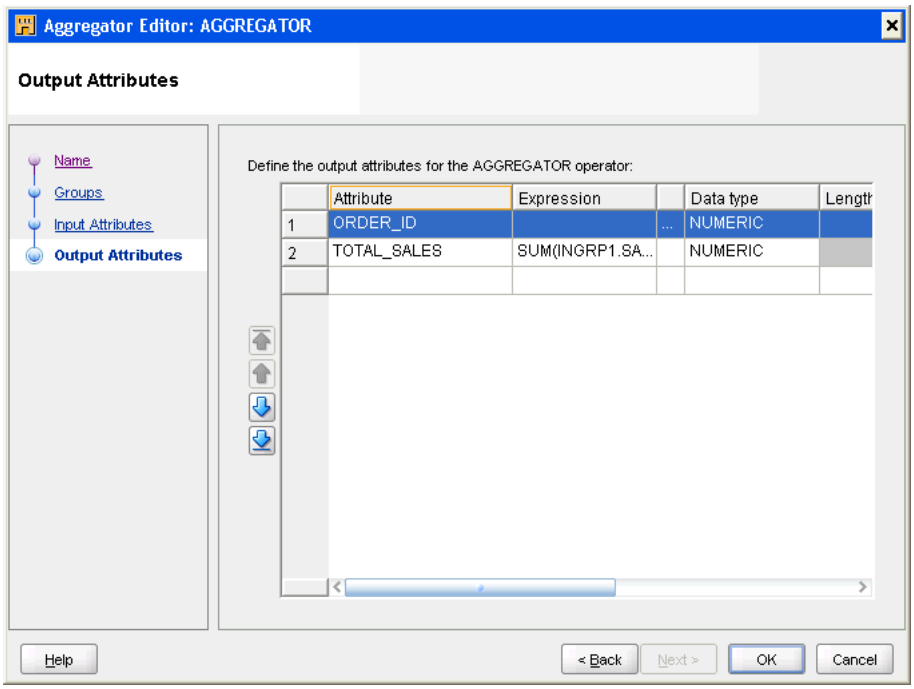
Depending on the operator, you can add and remove groups from the Groups tab. For example, you add input groups to Joiners and output groups to Splitters.

## Input and Output Tabs

The operator editor displays a tab for each type of group displayed on the Groups tab. Each of these tabs displays the attribute name, data type, length, precision, scale, seconds precision, and optional description. Certain operators such as the Table or View operators have only the Input/Output tab, instead of separate Input and Output tabs. Edit attribute information on these tabs.

Figure 5–4 shows an Output Attributes tab on the operator editor. In this example, the operator is an Aggregator, with separate Input and Output tabs.

*Figure 5–4 Output Tab on the Operator Editor*




The tab contains a table that you can use to define output attributes. Each row on this tab represents an attribute. The Mapping Editor disables properties that you cannot edit. For example, if the data type is `NUMBER`, you can edit the precision and scale but not the length.

You can add, remove, and edit attributes. To add an attribute, click on the Attribute column of an empty row, enter the attribute name and then provide the other attribute details such as data type, length, and description. To delete an attribute, right-click the grey cell to the left of the attribute and select **Remove**.

To assign correct values for data type, length, precision, and scale in an attribute, follow PL/SQL rules. When you synchronize the operator, Warehouse Builder checks the attributes based on SQL rules.

You can also change the order of the attributes listed in the Input and Output tabs. Select the row representing the attribute and use the arrow buttons to the left of the attributes to reorder attributes in a group. Alternatively, hold down the left-mouse button until you see a cross-hair and then drag the row to the position you want.

### Associating Expressions with Operator Attributes

Attributes in certain operators such as Expression, Joiner, Aggregator, and Lookup can have an expression associated with them. For such attributes, use the Expression column of the attribute to specify the expression used to create the attribute value. You can directly enter the expression in the Expression column. Figure 5–4 displays the Expression column for an output attribute in the Aggregator operator. To use the Expression Builder interface to define your expression, click the Ellipsis button to the right of the Expression column.

For example, in an Aggregator operator, you create output attributes that store the aggregated source columns. Use the Expression column for an output attribute to specify the expression used to create the attribute value.

## Using Display Sets

A display set is a graphical representation of a subset of attributes. Use display sets to limit the number of attributes visible in an operator and simplify the display of a complex mapping.

By default, operators contain three predefined display sets: ALL, MAPPED, and UNMAPPED. Table 5–1 describes the default display sets.

***Table 5–1 Default Sets***

| Display Set | Description |
|---|---|
| ALL | Includes all attributes in an operator |
| MAPPED | Includes only those attributes in an operator that are connected to another operator |
| UNMAPPED | Includes only those attributes that are not connected to other attributes |

### Defining Display Sets

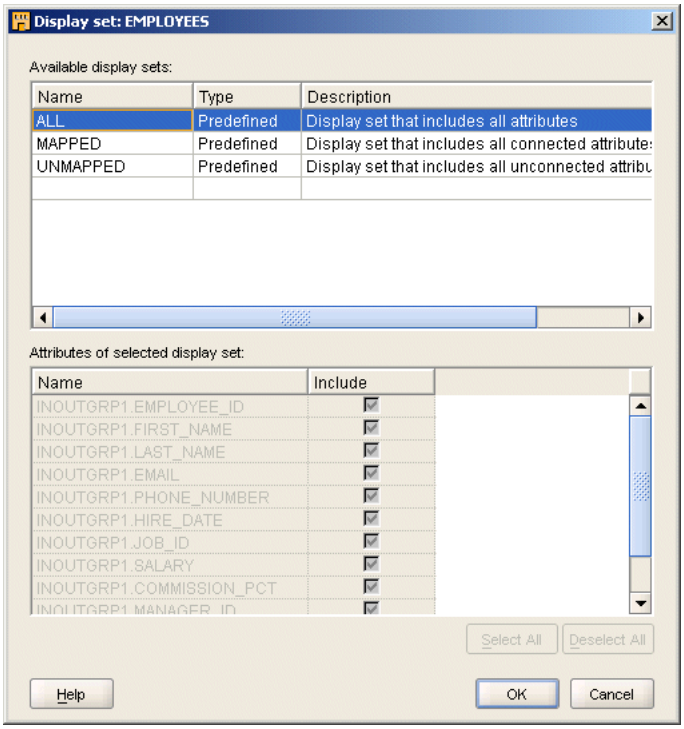
You can define display sets for any operator in a mapping.

**To define a display set:**

1. Right-click an operator, and select **Define Display Set**.

   The Display Set dialog box is displayed as shown in Figure 5–5.

***Figure 5–5 Display Set Dialog Box***




2. Click the row below UNMAPPED and enter a name and description for the new display set.

3. All available attributes for the operator appear in section called Attributes of selected display set. The Type column is automatically set to User defined.

You cannot edit or delete a Predefined attribute set.

4. In the Include column, select each attribute that you want to include in the display set.

   Click **Select All** to include all attributes and **Deselect All** to exclude all the attributes.

5. Click **OK.**

   The group for the operator now lists only those attributes contained within the Attribute Set selected for display.

### Selecting a Display Set

If a group contains more than one display set, you can select a different display set from a list using the Graph menu.

**To select a display set:**

1. Right-click a group in an operator.
2. Click **Select Display Set** and select the desired display set.

## Setting Mapping Properties

After you define a mapping, you can use the Property Inspector to set properties for the mapping.

You can set the following properties:

- **Mapping Properties:** Properties that affect the entire mapping. For example, you can set the Target Load Order parameter defines the order in which targets are loaded when the mapping is executed.
- **Operator Properties:** Properties that affect the operator as a whole. The properties you can set depend upon the operator type. For example, the steps for using Oracle source and target operators differ from the steps for using flat file source and target operators.
- **Group Properties:** Properties that affect a group of attributes. Most operators do not have properties for their groups. Examples of operators that do have group properties include the Splitter operator and the Deduplicator operator.
- **Attribute Properties:** Properties that pertain to attributes in source and target operators. Examples of attribute properties are data type, precision, and scale.

### Setting Operator, Group, and Attribute Properties

When you select an operator, group, or attribute on the Mapping Editor canvas, its associated properties are displayed in the Property Inspector. Set values for the required properties using the Property Inspector. The properties that you can set are documented in the chapters that discuss the operators.

**See Also:**

- "Source and Target Operators" on page 25-1
- "Data Flow Operators" on page 26-1
- "Using the Name and Address Operator to Cleanse and Correct Name and Address Data" on page 22-19
- "Using the Match Merge Operator to Eliminate Duplicate Source Records" on page 23-22

### Setting Mapping Properties

To set properties for a mapping, select the Mapping in the Projects Navigator. The Property Inspector displays the properties of the mapping.

You can set values for the following properties: Business Name, Physical Name, Description, Execution Type, and Target Load Order.

Use the Target Load Order configuration parameter to specify the order in which targets in the mapping are loaded as described in "Specifying the Order in Which Target Objects in a Mapping Are Loaded" on page 5-26.

## Specifying the Order in Which Target Objects in a Mapping Are Loaded

If your mapping includes only one target or is a SQL*Loader or ABAP mapping, target load ordering does not apply. Accept the default settings and continue with your mapping design.

When you design a PL/SQL mapping with multiple targets, Warehouse Builder calculates a default ordering for loading the targets. If you define foreign key relationships between targets, then Warehouse Builder creates a default order that loads the parent and then the child. If you do not create foreign key relationships or if a target table has a recursive relationship, then Warehouse Builder assigns a random ordering as the default.

You can override the default load ordering by setting the mapping property Target Load Order. If you make a mistake when reordering the targets, then you can restore the default ordering by selecting the Reset to Default option. Or you can select **Cancel** to discard your changes to the target order.

**To specify the loading order for multiple targets:**

1. Click whitespace in the mapping canvas to view the mapping properties in the Property Inspector.

   If the Property Inspector is not displayed, select **Property Inspector** from the View menu.

2. Go to the Target Load Order property and click the Ellipsis button on the right of this property.

   Warehouse Builder displays the Target Load Order dialog box in which TARGET2 is listed before TARGET1.

3. To change the loading order, select a target and use the buttons to move the target up or down on the list.

### Reset to Default

Use the Reset to Default button to instruct Warehouse Builder to recalculate the target loading order. You may want to recalculate if you made an error reordering the targets

or if you assigned an order and later changed the mapping design such that the original order became invalid.

# Configuring Mappings

After you define mappings, you can configure them to specify the physical properties of the mapping and the operators contained in the mapping. Configuring a mapping enables you to control the code generation, so that Warehouse Builder produces optimized code for the mapping and for your particular environment.

## Steps to Configure Mappings

Use the following steps to configure mappings.

1. In the Projects Navigator, right-click the mapping and select **Configure**.

   Warehouse Builder displays the Configuration tab that contains configuration parameters for the mapping.

   This tab contains the Deployable, Language, Generation Comments, and Referred Calendar parameters. It also contains the Runtime Parameters and Code Generation Options nodes. Additionally, each operator on the mapping is listed under the node representing the object or operator type. For example, if your mapping contains two tables and a Filter operator, the Table Operators node displays the configuration parameters for the two tables and the Filter Operator node displays the configuration parameters for the Filter operator.

2. Set the **Deployable** parameter to True.

3. Set **Language** to the type of code that you want to generate for the selected mapping.

   The options from which you can choose depend upon the design and use of the operators in the mapping. Warehouse Builder provides the following options: PL/SQL, SQL*PLUS, SQL*Loader, and ABAP (for an SAP source mapping).

4. If you want to schedule the mapping to run based on a previously defined schedule, click the Ellipsis button on the Referred Calendar parameter.

   The Referred Calendar dialog box is displayed. Any schedules created are listed here. Select the schedule that you want to associate with the current mapping.

   For instructions on creating and using schedules, see Chapter 11, "Scheduling ETL Jobs".

5. Expand **Code Generation Options** to enable performance options that optimize the code generated for the mapping.

   For a description of each option, see "Code Generation Options" on page 24-4.

6. (Optional) To define chunking for a PL/SQL mapping, use one of the following:

   - To configure serial chunking, expand **SCD Updates** and set the parameters described in "SCD Updates" on page 24-8.
   - To configure parallel chunking, expand **Chunking Options**, then **Details**, and set the parameters depending upon the method of parallel chunking you plan to use. Also, ensure that serial chunking is not enabled for this mapping.

     **See Also:** "Chunking Options" on page 24-7 for a description of the parallel chunking parameters

If you want to set Runtime parameters corresponding to parallel chunking, after you set values under Chunking Options, close the Mapping Configuration Editor and reopen it to display the parallel chunking-related Runtime parameters.

**Note:** You can perform chunking for PL/SQL mappings only.

7. Expand **Runtime Parameters** to configure your mapping for deployment.

   For a description of each runtime parameter, see "Runtime Parameters" on page 24-1.

8. Go to the node for each operator in the mapping to set their physical properties. The properties displayed under a particular node depend on the type of object. For example, for tables listed under the Table Operators node, you can configure the table parameters listed in "Configuring Tables" on page 2-48.

   **See Also:**

   - "Sources and Targets Reference" on page 24-8 for information about configuring sources and targets in a mapping
   - "Configuring Flat File Operators" on page 24-13 for information about configuring mappings with flat file sources and targets

## Synchronizing Operators and Workspace Objects

Many of the operators that you use in a mapping have corresponding definitions in the Warehouse Builder workspace. This is true of source and target operators such as table and view operators. This is also true of other operators, such as sequence and transformation operators, whose definitions you may want to use across multiple mappings. As you make changes to these operators, you may want to propagate those changes back to the workspace object.

You have the following choices in deciding the direction in which to propagate changes:

- **Synchronizing a Mapping Operator with its Associated Workspace Object**

  This enables you to propagate changes in the definition of a workspace object to the mapping operator that is bound to the workspace object.

  You can also synchronize all the operators in a mapping with their corresponding definitions in the workspace as described in "Synchronizing All Operators in a Mapping" on page 5-30.

- **Synchronizing a Workspace Object with a Mapping Operator**

  This enables you to propagate changes made to a mapping operator to its corresponding workspace definition. You can select a single operator and synchronize it with the definition of a specified workspace object.

Note that synchronizing is different from refreshing. The refresh command ensures that you are up-to-date with changes made by other users in a multiuser environment. Synchronizing matches operators with their corresponding workspace objects.

## Synchronizing a Mapping Operator with its Associated Workspace Object

After you begin using mappings in a production environment, changes may be made to the sources or targets that affect your ETL designs. Typically, the best way to manage these changes is through the Metadata Dependency Manager described in Chapter 14, "Managing Metadata Dependencies". Use the Metadata Dependency Manager to automatically evaluate the impact of changes and to synchronize all affected mappings at one time.

The Mapping Editor enables you to manually synchronize objects as described in this section.

### When Do You Synchronize from a Workspace Object to an Operator?

In the Mapping Editor, you can synchronize from a workspace object to an operator for any of the following reasons:

- **To manually propagate changes:** Propagate changes you made in a workspace object to its associated operator. Changes to the workspace object can include structural changes, attribute name changes, or attribute data type changes.

  To automatically propagate changes in a workspace object across multiple mappings, see Chapter 14, "Managing Metadata Dependencies".

- **To synchronize an operator with a new workspace object:** You can synchronize an operator with a new workspace object if, for example, you migrate mappings from one version of a data warehouse to a newer version and maintain different object definitions for each version.

- **To create a prototype mapping using tables:** When working in the design environment, you could choose to design the ETL logic using tables. However, for production, you may want to the mappings to source other workspace object types such as views, materialized views, or cubes.

### Synchronizing Physical and Business Names

While synchronizing from a workspace object to an operator, you can specify if the physical and business names of the operator, groups, and attributes should be synchronized. By default, the bound name of the operator, groups and attributes will be derived from the physical name of the corresponding workspace object. The synchronization behavior is controlled by a preference called Synchronize Name Changes listed under the Naming node of the Preferences dialog box.

If you select the Synchronize Name Changes preference under the Naming node of the Preferences dialog box, a synchronize operation on any operator synchronizes the physical and business names of the operator, groups, and attributes. If you deselect the Synchronize Name Changes preference, then the physical and business names of the operator, groups, and attributes are not synchronized.

> **See Also:** *Oracle Warehouse Builder Concepts* for more information about Warehouse Builder preferences

### Steps to Synchronize from a Workspace Object to an Operator

Use the following steps to synchronize an operator with the workspace object to which it is bound.

1. On the Mapping Editor canvas, select the operator that you want to synchronize. When the operator is displayed in maximized form, select the operator by clicking the operator name.
2. Right-click and select **Synchronize**.

The Synchronize dialog box is displayed.

3. In the Repository Object with which to Synchronize field, select the workspace object with which you want to synchronize the mapping operator.

   By default, the workspace object to which the mapping operator was originally bound is displayed in this field.

4. Under Direction of Synchronization, select **Inbound**.

5. In the Matching Strategy field, select the matching strategy to be used during synchronization.

   For more information about the matching strategy, see "Matching Strategies" on page 5-32.

6. In the Synchronize strategy field, select the synchronization strategy.

   Select Replace to replace the mapping operator definition with the workspace object definition. Select Merge to add any new metadata definitions and overwrite existing metadata definitions if they differ from the ones in the workspace object.

7. Click **OK** to complete the synchronization.

### Synchronizing All Operators in a Mapping

You can synchronize all operators in a mapping with their bound workspace objects using a single step. To do this, open the mapping containing the operators to be synchronized. With the Mapping Editor canvas as the active panel, from the Edit menu, select **Synchronize All**. The Synchronize All panel is displayed. Use this panel to define synchronization options.

The Synchronize All panel displays one row for each mapping operator that is bound to a workspace object. Select the box to the left of all the object names which you want to synchronize with their bound workspace objects. For each operator, specify values in the following columns:

- **From Repository:** Displays the workspace object to which the mapping operator is bound.

  To modify the workspace object to which an operator is bound, click the Ellipsis button to the right of the workspace object name. The Source dialog box is displayed. Click the list on this page to select the new workspace object and click **OK**.

- **To Mapping:** Displays the name of the mapping operator. This field is not editable.

- **Matching Strategy:** Select the matching strategy used while synchronizing operators. For more information about matching strategies, see "Matching Strategies" on page 5-32.

- **Synchronize Strategy:** Select the synchronization strategy used while synchronizing operators. You can select Replace or Merge as the synchronize strategy.

## Synchronizing a Workspace Object with a Mapping Operator

As you make changes to operators in a mapping, you may want to propagate those changes to a workspace object. By synchronizing, you can propagate changes from the following operators: Table, View, Materialized View, Transformation, and Flat File.

Synchronize from the operator to a workspace object for any of the following reasons:

- **To propagate changes:** Propagate changes that you made in an operator to its associated workspace object. When you rename the business name for an operator or attribute, Warehouse Builder propagates the first 30 characters of the business name as the bound name.
- **To replace workspace objects:** Synchronize to replace an existing workspace object.

Synchronizing from an operator has no impact on the dependent relationship between other operators and the workspace object. Table 5–2 lists the operators from which you can synchronize.

***Table 5–2 Outbound Synchronize Operators***

| Mapping Object | Create Workspace Objects | Propagate Changes | Replace Workspace Objects | Notes |
|---|---|---|---|---|
| External Table | Yes | Yes | Yes | Updates the workspace external table only and not the flat file associated with the external table. See *Oracle Warehouse Builder Sources and Targets Guide* for details. |
| Flat File | Yes | Yes | No | Creates a new, comma-delimited flat file for single record type flat files only. |
| Mapping Input Parameter | Yes | Yes | Yes | Copies input attributes and data types as input parameters |
| Mapping Output Parameter | Yes | Yes | Yes | Copies output attributes and data types as return specification for the function |
| Materialized View | Yes | Yes | Yes | Copies attributes and data types as columns |
| Table | Yes | Yes | Yes | Copies attributes and data types as columns. Constraint properties are not copied |
| Transformation | Yes | Yes | Yes | |
| View | Yes | Yes | Yes | Copies attributes and data types as columns |

### Steps to Synchronize a Workspace Object with a Mapping Operator

Use the following steps to synchronize from a mapping operator to a workspace object. Synchronization causes the workspace object to be updated with changes made to the mapping operator after the mapping was created.

1. On the Mapping Editor canvas, select the operator whose changes you want to propagate to the bound workspace object.
2. From the **Edit** menu, select **Synchronize.** Or, right-click the header of the operator and select **Synchronize.**

   The Synchronize Operator dialog box is displayed.
3. In the Repository Object with which to Synchronize field, select the workspace object that should be updated with the mapping operator definition changes.

   By default, Warehouse Builder displays the workspace object to which the mapping operator was initially bound.

4. In the Direction of Synchronization field, select **Outbound**.
5. (Optional) In the Matching strategy field, select the matching strategy used during synchronization. See "Matching Strategies" on page 5-32.
6. (Optional) In the Synchronize Strategy field select the synchronization strategy.

   Select Replace to replace the workspace object definition with the mapping operator definition. Select Merge to add any new metadata definitions and overwrite existing metadata definitions if they differ from the ones in the mapping operator.
7. Click **OK.**

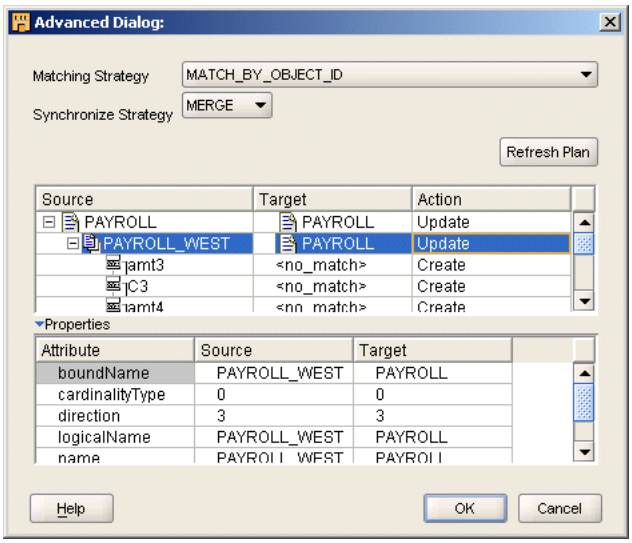
## Advanced Options for Synchronizing

Use the Synchronization Plan dialog box to view and edit the details of how Warehouse Builder synchronizes your selected objects. After you select from the Matching Strategies, click **Refresh Plan** to view the actions that Warehouse Builder takes.

In the context of synchronizing, *source* refers to the object from which to inherit differences and *target* refers to the object to be changed.

For example, in Figure 5–6, the flat file `PAYROLL_WEST` is the source and the Flat File operator `PAYROLL` is the target. Therefore, Warehouse Builder creates new attributes for the `PAYROLL` operator to correspond to fields in the flat file `PAYROLL_WEST`.

***Figure 5–6 Advanced Synchronizing Options***




### Matching Strategies

Set the matching strategy that determines how Warehouse Builder compares an operator to a workspace object. If synchronization introduces changes such as adding or deleting attributes in an operator, then Warehouse Builder refreshes the Mapping Editor. If synchronization removes an operator attribute, then data flow connections to or from the attribute are also removed. If synchronization adds an operator attribute, then the Mapping Editor displays the new attributes at the end of the operator. Data flow connections between matched attributes are preserved. If you rename an attribute in the source object, then this is interpreted as if the attribute were deleted and a new attribute added.

You can specify the following strategies for synchronizing an object in a mapping:

- Match by Object Identifier
- Match by Bound Name
- Match by Position

#### Match by Object Identifier

This strategy compares the unique object identifier of an operator attribute with that of a workspace object. The Match by object identifier is not available for synchronizing an operator and workspace object of different types, such as a View operator and a workspace table.

Use this strategy if you want the target object to be consistent with changes to the source object and if you want to maintain separate business names despite changes to physical names in the target object.

Warehouse Builder removes attributes from the target object that do not correspond to attributes in the source object. This can occur when an attribute is added to or removed from the source object.

#### Match by Bound Name

This strategy matches the bound names of the operator attributes to the physical names of the workspace object attributes. Matching is case-sensitive.

Use this strategy if you want bound names to be consistent with physical names in the workspace object. You can also use this strategy with a different workspace object if there are changes in the workspace object that would change the structure of the operator.

Warehouse Builder removes attributes of the operator that cannot be matched with those of the workspace object. Attributes of the selected workspace object that cannot be matched with those of the operator are added as new attributes to the operator. Because bound names are read-only after you have bound an operator to a workspace object, you cannot manipulate the bound names to achieve a different match result.

#### Match by Position

This strategy matches operator attributes with columns, fields, or parameters of the selected workspace object by position. The first attribute of the operator is synchronized with the first attribute of the workspace object, the second with the second, and so on.

Use this strategy to synchronize an operator with a different workspace object and to preserve the names of the attributes in the operator. This strategy is most effective when the only changes to the workspace object are the addition of extra columns, fields, or parameters at the end of the object.

If the target object has more attributes than the source object, then Warehouse Builder removes the excess attributes. If the source object has more attributes than the target object, then Warehouse Builder adds the excess attributes as new attributes.

## Example: Using a Mapping to Load Transaction Data

### Scenario

Your company records all its transactions as they occur, resulting in inserts, updates, and deletes, in a flat file called `record.csv`. These transactions must be processed in

the exact order they were stored. For example, if an order was first placed, then updated, then canceled and reentered, this transaction must be processed exactly in the same order.

An example data set of the source file `record.csv` is defined as:

```
Action,DateTime,Key,Name,Desc
I,71520031200,ABC,ProdABC,Product ABC
I,71520031201,CDE,ProdCDE,Product CDE
I,71520031202,XYZ,ProdXYZ,Product XYZ
U,71620031200,ABC,ProdABC,Product ABC with option
D,71620032300,ABC,ProdABC,Product ABC with option
I,71720031200,ABC,ProdABC,Former ProdABC reintroduced
U,71720031201,XYZ,ProdXYZ,Rename XYZ
```

You want to load the data into a target table such as the following:

```
SRC_TIMESTA KEY NAME    DESCRIPTION
----------- --- ------- ---------------------------
71520031201 CDE ProdCDE Product CDE
71720031201 XYZ ProdXYZ Rename XYZ
71720031200 ABC ProdABC Former ProdABC reintroduced
```

You must create ETL logic to load transaction data in a particular order using Warehouse Builder.

### Solution

Warehouse Builder enables you to design ETL logic and load the data in the exact temporal order in which the transactions were stored at the source. To achieve this result, you design a mapping that orders and conditionally splits the data before loading it into the target. Then, you configure the mapping to generate code in row-based operating mode. In row-based operating mode, Warehouse Builder generates code to process the data row by row using if-then-else constructions, as shown in the following example.

```
CURSOR
   SELECT
       "DATETIME$1"
   FROM
       "JOURNAL_EXT"
   ORDER BY "JOURNAL_EXT"."DATETIME" ASC
LOOP
  IF "ACTION" = 'I' THEN
      INSERT this row
  ELSE
   IF "ACTION" = 'U' THEN
      UPDATE this row
  ELSE
          DELETE FROM
                "TARGET_FOR_JOURNAL_EXT"
END LOOP;
```

This ensures that all consecutive actions are implemented in sequential order and the data is loaded in the order in which the transaction was recorded.

### Step 1: Import and Sample the Source Flat File, record.csv

In this example, the flat file `record.csv` stores all transaction records and a timestamp. Import this flat file from your source system using the Metadata Import

Wizard. Define the metadata for the flat file in Warehouse Builder using the Flat File Sample Wizard.

> **Note:** You can replace this flat file with a regular table if your system is sourced from a table. In this case, skip to Step 3: Design the Mapping.

### Step 2: Create an External Table

To simplify the use of a sampled flat file object in a mapping, create an external table (`JOURNAL_EXT`) using the Create External Table Wizard, based on the flat file imported and sampled in Step 1: Import and Sample the Source Flat File, record.csv.

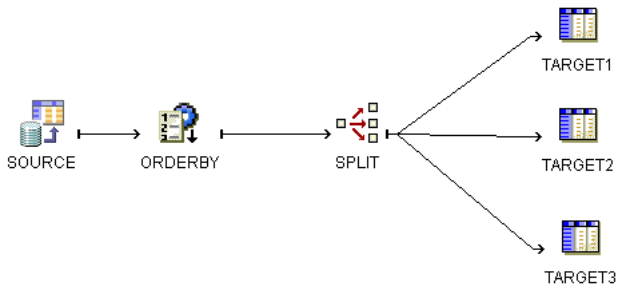
The advantage of using an external table instead of a flat file is that it provides you direct SQL access to the data in your flat file. Hence, there is no need to stage the data.

### Step 3: Design the Mapping

In this mapping, you move the transaction data from an external source, through an operator that orders the data, followed by an operator that conditionally splits the data before loading it into the target table.

Figure 5–7 shows you how the source is ordered and split.

***Figure 5–7 ETL Design***




The Sorter operator enables you to order the data and process the transactions in the exact order in which they were recorded at the source. The Splitter operator enables you to conditionally handle all the inserts, updates, and deletes recorded in the source data by defining a split condition that acts as the if-then-else constraint in the generated code. The data is conditionally split and loaded into the target table. In this mapping, the same target table is used three times to demonstrate this conditional loading. The mapping tables TARGET1, TARGET2, and TARGET3 are all bound to the same workspace table `TARGET`. All the data goes into a single target table.

The following steps show you how to build this mapping.

### Step 4: Create the Mapping

Create a mapping called `LOAD_JOURNAL_EXT` using the Create Mapping dialog box. Warehouse Builder then opens the Mapping Editor where you can build your mapping.

### Step 5: Add an External Table Operator

Drag and drop a mapping external table operator onto the Mapping Editor canvas and bind it to the external table `JOURNAL_EXT`.

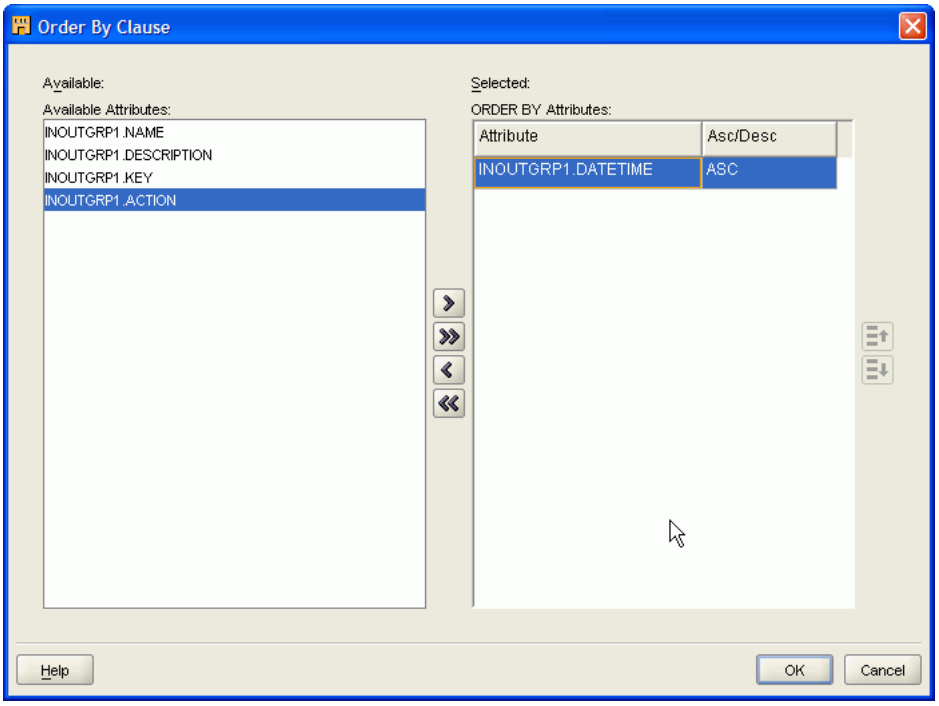
### Step 6: Order the Data

Add the Sorter operator to define an order-by clause that specifies the order in which the transaction data must be loaded into the target.

> **See Also:** "Sorter Operator" on page 26-37 for more information about using the Sorter operator

Figure 5–8 shows you how to order the table based on the timestamp of the transaction data in ascending order.

***Figure 5–8 Order By Clause Dialog Box***

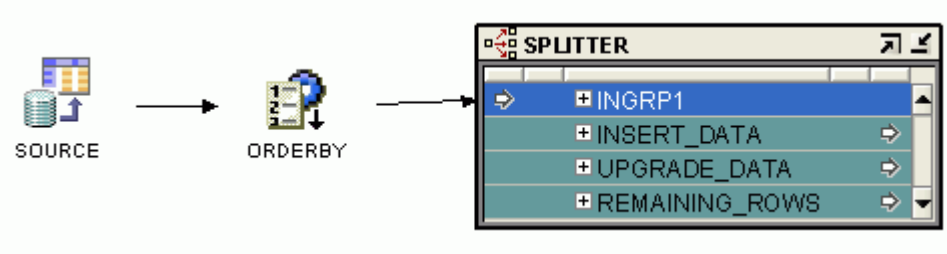



### Step 7: Define a Split Condition

Add the Splitter operator to conditionally split the inserts, updates, and deletes stored in the transaction data. This split condition acts as the if-then-else constraint in the generated code.

Figure 5–9 shows how to join the SOURCE operator with the ORDERBY operator that is linked to the Splitter operator.

***Figure 5–9 Adding the Splitter Operator***



Define the split condition for each type of transaction. For outgroup INSERT_DATA, define the split condition as `INGRP1.ACTION = 'I'`. For UPGRADE_DATA, define the split condition as `INGRP1.ACTION = 'U'`. In Warehouse Builder, the Splitter operator contains a default group called REMAINING_ROWS that automatically handles all Delete ('D') records.

#### Step 8: Define the Target Tables

Use the same workspace target table three times for each type of transaction: once for INSERT_DATA, once for UPDGRADE_DATA, and once for REMAINING_ROWS.

#### Step 9: Configure the Mapping LOAD_JOURNAL_EXT

After you define the mapping, you must configure the mapping to generate code. Because the objective of this example is to process the data strictly in the order in which it was stored, you must select row-based as the default operating mode. In this mode, the data is processed row by row and the insert, update, and delete actions on the target tables occur in the exact order in which the transaction was recorded at the source.

Do not select set-based mode as Warehouse Builder then generates code that creates one statement for all insert transactions, one statement for all update transactions, and a third one for all delete transactions. The code then calls these procedures one after the other, completing one action completely before following up with the next action. For example, it first handles all inserts, then all updates, and then all deletes.

**To configure a mapping for loading transaction data:**

1. From the Projects Navigator, right-click the `LOAD_JOURNAL_EXT` mapping and select **Configure**.
2. Expand the Runtime parameters node and set the Default Operating Mode parameter to **Row based**.

In this example, accept the default value for all other parameters. Validate the mapping before generating the code.

#### Step 10: Generate Code

After you generate the mapping, Warehouse Builder displays the results in the Log window.

When you inspect the code, you will see that Warehouse Builder implements all consecutive actions in row-based mode. This means that the data is processed row by row and Warehouse Builder evaluates all conditions in sequential order using if-then-else constructions. The resulting target table thus maintains the sequential integrity of the transactions recorded at source.

## Example: Using the Mapping Editor to Create Staging Area Tables

You can use the Mapping Editor with an unbound table operator to quickly create staging area tables.

The following instructions describe how to create a staging table based on an existing source table. You can also use these instructions to create views, materialized views, flat files, and transformations.

**To map a source table to a staging table:**

1. In the Mapping Editor, add a source table.

From the menu bar, select **Graph**, then **Add**, then **Data Sources/Targets,** and then **Table Operator.** Alternatively, drag and drop the source table from the Projects Navigator onto the Mapping Editor canvas.

2. Use the **Add Table Operator** dialog box to select and bind the source table operator in the mapping. From the Add Table Operator dialog box, select **Create unbound operator with no attributes.**

   The mapping should now resemble Figure 5–10, with one source table and one staging area table without attributes.

***Figure 5–10 Unbound Staging Table without Attributes and Source Table***

3. With the mouse pointer positioned over the group in the source operator, click and hold down the mouse button.
4. Drag the mouse to the staging area table group.

   Warehouse Builder copies the source attributes to the staging area table and connects the two operators.
5. In the Mapping Editor, select the unbound table that you added to the mapping. Right-click and select **Create and Bind.**

   Warehouse Builder displays the Create And Bind dialog box.
6. In the Create in field, specify the target module in which to create the table.

   Warehouse Builder creates the new table in the target module that you specify.

# Using Pluggable Mappings

You can reuse the data flow of a mapping by creating a pluggable mapping around the portion of the flow that you want to reuse. A *pluggable mapping* is a reusable grouping of mapping operators that works as a single operator. It is similar to the concept of a function in a programming language and is a graphical way to define a function.

Once defined, a pluggable mapping appears as a single mapping operator, nested inside a mapping. You can reuse a pluggable mapping more than once in the same mapping, or in other mappings. You can include pluggable mappings within other pluggable mappings.

Like any operator, a pluggable mapping has a *signature* consisting of input and output attributes that enable you to connect it to other operators in various mappings. The signature is similar to the input and output requirements of a function in a programming language.

> **See Also:** *Oracle Warehouse Builder Concepts* for more information about pluggable mappings

A pluggable mapping can be either *reusable* or *embedded*:

- **Reusable pluggable mapping:** A pluggable mapping is reusable if the metadata it references can exist outside of the mapping in question. You can store reusable pluggable mappings either as standalone pluggable mappings, which are private for your use, or in folders (libraries). Users who have access to these folders can use the pluggable mappings as templates for their work.
- **Embedded pluggable mapping:** A pluggable mapping is embedded if the metadata it references is owned only by the mapping or pluggable mapping in question. An embedded pluggable mapping is not stored as either a standalone mapping or in libraries on the Globals Navigator. It is stored only within the mapping or the pluggable mapping that owns it, and you can access it only by editing the object that owns it. To validate or generate the code for an embedded pluggable mapping, you must validate or generate the code for the object that owns it.

## Creating Pluggable Mappings

Pluggable mappings are usually predefined and used when required. You can create pluggable mappings either from within a mapping by using the Mapping Editor, or from the navigation tree by using the wizard. The wizard is the faster way to create a pluggable mapping because it makes some default choices and guides you through fewer choices. You can make additional choices later in the Pluggable Mapping Editor. The editor presents you with all the settings in a series of tabs.

The Pluggable Mappings node in the navigation tree contains the following two nodes:

- **Standalone:** Contains standalone pluggable mappings
- **Pluggable Mapping Libraries:** Contains a set of pluggable mappings providing related functionality that you would like to publish as a library.

You can create pluggable mappings under either of these nodes.

### Creating Standalone Pluggable Mappings

1. In the Projects Navigator, expand the project node and then the Pluggable Mappings node.
2. Right-click **Standalone**, and select **New Pluggable Mapping**.

   The Create Pluggable Mapping Wizard is displayed.
3. On the Name and Description page, enter a name and an optional description for the pluggable mapping. Click **Next**.
4. On the Signature Groups page, one input signature group INGRP1 and one output signature group OUTGRP1 are displayed. Create any additional input or output signature groups as described in "Signature Groups" on page 5-40. Click **Next**.
5. On the Input Signature page, define the input signature attributes for the pluggable mapping as described in "Input Signature" on page 5-40. Click **Next**.
6. On the Output Signature page, define the output signature attributes for the pluggable mapping as described in "Output Signature" on page 5-40. Click **Next**.
7. On the Summary page, review the options that you entered using the wizard. Click **Back** to modify an option. Click **Finish** to create the pluggable mapping.

   Warehouse Builder opens the Pluggable Mapping Editor and displays the name of the pluggable mapping on the title bar.

8. Use the Pluggable Mapping Editor to add the required operators and create a data flow between the operators. For more information, see "Adding Operators to Mappings" on page 5-13.

A pluggable mapping is considered as an operator by Warehouse Builder when it is used in a mapping. You can insert a pluggable mapping into any mapping. To use a pluggable mapping within a mapping, drag and drop the Pluggable Mapping operator from the Component Palette onto the canvas. The Add Pluggable Mapping dialog box is displayed. Select the required pluggable mapping and add it to the mapping.

### Signature Groups

The signature is a combination of input and output attributes flowing to and from the pluggable mapping. Signature groups are a mechanism for grouping the input and output attributes.

A pluggable mapping must have at least one input or output signature group. Most pluggable mappings are used in the middle of a logic flow and have input as well as output groups.

- To create an additional signature group, click an empty cell in the Group column, enter the name of the group, and specify whether the group is an input or output group using the Direction column. You can enter an optional description for the column in the Description column.
- To remove a signature group, right-click the grey cell to the left of the group name and select **Delete**.

Click **Next** to continue with the wizard.

### Input Signature

The input signature is the combination of input attributes that flow into the pluggable mapping. Define the input attributes for each input signature group that you created.

If you defined multiple input signature groups, select the group to which you want to add attributes from the Group list box. To add an attribute, click an empty cell in the Attribute column and enter an attribute name. Use the Data Type field to specify the data type of the attribute. Also specify other details for the attribute such as length, precision, scale, and seconds precision by clicking the corresponding field and using the arrows on the field or typing in a value. Note that some of these fields are disabled depending on the data type you specify.

To remove an attribute, right-click the grey cell to the left of the attribute and select **Delete**.

Click **Next** to continue with the wizard.

### Output Signature

The output signature is the combination of output attributes that flow out of the pluggable mapping. Define the output attributes for each output signature group that you created.

If you defined multiple output signature groups, select the group to which you want to add attributes from the Group list box. To add an attribute, click an empty cell in the Attribute column and enter the attribute name. Use the Data Type field to specify the data type of the attribute. Provide additional details about the attribute such as length, precision, and scale by clicking the corresponding field and using the arrows

or typing the values. Note that some of these fields are disabled depending on the data type you specify.

To remove an attribute, right-click the grey cell to the left of the attribute name and select **Delete**.

Click **Next** to continue with the wizard.

You can also add an Input Signature or an Output Signature from the palette of the Pluggable Mapping Editor. Note that a pluggable mapping can have only one Input Signature and one Output Signature. Also, pluggable mapping Input and Output signatures can only be added within pluggable mappings. They cannot be added to normal mappings.

### Creating Pluggable Mapping Folders

A pluggable mapping folder is a container for a set of related pluggable mappings. You can keep your pluggable mappings private, or you can place them into folders and then publish the folders so that others can access them for their design work.

**To create a pluggable mapping folder:**

1. In the Projects Navigator, expand the project node and then the Pluggable Mappings node.
2. Right-click the Pluggable Mapping Folders node and select **New Pluggable Mapping Folder**.

   The Create Pluggable Mapping Folder dialog box is displayed.
3. Enter a name and an optional description for the pluggable mapping folder.

   If you want to start the Create Pluggable Mapping wizard to create a pluggable mapping immediately after you create this pluggable mapping folder, select **Proceed to Pluggable Mapping Wizard**.
4. Click **OK**.

The library is displayed in the Projects Navigator. Create individual pluggable mappings within this library as described in "Creating Standalone Pluggable Mappings" on page 5-39.

You can also move a pluggable mapping to any library on the tree.

### Creating User Folders Within Pluggable Mapping Libraries

Within a pluggable mapping library, you can create user folders to group pluggable mappings using criteria such as product line, functional groupings, or application-specific categories.

User folders can contain user folders and other pluggable mappings. There is no limit on the level of nesting of user folders. You can also move, delete, edit, or rename user folders.

You can move or copy a user folder and its contained objects to the same pluggable mapping library, to any user folder belonging to the same library, or to a user folder belonging to a different library.

Deleting a user folder removes the user folder and all its contained objects from the repository.

**To create a user folder within a pluggable mapping library:**

1. Right-click the pluggable mapping library or the user folder under which you want to create the user folder and select **New**.

   The New Gallery dialog box is displayed.

2. In the Items section, select **User Folder**.

   The Create User Folder dialog box is displayed.

3. Enter a name for the user folder and click **OK**.

The user folder is created and added to the tree.

**To create a pluggable mapping within a user folder:**

1. Right-click the user folder and select **New**.

   The New Gallery dialog box is displayed.

2. In the Items section, select **Pluggable Mapping**.

   To create a new user folder within this user folder, select **User Folder**.

3. Click **OK**.

If you selected Pluggable Mapping in Step 2, the Create Pluggable Mapping Wizard is displayed. If you selected User Folder in Step 2, then the Create User Folder dialog box is displayed.

You can move pluggable mappings from within a user folder or a pluggable mapping library to the Standalone node or to a different user folder or library. To move, right-click the pluggable mapping, select **Cut**. Right-click the user folder or pluggable mapping library to which you want to copy the pluggable mapping and select **Paste**.

## Copying Operators Across Mappings and Pluggable Mappings

Operators enable you to create user-defined transformation logic in a mapping or pluggable mapping. Sometimes, you may want to reuse an operator that you previously defined in another mapping or pluggable mapping. Warehouse Builder supports copy-and-paste reuse of existing transformation logic, defined using operators or operator attributes, in other mappings or pluggable mappings. In the remainder of this section, the term *mappings* includes both mappings and pluggable mappings.

You can reuse transformation logic by copying the operator or operator attributes from the source mapping and pasting them into the required mapping. You can also copy and paste operator groups (input, output, and input/output).

### Steps to Copy Operators, Groups, or Attributes

Use the following steps to copy operators, groups, and attributes defined in a mapping to other mappings within the same project.

1. Open the mapping containing the operator, group, or attributes that you want to copy. This is your source mapping.

   See "Steps to Open a Previously Created Mapping" on page 5-11.

2. Open the mapping into which you want to copy the operator, group, or attributes. This is your target mapping.

3. In the source mapping, select the operator, group, or attribute. From the Edit menu, select **Copy**. To select multiple attributes, hold down the Ctrl key while selecting attributes.

*or*

Right-click the operator, group, or attribute and select **Copy**. If you selected multiple attributes, then ensure that you hold down the Ctrl key while right-clicking.

> **Note:** If you do not require the operator, group, or attributes in the source mapping, you can choose **Cut** instead of **Copy**. Cutting removes the object from the source mapping.
>
> Cut objects can be pasted only once, whereas copied objects can be pasted multiple times.

4. In the target mapping, paste the operator, group, or attributes.
   - To paste an operator, select **Paste** from the Edit menu. Or, right-click any blank space on the canvas and select **Paste** from the shortcut menu.
   - To paste a group, first select the operator into which you want to paste the group and then select **Paste** from the Edit menu. Or, right-click the operator into which you want to paste the group and select **Paste** from the shortcut menu.
   - To paste attributes, select the group into which you want to paste the attribute and then select **Paste** from the Edit menu. Or, right-click the group into which you want to paste the group and select **Paste** from the shortcut menu.

When you copy and paste an operator, the new operator has a UOID that is different from the source operator.

If the target mapping already contains an operator with the same name as the one that is being copied, an *_n* is appended to the name of the new operator. Here, *n* represents a sequence number that begins with 1.

### Information Copied Across Mappings

When you copy an operator, group, or attribute to a target mapping, the following information is copied.

- Object binding details
- Display sets details
- Physical and logical properties of operators and attributes

  If there are multiple configurations defined for the object, details of all configurations are copied.

> **Note:** When you copy an attribute and paste it into an operator that is of a different type than the source operator, only the name and data type of the operator are copied. No other details are copied.

## Limitations of Copying Operators, Groups, and Attributes

Following are the limitations of copying operators, groups, and attributes:

- You can copy operators, groups, or attributes from a mapping and paste them into another mapping within the same project only.

- Any connections that existed between the operator, group, or attributes in the source mapping are not copied to the target mapping. Only the operator, group, or attributes are pasted into the target mapping.
- For pluggable mappings, the connections between child operators within the source pluggable mapping are copied to the target mapping.
- Group properties of the source operator are not copied to the group in the target operator. However, all the attributes contained in the group are copied.
- Before you copy and paste a group from a source operator to a target operator, you must create the group in the target operator. The group is not created when you perform a paste operation.

**Note:** Copying and pasting a large number of operators may take a considerable amount of time.

# Grouping Operators in Mappings and Pluggable Mappings

Complex mappings and pluggable mappings contain many operators that are used to perform the required ETL task. Typically, each data transformation task may use one or more operators, which results in mappings that look cluttered and are difficult to comprehend. Trying to view all the operators at once means that each operator appears very small and the names are unreadable. Thus, grouping operators that perform related transformation tasks into separate folders helps reveal the overall transformation logic performed in the mapping. Warehouse Builder provides a method to group a set of operators into a folder so that unnecessary operators are hidden from the mapping canvas. In the remainder of this section, the term *mappings* refers to both mappings and pluggable mappings.

Grouping less interesting operators into a collapsible folder allows you to focus on the components that are important at a given time. It also uses less space on the canvas, thus enabling you to easily work on the mapping. When required, you can ungroup the folder to view or edit the operators that it contains.

Mappings can contain more than one grouped folder. You can also create nested folders in which one folder contains a set of mapping operators and one or more folders.

## Steps to Group Operators in Mappings and Pluggable Mappings

Use the following steps to group operators in mappings and pluggable mappings.

1. Open the mapping in which you want to group operators.

   See "Steps to Open a Previously Created Mapping" on page 5-11.

2. Select the operators that you want to group.

   To select multiple objects, hold down the Ctrl key while selecting objects. Or, hold down the left-mouse button and draw a rectangle that includes the objects you want to select.

3. In the toolbar at the top of the Mapping Editor canvas, click **Group Selected Objects**. Or, from the Graph menu, select **Group Selected Objects**.

   The selected operators are grouped into a folder, and the collapsed folder is displayed in the Mapping Editor. A default name, such as Folder1, is used for the folder.

4. (Optional) Rename the folder so that the name is more intuitive and reflects the task that the group of operators performs.

   **To rename the folder:**

   a. Right-click the folder and select **Open Details**.

   b. In the Edit Folder dialog box, enter the name of the folder in the Name field and click **OK**.

### Viewing the Contents of a Folder

When you group operators to create a folder for the selected operators, you can view the operators contained in the folder using one of the following methods.

- Use the tooltip for the folder

  Position your mouse over the folder. The tooltip displays the operators contained in the folder.

- Use Spotlighting to view folder contents

  Select the folder and click **Spotlight Selected Objects** on the toolbar. Or, select the folder and choose **Spotlight Selected Objects** from the Graph menu. The folder is expanded, and the operators it contains are displayed. All other operators in the mapping are hidden. This is called spotlighting. For more information about spotlighting, see "Spotlighting Selected Operators" on page 5-45.

- Double-click the folder

  Double-click the folder to expand it. All operators contained in the folder are displayed and the surrounding operators in the mapping are moved to accommodate the folder contents.

## Steps to Ungroup Operators in Mappings and Pluggable Mappings

Use the following steps to ungroup operators.

1. Open the mapping in which you want to ungroup operators.

   See "Steps to Open a Previously Created Mapping" on page 5-11.

2. Select the folder that you want to ungroup.

   You can select multiple folders by holding down the Ctrl key and selecting all the folders.

3. In the toolbar, click **Ungroup Selected Objects**. Or, from the Graph menu, select **Ungroup Selected Objects**.

   The operators that were grouped are now displayed individually on the Mapping Editor.

## Spotlighting Selected Operators

Spotlighting enables you to view only selected operators and their connections. All other operators and their connections are temporarily hidden. You can perform spotlighting on a single operator, a group of operators, a single folder, a group of folders, or any combination of folders and operators. When you select objects for spotlighting, the Mapping Editor layout is redisplayed such that the relationships between the selected objects are displayed clearly. The edges between spotlighted components are visible. However, edges between spotlighted components and other nonselected components are hidden.

To spotlight a folder containing grouped operators, in the Mapping Editor, select the folder and click **Spotlight Selected Items** on the toolbar. The folder is expanded and all the operators it contains are displayed. Click **Spotlight Selected Objects** again to toggle the spotlighting mode. The folder will appear collapsed again and the operators it contains are hidden.

When in Spotlight mode, you can perform all normal mapping operations, such as moving or resizing operators, modifying operator properties, deleting spotlighted operators, creating new operators, and creating connections between operators. You can perform any operation that does not affect temporarily hidden operators.

If you create an operator in Spotlight mode, the operator will remain visible in the Mapping Editor when you toggle out of Spotlight mode.

## Locating Operators, Groups, and Attributes in Mappings and Pluggable Mappings

Mappings and pluggable mappings contain numerous operators that are used to perform the required data transformation. Warehouse Builder provides a quick method of locating operators, groups, and attributes within mappings and pluggable mappings. When you search for objects, you can define the scope of the search and the search criteria. In the remainder of this section, the term *mapping* refers to both mappings and pluggable mappings.

### Types of Search

You can perform the following types of search:

- Regular Search

  Warehouse Builder searches for the search string in all the operators within the mapping. The search string is matched with the display name of operators, group names, and attribute names.

- Advanced Search

  This is a more sophisticated method of locating objects in mappings. You can locate operators, groups, and attributes by specifying which objects should be searched, and the search criteria.

Use the Advanced Find dialog box to perform both regular and advanced searches.

### Searching for Objects in Mappings and Pluggable Mappings

**To locate an operator, group, or attribute in a mapping or pluggable mapping:**

1. Open the mapping using the steps described in "Steps to Open a Previously Created Mapping" on page 5-11.
2. From the Search menu, select **Find**.

   The Advanced Find dialog box is displayed. Depending on the type of search that you want to perform, use one of the following sets of instructions.

   - Steps to Perform a Regular Search on page 5-46
   - Steps to Perform an Advanced Search on page 5-47

## Steps to Perform a Regular Search

Performing regular search involves the following steps:

1. Specifying the Object to Locate
2. Specifying the Method Used to Display Search Results
3. Clicking **Find**.

## Steps to Perform an Advanced Search

Performing an advanced search involves the following steps:

1. Specifying the Object to Locate
2. Specifying the Method Used to Display Search Results
3. Specifying the Search Scope
4. Specifying the Search Criteria
5. Clicking **Find**.

## Advanced Find Dialog Box

The Advanced Find dialog box enables you to search for operators, groups, or attributes within a mapping or pluggable mapping. By default, this dialog box displays the options required to perform a regular search. In a regular search, you search for an operator, group, or attribute using its display name. You can also specify how the search results should be displayed.

An advanced search provides techniques to define the scope of the search and the search criteria. To perform an advanced search, click **Show Advanced**. The additional parameters that you must define for an advanced search are displayed.

### Specifying the Object to Locate

Use the Find field to specify the object that you want to locate. A regular search locates objects containing the same display name as the one specified in the Find field.

While performing an advanced search, in addition to the display name, you can specify a string that you provided in the Description property, physical name, business name, or name of the workspace object to which an operator is bound.

You can use wildcards in the search criteria. For example, specifying "C*" in the Find field searches for an object whose name begins with C or c.

### Specifying the Method Used to Display Search Results

Use the Turn On Highlighting button on the Advanced Find dialog box to specify the method used to display the search results. This is a toggle button, and you can turn highlighting on or off.

#### Turn on Highlighting

When you turn highlighting on, all objects located as a result of the search operation are highlighted in yellow in the Mapping Editor. Highlighting enables you to easily spot objects that are part of the search results.

#### Turn off Highlighting

This is the default behavior. When you turn off highlighting, objects located as a result of the search operation are identified by the control being moved to these objects. When control is on a particular object, the border around the object is blue. When highlighting is off, the search results are presented one at a time.

For example, if an operator is found as a result of the search, the borders of the node representing the operator are blue.

### Specifying the Search Scope

You can restrict the objects searched by specifying the scope of the search. The In Selected field in the Scope section enables you to limit the scope of the search to within the objects selected on the canvas.

To locate an object within a specific set of operators, you first select all the operators on the canvas and then choose **In Selected**. You can select multiple operators by holding down the Shift key while selecting operators.

### Specifying the Search Criteria

To perform an advanced search, you must specify additional search criteria that will further refine the search.

**Find By**

Use the Find By list to specify which name should be used to search for an object. The options you can select are:

- **Display Name:** Search either the physical names or the business names, depending on the Naming Mode set.
- **Physical Name:** Search for an object containing the same physical name as the one specified in the Find field.
- **Business Name:** Search for an object containing the same business name as the one specified in the Find field.
- **Bound Name:** Search for an object containing the same bound name as the one specified in the Find field, if this property is available for the object.
- **Description:** Search for an object containing the same description as the one specified in the Find field, if this property is defined for the object.

### Match Options

Use the Match Options section to specify the matching options used while finding the object. The options that you can select are:

- **Match Case:** Locates objects whose name and case match the search string specified in the Find field.

  When searching by physical name, match case is set to false by default.

  Note that in physical name mode, everything a user creates will be uppercase, but the imported objects may be in mixed case.

  When searching by logical name, match case is set to true by default.
- **Whole Word Only:** Restricts matches to exclude those objects that do not match the entire search string, unless specifically overridden by a wildcard.
- **Regular Expression:** Supports the specification of a pattern, used for a Java regular expression as the search string. If Regular Expression is combined with Whole Word Only, then a boundary matcher "$" is appended to the search string pattern.

  For more information about regular expression support, see *Oracle Database SQL Language Reference*.

#### Find Options

Specifies options for managing the search operation. Select one of the following options:

- **Incremental:** Performs a search after any character is typed or if characters are removed from the search string. Use the Find button to find additional objects that match the search string
- **Wrap Find:** Continues the search operation with the first object when the last object in the set has been reached.
- **Find from Beginning:** Continues the search with the first object in the set.

#### Scope

Use this section to restrict the scope of the search.

**In Selected:** Select this option to locate the search string only among the objects currently selected in the Mapping Editor.

#### Direction

Use the Direction section to step through the objects in the search result set either forward or backward. Select **Next** to step forward through the result set and **Previous** to step backward through the result set.

## Debugging Mappings

You can use the Mapping Editor to debug complex data flows that you design in mappings. Once you begin a debug session and connect to a valid target schema, the debugging functions appear on the Mapping Editor toolbar and under Log window. You can run a debugging session using a defined set of test data, and follow the flow of data as it is extracted, transformed, and loaded to ensure that the designed data flow performs as expected. If you find problems, then you can correct them and restart the debug session to ensure that the problems have been fixed before proceeding to deployment.

When you modify a mapping that is being debugged, the mapping properties are changed. Except when display sets in operators are modified, the Mapping Debugger reinitializes to reflect the changes to the mapping.

#### Before You Begin

Ensure that you are connected to a Control Center and that the Control Center is running.

### General Restrictions in the Mapping Debugger

The following restrictions and limitations apply to the Mapping Debugger.

- Mappings run using the debug mode in the Mapping Editor are intended to be used for debug purposes only. Mappings run from the Mapping Editor do not perform as well as mappings that are run from the Control Center. This is attributed to the setup of temporary objects necessary to support the debugging capabilities. Use the Control Center to run mappings.
- You cannot pause an active debug run using the Pause button on the toolbar or the associated item in the debug menu.
- You cannot use the Repository Browser to view the results of a mapping run in debug mode.

- Only mappings that can be implemented as a PL/SQL package can currently be run in debug mode. ABAP mappings are not supported in the debugger.
- The Advanced Queue operator is not supported when you run mappings in debug mode.

## Starting a Debug Session

To start a debug session, open the mapping that you want to debug in the Mapping Editor. From the Debug menu, select **Start**. Or, click **Start** on the Mapping Editor toolbar. The Mapping Editor switches to debug mode with the debug panels appearing in the Log window, and the debugger connects to the appropriate Control Center for the project. The debug-generated code is deployed to the target schema specified by the location of the module that contains the map being debugged.

> **Note:** When the connection cannot be made, an error message is displayed and you have an option to edit the connection information and retry.

After the connection has been established, a message appears, indicating that you may want to define test data. When you have previously defined test data, then you are asked if you want to continue with initialization.

To debug a mapping, each source or target operator must be bound to a database object. Defining test data for the source and target operators is optional. By default, the debugger uses the same source and target data that is currently defined for the non-debug deployment of the map.

## Debug Panels of the Design Center

When the Mapping Editor is opened in Debug mode, the Log window displays two new panels: Info Panel and Data Panel.

### Info Panel

When the Mapping Editor is in Debug mode, the Info panel in the Log window contains the following tabs:

- **Messages:** Displays all debugger operation messages. These messages let you know the status of the debug session. This includes any error messages that occur while running the mapping in debug mode.
- **Breakpoints:** Displays a list of all breakpoints that you have set in the mapping. You can use the check boxes to activate and deactivate breakpoints. For more information, see "Setting Breakpoints" on page 5-52.
- **Test Data:** Displays a list of all data objects used in the mapping. The list also indicates which data objects have test data defined.

### Data Panel

When the Mapping Editor is in Debug mode, the Data panel is displayed in the Log window. The Data panel includes Step Data and watch point tabs, that contain input and output information for the operators being debugged. The Step Data tab contains information about the current step in the debug session. Additional tabs can be added for each watch that you set. These watch tabs allow you to keep track of and view data that has passed or will pass through an operator regardless of the currently active

operator in the debug session. Operators that have more than one input group or more than one output group display an additional list that enables you to select a specific group.

If an operator has more than one input or output group then the debugger will have a list in the upper-right corner, above the input or output groups. Use this list to select the group you are interested in. This applies both to the step data and to a watch.

## Defining Test Data

Every source or target operator in the mapping is listed on the Test Data tab in the lower Info tab panel. It also contains the object type, the source, and a check mark that indicates whether the database object has already been bound to the source or target operator.

The object type listed on the tab is determined by whether the column names in the data source that you select (for example, a table) matches the columns in the mapping operators. There are two possible types:

- **Direct Access**. When there is an exact match, the type is listed as Direct Access.
- **Deployed as View**. When you choose a data source with columns that do not match the mapping operator columns, you can choose how you want the columns mapped. This object is deployed as a view when you run the mapping and the type is listed as Deployed as View.

Click **Edit** to add or change the binding of an operator as well as the test data in the bound database objects. Before you can run the mapping in debug mode, each listed source or target operator must be bound and have a check mark. The need to have test data defined and available in the bound database object depends on what aspect of the data flow you are interested in focusing on when running the debug session. Typically, you will need test data for all source operators. Test data for target operators is usually necessary if you want to debug loading scenarios that involve updates or target constraints.

**To define or edit test data:**

1. From the Test Data tab in the Mapping Editor, select an operator from the list and click **Edit**. The Define Test Data dialog box is displayed.
2. In the Define Test Data dialog box, specify the characteristics of the test data that you want Warehouse Builder to use when it debugs. There are many characteristics that you can specify. For example, you can specify that the test data be from a new or existing database object or that you can or cannot manually edit the test data. Click **Help** on the Define Test Data dialog box for more information.

### Creating New Tables to Use as Test Data

When you create a new table using the Define Test Data dialog box, Warehouse Builder creates the table in the target schema that you specified when you started the debug run. Because the debugger does not automatically drop this table when you end the debug session, you can reuse it for other sessions. Constraints are not carried over for the new table. However, all other staging tables created by the debug session are dropped when the debug session ends.

When you create a new table, Warehouse Builder creates the new table in the connected runtime schema. The new table has an automatically generated name, and the value of the Debug Binding name changes to reflect the new table name. The new table has columns defined for it that exactly match the names and data types of the

mapping source or target attributes. In addition, any data that is displayed in the grid at the time the table is created is copied into the newly created table.

You can use both scalar and user-defined data types in tables that you create using the Define Test Data dialog box.

### Editing the Test Data

You can edit test data at any time using the Define Test Data dialog box. Editing test data is applicable for scalar data types only.

If you change the binding of the operator to another database object, you must reinitialize the debug session to implement the change before running the mapping again in debug mode.

> **Note:** The data loaded in the target definitions will be implicitly committed. If you do not want the target objects updated, then you should create copies of target objects by clicking **Create New Table**.

### Cleaning Up Debug Objects in the Runtime Schema

Debug tables, with names prefixed with `DBG$`, are created in the runtime schema when you debug mappings. Because multiple users, using multiple instances, can debug the same mapping, debug objects are created separately for each debug session. The debug objects for a session are automatically dropped at the end of the session. However, if the user abruptly exits the Design Center without exiting the mapping debugger, the debug objects for the debug session in progress are not dropped, and become stale objects.

However, you can clean up all debug objects in the runtime schema by using the *`OWB_ORACLE_HOME`*`/bin/admin/cleanupalldebugobjects.sql` script. This script drops all the stale objects prefixed by `DBG$` in the runtime repository user schema.

This script should be run by a Warehouse Builder user with administrator privileges. Before you run this script, determine if all the objects that are prefixed by `DBG$` in the runtime user schema are stale. Because the same mapping can be debugged using multiple instances, running this script will cause disruptions for other users debugging the same mapping.

## Setting Breakpoints

If you are interested in how a specific operator is processing data, you can set a breakpoint on that operator to cause a break in the debug session. This enables you to proceed quickly to a specific operator in the data flow without having to go through all the operators step by step. When the debug session gets to the breakpoint, you can run data through the operator step by step to ensure that it is functioning as expected.

**To set or remove a breakpoint:**

1. From the Mapping Editor, click an operator, select **Debug**, and then select **Set Breakpoint.** You can also click **Set Breakpoint** on the toolbar to toggle the breakpoint on and off for the currently highlighted operator.

   If you are setting the breakpoint, the name of the operator set as a breakpoint appears in the list on the Breakpoints tab on the Info panel. If you are removing the breakpoint, then the name is removed. Use the **Clear** button on the Breakpoint tab to remove breakpoints.

2. Deselect or select the breakpoints on the Breakpoint tab to disable or enable them.

## Setting Watches

The Step Data tab on the Data Panel always shows the data for the current operator. To keep track of data that has passed through any other operator irrespective of the active operator, you can set a watch.

Use watches to track data that has passed through an operator or for sources and targets, the data that currently resides in the bound database objects. You can also set watches on operators after the debug run has already passed the operator and look back to see how the data was processed by an operator in the data flow.

**To set a watch:**

From the Mapping Editor, select an operator. From the Debug menu, select **Set Watch.** You can also select the operator and click **Set Watch** on the Mapping Editor toolbar to toggle the watch on and off.

A separate Watch panel is displayed to view data for Constant, Mapping Input, Mapping Output, Pre Mapping, Post Mapping, and Sequence operators. Since these operators contain lesser information than other operators, information regarding more than one of these operators is displayed in the Watch panel. Thus, only one instance of Watch panel is displayed for these operators if you choose to watch values.

**To remove a watch:**

To remove a watch, select the operator on the Mapping Editor canvas. Then, click **Set Watch** on the Mapping Editor toolbar or select **Set Watch** from the Debug menu.

If a watch panel consists of non-data based operators such as Constant, Mapping Input, Mapping Output, Pre Mapping, Post Mapping, and Sequence, you can remove these operators by right-clicking the operator and selecting **Remove**. You can remove all these operators at one time by closing the Watch panel.

### Saving Watches

When you set a watch for an operator, Warehouse Builder automatically saves the watch points for this operator, unless you close the watch panels. When you end the debug session and start up again, the tabs for operator watches that you created are displayed.

If you do not want to save watch points, click **Set Watch** in the Mapping Debugger toolbar. Or close the tab related to the Watch point.

## Running the Mapping

After you have defined the test data connections for each of the data operators, you can initially generate the debug code by selecting **Reinitialize** from the Debug menu, or by clicking **Reinitialize** on the Mapping Editor toolbar. Warehouse Builder generates the debug code and deploys the package to the target schema that you specified.

You can run the debug session in one of the following modes:

- Continue processing until the next breakpoint or until the debug run finishes by using the Resume button on the toolbar or the associated menu item.
- Process row by row using the Step button on the toolbar or the associated menu item.
- Process all remaining rows for the current operator by using the Skip button on the toolbar or the associated menu item.

- Reset the debug run and go back to the beginning by using the Reset button or the associated item from the Debug menu.

### Selecting the First Source and Path to Debug

A mapping may have more than one source and more than one path to debug:

- When a mapping has more than one source, Warehouse Builder prompts you to designate the source with which to begin. For example, when two tables are mapped to a joiner, you must select the first source table that you want to use when debugging.
- There may be multiple paths that the debugger can walk through after it has finished one path. For example, this is the case when you use a splitter. Having finished one path, the debugger asks you whether you would like to complete the other paths as well.

The mapping finishes if all target operators have been processed or if the maximum number of errors as configured for the mapping has been reached. The debug connection and test data definitions are stored when you commit changes to the Warehouse Builder workspace. Breakpoint and watch settings are stored when you save the project.

As the debugger runs, it generates debug messages whenever applicable. You can follow the data flow through the operators. A red dashed box surrounds the active operator.

### Debugging Mappings with Correlated Commit

How a mapping is debugged depends on whether the mapping has the Correlated Commit parameter set to ON or OFF:

- When you begin a debug session for a mapping that has the Correlated Commit parameter set to ON, the mapping is not debugged using paths. Instead, all paths are executed and all targets are loaded during the initial stepping through the mapping regardless of what path is chosen. Also, if one of the targets has a constraint violation for the step, then none of the targets are loaded for that step.
- When you begin a debug session for a mapping that has the Correlated Commit parameter set to OFF, the mapping is debugged using one path at a time. All other paths are left unexecuted and all other targets are not loaded unless you reach the end of the original path and return to execute another path in the mapping.

For example, you have a mapping that has a source, S1, connected to a splitter that goes to two targets, T1 and T2:

- If Correlated Commit is OFF, then the mapping is debugged starting with S1. You can then choose either the path going to T1 or the path going to T2. If you choose the path to T1, then the data going to T1 is processed and displayed, and the target T1 is loaded. After T1 is completely loaded, you are given the option to go back, execute the other path, and load target T2.
- If Correlated Commit is ON, then the mapping is also debugged starting with S1, and you are given the option of choosing a path however in this case, the path you choose only determines the path that gets displayed in the Mapping Editor as you step through the data. All paths are executed simultaneously. This is also how a mapping using Correlated Commit is executed when the deployable code is run.

### Setting a Starting Point

You can select an operator as a starting point, even if it is not a source. To set an operator as a starting point, start a debug session, then select the operator and click **Set as Starting Point** on the Mapping Editor toolbar. Or, from the Debug menu, select **Set as Starting Point**.

When an operator is set as a starting point, Warehouse Builder combines all the upstream operators and sources into a single query, which is used as a source, and the operator is automatically used as the first source when stepping through the map. The operators that are upstream of the starting point operator are not steppable, and do not have displayable data, even if a watch point is set.

A good use of "set as starting point" would be for a mapping with three source tables that were all connected to a single Joiner operator. Each source table contains a large number of rows (more than 50000 rows), too many rows to efficiently step through in the debugger. In this case, set the Joiner operator as a starting point, and limit the row count for one of the source tables to a more manageable number of rows (500) by using the Test Data Editor. It would be best to limit the row count of the source table that is effectively driving the joiner (that is, the source with which all the other sources are joined in the join condition).

### Debugging Pluggable Submap Operators

You can also debug a map which contains one or more pluggable submap operators. This could include a user-defined pluggable submap operator from the pluggable folder, or a system-defined submap operator. When the debug session is started, the mapping will go through debug initialization and start stepping at the first executable operator, just as usual.

If during the course of stepping through the operator, the debugger reaches a pluggable submap operator, then that operator is highlighted as the current step operator just like any other operator. If you click **Step** at this point, then the debugger steps through all of the operators contained by the pluggable submap without changing the graphical context of the map to show the implementation of the pluggable map. If you click **Step Into**, then the graphical context of the map changes to the pluggable mapping implementation, and the current step operator is set to the first executable source operator inside the pluggable mapping. The first executable source operator for the pluggable submap is one of the operators connected from the input signature operator.

You can now step through the pluggable mapping just as you would any other type of map. When the pluggable submap operator contains targets, the debugger loads theses just as it does for a top-level map. When the final executable operator is done executing, then the next time you click **Step**, the context changes back to the top-level map and begins execution at the next executable operator following the pluggable submap that was just executed. When the pluggable submap has no output connections, and it is the final executable operator in the top-level map, then stepping is done.

You can set breakpoints and watch points on operators inside of a pluggable submap. Additionally, during normal editing, you can change the graphical context as you do in normal editing, by clicking **Visit Child Graph** and **Return to Parent Graph**.

## ReInitializing a Debug Session

When you have made changes to the mapping, or have bound source or target operators to different database objects, then you must reinitialize the debug session to continue debugging the mapping with the new changes. To reinitialize, click

**Reinitialize** on the toolbar or select Reinitialize from the Debug menu. Reinitializing both regenerates and redeploys the debug code. After reinitialization, the mapping debug session starts from the beginning.

## Scalability

Scalability when debugging a mapping applies both to the amount of data that is passed as well as to the number of columns displayed in the Step Data panel. The Define Test Data dialog box provides a row limit that you can use to limit the amount of data that flows through the mapping. Also, you can define your own data set by creating your own table and manipulating the records manually.

To restrict the number of columns displayed on the Step Data tab, or on a watch tab, you can use display sets. By default, every operator has a display set ALL and a display set MAPPED (to display only the mapped attributes). You can manually add display sets on sources by using the Mapping Editor directly. Select the Use Display Set option under the right mouse button on an input or output group to select the display set.

# 6

# Performing ETL Using Dimensional Objects

Oracle Warehouse Builder enables you to design mappings that perform ETL using dimensional objects. This chapter describes extracting data from, removing data from, and loading data into dimensional objects.

This chapter contains the following topics:

- Using Dimensions in ETL Mappings
- Using Cubes in ETL Mappings

## Using Dimensions in ETL Mappings

The Dimension operator enables you to perform ETL on dimensions, slowly changing dimensions (SCDs) and time dimensions. You can extract data from, load data into, or remove data from dimensions using the Dimension operator. The dimensions may be deployed in relational form to an Oracle Database or to an analytic workspace.

### Loading Data Into Dimensions

Use a Dimension operator as a target in a mapping to load data into dimensions and SCDs. Define a data flow from the operators that represent the source objects to the dimension or SCD.

Warehouse Builder loads data into the dimension starting from the highest level.

> **Note:** You cannot map a data flow to the surrogate identifier or the parent surrogate identifier reference of a level.

#### Loading Data into Type 1 Dimensions

While loading data into a dimension, Warehouse Builder checks if a similar record already exists in the dimension by comparing the business identifier of the source records with the business identifier of the existing dimension records.

**To load data into a dimension:**

1. Define a mapping as described in "Defining Mappings" on page 5-11.
2. Add a Dimension operator to the mapping. Ensure that this operator is bound to the dimension into which you want to load data.

   For information about adding operators to mappings, see "Adding Operators to Mappings" on page 5-13.

3. Add operators corresponding to the source objects and to any transformations that must be performed before the source data is loaded into the dimension.
4. Map the attributes from the source operators to intermediate transformation operators (if the source data is to be transformed before loading it into the dimension) and then to the target dimension. Complete mapping all attributes according to your requirements.

   If a record with the same business identifier as the one being loaded already exists in the dimension, the record is updated with the attribute values from the source; otherwise the source record is loaded into the dimension.
5. Set the Loading Type property of the Dimension operator to Load.
6. Validate the mapping by selecting the mapping in the Projects Navigator and clicking **Validate** on the toolbar.

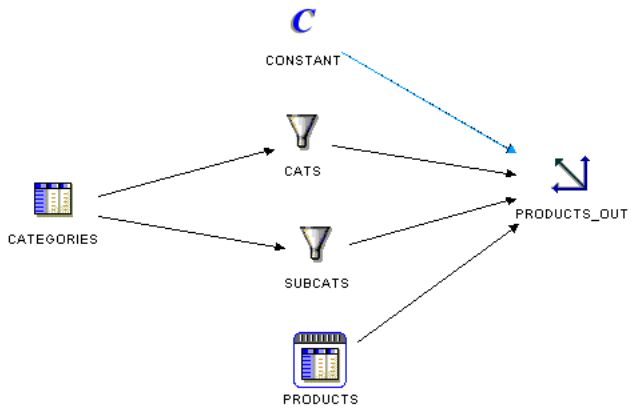
   Resolve errors, if any, that result from the validation process.
7. Generate the mapping by selecting the mapping in the Projects Navigator and clicking **Generate** on the toolbar.

   The generation results are displayed in a new Results tab in the Log window. Resolve generation errors, if any.

### Example: Loading Dimensions

Figure 6–1 displays a mapping that loads data into the `PRODUCTS` dimension, represented by the operator `PRODUCTS_OUT`. The source data is stored in two tables, `CATEGORIES` and `PRODUCTS`. The `CATEGORIES` table stores both the category and subcategory information. So the data from this table is filtered using two Filter operators `CATS` and `SUBCATS`. The filtered data is then mapped to the `CATEGORIES` level and the `SUBCATEGORIES` dimension levels. The `TOTAL` level of the dimension is loaded using a Constant operator. The data from the `PRODUCTS` table is mapped directly to the `PRODUCTS` level in the dimension.

***Figure 6–1 Loading the Products Dimension***




When you define a data flow to load a dimension, an in-line pluggable mapping that loads data into the dimension is created. To view this pluggable mapping, select the Dimension operator on the Mapping Editor canvas and click **Visit Child Graph** on the graphical toolbar.

### Loading Data into Type 2 Slowly Changing Dimensions (SCDs)

A Type 2 SCD stores both historic and current records. When the value of any Triggering Attribute in the Type 2 SCD is modified, the current record is marked as closed and a new record containing the changed values is created. A record is marked as closed by setting the value specified by the Default Expiration Time of Open Record property to the expiration date attribute. Regardless of the input connection defined in a mapping, the expiration date of a historic record is set using the Default Expiration Time of Open Record property.

> **Note:** When you load some Type 2 SCDs, if the target is an Oracle 9*i* database, only row-based mode is supported.
>
> A workaround is to switch on hierarchy versioning, by setting the parent surrogate identifier reference attribute as a trigger for all levels.

Before loading records into the Type 2 SCD, Warehouse Builder checks if a record with the same business identifier already exists in the Type 2 SCD. If the record does not exist, then Warehouse Builder adds the record to the Type 2 SCD. If the record already exists, then Warehouse Builder performs the following steps.

- Marks the existing record as closed by setting the value specified in the property Default Expiration Time of Open Record.
- Creates a new record using the changed attribute values.
  - If the effective date input for the level is not mapped, the effective time and expiration time are set using the Default Effective Time of Open Record and the Default Expiration Time of Open Record properties of the operator.
  - If the effective date input for the level is mapped, then the effective time of the new record is set to the value that is obtained from the effective date input data flow. The effective date input, if connected, represents the actual effective date of each individual new record.

> **Note:** To load multiple records for a particular business identifier during a single load, set the Support Multiple History Loading property for the Dimension operator that is bound to the Type 2 SCD.

#### Steps to Load Data into Type 2 SCDs

1. Define a mapping as described in "Defining Mappings" on page 5-11.
2. Add a Dimension operator to the mapping. Ensure that this operator is bound to the Type 2 SCD into which you want to load data.

   For information about adding operators to mappings, see "Adding Operators to Mappings" on page 5-13.
3. (Optional) Select the Dimension operator on the canvas by clicking the operator name, and use the Property Inspector to set the following properties:
   - Default Effective Time of Initial Record
   - Default Effective Time of Open Record
   - Default Expiration Time of Open Record
   - Type 2 Gap
   - Type 2 Gap Units

If you do not explicitly set values for these properties, the default values are used.

4. Set the Loading Type property of the Dimension operator to Load.
5. Add operators corresponding to the source objects and to any transformations that must be performed before the source data is loaded into the dimension.
6. Map the attributes from the source operators through intermediate transformation operators (if the source data is to be transformed before loading it into the dimension) and then to the target dimension operator. Complete mapping all attributes according to your requirements.

   **Note:** You cannot map attributes to the expiration date attribute of the Type 2 SCD.

7. Validate the mapping by selecting the mapping in the Projects Navigator and clicking **Validate** on the toolbar.

   Resolve errors, if any, that result from the validation process.
8. Generate the mapping by selecting the mapping in the Projects Navigator and clicking **Generate** on the toolbar.

   The generation results are displayed in a new Results tab in the Log window. Resolve generation errors, if any.

### Mapping Source Attributes to the Effective Data Attribute

In a mapping that loads a Type 2 SCD, if you map attributes from the source operator to the effective date attribute of a level, Warehouse Builder does the following:

- While loading the initial record, if the value of the source attribute is earlier than the value specified by the Default Effective Time of Initial Record property of the dimension operator (bound to the Type 2 SCD), Warehouse Builder uses the value from the source as the effective date of the record; otherwise Warehouse Builder takes the value specified in the Default Effective Time of Initial Record property as the effective date of the record.
- During subsequent loads for the record, if the record is being versioned, Warehouse Builder takes the effective time of the new record from the source. If no value is given for the effective time, then `SYSDATE` is used. Warehouse Builder sets the expiration time of the closed record to the effective time of the new record minus the gap.

If you do not map attributes from the source to the effective date attribute of a level, then Warehouse Builder does the following:

- While loading the initial record, Warehouse Builder uses the value specified in the Default Effective Time of Initial Record property as the effective date of the record.
- During subsequent loads for the record, if a new version is being created, Warehouse Builder takes the effective time of the new record from the source. If no value is given for the effective time, then `SYSDATE` is used. Warehouse Builder takes the expiration time of the previous version as the effective time of the new version minus the gap.

  For more information about the gap, see "Type 2 Gap" on page 25-17 and "Type 2 Gap Units" on page 25-17.

**Note:** Mapping to the Expiration Date attribute of a level is not allowed. While loading a record, the Default Expiration Time of Open Record property is used as the expiration date. The default value of this property is `NULL`.

#### Example: Values Assigned to Type 2 SCD Versioned Records

You create a mapping that loads the `Products` Type 2 SCD. The leaf level of this Type 2 SCD, `Product`, is loaded from a source table.The effective date attribute of the `Product` level is mapped from the source attribute `EFF_DATE`.

The Dimension operator has the following properties:

- Default Effective Time of Initial Record: 01-jan-2000
- Default Effective Time of Open Record: SYSDATE
- Default Expiration Time of Open Record: 01-jan-2099
- Type 2 Gap: 1

Consider a source `Product` level record with the value of `EFF_DATE` as 21-mar-2007 10.25.05.000000 PM.

When the initial `Product` level record is loaded, the values assigned to the record are:

Effective date: 01-jan-2000

Expiration date: 01-jan-2099

When the `Product` level record is versioned during a subsequent load on 21-mar-2007, the following occurs:

- The value of the source attribute overrides the Default Effective Time of Open Record property. Thus, the effective date stored in the new `Product` level record is 21-mar-2007 and the expiration date is set to 01-jan-2099.
- The initial `Product` level record is closed with the value of the expiration date set to 21-mar-200710.25.04.000000 PM.

### Loading Data into Type 3 Slowly Changing Dimensions (SCDs)

Use the following steps to load data into Type 3 SCDs.

1. Define a mapping as described in "Defining Mappings" on page 5-11.
2. Add a Dimension operator to the mapping. Ensure that this operator is bound to the Type 3 SCD into which you want to load data.

   For information about adding operators to mappings, see "Adding Operators to Mappings" on page 5-13.
3. Set the Loading Type property of the Dimension operator to Load.
4. Add operators corresponding to the source objects and to any transformations that must be performed before the source data is loaded into the Type 3 SCD.
5. Map the attributes from the source operators to intermediate transformation operators (if the source data is to be transformed before loading it into the dimension) and then to the target dimension operator. Complete mapping all attributes according to your requirements.

While loading data, Warehouse Builder checks if a record with the same business identifier exists in the Type 3 SCD. If the record does not exist, then it is added. If the record already exists, then the following steps are performed:

- The values of the versioned attributes are moved to the attributes that store the previous values of versioned attributes.
- The record with the values from the source record is updated.

6. Set the Loading Type property of the Dimension operator to Load.
7. Validate the mapping by selecting the mapping in the Projects Navigator and clicking **Validate** on the toolbar.

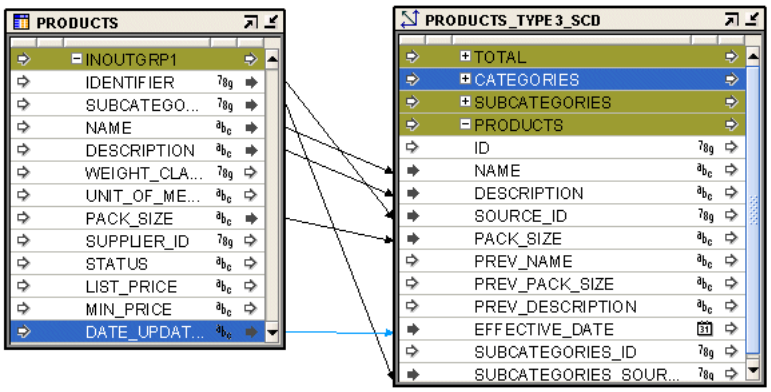
   Resolve errors, if any, that result from the validation process.
8. Generate the mapping by selecting the mapping in the Projects Navigator and clicking **Generate** on the toolbar.

   The generation results are displayed in a new Results tab in the Log window. Resolve generation errors, if any.

### Example: Loading Data into Type 3 SCDs

Figure 6–2 displays a mapping that loads data into the PRODUCTS level of the Type 3 SCD. In this mapping, the effective time of the current record is loaded from the source. You can also use the **Default Effective Time of Current Record** property to set a default value for the effective time of a level record.

***Figure 6–2 Loading a Type 3 SCD***




You cannot map a data flow to the attributes that represent the previous values of versioned attributes.

For example, in the mapping shown in Figure 6–2, you cannot map an attribute to the `PREV_PACK_SIZE` and `PREV_DESCRIPTION` attributes of the `PRODUCTS` level.

## Example: Loading Data Into Type 2 Slowly Changing Dimensions

The Type 2 SCD PRODUCTS_TYPE2 contains the levels Total, Categories, and Product. Product is the leaf level and its attribute Pack_size is the versioned attribute. The Effective_date and expiration_date attributes store the effective date and expiration date, respectively, for the product records.

The source data that is to be loaded into this dimension is stored in two tables: `Categories_tab` and `Product_information`. The name and the description of

the highest level in the Type 2 SCD are loaded using the `Total_desc` attribute of a Constant operator.

Use the following steps to load the PRODUCTS_TYPE2 Type 2 SCD.

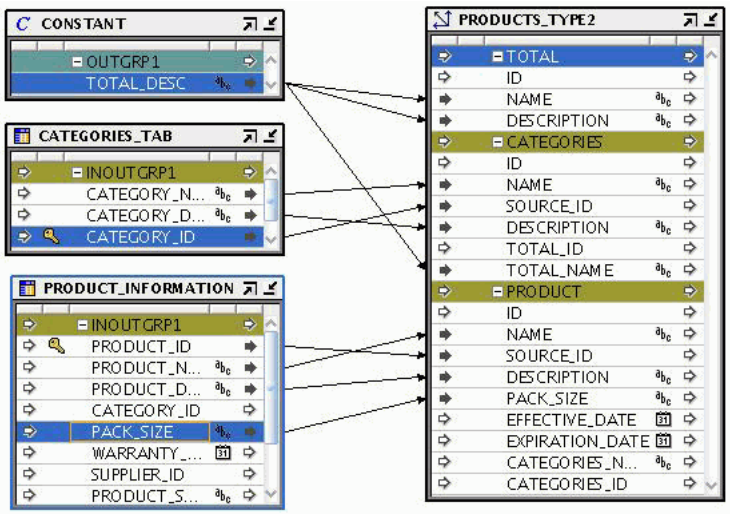
1. Define a mapping as described in "Defining Mappings" on page 5-11.
2. From the Projects Navigator, drag and drop the PRODUCTS_TYPE2 Type 2 SCD, the Categories_tab table, and the Product_information table onto the mapping canvas.
3. Set the Loading Type property of the Dimension operator to Load.

   Select the Dimension operator on the canvas. The Property Inspector displays the dimension properties. Loading Type is listed under the Dimension Properties node.
4. Drag and drop a Constant operator used to load the Total level onto the mapping canvas.

   Also, add an output attribute called Total_desc to this operator. Set the Expression property of the Total_desc operator to the value that you want assigned to the total_name and total_desc attributes.
5. Map the attributes from the source operators to the Type 2 SCD.

   Figure 6–3 displays the mapping with the source operator mapped to the target. Type 2 SCD.

***Figure 6–3 Loading Data Into a Type 2 SCD***




   Note that, in this example, the effective time of a record is loaded from the source. You can also choose to set this to a default value, such as `SYSDATE`, using the **Default Effective Time of Current Record** property.

   Because no values are explicitly assigned to the history logging properties, the default values are used.
6. Validate the mapping by selecting the mapping in the Projects Navigator and clicking **Validate** on the toolbar.

   Resolve errors, if any, that result from the validation process.
7. Generate the mapping by selecting the mapping in the Projects Navigator and clicking **Generate** on the toolbar.

The generation results are displayed in a new Results tab in the Log window. Resolve generation errors, if any.

You have now designed a mapping that loads data into a Type 2 SCD. To actually move the data from the source tables into the Type 2 SCD, you must deploy and execute this mapping. For more details about deploying and executing mappings, see "Starting ETL Jobs" on page 12-9.

## Extracting Data Stored in Dimensions

You can extract data stored in a workspace dimension, slowly changing dimension (SCD), or time dimension by using a Dimension operator as a source in a mapping.

---

**Note:** You cannot extract data from dimensions that use a MOLAP implementation.

---

### Extracting Data from Dimensions

Use the following steps to define a mapping that extracts data stored in dimensions.

1. Define a mapping as described in "Defining Mappings" on page 5-11.
2. Add a Dimension operator to the mapping. Ensure that this operator is bound to the dimension from which you want to extract data.

   For information about adding operators to mappings, see "Adding Operators to Mappings" on page 5-13.
3. Add operators corresponding to the target object and to any transformations that must be performed before the dimension data is loaded into the target.
4. Map the attributes from the dimension levels to the target operator or to intermediate operators that transform the source data. Complete mapping all attributes according to your requirements.
5. Validate the mapping by selecting the mapping in the Projects Navigator and clicking **Validate** on the toolbar.

   Resolve errors, if any, that result from the validation process.
6. Generate the mapping by selecting the mapping in the Projects Navigator and clicking **Generate** on the toolbar.

   The generation results are displayed in a new Results tab in the Log window. Resolve generation errors, if any.

### Extracting Data from Type 2 Slowly Changing Dimensions (SCDs)

Use a mapping containing a Dimension operator to extract data stored in a Type 2 SCD. The Dimension operator should be bound to the Type 2 SCD that contains the source data.

Because a Type 2 SCD stores multiple versions of a single record, you must specify the version of the record that should be extracted. To extract the current version of a record, set the Type 2 Extract/Remove Current Only property of the Dimension operator to Yes. To extract all records, including historic ones, set the Type 2 Extract/Remove Current Only property to No.

Additionally, you can set the following properties for the Dimension operator: Default Effective Time of Initial Record, Default Effective Time of Open Record, Default Expiration Time of Open Record, Type 2 Gap, and Type 2 Gap Units. These properties

enable you to assign values to the versioned attributes in the Type 2 SCD. All these properties have default values as displayed in the Property Inspector for the Dimension operator. If you do not explicitly assign values to these properties, then Warehouse Builder uses the default values.

**To define a mapping that extracts data from a Type 2 SCD:**

1. Define a mapping as described in "Defining Mappings" on page 5-11.
2. Add a Dimension operator to the mapping. Ensure that this operator is bound to the Type 2 SCD from which you want to extract data.

   For information about adding operators to mappings, see "Adding Operators to Mappings" on page 5-13.
3. Add operators corresponding to the target object and to any transformations that must be performed before the Type 2 SCD data is loaded into the target.
4. Map the attributes from the source Type 2 SCD to the target or to intermediate operators that transform the source data. Complete mapping all attributes according to your requirements.

   To specify the version of source records from the Type 2 SCD that should be extracted:

   - Set the Type 2 Extract/Remove Current Only property to Yes to extract the current record.
   - Set the Type 2 Extract/Remove Current Only property to No to extract historic records.
5. Validate the mapping by selecting the mapping in the Projects Navigator and clicking **Validate** on the toolbar.

   Resolve errors, if any, that result from the validation process.
6. Generate the mapping by selecting the mapping in the Projects Navigator and clicking **Generate** on the toolbar.

   The generation results are displayed in a new Results tab in the Log window. Resolve generation errors, if any.

### Extracting Data from Type 3 Slowly Changing Dimensions (SCDs)

Use a mapping containing a Dimension operator to extract data from a Type 3 SCD. The operator must be bound to the Type 3 SCD that contains the source data.

A Type 3 SCD uses separate attributes to store historic values of versioned attributes. Depending on whether you want to extract the current record or historic values, you map the attributes in the Type 3 SCD to the target operators.

**To define a mapping that extracts data from a Type 3 SCD:**

1. Define a mapping as described in "Defining Mappings" on page 5-11.
2. Add a Dimension operator to the mapping. Ensure that this operator is bound to the Type 3 SCD from which you want to extract data.

   For information about adding operators to mappings, see "Adding Operators to Mappings" on page 5-13.
3. Add operators for the target object and for any transformations that are needed before the dimension data is loaded into the target.
4. Map the attributes from the source Type 3 SCD either to the target operator or to operators that transform the source data.

- To extract historic data, map the attributes that represent the previous values of versioned attributes to the target or intermediate operator.
- To extract the current record, map the attributes that store the level attributes to the target or intermediate operator.

5. Validate the mapping by selecting the mapping in the Projects Navigator and clicking **Validate** on the toolbar.

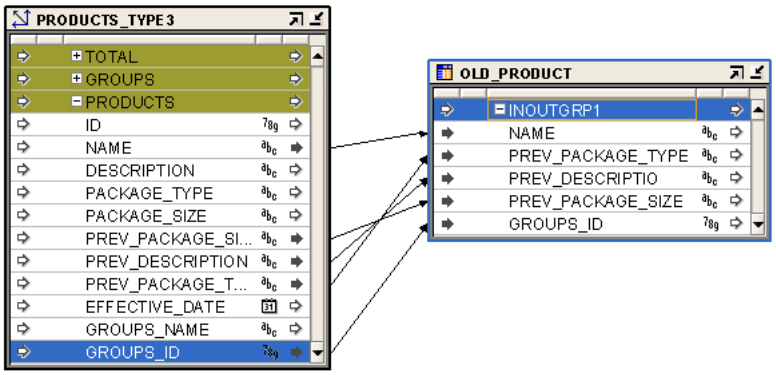
   Resolve errors, if any, that result from the validation process.

6. Generate the mapping by selecting the mapping in the Projects Navigator and clicking **Generate** on the toolbar.

   The generation results are displayed in a new Results tab in the Log window. Resolve generation errors, if any.

Figure 6–4 displays a mapping that sources the historic data records from the `PRODUCTS` Type 3 dimension. In this example, to source historic data, use the `PREV_DESCRIPTION`, `PREV_PACKAGE_TYPE`, or `PREV_PACKAGE_SIZE` attributes. To source current data, use `DESCRIPTION`, `PACKAGE_TYPE`, or `PACKAGE_SIZE`.

***Figure 6–4 Mapping That Sources Data from a Type 3 SCD***




## Removing Data from Dimensions

Use the Dimension operator to remove data from dimensions and SCDs. You create a mapping with the Dimension operator, the source objects containing the data that must be removed from the dimension, and any required transformation operators. Map attributes from the source or transformation operators to the Dimension operator. When the map is executed, the business identifier of the source record is compared to the business identifiers in the dimension. If the business identifiers match, then the corresponding record in the dimension is removed.

To remove data from dimensions or SCDs, set the Loading Type property of the Dimension operator to Remove.

### Effect of Surrogate Keys on Dimension Data Removal

When you remove data from a dimension that was created with surrogate keys, parent records of existing children are removed, but child records are left referencing nonexistent parents.

When you remove data from a dimension that was created with no surrogate keys, parent records of existing child records as well as the child records are removed. This is in effect a cascade operation.

## Example: Removing Data from Dimensions

The DF1_SINGLEH1_SCD1 is a dimension containing the levels Total, Region, Territory, and Salesrep. This dimension contains existing data that was loaded earlier using another mapping. The tables WBSALESREP_ALL, WB_REGIONS, WB_TERRITORIES, and WBSALESREPTERRITORIES contain the data that must be removed from the various levels in the dimension.

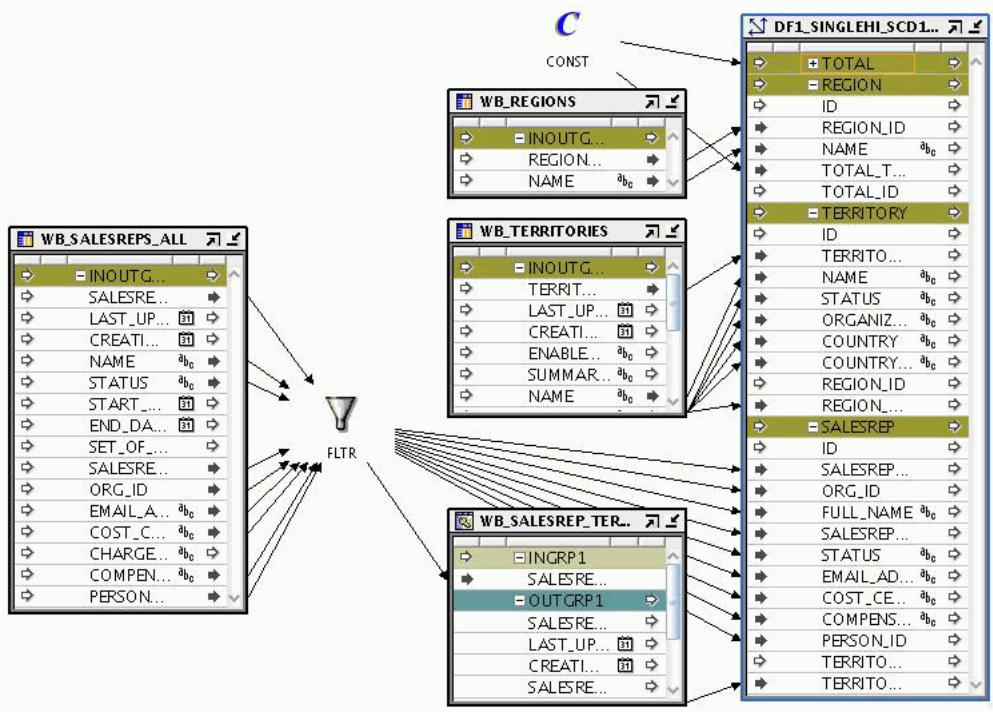
Use the following steps to remove data from the DF1_SINGLEH1_SCD1 dimension.

1. Define a mapping as described in "Defining Mappings" on page 5-11.
2. From the Projects Navigator, drag and drop the DF1_SINGLEH1_SCD1 dimension onto the mapping canvas.
3. Set the Loading Type property of the Dimension operator to Remove.
4. From the Projects Navigator, drag and drop the following tables onto the mapping canvas: WBSALESREP_ALL, WB_REGIONS, WB_TERRITORIES, and WBSALESREPTERRITORIES.
5. Add a Constant operator to the mapping. Create two output attributes ID and Name. To the Expression property of both attributes, assign the values that you want to remove from the Total level.

   For example, if you set the Expression attribute of ID to 100 and that of Name to Asia, then all level records, in the Total level, whose ID and Name match these values are deleted. Also, because the ID attribute of the Constant operator is mapped to the Region attribute, all child records of the Total level records that are removed are also removed.
6. Map the attributes from the source operators to the Dimension operator.

   Figure 6–5 displays the mapping with the attributes connected to the Dimension operator.

***Figure 6–5 Mapping that Removes Data From a Dimension***



The business identifier of the source records is compared to the business identifier of the dimension level record to which it is mapped. If the business identifier matches, then the corresponding record is removed from the dimension.

7. Validate the mapping by selecting the mapping in the Projects Navigator and clicking **Validate** on the toolbar.

   Resolve errors, if any, that result from the validation process.

8. Generate the mapping by selecting the mapping in the Projects Navigator and clicking **Generate** on the toolbar.

   The generation results are displayed in a new Results tab in the Log window. Resolve generation errors, if any.

You have now designed a mapping that removes data from a dimension. To actually remove the specified data from the dimension, you must deploy and execute this mapping. For more details about deploying and executing mappings, see "Starting ETL Jobs" on page 12-9.

## Using Cubes in ETL Mappings

The Cube operator enables you to extract data from, load data into, and remove data from cubes. To extract data from cubes, use the Cube operator as a source in a mapping.

> **Note:** You cannot extract data from cubes that use a MOLAP implementation.

Use the Loading Type property of the Cube operator to indicate if data is being loaded into the cube or removed from the cube. For cubes, the Loading Type property can have the following three values: LOAD, INSERT_LOAD, and REMOVE.

> **See Also:** "Loading Type" on page 25-11 for more information about the Loading Type property of cubes

### ACTIVE_DATE Attribute in Cubes

Cube operators contain an attribute called ACTIVE_DATE. This attribute represents the point in time that is used to determine which record in a Type 2 SCD is the active record. This property is applicable only when the cube that you are loading has one or more Type 2 SCDs.

If you do not map an attribute from the source to ACTIVE_DATE, `SYSDATE` is used as the default.

If you map a source attribute to ACTIVE_DATE, the value of the source attribute is used to determine which version of the Type 2 SCD record is referenced by the cube record.

For any cube that references a dimension in which the level is of a Type 2 SCD, the `WHERE` clause generated to determine the dimension member is as follows:

```
...
WHERE
(...
   (<dim_name>.DIMKEY = <lookup_for_dimension_dimkey> AND
           (<level>_EFFECTIVE_DATE <= ACTIVE_DATE AND
                   <level>_EXPIRATION_DATE >= ACTIVE_DATE) OR
           (<level>_EFFECTIVE_DATE <= ACTIVE_DATE AND
```

```
                    <level>_EXPIRATION_DATE IS NULL))
...)
```

### Loading Data from Type 2 SCDs into Cubes

If a mapping that loads a cube references at least one Type 2 SCD that has the Default Expiration Time of Open Record set to a non-NULL value, then the ACTIVE_DATE attribute of the Cube operator must be mapped from the source that contains the date value that defines the range for the dimension record.

If the ACTIVE_DATE attribute is not mapped from the source, then the `SYSDATE` value will define the date range for the dimension record.

When the ACTIVE_DATE attribute is mapped from the source, the source attribute value is used to perform a range comparison to determine which dimension record should be loaded.

The logic used to perform the lookup for the dimension member is described in the `WHERE` clause listed previously.

## Loading Data Into Cubes

When you load a cube, you map the data flow from the source to the attribute that represents the business identifier of the referencing level. Warehouse Builder performs a lookup on the dimensions and then stores the corresponding surrogate identifier in the cube table. For example, when you map the attributes from the dimension operator to the cube operator, a Lookup operator is created in cases where it is needed to lookup the surrogate identifier of the dimension.

Note that if there is a possibility of the lookup condition returning multiple rows, you must ensure that only one row is selected out of the returned rows. You can do this by using the Deduplicator operator or Filter operator.

Use the following steps to load data into a cube.

1. Define a mapping as described in "Defining Mappings" on page 5-11.
2. Add a Cube operator to the mapping. Ensure that this operator is bound to the cube into which you want to load data.

   For information about adding operators to mappings, see "Adding Operators to Mappings" on page 5-13.
3. Add operators corresponding to the source objects and to any transformations that must be performed before the source data is loaded into the dimension. Ensure that all source data objects are bound to the repository objects.
4. Map the attributes from the source operators to intermediate transformation operators (if the source data is to be transformed before loading it into the dimension) and then to the target cube. Complete mapping all attributes according to your requirements.
5. Set the Loading Type property of the Cube operator to Load.
6. Validate the mapping by selecting the mapping in the Projects Navigator and clicking **Validate** on the toolbar.

   Resolve errors, if any, that result from the validation process.
7. Generate the mapping by selecting the mapping in the Projects Navigator and clicking **Generate** on the toolbar.

The generation results are displayed in a new Results tab in the Log window. Resolve generation errors, if any.

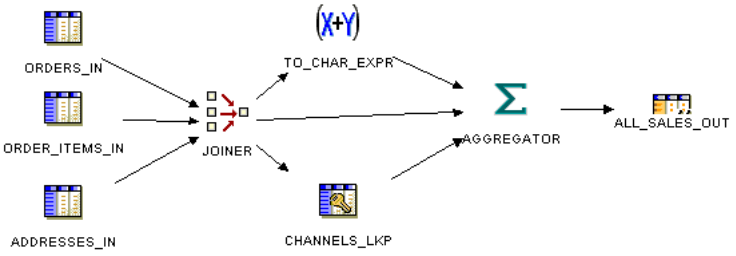
---

**Note:** If the source data for your cube contains invalid or null values for the dimension references, it is recommended that you use Orphan Management or DML Error Logging to avoid possible problems during subsequent cube loads.

---

Figure 6–6 displays a mapping that uses the Cube operator as a target. Data from three source tables is joined using a Joiner operator. An Aggregator operator is used to aggregate the joined data with the data from another source table. The output of the Aggregator operator is mapped to the Cube operator.

***Figure 6–6 Mapping that Loads a Cube***



# 7

# Using SQL*Loader, SAP, and Code Template Mappings

Oracle Warehouse Builder enables you use mappings to extract data from disparate sources such as flat files and SAP. Code Template (CT) mappings help in open connectivity and allow customization of how data is moved.

This chapter describes the steps used to create SAP and CT mappings. It also includes examples of performing ETL on SAP systems and other heterogeneous databases.

This chapter contains the following topics:


- Creating SQL*Loader Mappings to Extract Data from Flat Files
- Creating SAP Extraction Mappings
- Retrieving Data from the SAP System
- Creating Code Template (CT) Mappings
- Setting Options for Code Templates in Code Template Mappings
- Auditing the Execution of Code Template Mappings
- Using Code Template Mappings to Perform Change Data Capture (CDC)
- Using Control Code Templates
- Using Oracle Target CTs in Code Template Mappings
- Moving Data from Heterogeneous Databases to Oracle Database


## Creating SQL*Loader Mappings to Extract Data from Flat Files

Use the Flat File operator in a mapping to extract data from and load data into flat files. You can use Flat File operators as either sources or targets, but not a combination of both.

Define mappings to extract data from flat files as described in "Extracting Data from Flat Files" on page 7-2.

Define mappings to load data into flat files as described in "Loading Data into a Flat File" on page 7-3.

> **See Also:** "Best Practices for Designing SQL*Loader Mappings" on page 10-13 for more information about best practices to follow while using SQL*Loader mappings

### Subsequent Steps

After you design a mapping and generate its code, you can create a process flow or proceed directly with deployment followed by execution.

Use process flows to interrelate mappings. For example, you can design a process flow such that the completion of one mapping triggers an e-mail notification and starts another mapping. For more information, see Chapter 8, "Designing Process Flows".

Deploy the mapping, and any associated process flows you created, and then execute the mapping as described in Chapter 12, "Deploying to Target Schemas and Executing ETL Logic".

## Extracting Data from Flat Files

To extract data from a flat file, use a Flat File operator as a source in a mapping.

Alternatively, you can define an external table based on the flat file definition and use an External Table operator as a source. If you are loading large volumes of data, then loading from a flat file enables you to use the DIRECT PATH SQL*Loader option, which results in better performance. If you are not loading large volumes of data, then you can benefit from many of the relational transformations available when using external tables.

> **See Also:** *Oracle Warehouse Builder Sources and Targets Guide* for a comparison of external tables and flat files

As a source, the Flat File operator acts as a row set generator that reads from a flat file using the SQL*Loader utility. The targets in a flat file mapping can be relational objects such as tables. Note that an External Table operator cannot be a target, because external tables are read-only.

When you design a mapping with a Flat File source operator, you can use the following operators:

- Filter Operator
- Constant Operator
- Data Generator Operator
- Sequence Operator
- Expression Operator
- Transformation Operator

> **Note:** If you use the Sequence, Expression, or Transformation operators, you cannot use the SQL*Loader Direct Load setting as a configuration parameter.

When you use a flat file as a source, ensure that a connector is created from the flat file source to the relational target. It the connector is not created, the mapping cannot be deployed successfully.

### Defining a Mapping that Extracts Data from Flat Files

1. Import the flat file metadata into the Warehouse Builder workspace.

> **See Also:** *Oracle Warehouse Builder Sources and Targets Guide* for more information about importing flat file metadata

2. In the Projects Navigator, create a mapping as described in "Steps to Define a Mapping" on page 5-11.
3. From the Projects Navigator, drag and drop the flat file from which data is to be extracted onto the Mapping Editor canvas.
4. On the Mapping Editor canvas, add the operators that represent the target objects into which data extracted from the flat file is to be loaded. Also add the transformation operators needed to transform the source data.

   **See Also:** "Adding Operators to Mappings" on page 5-13 for information about adding operators

5. On the Mapping Editor canvas, create the data flows between the source, transformation, and target operators.
6. Validate the mapping by selecting **Validate** from the File menu. Rectify validation errors, if any.

## Loading Data into a Flat File

To load data into a flat file, use a Flat File operator as a target in a mapping.

A mapping with a flat file target generates a PL/SQL package that loads data into a flat file instead of loading data into rows in a table.

---

**Note:** A mapping can contain a maximum of 50 Flat File target operators at one time.

---

You can use an existing flat file with either a single record type or multiple record types. If you use a multiple-record-type flat file as a target, then you can only map to one of the record types. If you want to load all of the record types in the flat file from the same source, then you can drop the same flat file into the mapping as a target again and map to a different record type. For an example of this usage, see "Using Direct Path Loading to Ensure Referential Integrity in SQL*Loader Mappings" on page 10-18. Alternatively, create a separate mapping for each record type that you want to load.

### Creating Flat File Targets

Use one of the following methods to create a Flat File target operator:

- Import an existing flat file definition into the repository and use this flat file as a target in a mapping.
- Define a flat file using the Create Flat File Wizard and use this as a target in the mapping.
- Create a new flat file as described in "Creating a New Flat File Target" on page 7-4.

### Defining a Mapping That Loads Data into a Flat File

Use the following steps to define a mapping that loads data into a flat file.

1. In your target module, define the flat file into which you want to load data using one of the methods described in "Creating Flat File Targets" on page 7-3.
2. In the Projects Navigator, create a mapping as described in "Steps to Define a Mapping" on page 5-11.

3. From the Projects Navigator, drag and drop the flat file into which data is to be loaded onto the Mapping Editor canvas.
4. On the Mapping Editor canvas, add operators representing the source objects from which data is to be loaded into the flat file. Also add the transformation operators used to transform the source data.

   **See Also:** "Adding Operators to Mappings" on page 5-13 for information about adding operators

5. On the Mapping Editor canvas, create the data flows between the source, transformation, and target operators.
6. Validate the mapping by selecting **Validate** from the File menu. Rectify validation errors, is any.

### Creating a New Flat File Target

1. If you have not already done so, create a flat file module.

   A flat file module is necessary to enable you to create the physical flat file later in these instructions.
2. Define a mapping as described in "Defining Mappings" on page 5-11.
3. Drag and drop a Flat File operator onto the canvas.
4. On the Add Flat File Operator dialog box, select **Create Unbound Operator with No Attributes** and assign a name to the new target operator.
5. Edit the new operator as described in "Editing Operators" on page 5-21.

   Thus far, you have defined an operator that represents a flat file but have not created the actual flat file target.
6. To create the flat file in the database, right-click the operator and select **Create and Bind**.

   The dialog box prompts you to select a flat file module and enables you to assign a unique name to the flat file. When you click **OK**, Warehouse Builder displays the new target in the Files node, under the module that you specified.
7. Continue to define your mapping as described in "Steps to Perform Extraction, Transformation, and Loading (ETL) Using Mappings" on page 5-9.

## Creating SAP Extraction Mappings

After importing metadata from SAP tables, you must define the extraction mapping to retrieve data from the SAP system.

### Defining an SAP Extraction Mapping

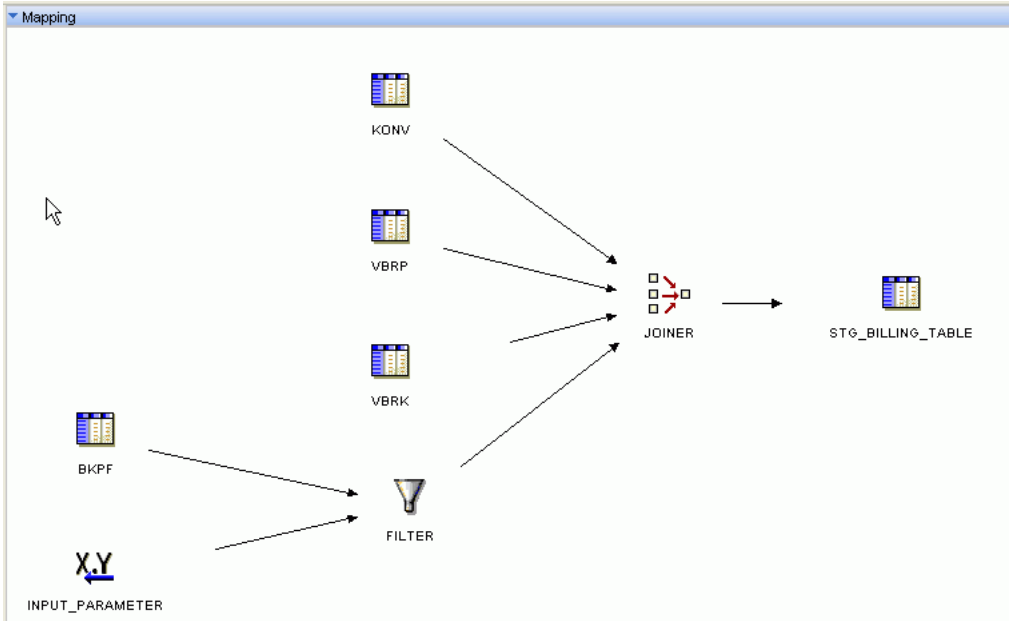
Use the Mapping Editor to create a mapping containing SAP tables. Creating a mapping with SAP tables is similar to creating mappings with other database objects. However, there are restrictions on the operators that can be used in the mapping. You can only use Table, Filter, Joiner, and Mapping Input Parameter mapping operators in a mapping containing SAP tables.

A typical SAP extraction mapping consists of one or more SAP source tables (transparent, cluster, or pooled), one or more Filter or Joiner operators, and a non-SAP target table (typically an Oracle Database table) to store the retrieved data.

> **Note:** You cannot have both SAP and non-SAP (Oracle Database) source tables in a mapping. However, you can use an Oracle Database table as a staging table.

Figure 7–1 displays a mapping that extracts data from an SAP source.

***Figure 7–1 SAP Extraction Mapping***




In this example, the Input Parameter holds a Date value, and the data from table `BKPF` is filtered based on this date. The Joiner operator enables you to join data from multiple tables, and the combined data set is stored in a staging table.

This section contains the following topics:

- Adding SAP Tables to the Mapping
- Setting the Loading Type
- Setting Configuration Properties for the Mapping
- Setting the Join Rank

## Adding SAP Tables to the Mapping

**To add an SAP table to a mapping:**

On the Mapping Editor, drag and drop the required SAP table onto the Mapping Editor canvas.

The editor places a Table operator on the mapping canvas to represent the SAP table.

## Setting the Loading Type

Use the Property Inspector to set the SQL*Loader properties for the tables in the mapping.

**To set the loading type for an SAP source table:**

1. On the Mapping Editor, select the SAP source table. The Property Inspector displays the properties of the SAP table.
2. Select a loading type from the Loading Type list. With ABAP code as the language for the mapping, the SQL*Loader code is generated as indicated in Table 7–1.

***Table 7–1 SQL*Loader Code Generated in ABAP***

| Loading Type | Resulting Load Type in SQL*Loader |
|---|---|
| INSERT | APPEND |
| CHECK/INSERT | INSERT |
| TRUNCATE/INSERT | TRUNCATE |
| DELETE/INSERT | REPLACE |
| All other types | APPEND |

## Setting Configuration Properties for the Mapping

Perform the following steps to configure a mapping containing SAP tables:

- Use the Configuration tab to define the code generation language as described in "Setting the Language Parameter" on page 7-6.
- Set ABAP specific parameters, and the directory and initialization file settings in the Configuration tab as described in "Setting the Runtime Parameters" on page 7-6.

### Setting the Language Parameter

The Language parameter enables you to choose the type of code you want to generate for a mapping. For mappings containing SAP source tables, Warehouse Builder automatically sets the language parameter to ABAP. Verify that this parameter has been set to ABAP.

### Setting the Runtime Parameters

With the Language set to ABAP, expand the Runtime Parameters node in the Configuration tab to display settings specific to ABAP code generation.

Some of these settings come with preset properties that optimize code generation. Oracle recommends that you retain these settings, as altering them may slow the code generation process.

The following Runtime parameters are available for SAP mappings:

- **Background Job**: Select this option to run the ABAP report as a background job in the SAP system. Enable this option for the longer running jobs. Foreground batch jobs that run for a long duration are considered hanging in SAP after a certain time. Therefore, it is ideal to run a background job for such extracts.
- **File Delimiter for Staging File:** Specifies the column separator in a SQL data file.
- **Data File Name:** Specifies the name of the data file that is generated when the ABAP code for the mapping is run in the SAP system.
- **SQL Join Collapsing:** Specifies the following hint, if possible, to generate ABAP code.

  ```
  SELECT < > INTO < > FROM (T1 as T1 inner join T2 as T2) ON <condition >
  ```

  The default setting is TRUE.

- **Primary Foreign Key for Join:** Specifies the primary key to be used for a join.
- **ABAP Report Name**: Specifies the name of the ABAP code file generated by the mapping. This is required only when you are running a custom function module to execute the ABAP code.
- **SAP System Version**: Specifies the SAP system version number to which you want to deploy the ABAP code. For MySAP ERP and all other versions, select SAP R/3 4.7. Note that different ABAP code is required for versions prior to 4.7.
- **Staging File Directory:** Specifies the location of the directory in the SAP system where the data file generated by ABAP code resides.
- **SAP Location**: Specifies the location of the SAP instance from where the data can be extracted.
- **Use Select Single:** Indicates whether Select Single is generated, if possible.
- **Nested Loop:** Specifies a hint to generate nested loop code for a join, if possible.

### Setting the Join Rank

You must set the Join Rank parameter only if the mapping contains the Joiner operator, and you want to explicitly specify the driving table. Unlike SQL, ABAP code generation is rule-based. Therefore, you must design the mapping so that the tables are loaded in the right order. Or you can explicitly specify the order in which the tables must be joined. To do this, from the Configuration tab, expand **Table Operators**, and then for each table, specify the Join Rank. The driving table must have the Join Rank value set to 1, with increasing values for the subsequent tables.

You can also let Warehouse Builder decide the driving table and the order of joining the other tables. In such cases, do not enter values for Join Rank.

## Retrieving Data from the SAP System

After designing the extraction mapping, you must validate, generate, and deploy the mapping, as you do with all mappings in Warehouse Builder.

**To generate the script for the SAP mapping:**

1. Right-click the SAP mapping and select **Generate.**

   The generation results are displayed in the Log window, under the Scripts node.

2. Expand the Scripts node, select the script name, and click **View Script** on the Log window toolbar.

   The generated code is displayed in the Code Viewer.

   You can edit, print, or save the file using the code editor. Close the Code Viewer to return to the Design Center.

3. To save the script, right-click the script and click **Save Script As** on the Log window toolbar.

After you generate the SAP mapping, you must deploy the mapping to create the logical objects in the target location. To deploy an SAP mapping, right-click the mapping and select **Deploy**. You can also deploy the mapping from the Control Center Manager.

For detailed information about deployment, see Chapter 12, "Deploying to Target Schemas and Executing ETL Logic".

When an SAP mapping is deployed, an ABAP mapping is created and stored in the Warehouse Builder runtime schema. Warehouse Builder also saves the `ABAP` file under *`OWB_ORACLE_HOME`*`\owb\deployed_files` directory, where *`OWB_ORACLE_HOME`* is the location of the Oracle Database home directory of your Warehouse Builder installation. Note that if you are using the Warehouse Builder installation that comes with Oracle Database, then this is the same as the database home.

Depending on whether data retrieval from the SAP system is fully automated, semiautomated, or manual, you must perform the subsequent tasks. This section consists of the following topics:

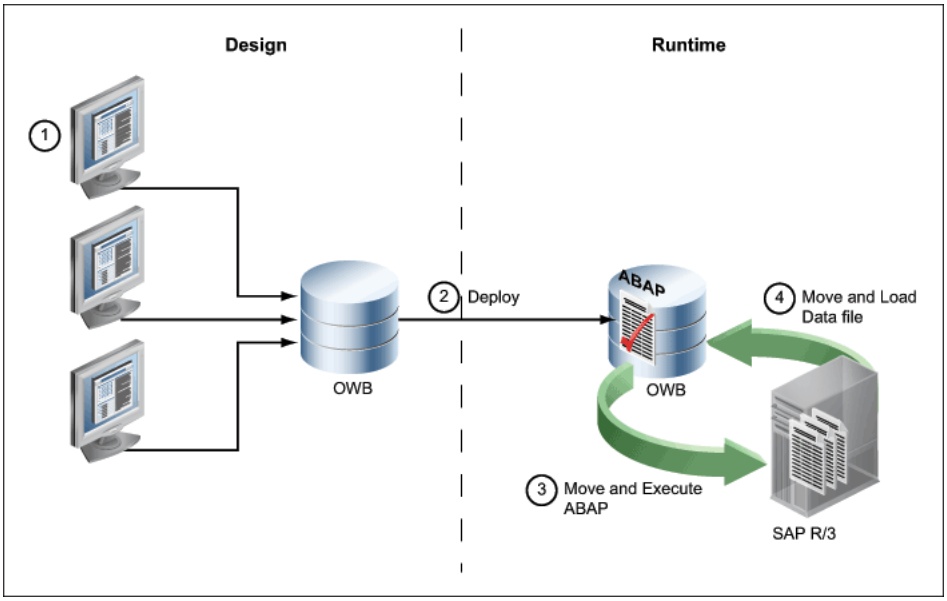

- "Automated System" on page 7-8
- "Semiautomated System" on page 7-9
- "Manual System" on page 7-11


## Automated System

In a completely automated system, as a Warehouse Builder user you have access to the predefined function module in the SAP system. This allows you to execute any ABAP code and retrieve data directly from the SAP system without being dependent on the SAP administrator.

Figure 7–2. displays a diagrammatic representation of the automated data retrieval mechanism.

***Figure 7–2 Automated Data Retrieval***




Because there is no dependence, you can automate the process of sending the ABAP code to the SAP system and retrieving the data file from the SAP system. Warehouse Builder will then use FTP to transfer the data file to the Warehouse Builder system, and load the target file with the retrieved data using SQL*Loader.

An automated system works as follows:

1. You design the extraction mapping and generate the ABAP code for this mapping.
2. Before deploying the mapping, ensure that you have set the following configuration parameters for the mapping:

- **ABAP Report Name**: The file that stores the ABAP code generated for the mapping.
- **SAP Location**: The location on the SAP system from where data is retrieved.
- **Data File Name**: Name of the data file to store the data generated by the execution of ABAP code.

Also ensure that you have provided the following additional connection details for the SAP location:

- **Execution Function Module**: Provide the name of the predefined SAP function module. Upon execution, this function module will take the ABAP report name as the parameter, and execute the ABAP code.
- **FTP Directory**: The directory on the Warehouse Builder system. The data file generated upon the execution of the function module will be sent using FTP to this directory.
- Also provide a user name that has write permissions on the FTP directory.

3. You then start the mapping. The following which the following tasks are automatically performed:
   - Warehouse Builder deploys the ABAP and uses RFC_ABAP_INSTALL_AND_RUN to both load the ABAP and execute it in SAP.

     The ABAP code is sent to the SAP system using a Remote Function Call (RFC).
4. In the SAP system, the code retrieves data from the source tables and creates a data file.

   This data file is stored in the location specified by Runtime parameter Staging File Directory.
5. Warehouse Builder uses FTP to transfer this data file back to the Warehouse Builder system.

   The file is stored in the location specified in the FTP Directory field.
6. Using SQL*Loader, Warehouse Builder loads the target table in the mapping with the data from the data file.

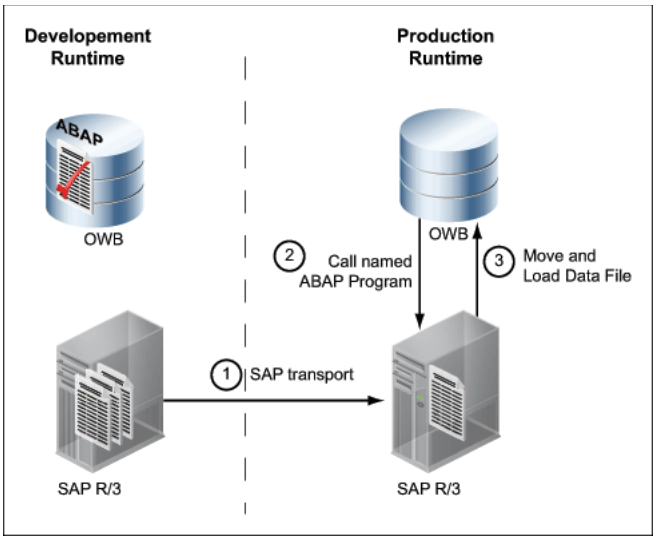
The advantage of this system is that you can create a fully automated end-to-end solution to retrieve SAP data. As a user, you just create the extraction mapping and run it from Warehouse Builder, which then creates the ABAP code, sends it to the SAP system, retrieves the resultant data file, and loads the target table with the retrieved data.

## Semiautomated System

In a semiautomated system, as a Warehouse Builder user, you do not have access to the predefined function module, and therefore cannot use this function module to execute ABAP code. You create an extraction mapping, deploy it, and then send the ABAP code to the SAP administrator who verifies the code before allowing you to run it in the SAP system.

Figure 7–3 displays a diagrammatic representation of a semi automated system.

***Figure 7–3 Semiautomated Implementation***




A semiautomated system works as follows:

1. You design the extraction mapping and generate the ABAP code for this mapping.
2. You then transport the ABAP code to the test system to test the code.
3. You then send the ABAP code to the SAP administrator, who loads it to the SAP repository.
4. The SAP administrator creates a new ABAP report name.
5. You can then call this ABAP report name to execute the ABAP code in the production environment.
6. Before you run the mapping in the SAP system, ensure that you have set the following configuration parameters for the mapping:
   - **ABAP Report Name**: The SAP administrator will provide the report name after verifying the ABAP code. You will then execute this ABAP file.
   - **SAP Location**: The location on the SAP system from where data is retrieved.
   - **Data File Name**: Name of the data file to store the data generated during execution of ABAP code.

   Also ensure that you have provided the following additional connection details for the SAP location:

   - **Execution Function Module**: Provide the name of the custom function module created by the SAP administrator. On execution, this function module takes the ABAP report name as the parameter, and executes the ABAP code. You must obtain the function module name from the SAP administrator.
   - **FTP Directory**: A directory on the Warehouse Builder system. The data file generated by the execution of the ABAP code is sent using FTP to this directory.
   - Also provide a user name that has Write permissions on the FTP directory.
7. In the production environment, when you run the mapping, Warehouse Builder generates the ABAP code and sends it to the SAP system using a Remote Function Call (RFC).

8. In the SAP system, the ABAP code is executed using the customized function module and a data file is generated.

   This data file is stored in the location specified by the Runtime parameter Staging File Directory.

9. Warehouse Builder uses FTP to transfer this data file back to the Warehouse Builder system.

   The file is stored in the location specified in the FTP Directory field.

10. Warehouse Builder uses SQL*Loader to load the target table with data from the data file.

## Manual System

In a manual system, your role as a Warehouse Builder user is restricted to generating the ABAP code for the mapping, and sending the ABAP code to the SAP administrator. The tasks involved in this system are:

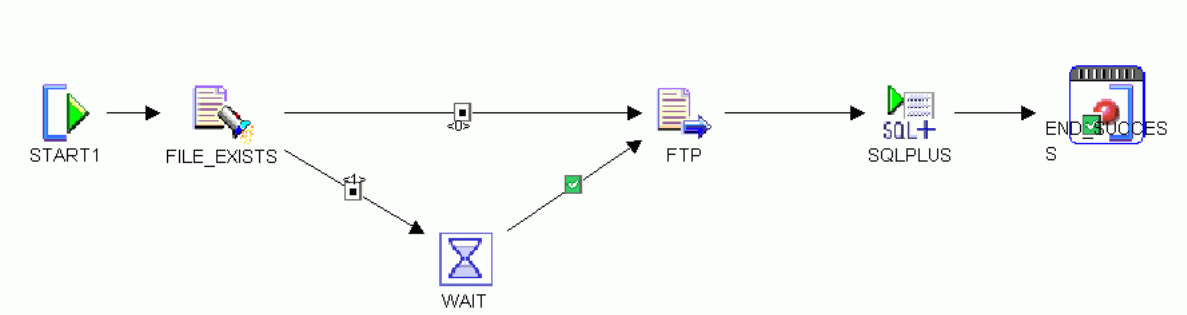
1. You create an extraction mapping, and generate the ABAP code for the mapping.
2. While designing the mapping, ensure that you specify the Data File Name to store the data file.
3. You send the ABAP code to the SAP administrator.
4. The SAP administrator executes the ABAP code in the SAP system.
5. On execution of the code, a data file is generated.

On the Warehouse Builder end, you can create a Process Flow to retrieve the data file. The process flow must contain the following activities.

1. A File Exists activity to check for the presence of the data file.
2. If the file exists, then an FTP activity transfers the file to the Warehouse Builder system.
3. If the file does not exist, then it must wait till the file is made available, and then perform an FTP.
4. Using SQL*Loader, the target table is loaded with data from the data file.

Figure 7–4 displays the process flow that retrieves the data file.

***Figure 7–4 Process Flow to Retrieve SAP Data***




In certain environments, the SAP administrator may not allow any other user to access the SAP system. In such cases, implementing the manual system may be the only viable option.

# Creating Code Template (CT) Mappings

Once you create or import a code template and deploy it, the template to perform a certain task on a certain platform is available in the workspace. To use this template to load or transform your data, you must create a mapping that uses this code template.

Some of the tasks that you can perform using code templates are:

- Integrate with heterogeneous databases such as DB2 or SQL Server by extracting data from these databases
- Leverage functionality beyond that of the current Code Template library. For example, you can construct new code templates to use Oracle Database functionality such as Data Pump to move data between Oracle systems at high speed.

You can also use code templates in situations where the code generated for PL/SQL mappings does not meet the requirements of your application.

### What are Code Template (CT) Mappings?

Mappings that contain an association with code templates are called *Code Template (CT) mappings*. Typically, they are used to extract or load data (both with and without transformations) from non-Oracle databases such as IBM DB2 and Microsoft SQL Server. You can also use Oracle Gateways to extract from and write to non-Oracle systems.

To extract data from an Oracle Database and transform and load it into another Oracle Database, you can either use Code Template mappings or create mappings under the Mappings node of the Oracle target module.

### When Can I Use Code Template (CT) Mappings?

Use Code Template mappings to extract data from, transform, or load data into Oracle and non-Oracle databases using code templates.

When moving data between Oracle databases, the main reason to use CT mappings is moving data using technologies other than database links. Code templates can be used to implement bulk data movement based on functionality such as Data Pump.

### Where Are Code Template Mappings Defined?

To create a Code Template mapping, use the Template Mappings node under a project in the Projects Navigator. This node is used to include non-Oracle mappings (not PL/SQL, SQL*Loader, or ABAP).

When you create a CT mapping, the Mapping Editor contains two tabs: Logical View and Execution View. Use the Logical View to define the mapping by adding mapping operators and creating data flows between operators. Use the Execution View to define execution units that specify how the mapping should be executed. For more information about execution units, see "Defining Execution Units" on page 7-19.

### What Operators Can Be Used in Code Template Mappings?

You can use any mapping operator, except the ones listed in "Mapping Operators Only Supported in Oracle Target CT Execution Units" on page 7-16, in CT mappings.

You can also use pluggable mappings in CT mappings. However, ensure that the pluggable mappings do not contain any of the operators listed in "Mapping Operators Only Supported in Oracle Target CT Execution Units" on page 7-16.

### What are the Types of Code Templates?

Code templates in Warehouse Builder are classified into the following categories:

- Load Code Template (Load CT)
- Integration Code Template (Integration CT)
- Control Code Template (Control CT)
- Change Data Capture Code Template (CDC CT)
- Oracle Target Code Template (Oracle Target CT)
- Function Code Template (Function CT)

For more details about the types of code templates, see Chapter 29, "Warehouse Builder Code Template Tools and Substitution Methods Reference".

## About Prebuilt Code Templates Shipped with Warehouse Builder

Warehouse Builder includes some prebuilt code templates that you can use in CT mappings to perform data transformations. These code templates, defined to perform certain ETL tasks on the specified source or target, are available under the BUILT_IN_CT node under the Public Code Templates node of the Globals Navigator.

Table 7–2 provides a brief description of the code templates supplied by Warehouse Builder and details any restrictions in their usage. Use specific Load CTs for your target staging area whenever possible as they are more optimized for performance. For example, if you are loading to an Oracle database, use LCT_FILE_TO_ORACLE_SQLLDR or LCT_FILE_TO_ORACLE_EXTER_TABLE instead.

For more details about these code templates, see the Oracle Data Integrator (ODI) documentation set. In ODI, code templates are called knowledge modules.

***Table 7–2 Prebuilt Code Templates Supplied by Warehouse Builder***

| Code Template Name | Code Template Type | Description |
|---|---|---|
| LCT_FILE_TO_ORACLE_EXTER_TABLE | Load CT | Loads data from a file to an Oracle Database staging area using the EXTERNAL TABLE SQL command.<br>This CT is more efficient than the LCT_FILE_TO_SQL when dealing with large volumes of data. However, the loaded file must be accessible from the Oracle Database computer. |
| LCT_FILE_TO_ORACLE_SQLLDR | Load CT | Loads data from a file to an Oracle Database staging area using the native SQL*LOADER command line utility. Because it uses SQL*LOADER, this CT is more efficient than LCT_FILE_TO_SQL when dealing with large volumes of data. |
| LCT_FILE_TO_SQL | Load CT | Loads data from an ASCII or EBCDIC file to any SQL-compliant database used as a staging area.<br>Consider using this Load CT if one of your source data stores is an ASCII or EBCDIC file. |
| LCT_ORACLE_TO_ORACLE_DBLINK | Load CT | Loads data from an Oracle database to an Oracle staging area database using the native database links feature. |
| LCT_SQL_TO_ORACLE | Load CT | Loads data from any Generic SQL source database to an Oracle staging area. This Load CT is similar to the standard LCT_SQL_TO_SQL, except that you can specify some additional specific Oracle Database parameters. |
| LCT_SQL_TO_SQL | Load CT | Loads data from a SQL-compliant database to a SQL-compliant staging area. |

***Table 7–2 (Cont.) Prebuilt Code Templates Supplied by Warehouse Builder***

| Code Template Name | Code Template Type | Description |
|---|---|---|
| LCT_SQL_TO_SQL_ROW_BY_ROW | Load CT | Loads data from a SQL-compliant database to a SQL-compliant staging area. This CT uses Jython scripting to read selected data from the source database and write the result into the staging temporary table created dynamically. |
| ICT_ORACLE_INCR_UPD | Integration CT | Loads your Oracle target table, in incremental update mode, to insert missing records and to update existing ones.<br>Inserts and updates are done in bulk set-based processing to maximize performance. You can also perform data integrity checks by invoking the Control CT.<br>**Note:** When you use this Integration CT, the following restrictions apply:<br>■ The Loading Type property of the target Table operator should be set to either INSERT_UPDATE or UPDATE_INSERT.<br>■ A unique key or primary key must be defined for the target Table operator. |
| ICT_ORACLE_INCR_UPD_MERGE | Integration CT | Loads your Oracle target table, in incremental update mode, to insert missing records and to update existing ones.<br>Inserts and updates are performed by the bulk set-based `MERGE` statement to maximize performance. It also enables performing data integrity checks by invoking the Control CT.<br>**Note:** When you use this Integration CT, the following restrictions apply:<br>■ The Loading Type property of the target Table operator should be set to either INSERT_UPDATE or UPDATE_INSERT.<br>■ A unique key or primary key must be defined for the target Table operator. |
| ICT_ORACLE_INCR_UPD_PL_SQL | Integration CT | Loads your Oracle target table to insert missing records and to update existing ones.<br>Use this CT if your records contain long or binary long object (BLOB) data types. Avoid using this CT to load large volumes of data because inserts and updates are performed in row-by-row PL/SQL processing.<br>**Note:** When you use this Integration CT, the following restrictions apply:<br>■ The Loading Type property of the target Table operator should be set to either INSERT_UPDATE or UPDATE_INSERT.<br>■ A unique key or primary key must be defined for the target Table operator. |
| ICT_ORACLE_SCD | Integration CT | Loads a Type 2 Slowly Changing Dimension.<br>This CT relies on the Slowly Changing Dimension metadata set on the target table to determine which records should be inserted as new versions or updated as existing versions. |
| ICT_SQL_CONTROL_APPEND | Integration CT | Loads your SQL-compliant target table in replace/append mode, with or without data integrity check.<br>When flow data must be checked using a Control CT, this CT creates a temporary staging table before invoking the Control CT. |

*Table 7–2 (Cont.) Prebuilt Code Templates Supplied by Warehouse Builder*

| Code Template Name | Code Template Type | Description |
|---|---|---|
| ICT_SQL_INCR_UPD | Integration CT | Loads your SQL-compliant target table, in incremental update mode, to insert missing records and to update existing ones.<br>You can also perform data integrity checks by invoking the Control CT. Because not all databases support the same bulk update syntax, updates are done row by row.<br>**Note:** When you use this Integration CT, the following restrictions apply:<br>▪ The Loading Type property of the target Table operator should be set to either INSERT_UPDATE or UPDATE_INSERT.<br>▪ A unique key or primary key must be defined for the target Table operator. |
| ICT_SQL_TO_FILE_APPEND | Integration CT | Integrates data in a target file from any SQL-compliant staging area in replace mode. |
| ICT_SQL_TO_SQL_APPEND | Integration CT | Enables you to use a staging area different from the target. It integrates data in a target SQL-compliant table from any SQL-compliant staging area in replace mode. |
| CCT_Oracle | Control CT | Checks for data integrity against constraints defined on an Oracle table. Rejects invalid records in the error table created dynamically. Can be used for static controls as well as flow controls. |
| CCT_SQL | Control CT | Checks for data integrity against constraints defined on a SQL-compliant database.<br>Rejects invalid records in the error table created dynamically. Can be used for static controls as well as flow controls. |
| JCT_DB2_UDB_CONSISTENT | CDC CT | Creates the infrastructure required for consistent Change Data Capture on IBM DB2 UDB tables using triggers. |
| JCT_DB2_UDB_SIMPLE | CDC CT | Creates the infrastructure required for simple Change Data Capture on IBM DB2 UDB tables using triggers. |
| JCT_MSSQL_CONSISTENT | CDC CT | Creates the journalizing infrastructure for consistent journalizing on Microsoft SQL Server tables using triggers.<br>Enables consistent Change Data Capture on Microsoft SQL Server. |
| JCT_MSSQL_SIMPLE | CDC CT | Creates the journalizing infrastructure for simple journalizing on Microsoft SQL Server tables using triggers.<br>Enables simple Change Data Capture on Microsoft SQL Server. |
| JCT_ORACLE_10G_CONSISTEN_MINER | CDC CT | Enables consistent Change Data Capture on Oracle tables. Creates the journalizing infrastructure for consistent journalizing on Oracle 10*g* tables. Changed data is captured by the Oracle 10*g* LogMiner-specific utility. |
| JCT_ORACLE_11G_CONSISTEN_MINER | CDC CT | Enables consistent Change Data Capture on Oracle tables. Creates the journalizing infrastructure for consistent journalizing on Oracle 11*g* tables. Changed data is captured by the Oracle 11*g* LogMiner-specific utility. |
| JCT_ORACLE_9I_CONSISTENT_MINER | CDC CT | Enables consistent Change Data Capture on Oracle tables. Creates the journalizing infrastructure for consistent journalizing on Oracle 9*i* tables. Changed data is captured by the Oracle 9*i* LogMiner-specific utility. |
| JCT_ORACLE_CONSISTENT | CDC CT | Enables consistent Change Data Capture on Oracle tables. Creates the journalizing infrastructure for consistent Change Data Capture on Oracle tables using triggers. |
| JCT_ORACLE_CONSISTENT_UPD_DATE | CDC CT | Enables consistent Change Data Capture on Oracle tables. Creates the infrastructure for consistent Change Data Capture on Oracle tables using a source tables column that indicates the last update date. |
| JCT_ORACLE_SIMPLE | CDC CT | Enables simple Change Data Capture on Oracle tables. Creates the journalizing infrastructure for simple Change Data Capture on Oracle tables using triggers. |

### Limitations of Using Certain Prebuilt Code Templates

- When you use ICT_ORACLE_INCR_UPD_MERGE, sequences are not supported.
- When you use ICT_SQL_CONTROL_APPEND in a mapping, an ORDER BY clause associated with this CT does not work. No error message is displayed during the execution of a CT mapping containing this CT. However, the rows are not ordered as specified in the ORDER BY property.
- When you use Incremental Update Integration CTs, the Loading Type property of the target Table operator should be set to either INSERT_UPDATE or UPDATE_INSERT. Also, the target Table operator must have Unique key or Primary key defined on it.
- ICT_SQL_TO_SQL_APPEND uses two different credentials (one credential to the source schema and another credential to the target schema as defined in the location) to perform the loading. As a result, the map can be executed successfully without a permission problem.
- ICT_SQL_CONTROL_APPEND uses a single credential (the credential to connect to the target schema) to perform the loading. In other words, the security behavior is similar to existing Warehouse Builder PL/SQL mapping. As a result, if the target schema has not been granted the permission to access the source schema, an "Insufficient privileges" error will be reported.

  In general, Oracle Data Integrator Knowledge Modules (KMs) with "multiple connections" property set to true in its KM falls into the first category described above. Please refer to Oracle Data Integrator documentation for details.

## Mapping Operators Only Supported in Oracle Target CT Execution Units

Certain transformation operators are designed to leverage functionality provided by the Oracle Database. This functionality is not available in other heterogeneous databases. Thus, you cannot assign execution units that contain these operators directly to Load CTs or Integration CTs. These operators are only supported if you add them to an execution unit that has an Oracle Target CT assigned to it.

The list of operators that you cannot use directly in CT mappings, if the execution unit containing these operators is associated with an Integration CT or Load CT, is as follows:

- Anydata Cast
- Construct Object
- Cube
- Dimension
- Expand Object
- LCR Cast
- LCR Splitter
- Lookup
- Mapping Input Parameter
- Mapping Output Parameter
- Match Merge
- Name and Address

- Pivot
- Post-Mapping Process
- Pre-Mapping Process
- Queue
- Set Operation
- Sorter
- Splitter
- Table Function
- Unpivot
- Varray Iterator

#### Using Restricted Mapping Operators in Code Template Mappings

Execution units that are associated with Oracle Target CTs enable you to use restricted mapping operators in CT mappings. You can use the operators listed in "Mapping Operators Only Supported in Oracle Target CT Execution Units" on page 7-16 in execution units, if the execution unit containing these operators is associated with an Oracle Target CT. Hybrid mappings can be constructed which leverage flexible integration capabilities using the loading code templates in addition with the powerful transformation capabilities supported through the use of the Oracle Target CTs.

Certain operations that cannot be performed using CT mappings can be performed using traditional Warehouse Builder mappings that are deployed as PL/SQL packages. You can perform such mappings as separate execution units.

## Steps to Perform ETL Using Code Template Mappings

Performing ETL using CT mappings involves the following steps:

1. (Optional) Creating Template Mapping Modules

   If you have not already done so, create a Mappings module that will contain the mapping.

2. Creating Mappings Using Code Templates
3. Defining Execution Units

   **Note:** After you define execution units, you can optionally configure the CT mapping parameters as described in the following sections:

   - "Configuring Mappings Reference" on page 24-1
   - "Configuring Code Template (CT) Mappings" on page 24-12

4. Starting the Control Center Agent (CCA)
5. Validating Code Template Mappings
6. Generating Code Template Mappings
7. Deploying Code Template Mappings
8. Executing Code Template Mappings

After you execute the CT mapping, you can view the execution results as described in "Viewing Execution Results for Code Template Mappings" on page 7-27.

You can also audit errors caused during the execution of the CT mapping as described in "Auditing the Execution of Code Template Mappings" on page 7-31.

## Creating Template Mapping Modules

A template mapping module is a container for the mappings that use code templates. You can create a mapping that contains a code template only from the Template Mappings node of the Design Center. Similar to other modules in Warehouse Builder, each template mapping module is associated with a location that specifies where the mappings in this module should be deployed. Mappings containing code templates are deployed to the Warehouse Builder Control Center Agent.

**To create a template mapping module:**

1. In the Projects Navigator, expand the project node under which you want to create a template mapping module.
2. Right-click the Template Mappings node and select **New Mapping Module**.

   The Create Module Wizard is displayed.
3. On the Welcome page, click **Next**.
4. On the Name and Description page, enter values for the following fields and click **Next**.

   **Name:** Enter the name of the template mapping module. The name should conform to the Warehouse Builder naming conventions.

   **Description:** Enter an optional description for the template mapping module.

   **Select the Module Status:** Select the status as Development, Quality Assurance, or Production.
5. On the Connection Information page, specify the details of the location pointing to the Control Center Agent (CCA) to which the CT mappings are deployed. Click **Next**.

   If you previously created a location corresponding to the Control Center Agent, select this location using the **Location** list. You can also use the default location corresponding to the CCA, DEFAULT_AGENT, that is created by Warehouse Builder.

   To create a location, enter the following details for the Control Center Agent.

   **Username:** User name for the OC4J user that you use to deploy to the CCA. To deploy to the Control Center Agent that is installed with Oracle Warehouse Builder, use oc4jadmin as the user name.

   **Password:** Password for the OC4J user that you use to deploy to the CCA.

   **Host:** The hostname of the computer on which the Control Center Agent is installed.

   **Port:** The value of the RMI port used by the OC4J server.

   **Port Type:** To deploy to the agent location associated with the CCA that is installed along with Oracle Warehouse Builder, use RMI as the port type.

   The other options you can choose for port type are OPMN and RMIS.

**Instance:** Name of the OC4J instance corresponding to the CCA. To deploy to the default CCA installed with Oracle Warehouse Builder, leave this field blank.

**Application Name:** Name of the application to which CT mappings should be deployed. To deploy to the default CCA installed with Oracle Warehouse Builder, use jrt as the application name.

6. On the Summary page, review the information that you entered in the wizard. Click **Finish** to create the template mapping module. Click **Back** to modify any entered values.

The Template mapping module is created and added to the navigator tree under the project.

## Creating Mappings Using Code Templates

To use the functionality defined by code templates in your environment, you create a Code Template (CT) mapping. The process to create a CT mapping is similar to a regular PL/SQL mapping, except for the additional step of defining execution units and associating them with code templates.

Every CT mapping must belong to a mapping module.

---

**Note:** Do not use CT mappings with Joiner Input Roles if the join is performed on 2 or more tables.

---

**To create a CT mapping:**

1. In the Projects Navigator, expand the project node and then the Template Mappings node under which you want to create a CT mapping.
2. Right-click the mapping module in which you want to create the CT mapping and select **New Mapping**.

   The Create Mapping dialog box is displayed.
3. Enter the name and an optional description for the CT mapping and click **OK**.

   The Mapping Editor for the CT mapping is displayed.
4. On the Logical View tab, add the required operators and establish data flows that perform the required data transformation.

   For more information about how to add operators and establish data flows between them, see Chapter 5, "Creating PL/SQL Mappings".

## Defining Execution Units

The Execution View tab of the Mapping Editor enables you to define execution units. Use execution units to break up your mapping execution into smaller, related units and to associate a part of your mapping with a code template. Warehouse Builder generates code separately for each execution unit that you define.

The Execution View tab of the Mapping Editor displays the operators and data flows from the Logical View in an iconized form. You cannot edit operators or create data flows in the Execution View. You must perform these operations using the Logical View. Create execution units as defined in "Creating Execution Units" on page 7-20.

**Note:** If you do not explicitly assign a code template to an execution unit, Warehouse Builder assigns a default code template to the execution unit. For more details about default code templates, see "Default Code Template for An Execution Unit" on page 7-22.

### Execution View Menu and Toolbars

When you select the Execution View tab, the Design Center displays an additional menu called Execution and an Execution toolbar. Use these to:

- Create and delete execution units
- Define default execution units
- Associate code templates with execution units

### Creating Execution Units

Create an execution unit for a group of operators for which code generation and execution will be controlled by a specific code template.

**To create an execution unit:**

1. In the Execution View of the mapping, select the operators that you want to group into an execution unit.

   You can do this by drawing a rectangle around the operators. Hold down the mouse button at the top of the first operator, drag the mouse so that you cover the required operators and then release the mouse button. Alternatively, hold down the Ctrl key and click the headers of all the operators that you want to include in an execution unit.

2. From the Execution menu, select **Create Execution Unit**. Or click **Create Execution Unit** on the Execution View toolbar.

   Warehouse Builder creates an execution unit for the selected operators and assigns a default name to it. To rename the execution unit, right-click the name and select **Open Details**. In the Edit Execution Unit dialog box, enter the new name for the execution unit.

**To associate a code template with an execution unit:**

1. If the Code Template panel is not displayed, from the View menu, select **Code Template**.

   The Code Templates tab is displayed in the Log window panel.

2. In the Execution View tab of the Mapping Editor, select the execution unit with which you want to associate a code template.

   **See Also:** "How Warehouse Builder Displays Code Templates that Can be Associated with Execution Units" on page 7-23

3. In the Code Templates tab, use the list to select the code template with which the selected execution unit should be associated.

   The code templates displayed in the list depend on the source and target platforms of the operators in the selected execution unit. They also depend on the nature of the execution unit. For example, you can associate a Load CT with an execution unit that does not contain any operators bound to data objects. You can associate

an Integration CT with an execution unit that contains one target operator bound to a repository data object.

### Adding Operators to an Execution Unit

**To add operators to an execution unit:**

1. In the Execution View of the Mapping Editor, select both the execution unit and the operators that you want to add to the execution unit.

   You can select multiple objects by holding down the Ctrl key while selecting objects.

2. From the Execution menu, select **Add Operator to Execution Unit**. Or click **Add Operator to Execution Unit** on the Execution View toolbar.

### Adding Operators to Multiple Execution Units

An operator may appear in more than one execution unit, such as when it is a target in one execution unit and a source in another.

For example, you have a Table operator `cust_tab` in the execution unit `CUST_EXEC_UNIT`. You can copy `cust_tab` to another execution unit `EX_UNIT_2` (which already exists) by selecting both `cust_tab` and `EX_UNIT_2` and then clicking **Add Operator to Execution Unit**. A copy of `cust_tab` is added to `EX_UNIT_2`.

Consider the Table operator `cust_tab`, which is a target in the execution unit `CUST_EXEC_UNIT`. Select `cust_tab` and then click **Create Execution Unit** to create an execution unit containing a copy of `cust_tab`. The label of this copy will be <cust_tab>, using the angle brackets as a visual cue that the operator appears in multiple execution units.

### Removing Operators from an Execution Unit

**To remove operators from an execution unit:**

1. In the Execution View of the Mapping Editor, select the operator or operators that you want to remove from an execution unit.

2. From the Execution menu, select **Remove Operator from Execution Unit**. Or click **Remove Operator from Execution Unit** on the Mapping Editor toolbar.

   The selected operators are removed from the execution unit and displayed separately in the Execution View.

---

**Note:** You cannot remove an operator from an execution unit when it is the only operator in the execution unit.

---

### Removing Execution Units

When you remove an existing execution unit, the operators that were part of the execution unit are not associated with any execution unit. You may need to associate these operators with other execution units, if required.

Note that if you remove execution units, you must regenerate and deploy updated code for the mapping before your changes take effect.

**To remove an execution unit:**

1. In the Execution View of the Mapping Editor, select the execution unit that you want to remove.

2. From the Execution menu, select **Remove Execution Unit**. Or click **Remove Execution Unit** on the Execution View toolbar.

   The execution unit is removed. Any operators that were contained in this execution unit are displayed individually, instead of grouped under the execution unit.

### Creating Default Execution Units

You can create a default set of execution units by clicking **Default Execution Units** or by selecting **Default Execution Units** from the Execution menu. Warehouse Builder first removes all existing execution units and then creates a new set of execution units such that all operators are assigned to some execution unit.

The operators are assigned to execution units based on factors such as their location. Operators that are at a different location will be assigned to different execution units. If all operators are at the same location, then the default may consist of a single execution unit containing all operators.

The names of the default execution units depend on whether the location associated with the operators in the execution units is known. When the execution unit location (location associated with operators in the execution unit) is known, the name of the default execution unit will be set to the location name followed by "_EU". If the location associated with the execution unit is not known, then the execution unit name starts with "MAP_EX_UNIT_",

### Default Code Template for An Execution Unit

If you do not explicitly associate a code template with an execution unit, during code generation, Warehouse Builder assigns a default code template. The default code template depends on the platform of the target operators. You can define default code templates for each platform.

For example, you can define a default Load CT, Integration CT, and Oracle Target CT for the Oracle Database platform. When you do not assign a code template to an execution unit that contains operators referencing Oracle Database objects, the Warehouse Builder Code Generator performs the following steps:

- Identifies the type of code template that should be used for that particular execution unit
- Retrieves the default code template that should be assigned to the execution unit using the platform of the location with which the execution unit is associated
- Assigns the retrieved code template to the execution unit

> **Note:** You can define default code templates for a platform only using OMB*Plus. For more information about OMB*Plus, see *Oracle Warehouse Builder API and Scripting Reference*.

If no default code templates are defined for a particular platform, Warehouse Builder picks a code template from the available code templates and assigns it to the execution unit. It then updates the platform definition and assigns the selected code template as the default code template definition for that platform.

A default code template can be assigned to an execution unit only if the CT mapping containing the code template and the default code template belong to the same project.

### How Warehouse Builder Displays Code Templates that Can be Associated with Execution Units

When associating a CT with execution units, the CTs displayed in the UI list are filtered by the source and the target platforms. For example, if the execution unit containing your source tables is an Oracle database, the CTs available for selection for this execution unit are ones that can be used for Oracle sources.

In certain simple mappings, you may create a single execution unit that contains the source and target operators. In this case, the list of available Integration CTs is limited by the following:

- Source Platform
- Platform
- The value set for the Multi-Connections property of the CT

## Starting the Control Center Agent (CCA)

The Control Center Agent is the agent that runs the code templates in the OC4J server. You must start the Control Center Agent before you deploy code templates or CT mappings.

### Starting the Control Center Agent (CCA)

On Windows, start the Control Center Agent by running the `ccastart.bat` script located in the *`OWB_ORACLE_HOME`*`/owb/bin/win32` directory.

On Unix, start the Control Center Agent by running the `ccastart` file located in the *`OWB_ORACLE_HOME`*`/owb/bin/unix` directory.

---

**Note:** It is recommended that you not run `ccastart.bat` or `ccastart` multiple times to start multiple CCA instances. If you need to run multiple CCA instances, then install separate CCA instances on the Application Server using the installation steps required to install a CCA on the Application Server.

---

### Stopping the Control Center Agent (CCA)

On Windows, stop the Control Center Agent by running the `ccashut.bat` script located in the *`OWB_ORACLE_HOME`*`/owb/bin/win32` directory. This script takes the password of the oc4jadmin user as an input parameter.

On Unix, stop the Control Center Agent by running the `ccashut` file located in the *`OWB_ORACLE_HOME`*`/owb/bin/unix` directory. This script takes the password of the oc4jadmin user as an input parameter.

## Validating Code Template Mappings

Validating a Code Template (CT) mapping verifies the metadata definitions and configuration parameters to ensure that they are valid according to the rules defined by Warehouse Builder. As part of validation, Warehouse Builder verifies the assignment of operators to execution units. It also ensures that the code generated to perform the ETL task defined in the CT mapping can be generated correctly.

During validation, Warehouse Builder checks the following rules:

- An operator cannot be connected to two or more downstream operators in different execution units.

- An execution unit cannot contain non-Oracle source or target operators and other restricted mapping operators.
- Only Oracle Target CTs can be assigned to an execution unit with restricted operators.
- A transformation or data target operator containing more than one incoming connection cannot exist in more than one execution unit.
- An execution unit cannot have outgoing connections to more than one execution unit.

## Generating Code Template Mappings

Generating a Code Template (CT) mapping creates the scripts necessary to perform the ETL task in the Warehouse Builder workspace. For CT mappings, Tcl scripts are generated.

When you generate a CT mapping, Warehouse Builder first validates it. After the validation is successful, the CT mapping is generated. Generation produces the following scripts, which are part of an `.ear` file:

- Variable script, `var.tcl`

  This script contains the variables that will be substituted in the substitution method calls of the code template script. The script collects all the defined mapping variables into a single Tcl variable that contains a map of name to value.

  This script stores all metadata, as described by the mapping, and this metadata will be used by the code template (via its substitution methods). During execution of the CT mapping, the substitution methods in the code template are executed in the Control Center Agent and their return values are computed from the appropriate metadata in the variable script. The metadata variable names do not directly match the names or patterns defined in the substitution methods; the implementation of the substitution methods performs that translation.

- Driver script, `driver.tcl`

  The driver script first invokes the variable script to load the metadata variable values into the Jacl interpreter of the Control Center Agent. Next, for each execution unit, it invokes the main method of the code template associated with that execution unit.

  The order of execution of the code templates is automatically derived from the topology of the mapping in the Execution View.

If any errors occurred during generation, use the Mapping Editor to correct them and then regenerate the CT mapping.

### Viewing Generation Results

The results of the generation are displayed in a Results tab in the Log window. The Results tab displays a parent node that indicates whether the generation was successful. Under this node is one that uses the same name as the CT mapping that you generated. This node contains the validation results under the Validation node and the generated scripts under the Scripts node. Expand the node containing the results you want to view.

### Viewing Generated Code

The Scripts node under the generation results contains the `var.tcl` and `driver.tcl` scripts that contain the code generated to create the CT mapping.

To view the generated code, expand the Scripts node. Right-click `var.tcl` or `driver.tcl` and click **View Script**. Or double-click the `.tcl` files to display their contents in a new tab in the Mapping Editor.

> **Note:** If the Mapping Editor already displays a tab that contains generated code for a CT mapping, ensure that you close this tab before you view the generated code for another CT mapping. If you do not close the tab containing previous generated code, then you may see conflicting results for the generated code.

### Sample Code Generated for CT Mappings

Following are some code examples from a code template and its associated metadata script, showing the substitution. The following code block displays a single procedural step of a code template, creating a temporary flow table.

```
proc 3_CREATE_FLOW_TABLE_I__main {  }
{
# Input Flow Parameters
  variable SRC_LOCATION
  variable TGT_LOCATION
  variable KM_PARAMS
  variable LOG_LEVEL
  variable INSERT
  variable UPDATE
  variable COMMIT
  variable SYNC_JRN_DELETE
  variable FLOW_CONTROL
  variable RECYCLE_ERRORS
  variable STATIC_CONTROL
  variable TRUNCATE
  variable DELETE_ALL
  variable CREATE_TARG_TABLE
  variable FLOW_TABLE_OPTIONS
  variable ANALYZE_TARGET
  variable OPTIMIZER_HINT

# Output parameters
  variable EXIT_CODE
  variable RETURN_VALUE {}

# Global variables
  global errorInfo
  global g_iud

        set g_iud ""
        set tgt_stmt [process "create table <%=snpRef.getTable(\"L\", \"INT_NAME\",
\"W\")%>\n(\n<%=snpRef.getColList(\"\", \"\\t\[COL_NAME\]\\t\[DEST_WRI_DT\] NULL\", \",\\n\", \"\",
\"\")%>,\n\tIND_UPDATE \tchar(1)\n)\n<%=snpRef.getUserExit(\"FLOW_TABLE_OPTIONS\")%>"]
        puts $tgt_stmt
        execjdbc $tgt_stmt "TGT_AC" "$TGT_LOCATION" "" "true" "false" "false"
}
```

Notice that a target SQL statement, variable `tgt_stmt`, is assigned an actual SQL statement which is then executed using the `execjdbc` procedure call. The `execjdbc` tcl procedure makes a Java call to execute the statement through JDBC. The target statement is a string produced by the `process` procedure. The <%...%> delimiters require special processing to substitute required components into the SQL string. The

snpRef tag is the prefix for a substitution method callout. snpRef.getTable is replaced by the actual table name. snpRef.getColList is another universal method to retrieve the list of table columns participating in the DML. In addition to snpRef, odiRef is supported (as in Oracle Data Integrator 10.2).

The substitution method (snpRef) calls are completed by a Warehouse Builder Tcl module which extracts the associated data from a variable of the metadata script. The following is an example of a metadata script section showing the table name and column list:

```
set M1_params {
  {CKM_CALL ""}
  {COLUMN_GENERIC_DATATYPE "NUMERIC VARCHAR"}
  {COLUMN_LIST "EMPLOYEES.EMPLOYEE_ID EMPLOYEES.LAST_NAME"}
  {COLUMN_LIST_ALIAS "EMPLOYEE_ID LAST_NAME"}
  {COLUMN_LIST_DATATYPE "NUMBER(6) VARCHAR2(25)"}
  {EXECUTION_UNIT_NAME "EX_UNIT_2"}
  {FROM_LIST "EMPLOYEES"}
  {FROM_LIST_ALIAS "EMPLOYEES"}
  {HAS_JRN "0"}
  {INSERT_COLUMN_LIST "EMPID ENAME"}
  {IS_DISTINCT "FALSE"}
  {JOURNAL_IN_CURRENT_SCHEMA "false"}
  {JOURNAL_IN_SOURCE_SCHEMA "false"}
  {JOURNAL_IN_STAGING_AREA "false"}
  {JRN_FILTER ""}
  {JRN_METHOD "NONE"}
  {JRN_TABLE "."}
  {KM_NAME "KM_IKM_ORACLE_INCREMENTAL_UPD"}
  {MAP_NAME "MAPPING_2"}
  {SELECT_STATEMENT "SELECT EMPLOYEES.EMPLOYEE_ID EMPLOYEE_ID,
         EMPLOYEES.LAST_NAME LAST_NAME
FROM
  EMPLOYEES  EMPLOYEES"}
  {SQL_STATEMENT "INSERT INTO TGT(EMPID, ENAME)
  (SELECT EMPLOYEES.EMPLOYEE_ID EMPLOYEE_ID,EMPLOYEES.LAST_NAME LAST_NAME
FROM
  EMPLOYEES  EMPLOYEES
  )
;"}
  {TARGET_COLUMN_DATATYPE "NUMBER(15) VARCHAR2(100)"}
  {TARGET_COLUMN_LIST "EMPID ENAME"}
  {TARGET_GENERIC_DATATYPE "NUMERIC VARCHAR"}
  {TARGET_NAME "TGT"}
}
```

## Deploying Code Template Mappings

To deploy a Code Template (CT) mapping, use the Control Center Manager. In the Control Center navigation tree, CT mappings are listed under the agent location that is associated with the mapping module containing the CT mappings.

Deploying a CT mapping copies the generated scripts to the CCA.

Before you deploy a CT mapping, ensure that you deploy all the code templates associated with it.

**To deploy a code template mapping:**

1. From the Design Center, open the Control Center by selecting **Control Center Manager** from the Tools menu.
2. In the Control Center navigation tree, expand the project node under which you created the CT mapping. Then expand the location node associated with the mapping module that contains the CT mapping.
3. Expand the mapping module node that contains the CT mapping.
4. Select the CT mapping to be deployed and, in the Object Details panel, select **Create** as the Deploy Action.
5. Click **Deploy**.

The Control Center Jobs panel contains a new entry for to the deployment job corresponding to the deployed CT mapping. If the deployment is successful, then the status displays a success message. If the deployment fails, then double-click the error message to view the details of the error.

## Executing Code Template Mappings

When you execute a Code Template (CT) mapping, the code template is used to perform the data integration task defined in the CT mapping. Before you execute a CT mapping, ensure that the CT mapping is deployed as described in "Deploying Code Template Mappings" on page 7-26.

To execute a CT mapping, in the Control Center Manager, select the CT mapping and click **Start**.

Or, from the Projects Navigator, right-click the CT mapping and select **Start**.

The CT mapping is executed and the ETL defined by it is performed.

Execution of CT mappings is controlled by a J2EE Runtime environment referred to as the CCA (Control Center Agent). The CCA is separate from the Runtime Platform. The CCA runs a mapping by executing Tcl/Java (Jacl) scripts. Because the execution is performed entirely by the CCA, it can be invoked separately from Service Oriented Architecture (SOA) interfaces.

---

**Note:** While executing complex CT mappings that could take more than a day to run, it is recommended that you split the job into smaller ones. The default transaction timeout for the OC4J is set to one day. If your job execution takes more than a day, then the execution will time out and unexpected errors may be encountered.

---

## Viewing Execution Results for Code Template Mappings

You can view execution results for a Code Template (CT) mapping by using the Results tab or Audit Information panel.

---

**Note:** The number of rows selected is not audited during CT mapping execution. Thus, the execution results do not contain the number of rows selected.

---

### Viewing Execution Results by Using the Results Tab

When you execute a CT mapping using the Design Center, a new Results tab is created in the Log window to display the CT mapping execution results. The Results

tab contains a node called Execution that contains the execution results. This displays details about the number of rows inserted, updated, and deleted during the CT mapping execution.

### Viewing Execution Results by Using the Audit Information Panel

The Audit Information panel enables you to view additional details about the CT mapping execution. You can also view results for previous executions of the CT mapping. To display the Audit Information panel, from the View menu, select **Audit Information**.

Expand the location node to which the CT mapping is deployed to view the jobs for this CT mapping. Because you can execute a CT mapping multiple times, each execution is represented by a job. Each job contains a node for execution units, which in turn contains a node for steps. Details such as number of rows inserted, updated, deleted, and selected are displayed. Use the Execute Statement tab to view the statement used to execute a step.

During the execution of a CT mapping or Change Data Capture process, the Audit Information panel displays the job and its child nodes, such as Execution Units (for CT mapping only), Tasks, and Steps, as they are being executed. The status for each job is also displayed. You can filter tasks for a Job in the Audit Information panel based on the status (such as Skipped, Warnings, and so on). The default is set to Filter None, which means that no filtering is performed. You can also sort Jobs by Timestamps.

Use the Message tab to view any message during the execution of the steps, such as an exception from the JDBC driver during the execution of a JDBC task.

#### Modes of Operation for the Audit Information Panel

The Audit Information panel can display audit information in the following modes:

- If a CT mapping is selected in the Projects Navigator, only the execution audit for that CT mapping is displayed.
- If you select a Function CT in the Projects Navigator, only execution audit information for the selected CT is displayed.
- If you select a module containing CDC tables in the Projects Navigator, the audit information for CDC is displayed.
- If you select an agent in the Locations Navigator, the execution audits for all the jobs executed in that agent (including CT mapping, CDC execution, and CT function deployment) are displayed.

## Setting Options for Code Templates in Code Template Mappings

When you use a code template in a CT mapping, the options that you can set for the code template depends on the type of code template.

Note that Warehouse Builder is not aware of the options that work with each platform or database. Thus, validating and generating your CT mappings does not automatically verify if the option that you set works with the platform or database associated with the code template. For example, Teradata does not support the TRUNCATE statement. Thus, if you set the Truncate property to true for a code template associated with a Teradata source or target, you will encounter errors while executing the CT mapping.

Table 7–3 describes the options that you can set for each type of code template in a CT mapping. You can add new options to code templates and use them in CT mappings.

*Table 7–3 Options for Code Templates in Code Template Mappings*

| Option Name | Description | Option applicable for Code Template Types |
|---|---|---|
| After ICT | Set to True to clean up work tables after the ICT has completed integration in the target. This property enables you to decide if you want to keep work tables after the mapping completes (primarily for debugging purposes). | Load CT |
| Commit | Set to True to indicate that a commit should be performed after the integration task is completed. | Integration CT |
| Create_targ_table | Set to True to create the target table. Set to False if the table already exists. | Integration CT |
| Delete_all | Set to True to delete all rows from the target table. | Integration CT |
| Delete_temporary_objects | Set to True to delete temporary objects created during the Integration CT execution, if any. | Integration CT<br>Load CT |
| Drop_check_table | Set to True to drop the check table. The check table contains statistics about the errors found. | Control CT |
| Drop_error_table | Set to True to delete the error table containing information about specific errors detected. | Control CT |
| Flow_Control | Set to True to activate flow control. Flow control detects errors prior to inserting data into the target table. | Integration CT<br>Control CT |
| Flow_table_options | Specify the options for flow table creation. | Integration CT |
| Insert | Set to True to indicate that the code template can insert new rows into the target table. | Integration CT |
| Deploy_files | Indicates if the CT always deploys the associated PL/SQL code.<br>Set to False to only deploy the PL/SQL code if the mapping does not exist in the database or if the mapping is not the same version as the deployed mapping. Set to True if a previous code template is creating the tables for this mapping. In general this should be set to false. | Oracle Target CT |
| Log_level | Represents the level of log reporting when the code template is executed. Valid values are between 0 and 5, with 5 representing the most detailed logging. | Integration CT<br>Load CT<br>CDC CT<br>Oracle Target CT |
| Static_control | Used for postintegration control. Detects errors on the target table after integration has been completed. | Integration CT |
| Target_location | Represents the Oracle location to which the PL/SQL mapping is to be deployed. | Oracle Target CT |
| Truncate | Set to True to truncate the target table before loading data. | Integration CT |
| Update | Set to True to indicate that the code template can update rows in the target table. | Integration CT |

## Setting Properties for Bound Operators in CT Mappings

For operators in CT mappings that are bound to data objects, you can set properties for the operator attributes. These attribute values are used in the substitution methods when the CT mapping is executed.

Operators, in CT mappings, that are bound to data objects have the following additional properties. These operators are listed under the Code Template Metadata Tags node in the Property Inspector.

### SCD

Use the SCD property to specify the role played by the attribute when using the Slowly Changing Dimension code template. The values you can set for this property are:

- **Surrogate Key:** The attribute is used as a unique record identifier (primary key) in the target table. You must map a sequence to this attribute.
- **Natural Key:** The attribute is mapped from the unique record identifier (business key of the record) in the source table.
- **Current Record Flag:** The attribute is used to identify the current active record. The current record has the flag set to 1 and old records have the flag set to 0. The attribute must be numeric and is loaded automatically, you do not need to map it.
- **Update Row on Change:** If a new value is loaded for this attribute, the value of its corresponding column is overwritten, for the record with the same natural key.
- **Add Row on Change:** If a new value is loaded for this attribute, a row with a new surrogate key but with the same natural key is inserted into the target table. The inserted record is marked as current.
- **Starting Timestamp:** The start time of the time period when the record is current. If the new row is inserted, then the Starting Timestamp of new record and Ending Timestamp of the old record are set to `SYSDATE`.
- **Ending Timestamp:** The end time of the time period when the record is current. For the current record, the ending timestamp column value is usually 01-01-2400.

### UD1

Set this property to True to include this attribute in code template functions using the UD1 tag.

### UD2

Set this property to True to include this attribute in code template functions using the UD2 tag.

### UD3

Set this property to True to include this attribute in code template functions using the UD3 tag.

### UD4

Set this property to True to include this attribute in code template functions using the UD4 tag.

### UD5

Set this property to True to include this attribute in code template functions using the UD5 tag.

### UPD

This property controls which columns are updated when Update Code Templates, such as ICT_SQL_ INCR_UPD or ICT_ORACLE_INCR_UPD_MERGE are used.

To specify which constraint to use for matching, use the Match by Constraint property of the target operator. If the UPD property is not set, then the operator's match by constraint key is used.

## Setting Configuration Properties for Code Template Mappings

You can set certain configuration properties for operators that are used in CT mappings. This section describes the configuration properties.

### Source File Operator

You can set the Source File Operator configuration property for Flat File operators that are used in CT mappings. This property identifies the Flat File operator with which the data file is associated.

For CT mappings that contain multiple Flat File operators, you must set the Source File Operator property. When your CT mapping contains only one Flat File operator, setting this property is optional.

Use the following steps to set the Source File Operator configuration property:

1. Right-click the CT mapping that contains the Flat File operators and select **Configure**.

   The Configuration tab for the CT mapping is displayed.

2. Click the Ellipsis on the SQL Loader Data Files property.

   The SQL Loader Data Files dialog box is displayed.

3. Select the file name on the left of the dialog box and, on the right, set the Source File Operator property.

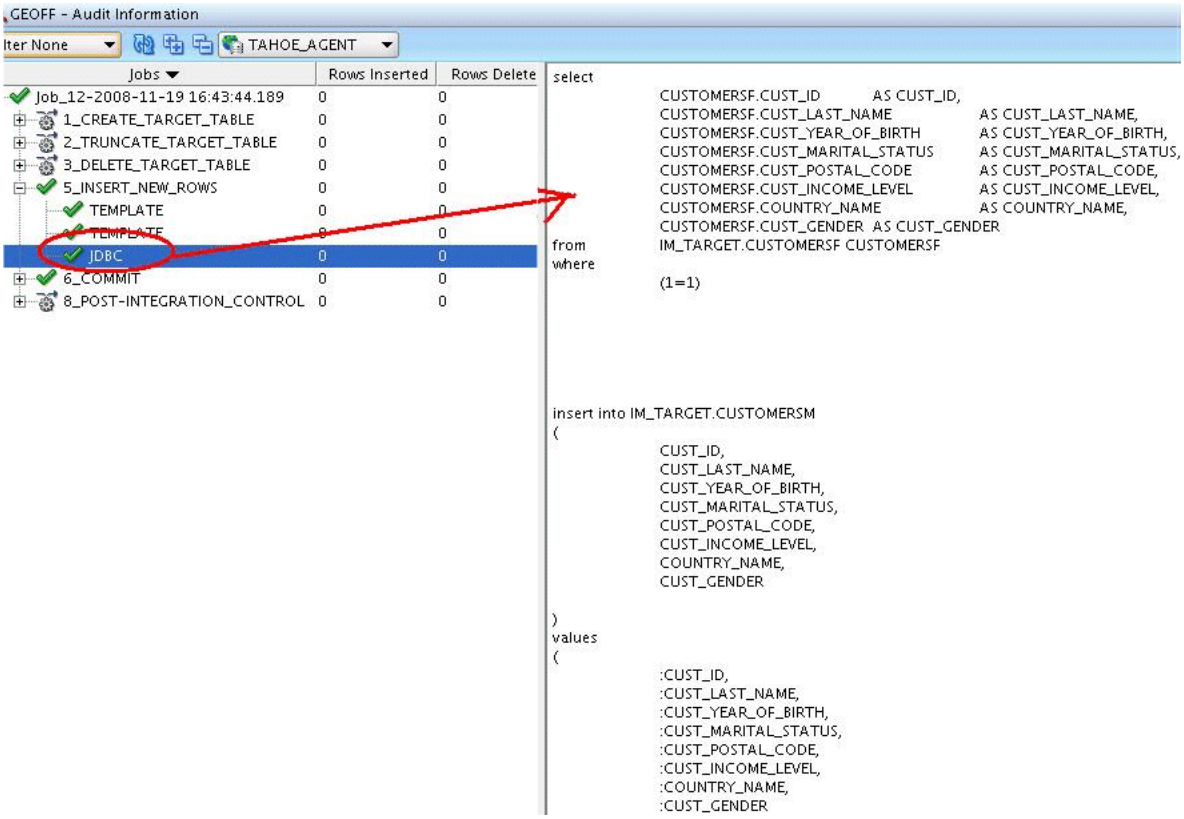
# Auditing the Execution of Code Template Mappings

The Audit Information panel in the Design Center provides detailed information about each task that was executed as part of a CT mapping execution. You can use this information to audit errors caused during the execution of CT mappings.

However, for the Audit Information panel to display execution details, you must set certain properties as described in "Prerequisites for Auditing Code Template Mappings" on page 7-32.

Figure 7–5 displays the Audit Information panel for a CT mapping execution.

***Figure 7–5 Audit Information Panel with Details about Executed Tasks***




The Audit Information panel displays the individual steps that are executed as part of the CT mapping execution. The details displayed for each step include the name of the task, the number of rows inserted, updated, or deleted for the task, and its status.

Detailed information about the steps within the CT mapping is displayed. For each step, you can see error messages, if any. You can also view the statements executed for each task. Expand the node representing each task to view details of the task execution.

Not all tasks are listed in the Audit Information panel. If the settings of a task exclude it from the audit, then the task is not displayed. For more information about including a code template task in audits, see "Prerequisites for Auditing Code Template Mappings" on page 7-32.

For example, in the Audit Information panel displayed in Figure 7–5, the CT mapping contains the following tasks: CREATE_TARGET_TABLE, TRUNCATE_TARGET_TABLE, DELETE_TARGET_TABLE, INSERT_NEW_ROWS, COMMIT, and POST_INTEGRATION_CONTROL. The statements executed as part of the INSERT_NEW_ROWS task are displayed in the panel on the right. Notice that task 4 is not listed in the Audit Information panel. This is because the settings of this task exclude it from the audit.

### Prerequisites for Auditing Code Template Mappings

For Warehouse Builder to provide detailed information about the execution of a CT mapping, you must set the following properties:

- Log Audit Level

Every task in a code template contains a property called Log Audit Level. You can set a value between 0 and 5 for this property. Setting a value of 0 means that the details of this task are not included in the audit. Set this property to a value between 1 and 5 to include the details for this task execution in the audit log.

- Log Level

  Every execution unit in a CT mapping contains a property called Log Level. This property represents the level of logging performed for the execution unit. You can set a value between 0 and 5 for this property. Set a value of 0 to exclude this execution unit from being logged. Set this property to a value between 1 and 5 to specify the level of detail used to log the execution of this execution unit.

### Steps to Audit the Execution of Code Template Mappings

1. In the Projects Navigator, select the CT mapping whose execution must be audited.
2. From the View menu, select **Audit Information**.

   The Audit Information panel is displayed in the Log window.
3. Expand the node that represents the CT mapping execution job.

   Separate nodes are displayed for each task in the CT mapping that is included in the audit.
4. Expand a task node to view the list of steps performed as part of this task.
5. To view the statements executed as part of a particular step, select the step and then select **Executed Statement**.

   The statements executed are displayed on the right.

## Using Code Template Mappings to Perform Change Data Capture (CDC)

In a typical data warehousing environment, you extract data from source systems, transform data, and then load it into the data warehouse. However, when the data in the source system changes, you must update the data warehouse with these changes. Change Data Capture (CDC) quickly identifies and processes only data that has changed and then makes this changed data available for further use. Warehouse Builder enables you to perform CDC by using CDC CTs.

### Types of Change Data Capture (CDC)

Warehouse Builder enables you to perform the following types of CDC:

- Consistent

  Consistent Change Data Capture ensures the consistency of the captured data. For example, you have the `ORDER` and `ORDER_LINE` tables (with a referential integrity constraint based on the fact that an `ORDER_LINE` record should have an associated `ORDER` record). When the changes to `ORDER_LINE` are captured, the associated `ORDER` change will also be captured, and vice versa.

  The set of available changes for which consistency is guaranteed is called the Consistency Window. Changes in this window should be processed in the correct sequence (`ORDER` followed by `ORDER_LINE`) by designing and sequencing integration interfaces into packages.

Although consistent Change Data Capture is more powerful, it is more difficult to set up. Use this method when referential integrity constraints must be ensured while capturing the data changes. For performance reasons, consistent Change Data Capture is also recommended when a large number of subscribers are required.

**Note:** You cannot journalize a model (or data stores within a model) using both consistent set and simple journalizing.

- Simple

  Simple Change Data Capture enables you to journalize one or more data stores. Each journalized data store is treated separately when capturing the changes.

  This approach has a limitation, illustrated in the following example: Suppose you need to process changes in the `ORDER` and `ORDER_LINE` tables (with a referential integrity constraint based on the fact that an `ORDER_LINE` record should have an associated `ORDER` record). If you have captured insertions into `ORDER_LINE`, then you have no guarantee that the associated new records in `ORDERS` have also been captured. Processing `ORDER_LINE` records with no associated `ORDER` records may cause referential constraint violations in the integration process.

## Change Data Capture Commands

Consistent Change Data Capture uses the following module commands: Start, Drop, Subscribe, Unsubscribe, Extend Window, Lock Subscriber, Unlock Subscriber, and Purge Data.

Simple Change Data Capture provides the following commands for modules and tables: Start, Drop, Subscribe, and Unsubscribe. The commands Lock Subscriber and Unlock Subscriber are available but not used.

### Start

The Start command sets up the Change Data Capture infrastructure.

### Drop

The Drop command removes the Change Data Capture infrastructure.

### Subscribe

The Subscribe command adds a subscriber to this Change Data Capture

### Unsubscribe

The Unsubscribe command removes a subscriber from this Change Data Capture

### Extend Window

The Consistency Window is a range of available changes in all the tables of the consistency set for which the insert/update/delete are possible without violating referential integrity. The Extend Window operation computes this window to take into account new changes captured since the latest Extend Window operation.

### Lock Subscriber

Although the extend window is applied to the entire consistency set, subscribers consume the changes separately. The Lock Subscriber operation performs a subscriber-specific snapshot of the changes in the consistency window. This snapshot includes all the changes within the consistency window that have not yet been consumed by the subscriber.

### Unlock Subscriber

The Unlock Subscriber command commits the use of the changes that where locked during the Lock Subscribers operations for the subscribers. This operation should be processed only after all the changes for the subscribers have been processed.

### Purge Data

After all subscribers have consumed the changes they have subscribed to, entries still remain in the capture tables and must be deleted. This deletion is performed by the Purge Data command.

## Example: Performing Change Data Capture Using Code Templates

### Scenario for Performing Change Data Capture

The `ORDERS` table, in the source schema `SRC`, stores order details. Data from this table is loaded into the `WH_ORDERS` table in the data warehouse target schema `WH_TGT`. The data in the `ORDERS` table changes when an order status or dispatch date is updated. The data in the data warehouse must be updated based on changes made to the source data.

You can set up Change Data Capture using CDC CTs and load only the changed data to the target table in the data warehouse.

### Steps to Perform Change Data Capture Using CDC CTs

Performing Change Data Capture includes the following steps:

1. Selecting the Objects for Change Data Capture
2. Creating the Mapping that Loads Changes
3. Deploying the Change Data Capture Solution
4. Starting the Change Data Capture Process
5. Adding a Subscriber to the Change Data Capture Process

### Selecting the Objects for Change Data Capture

You can perform Change Data Capture for any source or target for which code templates are available. The first step in performing CDC is to identify and select the objects for which you want to capture changes.

**To specify the objects for which data changes must be captured:**

1. If you have not already done so, create a module and import the source objects whose data changes you want to capture.

   In this example, the source schema `SRC` contains the `ORDERS` table whose changes you want to capture. Create an Oracle module called `SRC_MOD`, whose location points to the `SRC` schema, and import the `ORDERS` table into this module.

2. Double-click the `SRC_MOD` module to display the Edit Module dialog box.

3. In the left panel, click **CDC Code Template**. On the CDC Code Template page, in the Code Template list, select the code template that you imported to perform Change Data Capture.

   In this example, select PUBLIC_PROJECT/BUILT_IN_CT/JCT_ORACLE_SIMPLE.

4. In the left panel, click **CDC Tables**. On the CDC Tables page, use the buttons to move tables whose changes must be captured from the Available section to the Selected section.

   In this example, move the `ORDERS` table from the Available list to the Selected list.

The `ORDERS` table is now set up for Change Data Capture.

### Creating the Mapping that Loads Changes

Use the following steps to create the CT mapping that loads changes.

1. If you have not already done so, create a template mapping module to contain the CT mapping that performs Change Data Capture.

   Ensure that the location of this template mapping module is associated with the agent to which the CT mapping must be deployed.

   In this example, the template mapping module called `CDC_MAP_MOD` is associated with the DEFAULT_AGENT location. This location represents the OC4J server installed with Warehouse Builder.

2. Under the template mapping module created in Step 1, create a CT mapping that contains the ETL logic for performing CDC.

   The mapping created in this example is called `CDC_LOAD_ORDERS_MAP`.

3. Drag and drop the source object whose changes you want to capture.

   In this example, drag and drop the `ORDERS` table from the `SRC_MOD` module onto the Mapping Editor canvas.

4. Select the operator representing the source object. In the Property Inspector, under the Change Data Capture node, select **Enable**.

   Notice that the `ORDERS` Table operator contains three additional attributes: `JRN_SUBSCRIBER`, `JRN_FLAG`, and `JRN_DATE`.

5. In the Property Inspector, under the Change Data Capture node, set the Change Data Capture Filter property to define the condition that is used to select changed data for a particular subscriber.

6. Drag and drop the table that will store the changed data onto the canvas.

   In this example, the table `ORDERS_CHANGE` in the `SRC_MOD` module will store the changes to the `ORDERS` table.

7. Map the attributes from the source table, whose changes you want to capture, to the corresponding attributes in the target table that stores changed data.

   In this example, map the `order_Id`, `order_status`, `order_mode`, and `JRN_DATE` attributes from the `ORDERS` table to `order_id`, `order_status`, `order_mode`, and `change_date` attributes, respectively, in the `ORDERS_CHANGE` table.

8. In the Execution View of the CT mapping, create the execution units that you want to associate with code templates.

   In this example, select the `ORDERS` and `ORDERS_CHANGE` operators and click **Create Execution Unit** on the Execution View toolbar.

   Or, select `ORDERS` and `ORDERS_CHANGE` and select **Create Execution Unit** from the Execution menu.

9. If the Code Templates panel is not displayed in the Log window, from the View menu, select **Code Templates**.

10. Select the Integration/Load Code Template tab in the Code Templates panel.
11. In the Code Template for EX_UNIT1 field, select the code template to be used.

    In this example, select JCT_ORACLE_SIMPLE as the code template used to perform the ETL.
12. Validate the mapping and rectify any errors.

### Deploying the Change Data Capture Solution

After you set up the tables for Change Data Capture and create the CT mapping that loads changes from the source table, deploy the Change Data Capture solution using the following steps:

1. If you have not already done so, start the Control Center Agent as described in "Starting the Control Center Agent (CCA)" on page 7-23.
2. Deploy the target table that stores the changed data.

   In this example, deploy the `ORDERS_CHANGE` table using either the Design Center or Control Center Manager.
3. Deploy the CT mapping that loads changed data into the target table using the Design Center or Control Center Manager.

   If you get an error saying application already exists, go to the Control Center Manager, select Replace as the Deployment Option, and deploy the mapping.

   In this example, deploy the `CDC_ORDERS_LOAD_MAP` mapping using the Design Center or Control Center Manager.

### Starting the Change Data Capture Process

When you start the Change Data Capture process, Warehouse Builder creates triggers on the source table for which you want to capture changes and on the target table. For your CT mapping to run successfully, ensure that you grant the required privileges on the source schema to the user performing CDC.

To start the capture process, in the Projects Navigator, right-click the module containing the source table for which you want to capture changes, select **Change Data Capture,** then **Start**.

In this example, right-click `SRC_MOD`, select **Change Data Capture**, then **Start**.

A new tab is displayed in the Message Log containing messages about the CDC operation. During this process, Warehouse Builder generates DDL scripts to create triggers on the source table and deploys the DDL scripts.

### Adding a Subscriber to the Change Data Capture Process

After you start the Change Data Capture process, you must add a subscriber to the capture system. The subscriber consumes the changes generated by the Change Data Capture process.

**To add a subscriber:**

1. In the Projects Navigator, right-click the module containing the source table, select **Change Data Capture**, then **Subscribe**.

   The Add Subscriber dialog box is displayed containing the list of current subscribers.

   In this example, right-click `SRC_MOD`, select **Change Data Capture**, then **Subscribe**.

2. In the Subscriber Name column of the Subscriber to add section, enter a subscriber name.

   In this example, enter `Ord_subscriber` in the Subscriber Name column.

3. Click **OK** to close the Add Subscriber dialog box.

   The Log window displays a new panel for the Subscribe action that lists the actions being performed for the Subscribe action and the results of the actions. If this log displays errors, rectify them and then execute the steps to add a subscriber.

### Testing the Change Data Capture Process

After you set up your Change Data Capture process, you can optionally test this system to verify that changes to the source are being captured and made available to the subscriber.

**To test your Change Data Capture system:**

1. In SQL*Plus, log in to the schema that contains the source table for which you want to capture changes.

   In this example, log in to the `SRC` schema.

2. Insert a row into the source table.

   In this example, insert a row into the `ORDERS` table.

3. In the Control Center Manager, select the CT mapping that performs Change Data Capture and click **Start**.

   In this example, start the `CDC_LOAD_ORDERS_MAP` mapping.

The row that you just added should be inserted into the target table.

The Job Details panel displays the details about each operation being performed in the CT mapping.

### Performing Change Data Capture Actions in Warehouse Builder

After you set up your Change Data Capture solution, you manage the change capture process using predefined actions. If you publish the module or table with CDC tracking as a Web service, then the actions defined in these objects are contained in the generated Web service.

To perform these actions, you right-click the module or table with CDC tracking, select **Change Data Capture**, and use the following actions available here: Start, Drop, Subscribe, Unsubscribe, Extend Window, Purge Data, Lock Subscriber, and Unlock Subscriber. This section describes the CDC actions that you can perform.

Note that actions applicable depend on the type of Change Data Capture you are performing.

> **See Also:** "Change Data Capture Commands" on page 7-34 for information about the Change Data Capture commands

#### Starting the Change Data Capture

After you set up the source object for Change Data Capture, right-click the object and select **Start** to begin capturing changed data. For more details, see "Starting the Change Data Capture Process" on page 7-37.

#### Stopping a Change Data Capture System

The Drop action stops the Change Data Capture process and drops the capture objects.

To stop a Change Data Capture process, in the Projects Navigator, right-click the data object or module, select **Change Data Capture**, then **Drop**. Warehouse Builder displays a prompt asking if you want to stop the capture and drop capture objects. Click **Yes**.

#### Subscribing to a Change Data Capture System

The Subscribe action enables you to add a subscriber to the Change Data Capture system. For more information about adding subscribers, see "Adding a Subscriber to the Change Data Capture Process" on page 7-37.

#### Removing a Subscriber from a Change Data Capture System

The Unsubscribe action enables you to remove a subscriber from the Change Data Capture system. Once you remove a subscriber, change data that is being captured is not available to the subscriber.

**To remove a subscriber:**

1. In the Projects Navigator, right-click the data object or module from which you want to remove a subscriber and, from the Change Data Capture menu, select **Unsubscribe**.

   The Remove Subscriber dialog box is displayed. The Current Subscribers section contains the list of current subscribers of the Change Data Capture system.

2. Select the subscriber you want to remove from the Current Subscriber section and use the buttons to move the subscriber to the section titled Subscribers to remove.

3. Click **OK**.

   The Log window displays the details of the action being performed. Check this log to verify that the Unsubscribe action succeeded.

### Extending the Change Data Capture Window

Use the Extend Window action to extend the subscription window to receive a new set of change data. To extend the window and receive the latest changed data, right-click the module or table in the Projects Navigator, select **Change Data Capture**, and then **Extend Window**. Warehouse Builder displays a prompt asking if you want to extend the window. Click **Yes**.

#### Purging Capture Data

After subscribers consume the change data that they subscribed to, use the Purge Data action to remove the change data from the capture tables. To purge capture data, right-click the module or table in the Projects Navigator, select **Change Data Capture**, and then **Purge Data**. Warehouse Builder displays a prompt asking if you want to purge all capture data. Click **Yes**.

#### Locking a Subscriber

The Lock Subscriber operation enables you to lock a subscriber so that a subscriber-specific snapshot of the changes in the consistency window can be taken. The snapshot includes all the changes within the consistency window that have not yet been consumed by the subscriber.

**To lock a subscriber:**

1. In the Projects Navigator, right-click the data object or module from which you want to lock a subscriber, select **Change Data Capture**, then **Lock Subscriber**.

   The Lock Subscriber dialog box is displayed. The Current Subscribers section contains the list of current subscribers of the Change Data Capture system.

2. Select the subscriber you want to lock from the section titled Current Subscriber and use the buttons to move the subscriber to the section titled Subscribers to lock.

3. Click **OK**.

   The Log window displays the details of the action being performed. Check this log to verify that the Lock Subscriber action succeeded.

### Unlocking a Subscriber

Use the Unlock Subscriber action to commit the changes that where locked during the Lock Subscriber operation for the subscriber.

**To unlock a subscriber:**

1. In the Projects Navigator, right-click the data object or module from which you want to unlock a subscriber, select **Change Data Capture**, then **Lock Subscriber**.

   The Unlock Subscriber dialog box is displayed. The Current Subscribers section contains the list of current subscribers of the Change Data Capture system.

2. Select the subscriber you want to unlock from the section titled Current Subscriber and use the buttons to move the subscriber to the section titled Subscribers to lock.

3. Click **OK**.

   The Log window displays the details of the action being performed. Check this log to verify that the Unlock Subscriber action succeeded.

## Using Control Code Templates

Control Code Templates (Control CTs) enable you to maintain data integrity by checking if the records in a data object are consistent with defined constraints. Use Control CTs when you want to ensure that data that violates constraints specified for a data object is not loaded into the data object.

The constraints checked by a Control CT are Check constraints, Primary Key, Alternate Key, and Not Null.

Use Control CTs to check the following:

- Consistency of existing data

  Set the STATIC_CONTROL property to True to check the data currently in the data object.

- Consistency of the incoming data, before loading the records to a target.

  Set the FLOW_CONTROL property to True. The Control CT simulates the constraints of the target data object on the resulting flow prior to writing to the target.

Control CTs can check either an existing table or the temporary table created by an Integration CT.

### How Does a Control CT Work?

A Control CT accepts a set of constraints and the name of the table to check. It either creates an error table to which all rejected records are written or removes the erroneous records from the checked result set.

In both cases, a Control CT usually performs the following tasks:

1. Creates the error table. The error table contains the same columns as the target table and additional columns to trace error messages, check origin, and check date.
2. Isolates the erroneous records in the error table for each primary key, alternate key, foreign key, condition, and mandatory column that needs to be checked.
3. If required, remove erroneous records from the table that has been checked.

### Control CT Operating Modes

Control CTs can operate in the following modes:

- STATIC_CONTROL

  The Control CT reads the constraints of the table and checks them against the data of the table. Records that do not match the constraints are written to the error table.

- FLOW_CONTROL

  The Control CT reads the constraints of the target table and checks these constraints against the data contained in the "I$" flow table of the staging area. Records that violate these constraints are written to the error table.

## Example: Checking Data Constraints Using Control CTs

### Scenario

Employee data is stored in a file called `EMP.dat`. You must load this data into the target table `EMP_TGT`. During the load, any records that violate constraints defined on the target table are written to the error table associated with the target table.

The target table already exists in an Oracle module called `WH_TGT`.

### Steps to Log Constraint Violations While Loading Data Into a Target Table

Checking data constraints using Control CTs includes the following steps:

1. Creating the Source Module and Importing Source Objects
2. Creating the Code Template Mapping that Extracts Data, Checks Data Integrity, and Loads Data into an Oracle Target

### Creating the Source Module and Importing Source Objects

Because the source is a flat file, you create a flat file module in the Design Center and import the flat file into this module.

**To create a source module and import the source flat file:**

1. In the Projects Navigator, right-click the Files node and select **New Flat File Module**.

   The Create Module Wizard is displayed. Use this wizard to create a flat file module.

For more information about creating flat file modules, see *Oracle Warehouse Builder Sources and Targets Guide*.

2. Right-click the flat file module that you created and select **New File**.

   The Create Flat File Wizard is displayed. Use this wizard to define and sample the source flat file.

   For more information about creating flat files and sampling them, see *Oracle Warehouse Builder Sources and Targets Guide*.

### Creating the Code Template Mapping that Extracts Data, Checks Data Integrity, and Loads Data into an Oracle Target

1. If you have not already done so, create a template mapping module to contain the CT mapping that performs the data integrity check.

   Ensure that you set the location details of this mapping module to the Agent to which the mapping must be deployed.

   This example uses a mapping module called `CKM_MAP_MOD` that is associated with the DEFAULT_AGENT location. This location points to the OC4J server installed with Warehouse Builder.

2. Create a CT mapping that will contain the logic for extracting, checking, and loading data.

   In this example, create a CT mapping called `EMP_LOAD_CKM_MAP`.

3. Drag and drop the source file from the source File source module onto the Mapping Editor canvas.

   In this example, drag and drop the file EMP.dat from the File module onto the canvas.

4. Drag and drop the target table onto the canvas.

   In this example, drag and drop the `EMP_TGT` operator onto the canvas.

   The Table operator properties, in the Property Inspector, contain a node called Control CT. All existing constraints and data rules are displayed in the properties under this section. Use the properties in this group to define how data rules and integrity constraints should be applied.

5. Map the source attributes to the corresponding target attributes.

6. In the Execution View of the mapping, perform the following tasks:

   - Create an execution unit for Flat File operator and associate this execution unit with the LCT_FILE_TO_ORACLE_EXTER_TABLE code template.
   - Create an execution unit containing the target table. Associate this execution unit with the CCT_ORACLE code template.

7. Validate and generate the CT mapping. In the Projects Navigator, right-click the CT mapping and select **Generate**.

8. Deploy the CT mapping. In the Projects Navigator, right-click the CT mapping and select **Deploy**.

9. Execute the CT mapping. In the Projects Navigator, right-click the CT mapping and select **Start**.

   The records that violate constraints defined on the `EMP_TGT` table are not loaded into the target. These records are written to the error table associated with the target table.

# Using Oracle Target CTs in Code Template Mappings

Oracle Target CTs provide a method for using operators that are otherwise only supported in Warehouse Builder PL/SQL mappings. You can use these operators to define your data transformation, create an execution unit containing these transformation operators, and then associate the execution unit with an Oracle Target CT.

## Example: Using Oracle Target Code Templates

### Scenario

You want to aggregate source data available in two different sources and then load it into the target table. The first source is an Oracle module that contains the source tables `CUSTOMERS`, `TIMES`, and `SALES`. The second source is an XML module that contains the tables `CHANNEL` and `COUNTRY`.

The transformation required on the source data is to join data from all the source tables, aggregate it, and then load the aggregated data into the target table `SUMMARY_SALES`. Use the Joiner operator to join data from the source tables. The aggregation is performed using the Aggregator operator that leverages the Oracle Database SQL function `CUBE`. The summarized data is loaded into the target table.

### Steps to Transform Source Data Using Oracle Target CTs

Transforming source data using Oracle Target CTS involves the following tasks:

1. Creating the Source Module and Importing Source Objects
2. Creating the Target Module and Target Table
3. Creating the CT Mapping that Transforms Source Data Using Oracle Target CTs

### Creating the Source Module and Importing Source Objects

In the Projects Navigator, create a source module and its associated location. Import the source objects into this module.

In this example, create an XML module that represents the XML source data and import the `CHANNEL` and `COUNTRY` tables. The location associated with this XML module should point to the XML source. Create an Oracle module whose location points to the `SH` sample schema in the Oracle Database. Import the `CUSTOMERS`, `TIMES`, and `SALES` tables into this module.

For more information about creating modules, see *Oracle Warehouse Builder Sources and Targets Guide*.

### Creating the Target Module and Target Table

1. If you have not already done so, create an Oracle module to store the target table.
2. Under this Oracle module, create the target table that will store the transformed data. Right-click the Tables node, select **New Table** and use the Table Editor to define the table.
3. Generate the target table by right-clicking the table name and selecting **Generate**. Rectify generation errors, if any.
4. Deploy the target table by right-clicking the table name and selecting **Deploy**.

In this example, the module `WH_TGT` contains the target table `SUMMARY_SALES`.

### Creating the CT Mapping that Transforms Source Data Using Oracle Target CTs

1. If you have not already done so, create a template mapping module to contain the CT mapping that performs the required data transformation.

   Ensure that you set the location details of this mapping module to the agent to which the mapping must be deployed.

   In this example, a mapping module called `ETL_MAP_MOD` is associated with the `DEFAULT_AGENT` location. This location points to the OC4J server installed with Warehouse Builder.

2. Create a CT mapping to contain the required ETL logic for extracting, transforming, and loading data.

   In this example, create a mapping called `LOAD_SUMMARY_SALES_MAP`.

3. Drag and drop the source tables onto the Mapping Editor canvas.

   In this example, drag and drop the `CHANNEL` and `COUNTRY` tables from the XML source module and the `CUSTOMERS`, `TIMES`, and `SALES` tables from the Oracle module.

4. Drag and drop the operators that you must use to perform the required data transformation. Connect the operator attributes.

   In this example, you add the following operators:

   - A Joiner operator to join the data in the source tables. Set the Join Condition property for the Joiner operator to define how the source tables should be joined. In this example, you use the columns that are common between a pair of tables to join data from those tables.
   - An Aggregator operator to aggregate the output of the Joiner operator. Data is aggregated based on the `CHANNEL_DESC` and `COUNTRY_ISO_CODE` attributes and the SQL function `CUBE` is leveraged to perform aggregation.

     Thus, in the Group by clause of the Aggregator operator, specify the following:

     CUBE(INGRP1.CHANNEL_DESC,INGRP1.COUNTRY_ISO_CODE)

5. Drag and drop the target table onto the canvas.

   In this example, drag and drop the `SUMMARY_SALES` table onto the canvas.

6. Create the data flows between the source and transformation operators. Map the transformed output to the target table.

   In this example, the tables `CHANNEL`, `CUSTOMERS`, `COUNTRY`, `TIMES`, and `SALES` are mapped to the Input groups of the Joiner operator. The Output group of the Joiner operator is mapped to the Aggregator operator. The output of the Aggregator operator is mapped to the target table `SUMMARY_SALES`.

7. In the Execution View of the CT mapping, create the execution units required to perform the data transformation.

   Your CT mapping should look like the one in Figure 7–6.

   In this example, perform the following:

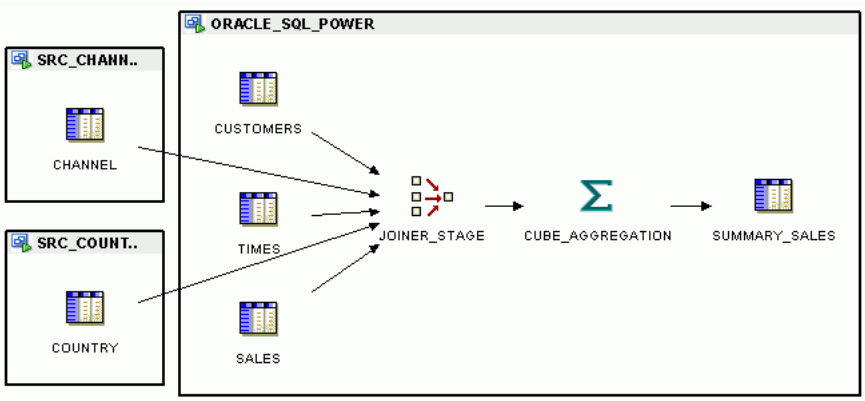
   - Create an execution unit for the `CHANNEL` table. Associate this execution unit with the LCT_SQL_TO_SQL code template.
   - Create an execution unit for the `COUNTRY` table. Associate this execution unit with the LCT_SQL_TO_SQL code template.

- Create an execution unit containing the tables CUSTOMERS, SALES, and TIMES, the Joiner operator, the Aggregator operator, and the SUMMARY_SALES table. Associate this execution unit with the Oracle Target CT DEFAULT_ORACLE_TARGET_CT.

8. Validate and generate the CT mapping. In the Projects Navigator, right-click the CT mapping and select **Generate**.
9. Deploy the CT mapping. In the Projects Navigator, right-click the CT mapping and select **Deploy**.
10. Execute the CT mapping to extract data from the XML and Oracle source, transform it, and load it into the Oracle target table. In the Projects Navigator, right-click the CT mapping and select **Start**.

Figure 7–6 displays the Execution View of the CT mapping that enables you to perform the required data transformation using Oracle Target CTs.

***Figure 7–6 Mapping That Uses Oracle Target CTs to Transform Source Data***




# Moving Data from Heterogeneous Databases to Oracle Database

You can use code templates to extract data from heterogeneous databases such as SQL Server, DB2, and Teradata. The code templates used to perform the data transfer depends on the source and target database.

Warehouse Builder provides a set of code templates that you can use to transfer data between different databases. These code templates are located in the BUILT_IN_CT node under the Public Code Templates node of the Globals Navigator. Each code template performs a certain set of tasks on a certain platform.

## Example: Moving Data from IBM DB2 to Oracle Database Using Integration CTs and Load CTs

### Scenario for Extracting Data

The tables ORDERS and ORDER_DETAILS are stored in an IBM DB2 database. You must extract data from these two tables, transform it, and store it in a table called ORDERS_AGG_CUST in an Oracle database. The transformation consists of joining the data in these two tables and then aggregating the data for each customer.

### Before You Extract Data from IBM DB2

- Ensure that you have the drivers required to access an IBM DB2 database. The files that you need are `db2jcc.jar` and `db2jcc_license_cu.jar`. Copy these files to the *`OWB_ORACLE_HOME`*`/owb/lib/ext` directory.
- In the script that starts the CCA, add the statement that loads the required libraries for the DB2 driver.

  On Unix, add the following statement to *`OWB_ORACLE_HOME`*`/owb/bin/unix/ccastart`:

  ```
  -Dapi.ext.dirs=$OWB_HOME/owb/lib/ext
  ```

  On Windows, add the following statement to *`OWB_ORACLE_HOME`*`/owb/bin/win32/ccastart.bat`:

  ```
  -Dapi.ext.dirs=%OWB_HOME%\owb\lib\ext
  ```

> **See Also:** *Oracle Warehouse Builder Sources and Targets Guide* for more information about modifying the CCA start script

## Steps to Extract Data from IBM DB2, Transform Data, and Load it into an Oracle Database

To extract data from IBM DB2, transform the data, and then load the data into an Oracle Database:

1. Create the Source Module
2. Create the Target Module and Target Table
3. Create the CT Mapping that Extracts, Transforms, and Loads Data

### Create the Source Module

In the Projects Navigator, create a DB2 module that represents the source data. The location associated with this module should point to the DB2 database containing the source objects.

In this example, create a DB2 module whose location points to the DB2 database containing the `ORDERS` and `ORDER_DETAILS` tables.

For more information about creating a DB2 module, see *Oracle Warehouse Builder Sources and Targets Guide*.

### Create the Target Module and Target Table

Use the following steps to create the target module and the target table.

1. If you have not already done so, create an Oracle module to store the target table.
2. Under this Oracle module, create the target table that will store the transformed data. Right-click the Tables node, select **New Table,** and use the Table Editor to define the table.
3. Generate the target table by right-clicking the table name and selecting **Generate**. Rectify generation errors, if any.
4. Deploy the target table by right-clicking the table name and selecting Deploy.

In this example, the module `WH_TGT` contains the target table `ORDERS_AGG_CUST`.

### Create the CT Mapping that Extracts, Transforms, and Loads Data

Use the following steps to create the CT mapping that extracts data from DB2 tables, transforms it, and then loads it into and Oracle Database table.

1. If you have not already done so, create a template mapping module to contain the CT mapping that performs the required ETL.

   Ensure that you set the location details of this mapping module to the agent to which the mapping must be deployed.

   In this example, a mapping module called `ETL_MAP_MOD` is associated with the DEFAULT_AGENT location. This location points to the OC4J server installed with Warehouse Builder.

2. Create a mapping to contain the required ETL logic for extracting, transforming, and loading data.

   In this example, create a mapping called `LOAD_DB2_TO_ORACLE_MAP`.

3. Drag and drop the source tables from the DB2 source module onto the Mapping Editor canvas.

   In this example, drag and drop the `ORDERS` and `ORDER_DETAILS` tables from the DB2 module source module.

4. Drag and drop the operators that you must use to perform the required data transformation. Connect the operator attributes.

   In this example, you add the following operators:

   - A Joiner operator to join the data in the `ORDERS` and `ORDER_DETAILS` tables. Set the Join Condition property for the Joiner operator.
   - An Aggregator operator to aggregate the output of the Joiner operator. Aggregate the data based on the `CUSTOMER_ID` attribute.

5. Drag and drop the target table onto the canvas.

   In this example, drag and drop the `ORDERS_TGT` operator onto the canvas.

6. Map the transformed output to the target table.

7. In the Execution View of the mapping, perform the following:

   - Create an execution unit for the `ORDERS` and `ORDER_DETAILS` operators. Associate this execution unit with the LCT_SQL_TO_ORACLE code template.
   - Create an execution unit containing the Joiner, Aggregator, and `ORDERS_AGG_CUST` table. Associate this execution unit with the ICT_ORACLE_INCR_UPD code template.

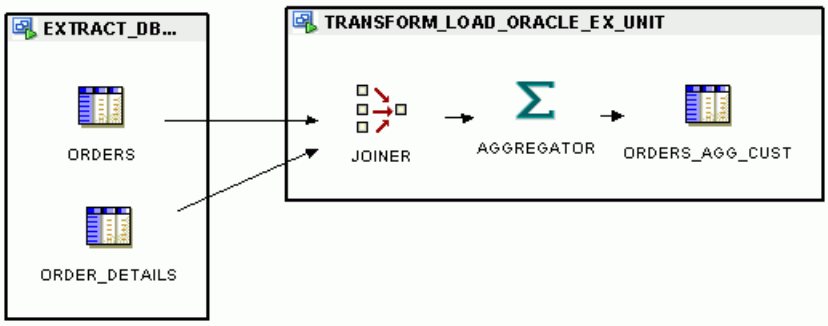
8. Validate and generate the CT mapping. In the Projects Navigator, right-click the CT mapping and select **Generate**.

9. Deploy the CT mapping. In the Projects Navigator, right-click the CT mapping and select **Deploy**.

10. Execute the CT mapping to extract data from the source DB2 tables, transform it, and load it into the Oracle target table. In the Projects Navigator, right-click the CT mapping and select **Start**.

Figure 7–7 displays the Execution View of the mapping LOAD_DB2_TO_ORACLE_MAP.

***Figure 7–7 Mapping to Extract Data from IBM DB2, Transform Data, and Load it into Oracle Database***



# 8

# Designing Process Flows

After you design mappings that define the operations for moving data from sources to targets, you can create and define process flows. A process flow allows activities to be linked together to define flow of control among different activities. Supported flow of control constructs include conditional branches, loops, parallel flows or serial dependencies. Activities can be mappings, transformations, or external commands such as e-mail, FTP commands, and operating system executables.

You can use process flows to manage dependencies between mappings. To schedule mappings, process flows, and other executable objects, see "Defining Schedules" on page 11-2.

This chapter contains the following topics:


- Overview of Process Flows
- Example: Creating a Basic Process Flow
- Steps for Defining Process Flows
- Adding Activities to Process Flows
- Creating and Using Activity Templates
- About Transitions
- About Expressions
- Defining Transition Conditions
- Example: Using Process Flows to Access Flat Files with Variable Names
- Example: Using Process Flows to Transfer Remote Files


## Overview of Process Flows

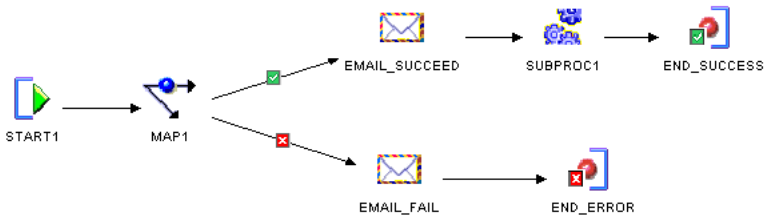
A process flow describes dependencies between Warehouse Builder mappings and external activities such as e-mail, FTP, and operating system commands. Use process flows to sequence individual steps in the ETL process. The individual steps often include mappings, but can also include manual activities or external activities such as FTP or e-mail.

Each process flow begins with a Start activity and concludes with an End activity for each stream in the flow. A process flow is considered as a type of activity, so a process flow can start other process flows.

Figure 8–1 shows an example of a process flow that starts a mapping (MAP1). If the mapping completes successfully, then Warehouse Builder sends an e-mail notification (EMAIL_SUCCEED) and starts another process flow (SUBPROC1). If the mapping

fails, then Warehouse Builder sends an email (EMAIL_FAIL) and ends the process flow.

***Figure 8–1 Sample Process Flow***




When you design a process flow in Warehouse Builder, you use an interface known as the Process Flow Editor. Alternatively, you can create and define process flows using the Warehouse Builder scripting language, OMB*Plus, as described in *Oracle Warehouse Builder API and Scripting Reference*.

## About Process Flow Modules and Packages

Process flows must be grouped together into process flow packages, which in turn are grouped together in process flow modules. Together, the process flow modules and packages provide two levels to manage and deploy process flows. You can validate, generate, and deploy process flows at either the module or the package level.

You can design a process flow that starts other process flows as long as they are in the same module. You can copy process flows from one package to another package in the same or a different module, and you can copy packages to a different module. To do so, use the Copy and Paste commands available under **Edit** on the Design Center main menu.

For example, Figure 8–1 shows a process flow PROC1 that includes process flow SUBPROC1. For PROC1 to run successfully, SUBPROC1 and PROC1 can be in the same or separate process flow modules, but they must be deployed to the same location.

### Deploying Process Flows to Workflow Engines

Warehouse Builder process flows comply with the XML Process Definition Language (XPDL) standard set forth by the Workflow Management Coalition (WfMC). When you generate a process flow, Warehouse Builder generates an XML file in the XPDL format. The generated XPDL can be used to integrate with any workflow engine that supports the WfMC standard.

Warehouse Builder provides integration with Oracle Workflow. From the Warehouse Builder Control Center, you can deploy process flow packages or modules to Oracle Workflow.

# Example: Creating a Basic Process Flow

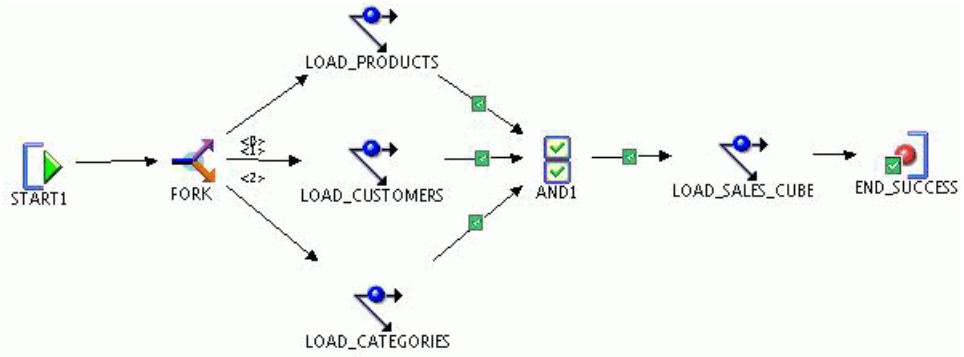
The cube `SALES_CUBE` is loaded using data in the `PRODUCTS`, `CATEGORIES`, and `CUSTOMERS` dimensions. These dimensions, in turn, are loaded using one or more transaction tables. Data must be loaded into the cube only if all the dimensions are loaded successfully.

You create separate mappings to load the cube and dimensions. However, you want a sequential flow in which the three dimensions `PRODUCTS`, `CATEGORIES`, and

CUSTOMERS are loaded first and, if these dimensions are loaded successfully, the cube SALES_CUBE is loaded. Use a process flow to link these mappings and create a sequential flow.

Figure 8–2 displays the process flow that loads the SALES_CUBE. This process flow is created after you complete the steps listed in "Steps to Define the Process Flow" on page 8-3.

***Figure 8–2 Process Flow that Loads a Cube***




### Before You Begin

Create the following mappings:

- LOAD_PRODUCTS: This mapping transforms source data stored in transaction tables and loads transformed data into the PRODUCTS dimension.
- LOAD_CATEGORIES: This mapping transforms source data stored in transaction tables and loads transformed data into the CATEGORIES dimension
- LOAD_CUSTOMERS: This mapping transforms source data stored in transaction tables and loads transformed data into the CUSTOMERS dimension
- LOAD_SALES_CUBE: This mapping loads data into the cube SALES_CUBE, using the dimensions PRODUCTS, CATEGORIES, and CUSTOMERS.

### Steps to Define the Process Flow

1. Create a Oracle Workflow location to which the process flow will be deployed.

   See "Creating Oracle Workflow Locations" on page 8-5 for information about creating Oracle Workflow locations.

2. In the Projects Navigator, expand the project node under which you want to create the process flow and then expand the Process Flows node.

3. Right-click the Process Flow Modules node and select **New Process Flow Module**.

   The Create Process Flow Module wizard is displayed.

4. On the Name and Description page, provide a name and an optional description for the process flow.

5. On the Connection Information page, in the Location field, select the location created in step 1. Click **Finish** to create the process flow module.

   The Create Process Flow Package dialog box is displayed.

6. Enter a name and an optional description for the process flow package and click **OK**.

The Create Process Flow dialog box is displayed.

7. Enter the name and an optional description for the process flow and click **OK**.

   The Process Flow Editor is displayed. The editor canvas contains the Start activity named START1 and a Stop activity called End_success. Use the editor to add other activities that are part of your process flow and to define relationships between them.

8. From the Component Palette, drag a Fork activity and drop it on to the editor canvas.

9. From the Projects Navigator, drag the following mappings and drop them on to the editor canvas: `LOAD_PRODUCTS`, `LOAD_CATEGORIES`, and `LOAD_CUSTOMERS`.

   When you drag a mapping and drop it onto the canvas, the mapping activity is displayed on the canvas with a default name such as MAPPING_*n*. The activity name highlighted in blue so that you can change the name, if required. For each activity, enter the same name as the mapping. For example, for the `LOAD_PRODUCTS` mapping, enter the name of the activity as `LOAD_PRODUCTS`.

   Position these activities in a vertical line, one below the other, to the right of the Fork activity.

10. Create the following transitions:

    - From the Fork activity to the `LOAD_PRODUCTS` activity
    - From the Fork activity to the `LOAD_CATEGORIES` activity
    - From the Fork activity to the `LOAD_CUSTOMERS` activity

    To create a transition, select the source activity. The cursor is displayed as a small horizontal arrow on the activity. Drag and drop on the target activity.

11. From the Component Palette, drag and drop an AND activity on to the editor.

12. Define a conditional transition, with the condition defined as SUCCESS, from the `LOAD_PRODUCTS` activity to the And activity.

    A conditional transition is one that is based on a predefined condition. To define a condition for the transition:

    a. On the editor canvas, select the transition.

       The Property Inspector displays the properties of the selected transition.

    b. Click the Ellipsis button on the Condition property.

       The Edit Property dialog box is displayed.

    c. Select Enumerated Condition. In the list below this option, select Success and click **OK**.

13. Define a conditional transition, with the condition defined as SUCCESS, from the LOAD_CATEGORIES activity to the And activity.

14. Define a conditional transition, with the condition defined as SUCCESS, from the `LOAD_CUSTOMERS` activity to the And activity.

15. From the Projects Navigator, drag and drop the mapping `LOAD_SALES_CUBE` onto the editor canvas. Enter the name of the activity as `LOAD_SALES_CUBE`.

16. Create a transition from the AND activity to the `LOAD_SALES_CUBE` activity. Select the AND activity to display a small horizontal arrow. Drag and drop on to the `LOAD_SALES_CUBE` activity.

17. Create a conditional transition, with the condition defined as SUCCESS, from the LOAD_SALES_CUBE activity to the End_success activity.

# Steps for Defining Process Flows

### Before You Begin

To enable deployment of process flows, install Oracle Workflow as described in the *Oracle Warehouse Builder Installation and Administration Guide for Windows and UNIX*.

To define a process flow, refer to the following sections:

1. (Optional) Creating Oracle Workflow Locations
2. Creating Process Flow Modules
3. Creating Process Flow Packages
4. Creating Process Flows
5. Creating and Using Activity Templates
6. Adding Activities
7. Connecting Activities
8. Activities in Process Flows
9. Using Parameters and Variables
10. Configuring Process Flows Reference
11. Validating and Generating Process Flows
12. Scheduling Process Flows (optional)

    When you are satisfied that the process flow runs as expected, you can schedule the process flow to run on a single day or multiple days as described in "Defining Schedules" on page 11-2.
13. Deploying Process Flows, see "Steps in the Deployment and Execution Process" on page 12-5.

## Creating Oracle Workflow Locations

Use an Oracle Workflow location to deploy process flows. This location corresponds to the Oracle Workflow schema.

**To create an Oracle Workflow location:**

1. In the Locations Navigator, expand the Locations node and then the Process Flows and Schedules node.
2. Right-click the Oracle Workflow node and select **New Oracle Workflow Location**.

   The Create Oracle Workflow Location dialog box is displayed.
3. On the Details page, provide information in the following fields:
   - **Name:** Represents the name of the Oracle Workflow location.
   - **Description:** Represents the description of the location. Providing a description is optional.
   - **Connection Type:** Represents the type of connection to Oracle Workflow. Select one of the following options:

- **Host:Port:Service:** Makes a connection using the Easy Connect Naming method, which requires no prior setup. Enter the following additional information.

  **Host:** The name of the system where Oracle Database is installed with Oracle Workflow Manager.

  If the client software is installed on the same system as Oracle Database, you can enter localhost instead of the computer name.

  **Port:** The SQL port number for the Oracle Database.

  **Service:** The service name of the Oracle database.

- **SQL*NET Connection:** Makes a connection using a net service name previously defined using a tool such as Oracle Net Configuration Assistant. The net service name provides a convenient alias for the connection information.

  In the Net Service Name field, enter the name of the predefined connection.

- **Schema:** Represents the user name for the Workflow schema.
- **Password:** Represents the password for the user specified in the Schema field.
- **Version:** Represents the version of Oracle Workflow.

## Creating Process Flow Modules

Before working with process flows, create a process flow module. The module is a container using which you can validate, generate, and deploy a group of process flows. Process flow modules include process flow packages which include process flows.

**To create a process flow module:**

1. Right-click the **Process Flow Modules** node in the Projects Navigator and select **New Process Flow Module.**

   Warehouse Builder displays the Welcome page for the Create Module Wizard.

2. Click **Next.**

   On the Name and Description page, type a module name that is unique within the project. Enter an optional text description.

3. Click **Next.**

   The wizard displays the Connection Information page.

   You can accept the default location that the wizard creates for you based on the module name. Alternatively, select an existing location from the list. Click **Edit** to enter the connection information and test the connection.

4. Click **Next.**

   The wizard displays the Finish page. Verify the name and deployment location of the new process flow module.

   When you click **Finish,** Warehouse Builder stores the definition for the module, inserts its name in the Projects Navigator, and prompts you to create a process flow package.

### Creating User Folders Within a Process Flow Module

Within a process flow module, you can create user folders to group process flow packages based on criteria such as product line, functional groupings, or application-specific categories.

User folders can contain other user folders and other process flow packages. There is no limit on the level of nesting of user folders. You can also move, delete, edit, or rename user folders. To move a user folder, select the user folder in the Projects Navigator and click **Cut** on the toolbar. Then, select the process flow module into which the user folder is to be moved and click **Paste**.

You can move process flow packages that are contained in a user folder either to the corresponding parent process flow module or to another process flow module.

Deleting a user folder removes the user folder and all its contained objects from the repository.

**To create a user folder within a process flow module:**

1. In the Projects Navigator, expand the Process Flows node. Right-click the Process Flow module or user folder under which you want to create a user folder and select **New**.

   The New Gallery dialog box is displayed.

2. In the Items section, select **User Folder**.

   The Create User Folder dialog box is displayed.

3. Specify a name for the user folder and click **OK**.

The user folder is created and added to the tree under the Process Flow module.

**To create a process flow package within a user folder:**

1. In the Projects Navigator, expand the Process Flows node. Right-click the user folder and select **New**.

   The New Gallery dialog box is displayed.

2. In the Items section, select **Process Flow Package**.

3. Click **OK**.

The Create Process Flow Package dialog box is displayed. Use this dialog box to create a process flow. Subsequently, create the required process flows within this process flow package.

## Creating Process Flow Packages

After you create a Process Flow module, you can create a process flow package. The process flow package is an additional grouping mechanism from which you can deploy process flows.

**To create a process flow package:**

1. Right-click a process flow module in the Projects Navigator and click **New Process Flow Package**.

   Warehouse Builder displays the Create Process Flow Package dialog box.

2. Type a name and optional description for the process flow package.

   If you intend to integrate with Oracle Workflow, note that Oracle Workflow restricts package names to 8 bytes.

3. Click **OK.**

   Warehouse Builder prompts you to create a process flow.

## Creating Process Flows

After you create a module and package for process flows, you can create a process flow.

**To create a process flow:**

1. Right-click a process flow package in the Projects Navigator and click **New Process Flow**.

   Warehouse Builder displays the Create Process Flow dialog box.

2. Type a name and optional description for the process flow.

   **Note:** If you intend to schedule a process flow, there is an additional consideration. For any ETL object that you want to schedule, the limit is 25 characters for physical names and 1,995 characters for business names. Follow this additional restriction to enable Warehouse Builder to append to the process flow name the suffix _job and other internal characters required for deployment and running the process flow.

3. Click **OK.**

   Warehouse Builder runs the Process Flow Editor and displays the process flow with a Start activity and an End_Success activity.

4. You can now model the process flow with activities and transitions.

5. Continue with the steps listed in "Steps for Defining Process Flows" on page 8-5.

# Adding Activities to Process Flows

You can add activities in a process flow by using the Projects Navigator.

## About Activities

Activities represent units of work for the process flow, such as starting a mapping or verifying the existence of a file on a drive or directory. When you design a process flow in Warehouse Builder, you select activities from the Component Palette, drag them onto the canvas, and set their parameters. Warehouse Builder includes the following types of activities:

- **Oracle Warehouse Builder Specific Activities:** These activities enable you to start Warehouse Builder objects such as mappings, transformations, or other process flows. The process flow runs the object and provides a commit statement.
- **Utility Activities:** These activities enable you to perform services such as sending e-mails and transferring files.
- **Control Activities:** These activities enable you to control the progress and direction of the process flow. For instance, use the Fork activity to run multiple activities concurrently.

For the utility and control type activities, you can reuse their parameters by defining activity templates as described in "Creating and Using Activity Templates" on page 8-11. For e-mail, for example, use an e-mail template to specify the SMTP server

name and port number, the list of addresses, and the priority. Then you can reuse that template when you add email activities to a process flow.

For a description of each activity, see "Using Activities in Process Flows" on page 27-1.

## Adding Activities

**To add an activity to a process flow:**

1. Open the process flow by right-clicking the process flow in the Projects Navigator and selecting **Open**.
2. View the activities listed in the Component Palette.

   By default, the palette lists all activities. To find a particular activity, use the list box on the palette to narrow the displayed list to one of the following types of activities: Oracle Warehouse Builder Specific activities, Utility activities, and Control activities.
3. Select an activity from the palette and drag it onto the canvas.

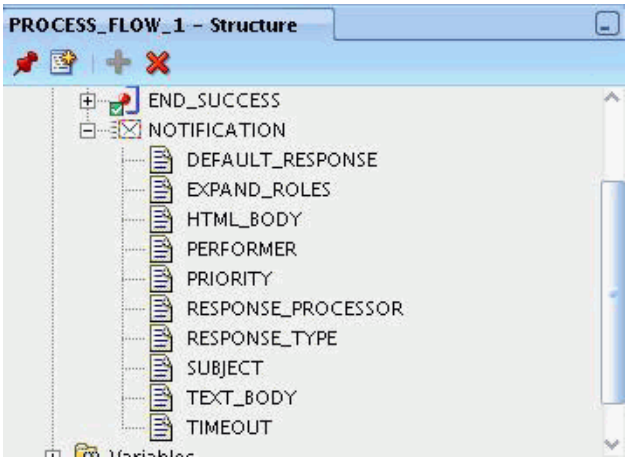
   The editor displays the activity on the canvas with the name highlighted in blue.
4. To accept the default name, press **Enter.** To change the name, type in the new name.

   The editor lists the activity in the Structure panel. The properties of this activity are displayed in the Property Inspector.
5. In the Property Inspector**,** add parameters for the activity by clicking **New Process Activity Parameter** at the top of the Structure panel.

   The parameters for an activity vary according to the type of activity. For each activity, Warehouse Builder defines read-only parameters Name, Direction, and Data Type. And for each parameter, you can specify values for Binding, Literal, Value, and Description in the Property Inspector.

   For example, Figure 8–3 shows the parameters for a Notification activity. The parameters include DEFAULT_RESPONSE, EXPAND_ROLES, HTML_BODY, PERFORMER, PRIORITY, RESPONSE_PROCESSOR, RESPONSE_TYPE, SUBJECT, TEXT_BODY, and TIMEOUT.

***Figure 8–3 Parameters for a Notification Activity***



## Parameters for Activities

Each parameter has the following properties:

### Name

This is a name property of the activity parameter. For information about a specific parameter, look up the activity by name under Example 27, "Activities in Process Flows".

### Direction

The direction property is read-only for parameters that are not created by the user. A direction of IN indicates that the parameter is an input parameter for the activity.

### Data Type

The data type property is read-only for parameters that are not created by the user. Warehouse Builder assigns the appropriate data type for all default parameters.

### Binding

Use the binding property to pass in parameters from outside the process flow for parameters that are not created by the user. If you assign a parameter in **Binding,** then it overrides any text you assign to **Value.**

### Literal

If you enter a value for the parameter in the field **Value,** then indicate whether the value is a literal or an expression. The literal data types follow the PL/SQL literal value specification except for calendar data types. These data types are represented in a standard format as the process flow interacts with data sources from different locations.

The values you can select for Literal are True or False. When you set Literal to False, then the value entered for the Value property must be a valid PL/SQL expression which is evaluated at the Control Center. When you set Literal to True, then the value depends on the type of activity. If the activity is a PL/SQL object, such as a mapping or process flow, then the Value is a PL/SQL snippet. If the activity is not a PL/SQL object, then the Value is language-dependent.

Table 8–1 provides the Literal value type, format, and some examples.

***Table 8–1 Example of Literal Value Types***

| Literal Value Type | Format | Example |
|---|---|---|
| DATE | YYYY-MM-DD | 2006-03-21 |
| DATE | YYYY-MM-DD HH24:MI:SS | 2006-03-21 15:45:00 |
| TIMESTAMP | YYYY-MM-DD HH24:MI:SS.FF9 | 2006-03-21 15:45:00.000000000 |
| TIMESTAMP_TZ | YYYY-MM-DD HH24:MI:SS.FF9 TZH:TZM | 2006-03-21 15:45:00.000000000 +01:00 |
| YMINTERVAL | [+-]YYYYYYYYY-MM | +000000001-01 |
| DMINVERVAL | [+-]DDDDDDDDD HH24:MI.SS.FF9 | +000000001 01:01:01.000000001 |

### Value

This is the value of the parameter. For some parameters, Warehouse Builder enables you to select from a list of values. For other parameters, Warehouse Builder assigns

default values that you can override by entering a new value or using the field **Binding.** In the absence of a list of possible values or a default value, you must enter a value.

#### Description

You can enter an optional description for each property.

## Creating and Using Activity Templates

In designing process flows, you may want to reuse existing activities. For example, each time a mapping fails in a process flow, you may want to send an e-mail to the same group of administrators. You create a template for the Email activity once, and then use and edit the activity in many process flows.

**To create an activity template:**

1. In the Projects Navigator, navigate to the Activity Templates node under the Process Flows node.
2. To create a folder for containing templates, right-click the Activity Templates node and select **New Activity Template Folder**.
3. Assign a name to the activity template folder and click **OK**.

   Consider creating a folder for each type of template that you plan to create. For instance, you could create separate folders to contain Email and Ftp templates.
4. The Create Activity Template Wizard is displayed.

   > **Note:** If the wizard does not appear automatically, then right-click a folder and select **New Activity Template**.

   Follow the prompts in the Create Activity Template Wizard to complete the Name and Description Page, the Parameters Page, and the Summary page.
5. See "Using Activity Templates" on page 8-12 for instructions about how to use the template in a process flow.

### Name and Description Page

The rules for naming objects in the activity template depend on the naming mode that you select in that Naming Preferences section of the Preferences dialog box. Warehouse Builder maintains a business and a physical name for each object in the workspace. The business name is its descriptive business name. The physical name is the name that Warehouse Builder uses when generating code.

When you name objects while working in one naming mode, Warehouse Builder creates a default name for the other mode. So, when working in the business name mode, if you assign an activity template name that includes mixed cases, special characters, and spaces, then Warehouse Builder creates a default physical name for the objects.

Assign a name and select the type of activity template that you want to create. Also, write an optional description for the template.

### Naming Activities

In the physical naming mode, an activity name can be from 1 to 30 alphanumeric characters and blank spaces are not allowed. In the business naming mode, the limit is 200 characters and blank spaces and special characters are allowed. In both naming modes, the name should be unique across the project.

### Describing Activities

The description can be up to 4,000 alphanumeric characters and can contain blank spaces. Specifying a description for an activity template is optional.

### Activity Templates

The following activity templates are available from the list.

- Assign
- Email
- Enterprise Java Bean
- FTP
- File Exists
- Java Class
- Manual
- Notification
- OMBPlus
- Set Status
- Sqlplus
- User Defined
- Wait

## Parameters Page

The wizard displays parameters based on the type of activity that you previously selected in the Activity Templates list.

Enter default values for the activity. When you use the activity template in a process flow, you can retain or edit the default values. For example, an Email activity template contains the parameters FROM_ADDRESS and REPLY_TO_ADDRESS. When you use an Email activity template in a process flow, you can overwrite the default values of these parameters with different values.

## Using Activity Templates

Complete the following steps to use an activity template:

1. In the Projects Navigator, navigate to the process flow module under the Process Flows node.
2. To open the Process Flow Editor, right-click the Process Flow and select **Open**.
3. From the Graph menu, select **Add**, then **Available Objects.**

   The Add Available Objects dialog box is displayed.
4. Select the activity template you want to use and click **OK**.

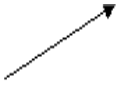
The activity template is added to the Process Flow Editor canvas. Activity templates in a process flow acts like regular activities.

Alternatively, instead of Steps 3 and 4, you can drag and drop an activity template from the Projects Navigator on to the Process Flow Editor canvas.

To edit the activity, select the activity on the canvas and use the Structure tab to modify the activity details.

# About Transitions

Use transitions to indicate the sequence and conditions in which activities occur in the process flow. You can use transitions to run an activity based on the completion state of the preceding activity.



When you add a transition to the canvas, by default, the transition has no condition applied to it. The process flow continues once the preceding activity completes, regardless of the ending state of the previous activity.

A transition with no condition applied to it has different semantics depending on the source activity type. If the activity type is FORK, then it may have multiple unconditional transitions in which each transition begins a new flow in the process flow. If the source activity type is not FORK, then there may be only one unconditional transition which is used when no other conditional transition is activated (for example, the final `ELSE` condition in an `IF...THEN...ELSIF...ELSE...END` PL/SQL statement).

## Rules for Valid Transitions

For a transition to be valid, it must conform to the following rules:

- All activities, apart from START and END, must have at least one incoming transition.
- Only the AND and OR activities can have more than one incoming transition.
- Only a FORK activity can have more than one unconditional outgoing transition.
- A FORK activity can have only unconditional outgoing transitions.
- An activity that has an enumerated set of outcomes must have either an outgoing transition for each possible outcome or an unconditional outgoing transition.
- An activity can have zero or more outgoing complex expression transitions.
- An activity, with an outgoing complex expression transition, must have an unconditional outgoing transition.
- An END_LOOP transition must have only one unconditional transition to its associated FOR_LOOP or WHILE_LOOP activity.
- The transition taken by the `exit` outcome of a FOR_LOOP or WHILE_LOOP must not connect to an activity that could be carried on as a result of the "loop."

## Connecting Activities

**To create dependencies using transitions:**

1. When working in the Select mode in the Process Flow Editor, place your mouse pointer along the right border of the activity icon along its center line.

The editor displays the cursor as a small horizontal arrow, indicating that you can now use the mouse button to connect activities.

2. Press the left mouse button and scroll toward the next activity. As you begin to scroll, the cursor appears as an arrow with a plus sign under it. Continue to scroll towards the next activity until the plus sign under the cursor arrow changes to a circle. Release the mouse button to connect the two activities.

   The editor displays an arrow between the two activities, assigns a default name to the transition, and displays the transition in the Structure panel. The properties of the transition are displayed in the Property Inspector.

3. In the Property Inspector, view or edit the following attributes for the transition:

   **Name:** The editor assigns a default name that you can change.

   **Description:** You can enter an optional description for the transition.

   **Condition:** Transitions that you initially draw on the canvas are unconditional by default. To override the default and apply new conditions, select the transition. The Property Inspector displays the transition properties. Click the Ellipsis button to the right of the Condition field and, in the Edit Property dialog box, select the condition that you want to apply to the transition. When you select a condition, then the editor displays the associated icon imposed onto the transition line on the canvas.

   **Source:** This property is read-only and indicates the first activity in the connection.

   **Target:** This property is read-only and indicates the second activity in the connection.

## Configuring Activities

Some activities, such as Sqlplus, require additional configuration. These configuration details for a given activity are listed in Chapter 27, "Activities in Process Flows".

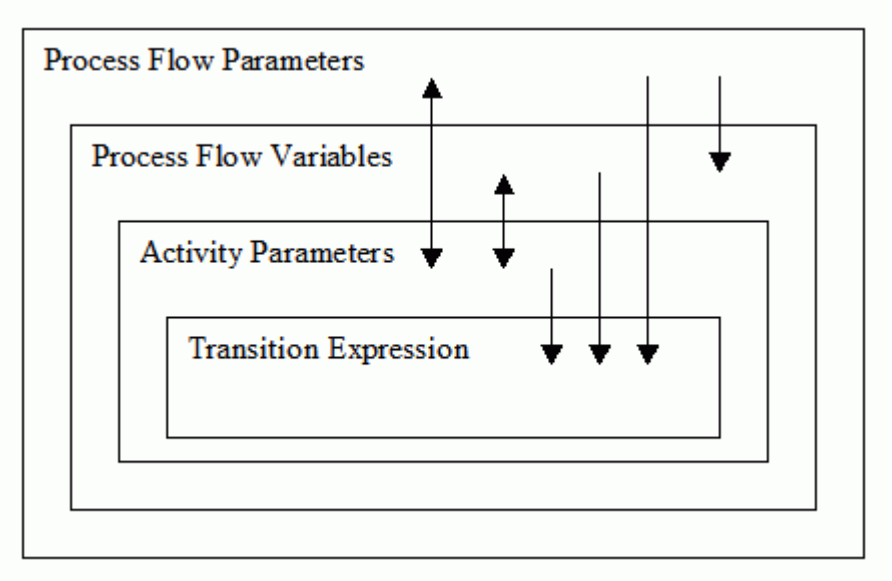
## Using Parameters and Variables

Process flows and activities support the PL/SQL parameter passing concept, allowing data to be passed and reused through parameterization. This is accomplished through data stores, which are implemented as either parameters or variables. The process flow allows the data to be passed between data stores.

- Parameters allow passing of data between a process flow and its activities or subprocesses.
- Variables allow the storage of transient data, which is then maintained for the lifetime of running the process flow. Variables are used to pass data between activities.

Figure 8–4 shows the direction in which the data is passed.

***Figure 8–4 Relationship Between the Scope and the Direction in Which the Data is Passed***




Process flows adhere to the following rules for allowing the data to be passed between data stores:

1. Process flow variables can be initialized from process flow parameters, but the reverse is not allowed.
2. Activity parameters can pass data bidirectionally between process flow variables and process flow parameters.
3. Transition expressions can be evaluated against their source activity parameters, process flow parameters, and process flow variables.
4. A data store cannot be accessed from another data store within the same scope.

## Using a Namespace

The namespace allows a data store of an inner scope to hide the data store of an outer scope, similar to PL/SQL. By qualifying the data store name with the process flow name or activity, you can reference the hidden data store name. For example:

```
My_PROC.VAR1
```

The namespace does not allow referencing of data from another data store within the same scope.

## Using Bindings

A data store may be bound to another data store in an outer scope, which supports the passing of data in both directions.

Process flow bindings follow the same semantics as PL/SQL with the following rules:

1. All the data is passed within the process flow by value.
2. Variables can be initialized through a binding. They cannot return a value.
3. An INOUT parameter can be bound to an IN parameter in an outer scope. The output value, which is passed by value, is audited and then discarded.

Because a variable cannot pass data out to a process flow parameter, this is accomplished by the use of an Assign operator, which can be bound to the variable and the parameter.

# About Expressions

Oracle Warehouse Builder supports the use of PL/SQL expressions for the derivation of parameter values and the use of 'complex expression' transitions.

The expression must produce a correctly typed value for data store. Automatic conversion from VARCHAR is supported. When the expression is associated with a transition a Boolean result is expected.

During evaluation, an expression has access to the outer scope that encloses it. So, an expression for an activity parameter can use process flow variables and process flow parameters in its evaluation.

The PL/SQL expression is run in the context of the Control Center user who requested the process of the activity. However, if the Oracle Workflow schema is hosted in a remote database instance, the effective user of the generated database link will be used instead. A different Control Center user may be selected by configuring the process flow and specifying an evaluation location. Thus, the expression may reference any PL/SQL function that is accessible to the Control Center user.

## Global Expression Values

Warehouse Builder makes additional data values available to the expression from the current activity and the owning process flow.

Table 8–2 lists these global expression values.

***Table 8–2 Global Expression Values***

| Identifier | Type | Description |
|---|---|---|
| NUMBER_OF_ERRORS | NUMBER | Number of errors reported on completion of activity execution |
| NUMBER_OF_WARNINGS | NUMBER | Number of warnings reported on completion of activity execution |
| RETURN_RESULT | VARCHAR2(64) | Textual representation of result. For example, 'SUCCESS,' 'WARNING,' 'ERROR' |
| RETURN_RESULT_NUMBER | NUMBER | Enumeration of RESULT_RESULT1 = SUCCESS2 = WARNING3 = ERROR |
| RETURN_CODE | NUMBER | An integer, 0 to 255, specific to the activity, synonymous with an Operating System return code |
| PARENT_AUDIT_ID | NUMBER | The audit ID of the calling Process Flow |
| AUDIT_ID | NUMBER | The audit ID of the activity |

Table 8–3 lists the additional constants provided.

***Table 8–3 Additional Constants***

| Identifier | Type | Description |
|---|---|---|
| SUCCESS | NUMBER | SUCCESS enumerated value |

*Table 8–3 (Cont.) Additional Constants*

| Identifier | Type | Description |
|---|---|---|
| WARNING | NUMBER | WARNING enumerated value |
| ERROR | NUMBER | ERROR enumerated value |

## Defining Transition Conditions

Use the Transition Editor to specify one of the enumerated conditions or to write an expression for a complex condition. The enumerated conditions include success, warning, and error. These are displayed on the canvas as shown in Table 8–4.

*Table 8–4 Types of Conditions for Transitions*

| Icon | Transition Condition | Description |
|---|---|---|
| | Success | The process flow continues only if the preceding activity ends in success. |
| | Warning | The process flow continues only if the preceding activity ends with warnings. |
| | Error | The process flow continues only if the preceding activity ends in error. |
| | Complex | The process flow continues only if the preceding activity returns a value that meets the criteria you specify in an expression. |
| | Extended | The process flow continues only if the preceding notification activity ends with an extended result. |

The extended transition condition is valid only for Notification activities, because this is the only type of activity that returns an extended result. The activity acquires this icon when it is set to an outcome of `#MAIL, #NOMATCH, #TIE, or #TIMEOUT`.

Table 8–5 lists the output and the description of the Extended transition.

*Table 8–5 Output and Description of the Extended Transition*

| Output | Description |
|---|---|
| #NOMATCH | Result of a voting notification where no candidate acquired the minimum number of votes to win. |
| #TIE | Result of a voting notification where the result was a tie. |
| #MAIL | A mail error occurred for the notification. Some recipients did not receive an e-mail notification, so it was canceled. |
| #TIMEOUT | The notification did not receive a response within the configured amount of time. |

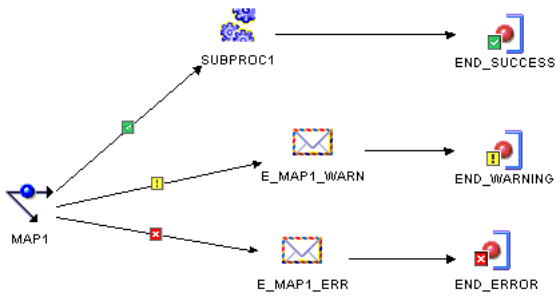
If the activity has only one outgoing activity, then you can specify any of the conditions listed in Table 8–4 or leave the transition as unconditional.

The rules for using multiple outgoing transitions depend on the type of activity. The general rule is that you can use an unlimited number of complex conditions in addition to one of each of the following: SUCCESS, WARNING, ERROR, and UNCONDITIONAL. The exception to this rule is when you use control activities such as AND, FORK, and OR.

When you add multiple outgoing transitions from an activity, ensure that the conditions do not conflict. A conflict occurs when the process flow logic evaluates that more than one outgoing transition is true.

Figure 8–5 shows a portion of a process flow in which different activities are triggered based on the three possible completion states of MAP1. Because only one of these conditions can be satisfied at a time, there is no conflict. If you attempt to add an unconditional transition or another conditional transition, then two transition conditions would be true and the process flow would be invalid.

***Figure 8–5 Outgoing Transition Conditions***




# Example: Using Process Flows to Access Flat Files with Variable Names

### Scenario

Your company relies on a legacy system that writes data to a flat file on a daily basis and assigns a unique name to the file based on the date and time of its creation. You would like to create a mapping that uses the generated flat files as a source, and transforms and loads the data to a relational database. However, mappings require files to have permanent names and, in this situation, the name of the source file changes each time the file is created.

### Solution

In Warehouse Builder, you can design a process flow that locates the generated file in a specific directory, renames it to a permanent name that you designate, and starts a dependent mapping. You can now use the permanent flat file name as the source for your mapping.

### Case Study

This case study describes how to create a process flow and a mapping to extract data from a legacy system that generates flat files with variable names. The process flow relies on the use of a User Defined activity. Assume the following information for the purposes of this case study:

- **Generated Flat File:** The legacy system generates a flat file containing sales data on a daily basis. It saves the file to the `c:\staging_files` directory and names the file based on the time and date, such as `sales010520041154.dat`. Every generated file is saved to the same directory and begins with the word `sales`, followed by the timestamp information.
- **Permanent Flat File Name:** You decide to rename the generated file name to `s_data.dat`. This is the name that you reference as the flat file source in the mapping.

- **Process Activity:** You design a process flow named `OWF_EXT` to execute batch commands in DOS to copy the generated file, save it as `s_data.dat`, and delete the originally generated file.

Your objective is to create logic that ensures the generated flat file is renamed appropriately before it triggers the execution of a mapping.

**To extract data from a generated flat file with a name that varies with each generation, refer to the following sections:**

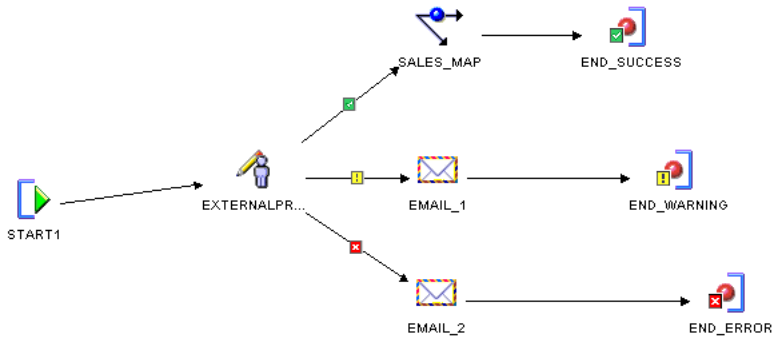
1. "Creating the Process Flow" on page 8-19
2. "Setting Parameters for the User Defined Activity" on page 8-19
3. "Configuring the User Defined Activity" on page 8-21
4. "Designing the Mapping" on page 8-21
5. "Deploying and Executing" on page 8-22

## Creating the Process Flow

Create a process flow that starts a mapping on the condition that the User Defined activity completes successfully. For more information about creating the process flow, see "Steps for Defining Process Flows" on page 8-5.

Figure 8–6 displays the process flow you create to extract data from a generated flat file.

***Figure 8–6 Process Flow with User Defined Activity Transitioning to a Mapping***




## Setting Parameters for the User Defined Activity

This section describes how to specify the DOS commands for renaming the generated file. The DOS commands that you issue from the User Defined activity should be similar to the following:

```
copy c:\staging_files\sales*.* c:\staging_files\s_data.dat
del c:\staging_files\sales*.*
```

The first command copies the temporary file into a file with a fixed name `s_data.dat`. The second command deletes the originally generated file.

You can either direct Warehouse Builder to a file containing the script of commands or you can store the commands in the Warehouse Builder user interface. Choose one of the following methods:

- Method 1: Write a script Within Warehouse Builder

- Method 2: Call a script maintained outside of Warehouse Builder

### Method 1: Write a script Within Warehouse Builder

Choose this method when you want to maintain the script in Warehouse Builder. Consider using this method when the script is small and need not be very flexible.

For this method, write or copy and paste the script into the Value column of the SCRIPT parameter. In the COMMAND parameter, enter the path to the DOS shell command, such as `c:\winnt\system32\cmd.exe`. Also, type the `${Task.Input}` variable into the Value column of the PARAMETER_LIST parameter.

Although this case study does not illustrate it, you can use substitution variables in the script when you maintain it in Warehouse Builder. This prevents you from having to update activities when server files, accounts, and passwords change.

Table 8–6 lists the substitute variables that you can type for the User Defined activity. *Working* refers to the computer hosting the Runtime Service, the *local* computer in this case study. *Remote* refers to a server other than the Runtime Service host. You designate which server is remote and which is local when you configure the activity, as described in "Configuring the User Defined Activity" on page 8-21. These values are set when you register the locations at deployment.

***Table 8–6 Substitute Variables for the User Defined Activity***

| Variable | Value |
|---|---|
| ${Working.Host} | The host value for the location of the Runtime Service host |
| ${Working.User} | The user value for the location of the Runtime Service host |
| ${Working.Password} | The password value for the location of the Runtime Service host |
| ${Working.RootPath} | The root path value for the location of the Runtime Service host |
| ${Remote.Host} | The host value for a location other than the Runtime Service host |
| ${Remote.User} | The user value for a location other than the Runtime Service host |
| ${Remote.Password} | The password value for a location other than the Runtime Service host |
| ${Remote.RootPath} | The root path value for a location other than the Runtime Service host |
| ${Deployment.Location} | The deployment location |

### Method 2: Call a script maintained outside of Warehouse Builder

If extra maintenance is not an issue, you can point Warehouse Builder to a file containing a script including the necessary commands. This method is more flexible, as it enables you to pass in parameters during execution of the process flow.

The following example shows how to call an external process script outside of Warehouse Builder and illustrates how to pass parameters into the script during execution of the process flow. This example assumes a Windows operating system. For other operating systems, issue the appropriate equivalent commands.

#### To call a script outside the User Defined activity:

1. Write the script and save it to the file directory. For example, you can write the following script and save it as `c:\staging_files\rename_file.bat`:

   ```
   copy c:\staging_files\%1*.dat c:\staging_files\s_data.dat

   del c:\staging_files\%1*.dat
   ```

This sample script passes a parameter %1 to the script during the execution of the process flow. This parameter represents a string containing the first characters of the temporary file name, such as `sales010520041154.`

2. Select the Start activity on the canvas to view and edit activity parameters in the Structure view.

   To add a start parameter, select the Start activity on the canvas, and click **Add New Activity Parameter** on the Structure tab. Create a start parameter named `FILE_STRING`. During execution, Warehouse Builder will prompt you to type a value for `FILE_STRING` to pass on to the `%1` parameter in the `rename_file.bat` script.

3. Select the User Defined activity on the canvas and edit its parameters.

   For the COMMAND parameter, enter the path to the script in the column labeled Value. If necessary, use the scroll bar to scroll down and reveal the column. For this example, enter `c:\staging_files\rename_file.bat.`

   For PARAMETER_LIST, click the row labeled Binding and select the parameter that you defined for the start activity, FILE_STRING.

   Accept the defaults for all other parameters for the external process.

## Configuring the User Defined Activity

When you apply conditions to the outgoing transitions of a User Defined activity, you must define the meaning of those conditions when you configure the User Defined activity.

**To configure the User Defined activity:**

1. Right-click the process flow on the navigation tree and select **Configure.**

   The configuration properties for the process flow are displayed in a new tab.

2. Expand the User Defined Activities node, then the User Defined activity, and the Path Settings node. Warehouse Builder displays the configuration settings.

3. Complete this step if you wrote the script in the Warehouse Builder user interface using the substitution variables related to Remote Location, Working Location, and Deployment Location. Use the list to select the values.

   Because this case study does not use substitution variables, accept the default values.

4. Set the Deployed Location to the computer where you deploy the process flow.

5. Under the Execution Settings node, set **Use Return as Status** to true.

   This ensures that the process flow uses the external process return codes for determining which outgoing transition to activate. For the process flow in this case study, if the external process returns a success value, the process flow continues down the success transition and executes the downstream mapping.

## Designing the Mapping

Now you can design a mapping with `s_data.dat` as the source. You can create a PL/SQL mapping or a SQL*Loader mapping. For PL/SQL, map the flat file source to an external table and design the rest of the mapping with all the operators available for a PL/SQL mapping. For SQL*Loader, map the flat file source to a staging table and limit the mapping to those operators permitted in SQL*Loader mappings.

## Deploying and Executing

Deploy the mapping. Also, deploy the process flow package or module containing the process flow `OWF_EXT`.

Execute the process flow manually. When you execute the process flow, Warehouse Builder prompts you to enter values for the parameter that you created to pass into the script, FILE_STRING. For this case study, enter `?sales` where the question mark is the separator. The external activity then executes the command `rename_file.bat sales`.

## Subsequent Steps

After you successfully execute the process flow manually, consider creating a schedule.You can define a daily schedule to execute the process flow and, therefore, the mapping. Use schedules to plan when and how often to execute operations such as mappings and process flows that you deploy through Warehouse Builder.

> **See Also:** "Defining Schedules" on page 11-2 for information about defining schedules

# Example: Using Process Flows to Transfer Remote Files

### Scenario

Developers at your company designed mappings that extract, transform, and load data. The source data for the mapping resides on a server separate from the server that performs the ETL processing. You would like to create logic that transfers the files from the remote computer and triggers the dependent mappings.

### Solution

In Warehouse Builder, you can design a process flow that executes file transfer protocol (FTP) commands and then starts a mapping. For the process flow to be valid, the FTP commands must involve transferring data either from or to the server with the Runtime Service installed. To move data between two computers, neither of which hosts the Runtime Service, first transfer the data to the Runtime Service host computer and then transfer the data to the second computer.

You can design the process flow to start different activities depending upon the success or failure of the FTP commands.

### Case Study

This case study describes how to transfer files from one computer to another and start a dependent mapping. The case study provides examples of all the necessary servers, files, and user accounts.

- **Data host computer:** For the computer hosting the source data, you need a user name and password, host name, and the directory containing the data. In this case study, the computer hosting the data is a UNIX server named `salessrv1`. The source data is a flat file named `salesdata.txt` located in the `/usr/stage` directory.
- **Runtime Service host computer:** In this case study, Warehouse Builder and the Runtime Service are installed on a computer called `local` with a Windows operating system. `local` executes the mapping and the process flow.
- **Mapping:** This case study includes a mapping called `salesresults` that uses a copy of `salesdata.txt` stored on `local` at `c:\temp` as its source.

- **FTP Commands:** This case study illustrates the use of a few basic FTP commands on the Windows operating system.

Your objective is to create logic that ensures the flat file on `salessrv1` is copied to the `local` computer, and then, trigger the execution of the `salesresults` mapping.

**To transfer files and start a dependent mapping, see the following sections:**

1. "Defining Locations" on page 8-23.
2. "Creating the Process Flow" on page 8-23
3. "Setting Parameters for the FTP Activity" on page 8-24
4. "Configuring the FTP Activity" on page 8-26
5. "Registering the Process Flow for Deployment" on page 8-26

After you complete the instructions in the above sections, you can run the process flow.

## Defining Locations

Locations are logical representations of the various data sources and destinations in the warehouse environment. In this scenario, the locations are the logical representations of the host and path name information required to access a flat file. Warehouse Builder requires these definitions for deploying and running the process flow. When you deploy the process flow, Warehouse Builder prompts you to type the host and path name information associated with each location. You must define locations for each computer involved in the data transfer.

To define locations, right-click the appropriate Locations node in the Locations Navigator and select **New**. For salessrv1, right-click Files under the Locations node and create a location named `REMOTE_FILES`. Repeat the step for local and create the location `LOCAL_FILES`.

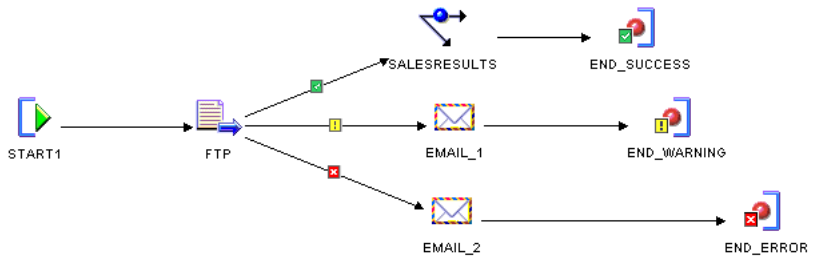
For the remote location, enter the host name, root path to the file, user name, and password. Warehouse Builder keeps the password secure. For the local location, only the host name is necessary.

## Creating the Process Flow

Use the Process Flow Editor to create a process flow with an FTP activity that transitions to the `salesresults` mapping on the condition of success.

Your process flow should appear similar to Figure 8–7.

***Figure 8–7 Process Flow with FTP Transitioning to a Mapping***



## Setting Parameters for the FTP Activity

This section describes how to specify the commands for transferring data from the remote server `salessrv1` to the `local` computer. You specify the FTP parameters by entering values for the FTP activity parameters on the Activity View.

Warehouse Builder offers you flexibility on how you specify the FTP commands. Choose one of the following methods:

- **Method 1: Write a script in Warehouse Builder:** Choose this method when you want to maintain the script in Warehouse Builder or when password security to servers is a requirement.

  For this method, write or copy and paste the script into the Value column of the SCRIPT parameter. In the COMMAND parameter, enter the path to the FTP executable, such as `c:\winnt\system32\ftp.exe.` Also, enter the `Task.Input` variable into the Value column of the PARAMETER_LIST parameter.

- **Method 2: Call a script maintained outside of Warehouse Builder:** If password security is not an issue, you can direct Warehouse Builder to a file containing a script including the FTP commands and the user name and password.

  To call a file on the file system, enter the appropriate command in PARAMETER_LIST to direct Warehouse Builder to the file. For a Windows operating system, enter the following:

  ```
  ?"-s:<file path\file name>"?
  ```

  For example, to call a file named `move.ftp` located in a temp directory on the C drive, enter the following:

  ```
  ?"-s:c:\temp\move.ftp"?
  ```

  Leave the SCRIPT parameter blank for this method.

### Example: Writing a Script in Warehouse Builder for the FTP Activity

The following example illustrates Method 1. It relies on a script and the use of substitution variables. The script navigates to the correct directory on `salessrv1` and the substitution variables are used for security and convenience.

This example assumes a Windows operating system. For other operating systems, issue the appropriate equivalent commands.

**To define a script within the FTP activity:**

1. Select the FTP activity on the canvas to view and edit activity parameters in the Property Inspector.
2. For the COMMAND parameter, enter the path to the FTP executable in the column labeled Value. If necessary, use the scroll bar to scroll to the right and reveal the column labeled Value.

   For Windows operating systems, the FTP executable is often stored at `c:\winnt\system32\ftp.exe`.
3. For the PARAMETER_LIST parameter, enter the `Task.Input` variable.

   When defining a script in Warehouse Builder and using Windows FTP, you must enter `?"-s:${Task.Input}"?` into PARAMETER_LIST.

   For UNIX, enter the following: `?"${Task.Input}"?`.
4. Navigate to and highlight the SCRIPT parameter in the Structure tab.

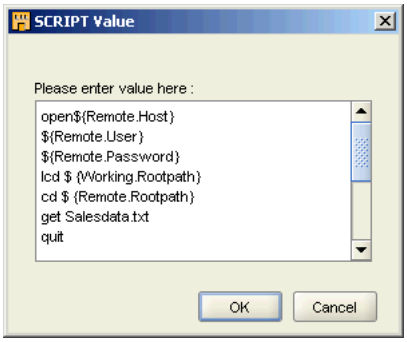
The Property Inspector displays the properties of the SCRIPT parameter.

5. Click the Ellipsis button to the right of the Value field displayed in the Property Inspector.

   Warehouse Builder displays the SCRIPT Value editor. Write or copy and paste FTP commands into the editor.

   Notice that the script in Figure 8–8 includes `${Remote.User}` and `${Remote.Password}`. These are substitution variables. See "Using Substitution Variables" on page 8-25 for more details.

***Figure 8–8 SCRIPT Value Editor Using Substitution Variables***




## Using Substitution Variables

Substitution variables are available only when you choose to write and store the FTP script in Warehouse Builder.

Use substitution variables to prevent having to update FTP activities when server files, accounts, and passwords change. For example, suppose that you create 10 process flows that use FTP activities to access a file on `salessrv1` under a specific directory. If the file is moved, then, without the use of substitution variables, you must update each FTP activity individually. With the use of substitution variables, you need only update the location information as described in "Defining Locations" on page 8-23.

Substitution variables are also important for maintaining password security. When Warehouse Builder executes an FTP activity with substitution variables for the server passwords, it resolves the variable to the secure password that you provided for the associated location.

Table 8–7 lists the substitute variables that you can provide for the FTP activity. `Working` refers to the computer hosting the Runtime Service, the *local* computer in this case study. `Remote` refers to the other server involved in the data transfer. You designate which server is remote and which is local when you configure the FTP activity. For more information, see "Configuring the FTP Activity" on page 8-26.

***Table 8–7 Substitute Variables for the FTP Activity***

| Variable | Value |
|---|---|
| ${Working.RootPath} | The root path value for the location of the Runtime Service host |
| ${Remote.Host} | The host value for the location involved in transferring data to or from the Runtime Service host |
| ${Remote.User} | The user value for the location involved in transferring data to or from the Runtime Service host |

*Table 8–7 (Cont.) Substitute Variables for the FTP Activity*

| Variable | Value |
|---|---|
| ${Remote.Password} | The password value for the location involved in transferring data to or from the Runtime Service host |
| ${Remote.RootPath} | The root path value for the location involved in transferring data to or from the Runtime Service host |

## Configuring the FTP Activity

As part of configuring the complete process flow, configure the FTP activity.

**To configure the FTP activity:**

1. Right-click the process flow on the navigation tree and select **Configure.**
2. Expand the FTP activity and the Path Settings. Warehouse Builder displays the configuration settings.
3. Set Remote Location to REMOTE_LOCATION and Working Location to LOCAL_LOCATION.
4. Click to select the **Use Return as Status**. This ensures that the process flow uses the FTP return codes for determining which outgoing transition to activate. For the process flow in this case study, if FTP returns a success value of 1, the process flow continues down the success transition and executes the `salesresults` mapping.

## Registering the Process Flow for Deployment

After you complete these instructions, you can deploy and run the process flow. To deploy the process flow, start the Deployment Manager by right-clicking and selecting **Deploy** from either the process flow module or package on the navigation tree. The Deployment Manager prompts you to register the REMOTE_LOCATION and the LOCAL_LOCATION.

Now you can run the process flow.

# 9

# Defining Custom Transformations

One of the main functions of an extract, transformation, and loading (ETL) tool is to transform data. Oracle Warehouse Builder provides several methods of transforming data. This chapter discusses transformations and describes how to create custom transformations using Warehouse Builder. It also describes how to import transformation definitions.

This chapter contains the following topics:


- About Transforming Data Using Warehouse Builder
- Defining Custom Transformations
- Editing Custom Transformations
- Importing Transformations
- Example: Reusing Existing PL/SQL Code
- Using Functions In Non-Oracle Platforms
- Configuring Functions


## About Transforming Data Using Warehouse Builder

Warehouse Builder provides an intuitive user interface that enables you to define transformations required for your source data. Use one of the following methods to transform source data.

- **Transformations:** The Design Center includes a set of transformations used to transform data. You can use the predefined transformations provided by Warehouse Builder or define custom transformations that suit your requirements.

  Custom transformations can be deployed to Oracle Database just like any other data object that you define in an Oracle module. For more information about transformations, see "About Transformations" on page 4-6.

- **Operators:** The Mapping Editor includes a set of prebuilt transformation operators that enable you to define common transformations when you define how data will move from source to target. Transformation operators are prebuilt PL/SQL functions, procedures, package functions, and package procedures. They take input data, perform operations on it, and produce output.

  In addition to the prebuilt operators, you can use custom transformations that you define in the Mapping Editor through the Transformation operator. For more information about these operators, see Chapter 26, "Data Flow Operators."

## Benefits of Using Warehouse Builder for Transforming Data

Warehouse Builder enables you to reuse PL/SQL as well as to write your own custom PL/SQL transformations. These custom transformations can be used in Warehouse Builder mappings.

All major relational database management systems support SQL and all programs written in SQL can be moved from one database to another with very little modification. This means that all the SQL knowledge in your organization is fully portable to Warehouse Builder. Warehouse Builder enables you to import and maintain any existing complex custom code.

# Defining Custom Transformations

Custom transformations include procedures, functions, and packages. Warehouse Builder provides wizards to create each type of custom transformation. Custom transformations can belong to the public Oracle Custom library or to a module in a project.

### Custom Transformations in the Public Oracle Custom Library

Custom transformations that are part of the public Oracle custom library can be used across all projects of the workspace in which they are defined. For example, you create a function called `ADD_EMPL` in the public Oracle Custom library of the workspace `REP_OWNER`. This procedure can be used across all the projects in `REP_OWNER`.

Use the Custom node of the Public Transformations node in the Globals Navigator to define custom transformations that can be used across all projects in the workspace.

**To create a custom transformation in the Public Oracle Custom Library:**

1. From the Globals Navigator, expand the Public Transformations node, and then the Oracle node.
2. Right-click the Custom node and select **New**.

   The New Gallery dialog box is displayed containing the type of transformations that you can create. This includes functions, procedures, and packages. Note that PL/SQL types can be created only as part of a package.
3. Select the type of transformation you want to create a click **OK**.
4. For table functions, Warehouse Builder displays the Create Table Function wizard. Use the wizard to define the table function as described in "Defining Table Functions" on page 9-4.

   For functions, procedures, and packages, the Create Function dialog box, Create Procedure dialog box, or Create Package dialog box respectively, is displayed. Provide a name and an optional description and click **OK**. For packages, the package is added to the Projects Navigator. For functions and procedures, the editor is displayed. Use the editor to define the function or procedure.

   See "Defining Functions and Procedures" on page 9-3 and "Defining PL/SQL Types" on page 9-7.

### Custom Transformations in a Project

Sometimes, you may need to define custom transformations that are required only in the current module or project. In such cases, you can define custom transformations in an Oracle module of a project. When you define a custom transformation in an Oracle module, the transformation is accessible from all the modules of the project in which it is defined. For example, consider the workspace owner called `REP_OWNER`, that

contains two projects, PROJECT1 and PROJECT2. In the Oracle module called SALES of PROJECT1, you define a procedure called CALC_SAL. This procedure can be used in all modules belonging to PROJECT1, but is not accessible in PROJECT2.

**To define a custom transformation in an Oracle module:**

1. From the Projects Navigator, expand the Oracle warehouse module node under which you want to define a custom transformation.
2. Right-click the Transformations node and select **New**.

   The New Gallery dialog box is displayed.
3. Select the type of transformation you want to create and click **OK**.

   For functions and procedures, Warehouse Builder displays the Create Function or Create Procedure dialog box. Provide a name and an optional description and click **OK**. The editor for that transformation is displayed. Use the tabs on the editor to define the transformation. For packages, after you define a name and description and click **OK**, the package is added to the Projects Navigator. You can then define the transformations that are part of the package.

   For table functions, Warehouse Builder displays the Welcome page of the Create Table Function Wizard. Note that you can create PL/SQL types only under a package.

   **See Also:** For more information about defining each type of transformation, see the following sections:

   - "Defining Functions and Procedures" on page 9-3
   - "Defining PL/SQL Types" on page 9-7
   - "Defining Table Functions" on page 9-4

## Defining Functions and Procedures

Complete the following steps using the Function Editor or Procedure Editor to define a function or procedure.

- Naming the Custom Transformation
- Defining the Parameters
- Specifying the Implementation

**Note:** You cannot copy and paste functions across platforms. For example, you cannot copy a function from an Oracle module and paste it into a SQL Server module.

### Naming the Custom Transformation

Use the Name and Description page or the Name tab to describe the custom transformation. Specify the following details on this page:

- **Name:** Represents the name of the custom transformation. For more information about naming conventions, see "Naming Conventions for Data Objects" on page 2-8.
- **Description:** Represents the description of the custom transformation. This is an optional field.

### Defining the Parameters

Use the Parameters tab to define, modify, or delete the parameters, both input and output, of the transformation. For functions, an additional field called Return Type is displayed. The Return Type field represents the data type of the value returned by the function. Select a return type from the available options in the list.

For transformations defined in an Oracle module, specify the following details for each parameter:

- **Name:** Enter the name of the parameter.
- **Data Type:** Select the data type of the parameter from the list.
- **I/O:** Select the type of parameter. The options available are Input, Output, and Input/Output.
- **Required:** Select **Yes** to indicate that a parameter is mandatory or **No** to indicate that it is not mandatory.
- **Default Value:** Enter the default value for the parameter. The default value is used when you do not specify a value for the parameter when you execute the function or procedure.

Transformations defined in a DB2 module contain the following details for each parameter: Name, Data Type, Length, Precision, Scale.

Length is applicable to character data types only and represents the length of the parameter. Precision represents the total number of digits allowed for the parameter and is applicable to numeric data types only. Scale represents the total number of digits to the right of the decimal point and is applicable to numeric data types only.

Transformations defined in a SQL Server module contain the following details for each parameter: Name, Data Type, Length, Precision, Scale, Required, Default Value.

### Specifying the Implementation

Use the Implementation tab to specify or modify the implementation details, such as the code, of the transformation. Click **Generate** to validate and generate the implementation code.

Ensure that you save changes made to the function or procedure definition before you close the editor. You can save changes by clicking **Save All** on the toolbar or selecting **Save** from the File menu.

## Defining Table Functions

Table functions are functions that take a set of rows as input and produce a set of rows as output. The input to the table function can be scalar data types, collection data types (PL/SQL records, Varrays, and nested tables), or Ref Cursors. The output of table functions is either a nested table or a Varray. Table functions can be queried like a regular database table.

Parallelization eliminates the need for intermediate staging of table function output by allowing you to stream rows returned by the table function directly to the next process.

Table functions enable you to define and use more flexible and powerful transformations. You can create your own specialized transformations, without using the transformation operators provided, to perform tasks such as user-defined aggregations and data mining. Table functions provide support for parallel and pipelined execution of transformations, resulting in better performance.

> **See Also:** *Oracle Database SQL Language Reference* for more information about table functions

Use the following steps to define a table function:

1. Naming the Table Function
2. Specifying the Return Type
3. Specifying Table Function Input and Output Parameters
4. Specifying Parallelism Options (optional)
5. Specifying Data Streaming Options (optional)
6. Specifying the Table Function Implementation

### Naming the Table Function

Use the following fields on the Name page to describe the table function.

**Name:** Represents the name of the table function. The name must follow the naming conventions for Warehouse Builder objects.

To rename a table function, select the name and enter the new name. Note that when you rename a table function, you must deploy it again. Also synchronize any mappings that use the table function.

**Description:** Represents an optional description, up to 4,000 characters long, for the table function.

### Specifying the Return Type

The return type for table functions can be the following collection types: nested tables and Varrays. The Return Type page displays the collection types that you can select as the return type. Select the collection type that you want to use as the return type of the table function.

For table functions defined under an Oracle module, you can use following as return type:

- Nested tables and Varrays defined in an Oracle module that is contained by the project in which the table function is defined
- Public nested tables that are defined as part of a package in the Globals Navigator

For public table functions, defined using the Globals Navigator, you can only use public nested tables or public Varrays as a return type.

### Specifying Table Function Input and Output Parameters

Use the Parameters page or Parameters tab to define the input parameters of the table function. For each parameter, enter the following details:

- **Name:** Enter the name of the parameter.
- **Type:** Select the data type of the parameter from the list.

  Parameters can be Oracle scalar data types or user-defined collection types, except nested tables. Typically the input parameters of table functions are the collection types such as Record Type, Table Type, or Ref Cursor Type. Since collection types are user-defined, you must define these types before you use them as data types for a table function parameter. Note that you can define collection types only as part of a public package or a package within an Oracle module.

- **I/O:** Select the type of parameter. The only option available for table functions is Input.
- **Required:** Select Yes to indicate that a parameter is mandatory and No to indicate that it is not mandatory.
- **Default Value:** Enter the default value for the parameter. The default value is used when you do not specify a value for the parameter when you execute the table function.

To modify a parameter, select the parameter value and enter the new value. Redeploy the table function after you make this change.

### Specifying Parallelism Options

You can parallelize the execution of table functions to eliminate the need for staging tables. When the execution of a table function is parallelized, the rows returned by the table function can be streamed directly into the next process without intermediate staging. This enables multithreaded, concurrent execution of table functions.

Parallel execution of table functions is performed using multiple slave processes. For a table function to be executed in parallel, you must specify one input parameter, of type Ref cursor, that is used for data partitioning.

Provide the following details to parallelize the execution of your table function:

- **Parallel:** Select this option to indicate that the execution of the table function should be parallelized.

  Note that this option is enabled only when one or more input parameters are of type Ref Cursor.
- **Partition Method:** Select the partition method. You can choose Any, Range, or Hash as the partition method.
- **Parameters:** Select the input parameter on which partitioning should be performed. Only parameters of type Ref cursor can be selected as partitioning parameters. Thus this field lists only input parameters of type Ref Cursor.
- **Attributes for Partitioning:** Select the attributes in the Ref cursor on which partitioning should be performed. The Available Attributes section lists the attributes of the Ref cursor on which the table function input parameter is based. Select the attributes and use the arrows to move them to the Selected Attributes section.

### Specifying Data Streaming Options

Use the Order page to perform streaming on table functions. When you perform data streaming, the table function orders or clusters rows that it fetches from cursor arguments. Ordering or clustering is performed using a particular key or key columns. Clustering causes rows that have the same input key values to appear together, but does not perform any ordering of rows.

To perform data streaming, enter the following information on this page.

- **Ordering Method:** Specify the method used for data streaming. You can select Order By to order rows or Cluster By to cluster rows.
- **Attributes for Ordering:** Select the attributes on which the ordering or clustering is performed. The Available Attributes section lists the attributes of the Ref cursor input parameter. Select one or more attributes and use the arrows to move the attributes to the Selected Attributes section.

### Specifying the Table Function Implementation

On the Implementation page, specify the following details:

- **Pipelined:** Select the **Pipelined** option to create a pipelined table function. Pipelining iteratively returns rows as they are produced, instead of returning them in a single batch after all the table function processing is complete. Pipelining enables table functions to return rows faster and reduces the memory required to cache table function results. Thus query response times are reduced. Pipelining enables tables functions to be used as a virtual table.
- **Implementation:** In the Implementation section, a sample code is provided with comments for each part of the table function definition. Click **Code Editor** to display the Code Editor that enables you to edit the default sample code and enter the code for your table function.

## Defining PL/SQL Types

Use the Create PL/SQL Type Wizard to create PL/SQL types. PL/SQL types must be defined within a package and they cannot exist independently.

### About PL/SQL Types

PL/SQL types enable you to create collection types, record types, and REF cursor types in Warehouse Builder. You use PL/SQL types as parameters in subprograms or as return types for functions. Using PL/SQL types as parameters to subprograms enables you to process arbitrary number of elements. Use collection types to move data into and out of database tables using bulk SQL. For more information about PL/SQL types, see *Oracle Database PL/SQL Language Reference*.

Warehouse Builder enables you to create the following PL/SQL types:

- PL/SQL Record types

  Record types enable you to define records in a package. A record is a composite data structure that contains multiple fields. Use records to hold related items and pass them to subprograms using a single parameter.

  For example, an `EMPLOYEE` record can contain details related to an employee such as ID, first name, last name, address, date of birth, date of joining, and salary. You can create a record type based on the `EMPLOYEE` record and use this record type to pass employee data between subprograms.

- REF Cursor types

  REF cursor types enable you to define REF cursors within a package. REF cursors are not bound to a single query and can point to different result sets. Use REF cursors when you want to perform a query in one subprogram and process the results in another subprogram. REF cursors also enable you to pass query result sets between PL/SQL stored subprograms and various clients such as an OCI client or an Oracle Forms application.

  REF cursors are available to all PL/SQL clients. For example, you can declare a REF cursor in a PL/SQL host environment such as an OCI or Pro*C program, then pass it as an input host variable (bind variable) to PL/SQL. Application development tools such as Oracle Forms, which have a PL/SQL engine, can use cursor variables entirely on the client side. Or, you can pass cursor variables back and forth between a client and the database server through remote procedure calls.

- Nested Table types

Use nested table types to define nested tables within a package. A nested table is an unordered set of elements, all of the same data type. They are similar to one-dimensional arrays with no declared number of elements. Nested tables enable you to model multidimensional arrays by creating a nested table whose elements are also tables.

For example, you can create a nested table type that can hold an arbitrary number of employee IDs. This nested table type can then be passed as a parameter to a subprogram that processes only the employee records contained in the nested table type.

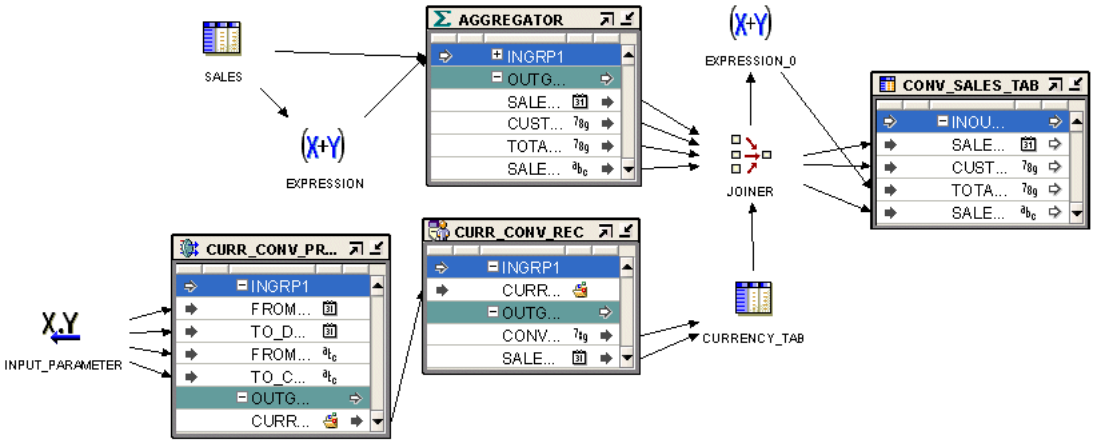
## Usage Scenario for PL/SQL Types

The SALES table stores the daily sales of an organization that has offices across the world. This table contains the sale ID, sale date, customer ID, product ID, amount sold, quantity sold, and currency in which the sale was made. Management wants to analyze global sales for a specified time period using a single currency, for example the U.S. Dollar. Thus all sales values must be converted to U.S. Dollar. Because the currency exchange rates can change every day, the sales amounts must be computed using the exchange rate of the sale currency on the sale date.

### Solution Using PL/SQL Record Types

Figure 9–1 displays the mapping that you use to obtain the sales amount in a specified currency using PL/SQL record types.

***Figure 9–1 PL/SQL Record Type in a Mapping***




The mapping takes the individual sales data stored in different currencies, obtains the sales value in the specified currency, and loads this data into a target table. Use the following steps to create this mapping.

1. In the Globals Navigator, create a package. In this package, create a procedure called CURR_CONV_PROC.

   This procedure obtains the currency conversion values on each date in a specified time interval from a Web site. The input parameters of this procedure are the sales currency, the currency to which the sale value needs to be converted, and the time interval for which the currency conversion is required. This data is stored in a PL/SQL record type of type CURR_CONV_REC. This record type contains two attributes: date and conversion value.

You create the PL/SQL record type as part of the package.

2. Create a mapping that contains a Transformation operator. This operator is bound to the `CURR_CONV_PROC` procedure.
3. Use a Mapping Input Parameter operator to provide values for the input parameters of the Transformation operator.

   The output group of the Transformation operator is a PL/SQL record type of type `CURR_CONV_REC`.
4. Use an Expand Object operator to obtain the individual values stored in this record type and store these values in the table `CURRENCY_TAB`.
5. Use an Aggregator operator to aggregate sales details for each order.

   The `SALES` table is a transactional table and stores data in normalized form. To obtain the aggregate sales for each order, use an Aggregator operator to aggregate sales data.
6. Use a Joiner operator to join the aggregated sales details, which is the output of the Aggregator operator, with the data in the `CURRENCY_TAB` table. The sale date is used as the join condition.
7. Use the Expression operator to multiply the sales amount with the currency exchange rate to get the total sales in the required currency. Load the converted sales data into the `CONV_SALES_TAB` table.

### Creating PL/SQL Types

You can create PL/SQL types in the Projects Navigator or Globals Navigator of the Design Center.

Use the Create PL/SQL Types Wizard to create PL/SQL types. To display the Create PL/SQL Types Wizard, right-click the PL/SQL Types node under a package, and select **New PL/SQL Type**. The Welcome page of the Create PL/SQL Types Wizard is displayed. Click **Next** and then the wizard guides you through the following pages:

- Name and Description Page
- Attributes Page
- Return Type Page
- Summary Page

### Name and Description Page

Use the Name and Description page to provide the name and an optional description for the PL/SQL type. Also use this page to select the type of PL/SQL type that you want to create.

You can create any of the following PL/SQL types:

- PL/SQL record type
- REF cursor type
- Nested table type

For more information about each PL/SQL type, see "About PL/SQL Types" on page 9-7.

After specifying the name and selecting the type of PL/SQL type to create, click **Next**.

### Attributes Page

Use the Attributes page to define the attributes of the PL/SQL record type. You specify attributes only for PL/SQL record types. A PL/SQL record must have at least one attribute.

For each attribute, define the following:

- **Name:** The name of the attribute. The name should be unique within the record type.
- **Data Type:** The data type of the attribute. Select the data type from the list.
- **Length:** The length of the data type, for character data types.
- **Precision:** The total number of digits allowed for the attribute, for numeric data types.
- **Scale:** The total number of digits to the right of the decimal point, for numeric data types.
- **Seconds Precision:** The number of digits in the fractional part of the datetime field. It can be a number between 0 and 9. Seconds Precision is used only for `TIMESTAMP`, `TIMESTAMP WITH TIME ZONE`, and `TIMESTAMP WITH LOCAL TIME ZONE` data types.

Click **Next** to proceed to the next step.

### Return Type Page

Use the Return Type page to select the return type of the PL/SQL type. You must specify a return type only while creating REF cursors and nested tables.

**To define REF cursors:**

The return type for a REF cursor can only be a PL/SQL record type. If you know the name of the PL/SQL record type, then you can search for it by entering the name in the **Search For** field and clicking **Go**.

The area below the Search For field displays the available PL/SQL types. These PL/SQL types are grouped under the two nodes: Public and Private. Expand the Public node to view the PL/SQL types that are part of the Oracle Shared Library. The types are grouped by package name. The Private node contains PL/SQL types that are created as part of a package in an Oracle module. Only PL/SQL types that belong to the current project are displayed. Each Oracle module is represented by a node. Within the module, the PL/SQL types are grouped by the package to which they belong.

**To define nested tables:**

For nested tables, the return type can be a scalar data type or a PL/SQL record type. Select one of the following options based on what the PL/SQL type returns:

- Select a scalar type as return type

  This option enables you to create a PL/SQL type that returns a scalar type. Use the list to select the data type.

- Select a PL/SQL record as return type

  This option enables you to create a PL/SQL type that returns a PL/SQL record type. If you know the name of the PL/SQL record type that is returned, then type the name in the **Search For** field and click **Go**. The results of the search are displayed in the area below the option.

You can also select the return type from the list of available types displayed. The area below this option contains two nodes: Public and Private. The Public node contains PL/SQL record types that are part of the Oracle Shared Library. The PL/SQL record types are grouped by the package to which they belong. The Private node contains the PL/SQL record types created as transformations in each Oracle module in the current project. These are grouped by module. Select the PL/SQL record type that the PL/SQL type returns.

Click **Next** to proceed with the creation of the PL/SQL type.

#### Summary Page

The Summary page displays the options that you have chosen on the wizard pages. Review the options. Click **Back** to modify any options. Click **Finish** to create the PL/SQL type.

## Editing Custom Transformations

You can edit the definition of a custom transformation using the editors. Make sure you edit properties consistently. For example, if you change the name of a parameter, then you must also change its name in the implementation code.

After editing a custom transformation, ensure that you do the following:

- Synchronize any mapping operators that reference the edited transformation with the edited transformation

  Synchronization updates the definition of the transformation in the mapping with the changes made while editing.
- Redeploy the mapping containing the edited transformation

### Editing Function or Procedure Definitions

The Edit Function dialog box enables you to edit function definitions. To edit a procedure definition, use the Edit Procedure dialog box.

**Use the following steps to edit functions, procedures, or packages:**

1. From the Projects Navigator, expand the Oracle module in which the transformation is created. Then expand the Transformations node.

   To edit a transformation that is part of the public Oracle Custom library, from the Globals Navigator, expand the Public Transformations node, and then the Custom node.
2. Right-click the name of the function, procedure, or package that you want to edit and select **Open.** Or, double-click the name of the function, procedure, or package.

   For functions, the Function Editor is displayed. For procedures, the Procedure Editor is displayed. Use the following tabs to edit the function or procedure definition:

   - Name tab, see Naming the Custom Transformation on page 9-3
   - Parameters tab, see Defining the Parameters on page 9-4
   - Implementation tab, see Specifying the Implementation on page 9-4

   For packages, Warehouse Builder displays the Edit Transformation Library dialog box. You can only edit the name and description of the package. You can edit the

functions and procedures contained within the package using the steps used to edit functions or packages.

## Editing PL/SQL Types

The Edit PL/SQL Type dialog box enables you to edit the definition of a PL/SQL type. Use the following steps to edit a PL/SQL type:

1. From the Projects Navigator, expand the Oracle module that contains the PL/SQL type. Then expand the Transformations node.

   To edit a PL/SQL type stored in the public Oracle Custom library, expand the Public Transformations node in the Globals Navigator, and then the Custom node.

2. Expand the package that contains the PL/SQL type and then the PL/SQL Types node.

3. Right-click the name of the PL/SQL type that you want to edit and select **Open**. Or, double-click the name of the PL/SQL type.

   The Edit PL/SQL Type dialog box is displayed. Use the following tabs to edit the PL/SQL type:

   - Name Tab
   - Attributes Tab
   - Return Type Tab

### Name Tab

The Name tab displays the name and the description of the PL/SQL type. Use this tab to edit the name or the description of the PL/SQL type.

To rename a PL/SQL type, select the name and enter the new name.

### Attributes Tab

The Attributes tab displays details about the existing attributes of the PL/SQL record type. This tab is displayed for PL/SQL record types only. You can modify existing attributes, add new attributes, or delete attributes.

To add a new attribute, click the **Name** column of a blank row specify the details for the attribute. To delete an attribute, right-click the gray cell to the left the row that represents the attribute and select **Delete**.

### Return Type Tab

Use the Return Type tab to modify the details of the return type of the PL/SQL type. For a REF cursor type, the return type must be a PL/SQL record. For a nested table, the return type can be a PL/SQL record type or a scalar data type.

## Editing Table Functions

You can edit the definition of a table function and modify its specification.

**To edit table functions:**

1. Expand the Oracle Module that contains the table function, the Transformations node, and then the Table Functions node.

   For a global table function, expand the Public Transformations node and then the Custom node.

2. If the table function belongs to a package, first expand the package node. Right-click the name of the table function that you want to edit and select **Open**. Or, double-click the name of the table function.

   The Table Function Editor is displayed.

3. Use the following tabs to edit the table function:
   - Name tab, see "Naming the Table Function" on page 9-5
   - Return Type tab, see "Specifying the Return Type" on page 9-5
   - Parameters tab, see "Specifying Table Function Input and Output Parameters" on page 9-5
   - Partitions tab, see "Specifying Parallelism Options" on page 9-6
   - Order tab, see "Specifying Data Streaming Options" on page 9-6
   - Implementation tab, see "Specifying the Table Function Implementation" on page 9-7

## Importing Transformations

Use the Import Metadata Wizard to import PL/SQL functions, procedures, and packages into a Warehouse Builder project. You can also import scalar functions from IBM DB2 and SQL Server databases.

You can edit, save, and deploy the imported PL/SQL functions and procedures. You can also view and modify imported packages.

**To import transformations in to a project:**

1. From the Projects Navigator, expand the project node and then the Databases node.
2. Expand the node corresponding to the database from which you want to import transformations.

   For example, to import PL/SQL functions from an Oracle database, right-click the Oracle node. To import scalar functions from an IBM DB2UDB database, right-click the DB2 node.

3. Right-click the module into which you want to import transformations, select **Import,** and then **Database Objects**.

   Warehouse Builder displays the Welcome page of the Import Metadata Wizard.

4. Click **Next.**
5. In the Object Type field of the Filter Information page, select **PL/SQL Transformation** to import PL/SQL transformations into an Oracle module or select **Transformation** to import scalar functions into an IBM DB2 UDB or a SQL Server module.
6. Click **Next.**

   The Import Metadata Wizard displays the Object Selection page.

7. Select a function, procedure, or package from the Available Objects list. Move the objects to the Selected Objects list by clicking the right arrow to move a single object or the Move All button to move multiple objects.
8. Click **Next.**

   The Import Metadata Wizard displays the Summary and Import page.

9. Verify the import information. Click **Back** to revise your selections.
10. Click **Finish** to import the selected PL/SQL transformations.

    Warehouse Builder displays the Import Results page.
11. Click **OK** proceed with the import. Click **Undo** to cancel the import process.

    The imported PL/SQL information appears under the Transformations node of the module into which you imported the data.

## Restrictions on Using Imported PL/SQL

The following restrictions apply to the use of imported PL/SQL:

- You cannot edit imported PL/SQL packages.
- Wrapped PL/SQL objects are not readable.
- You can edit the imported package body but not the imported package specification.

# Example: Reusing Existing PL/SQL Code

### Scenario

A movie rental company periodically updates the customer rental activity in its `CUST_RENTAL_ACTIVITY` table, where it stores the rental sales and overdue charges data for each customer. This table is used for different mailing campaigns. For example, in their latest mailing campaign, customers with high overdue charges are offered the company's new pay-per-view service.

Currently, the movie rental company uses a PL/SQL package to consolidate their data. The existing PL/SQL package needs to be maintained manually by accessing the database. This code runs on an Oracle 8*i* database.

```
CREATE OR REPLACE PACKAGE RENTAL_ACTIVITY AS
  PROCEDURE REFRESH_ACTIVITY(SNAPSHOT_START_DATE IN DATE);
END RENTAL_ACTIVITY;
/
CREATE OR REPLACE PACKAGE BODY RENTAL_ACTIVITY AS
 PROCEDURE REFRESH_ACTIVITY(SNAPSHOT_START_DATE IN DATE) IS
   CURSOR C_ACTIVITY IS
    SELECT
      CUST.CUSTOMER_NUMBER CUSTOMER_NUMBER,
      CUST.CUSTOMER_FIRST_NAME CUSTOMER_FIRST_NAME,
      CUST.CUSTOMER_LAST_NAME CUSTOMER_LAST_NAME,
      CUST.CUSTOMER_ADDRESS CUSTOMER_ADDRESS,
      CUST.CUSTOMER_CITY CUSTOMER_CITY,
      CUST.CUSTOMER_STATE CUSTOMER_STATE,
      CUST.CUSTOMER_ZIP_CODE CUSTOMER_ZIP_CODE,
      SUM(SALE.RENTAL_SALES) RENTAL_SALES,
      SUM(SALE.OVERDUE_FEES) OVERDUE_FEES
    FROM  CUSTOMER CUST, MOVIE_RENTAL_RECORD SALE
    WHERE SALE.CUSTOMER_NUMBER = CUST.CUSTOMER_NUMBER AND
          SALE.RENTAL_RECORD_DATE >= SNAPSHOT_START_DATE
    GROUP BY
    CUST.CUSTOMER_NUMBER,
    CUST.CUSTOMER_FIRST_NAME,
    CUST.CUSTOMER_LAST_NAME,
    CUST.CUSTOMER_ADDRESS,
    CUST.CUSTOMER_CITY,
```

```
    CUST.CUSTOMER_STATE,
    CUST.CUSTOMER_ZIP_CODE;

   V_CUSTOMER_NUMBER NUMBER;
   V_CUSTOMER_FIRST_NAME VARCHAR2(20);
   V_CUSTOMER_LAST_NAME VARCHAR2(20);
   V_CUSTOMER_ADDRESS VARCHAR(50);
   V_CUSTOMER_CITY VARCHAR2(20);
   V_CUSTOMER_STATE VARCHAR2(20);
   V_CUSTOMER_ZIP_CODE VARCHAR(10);
   V_RENTAL_SALES NUMBER;
   V_OVERDUE_FEES NUMBER;

BEGIN
   OPEN C_ACTIVITY;
   LOOP
    EXIT WHEN C_ACTIVITY%NOTFOUND;
    FETCH
     C_ACTIVITY
    INTO
     V_CUSTOMER_NUMBER,
     V_CUSTOMER_FIRST_NAME,
     V_CUSTOMER_LAST_NAME,
     V_CUSTOMER_ADDRESS,
     V_CUSTOMER_CITY,
     V_CUSTOMER_STATE,
     V_CUSTOMER_ZIP_CODE,
     V_RENTAL_SALES,
     V_OVERDUE_FEES;

    UPDATE CUST_ACTIVITY_SNAPSHOT
    SET
     CUSTOMER_FIRST_NAME = V_CUSTOMER_FIRST_NAME,
     CUSTOMER_LAST_NAME = V_CUSTOMER_LAST_NAME,
     CUSTOMER_ADDRESS = V_CUSTOMER_ADDRESS,
     CUSTOMER_CITY = V_CUSTOMER_CITY,
     CUSTOMER_STATE = V_CUSTOMER_STATE,
     CUSTOMER_ZIP_CODE = V_CUSTOMER_ZIP_CODE,
     RENTAL_SALES = V_RENTAL_SALES,
     OVERDUE_FEES = V_OVERDUE_FEES,
     STATUS_UPDATE_DATE = SYSDATE
    WHERE
     CUSTOMER_NUMBER = V_CUSTOMER_NUMBER;

    IF SQL%NOTFOUND THEN
       INSERT INTO CUST_ACTIVITY_SNAPSHOT
       ( CUSTOMER_NUMBER,
         CUSTOMER_FIRST_NAME,
         CUSTOMER_LAST_NAME,
         CUSTOMER_ADDRESS,
         CUSTOMER_CITY,
         CUSTOMER_STATE,
         CUSTOMER_ZIP_CODE,
         RENTAL_SALES,
         OVERDUE_FEES,
         STATUS_UPDATE_DATE )
       VALUES
       ( V_CUSTOMER_NUMBER,
         V_CUSTOMER_FIRST_NAME,
         V_CUSTOMER_LAST_NAME,
```

```
          V_CUSTOMER_ADDRESS,
          V_CUSTOMER_CITY,
          V_CUSTOMER_STATE,
          V_CUSTOMER_ZIP_CODE,
          V_RENTAL_SALES,
          V_OVERDUE_FEES,
          SYSDATE );
      END IF;
    END LOOP;
END REFRESH_ACTIVITY;
END RENTAL_ACTIVITY;
/
```

### Solution

This case study highlights the benefits of importing an existing custom PL/SQL package into Warehouse Builder and using its functionality to automatically maintain, update, and regenerate the PL/SQL code. Warehouse Builder enables you to automatically take advantage of new database features and upgrades by generating code that is optimized for new database versions. For example, if you have a PL/SQL package for Oracle 8*i*, then by importing it into Warehouse Builder you can generate code for Oracle 8*i*, Oracle 9*i*, Oracle 10*g*, or Oracle 11*g*.

Also, by importing a custom package and re-creating its operations through a Warehouse Builder mapping, you can transparently run and monitor the operations. Otherwise, you must manually access the database to verify and update the code. Warehouse Builder also enables you to perform lineage and impact analysis on all ETL operations while the Runtime Audit Browser monitors the running of the code and logs errors.

### Case Study

You can migrate the PL/SQL code into Warehouse Builder using the following steps:

- Step 1: Import the Custom PL/SQL Package
- Step 2: Create a "Black Box" Mapping by using a custom transformation in a Warehouse Builder mapping
- Step 3: Reimplement Custom Code into a Mapping by reimplementing the legacy PL/SQL code into a new Warehouse Builder mapping and phasing out the custom package
- Step 4: Generate Code for Oracle Database 11g

Follow these steps to handle a custom PL/SQL package in Warehouse Builder.

### Step 1: Import the Custom PL/SQL Package

In the Projects Navigator, expand the Transformations node under the Oracle module into which you want to import the PL/SQL package `refresh_activity(DATE)`. Use the Import Metadata Wizard to import the package by right-clicking Transformations, selecting **Import**, and then **Database Objects**. On the Filter Information page of this wizard, indicate that you are importing a PL/SQL Transformation.

After you finish the import, the package `refresh_activity(DATE)` appears under the Packages node of the Transformations folder.

### Step 2: Create a "Black Box" Mapping

You can use the `refresh_activity(DATE)` procedure directly in a mapping without making any changes to it. In the mapping, you add a Post-Mapping Process operator to the mapping, with the package `refresh_activity(DATE)` selected.

In this example, you can immediately take advantage of the existing custom code. The learning curve and investment on resources is minimal. You may decide to maintain all the existing and developed PL/SQL code in this manner, using Warehouse Builder only to develop new processing units. Warehouse Builder enables you to use mappings that use the legacy code along with the new mappings you create. In such a case, although you can generate code for these mappings in Warehouse Builder, you cannot use Warehouse Builder features to maintain, update, or audit the code.

Because the legacy code is used as a "black box" that is not transparent to Warehouse Builder, you still need to maintain the legacy code manually. Thus, you cannot take advantage of the Warehouse Builder features, such as runtime audit browser, lineage and impact analysis, and optimized code generation, that rely on infrastructure code and metadata available for Warehouse Builder generated mappings.

Follow the next steps to take advantage of these features in Warehouse Builder and to automatically maintain, monitor, and generate your PL/SQL code.

### Step 3: Reimplement Custom Code into a Mapping

To take advantage of the code generation, maintenance, and auditing features, you can reimplement the legacy PL/SQL code functionality using a mapping and phase out the custom "black box" package. The mapping created to provide the PL/SQL code functionality is called `Rental_Activity`.

The recommended method is to test out this new mapping by running it side by side with the "black box" mapping. If the testing is successful and the new mapping can perform all the operations included in the custom code, then the "black box" mappings can be phased out. Warehouse Builder enables you to maintain, update, and generate code from a mapping without performing manual updates in the database.

Figure 9–2 shows a sample of code generated from the `Rental_Activity` mapping that replicates the operations of the custom PL/SQL package for the movie rental company.

***Figure 9–2 Sample Code***

```
Code Viewer: RENTAL_ACTIVITY [0 error(s), 4 warning(s)]
Code  Edit  Search  View
160         UPDATE
161         /*+ APPEND PARALLEL(CUST_ACTIVITY, DEFAULT, DEFAULT) */
162           "CUST_ACTIVITY"
163         SET
164           "CUSTOMER_FIRST_NAME" = "CUST_ACT_0_CUSTOMER_FIRST_NAME",
165     "CUSTOMER_LAST_NAME" = "CUST_ACT_0_CUSTOMER_LAST_NAME",
166     "CUSTOMER_ADDRESS" = "CUST_ACT_0_CUSTOMER_ADDRESS",
167     "CUSTOMER_CITY" = "CUST_ACT_0_CUSTOMER_CITY",
168     "CUSTOMER_STATE" = "CUST_ACT_0_CUSTOMER_STATE",
169     "CUSTOMER_ZIP_CODE" = "CUST_ACT_0_CUSTOMER_ZIP_CODE",
170     "RENTAL_SALES" = "CUST_ACT_0_RENTAL_SALES",
171     "OVERDUE_FEES" = "CUST_ACT_0_OVERDUE_FEES",
172     "STATUS_UPDATE_DATE" = "CUST_ACT_0_STATUS_UPDATE_DATE"
173         WHERE
174           "CUSTOMER_NUMBER" = "CUST_ACT_0_CUSTOMER_NUMBER";
175         IF SQL%NOTFOUND THEN
176           INSERT INTO
177             "CUST_ACTIVITY"
178             ("CUSTOMER_FIRST_NAME",
179             "CUSTOMER_LAST_NAME",
180             "CUSTOMER_ADDRESS",
181             "CUSTOMER_CITY",
182             "CUSTOMER_STATE",
183             "CUSTOMER_ZIP_CODE",
184             "RENTAL_SALES",
185             "OVERDUE_FEES",
186             "STATUS_UPDATE_DATE",
187             "CUSTOMER_NUMBER")
188           VALUES
189             ("CUST_ACT_0_CUSTOMER_FIRST_NAME",
190             "CUST_ACT_0_CUSTOMER_LAST_NAME",
191             "CUST_ACT_0_CUSTOMER_ADDRESS",
192             "CUST_ACT_0_CUSTOMER_CITY",
193             "CUST_ACT_0_CUSTOMER_STATE",
194             "CUST_ACT_0_CUSTOMER_ZIP_CODE",
195             "CUST_ACT_0_RENTAL_SALES",
196             "CUST_ACT_0_OVERDUE_FEES",
197             "CUST_ACT_0_STATUS_UPDATE_DATE",
198             "CUST_ACT_0_CUSTOMER_NUMBER");
199         END IF;
Line 1 Column 1   Read Only   [PL/SQL] Row based   Windows: CR/LF
```

### Step 4: Generate Code for Oracle Database 11*g*

If you upgrade to Oracle 9*i* version of the database, you only need to redeploy the `Rental_Activity` mapping created in Step 3. Warehouse Builder generates code optimized for the new database version.

Figure 9–3 shows the `MERGE` statement from a sample of code generated for the same mapping for Oracle 9*i*.

***Figure 9–3 Sample Code for Oracle9i***

```
Code Viewer: RENTAL_ACTIVITY [0 error(s), 3 warning(s)]
Code  Edit  Search  View
38
39    MERGE
40    /*+ APPEND PARALLEL(CUST_ACTIVITY, DEFAULT, DEFAULT)  */
41    INTO
42      "CUST_ACTIVITY"
43    USING
44      (SELECT
45        "AGG"."CUSTOMER_FIRST_NAME$0" "CUSTOMER_FIRST_NAME",
46        "AGG"."CUSTOMER_LAST_NAME$0" "CUSTOMER_LAST_NAME",
47        "AGG"."CUSTOMER_ADDRESS$0" "CUSTOMER_ADDRESS",
48        "AGG"."CUSTOMER_CITY$0" "CUSTOMER_CITY",
49        "AGG"."CUSTOMER_STATE$0" "CUSTOMER_STATE",
50        "AGG"."CUSTOMER_ZIP_CODE$0" "CUSTOMER_ZIP_CODE",
51        "AGG"."RENTAL_SALES$0" "RENTAL_SALES",
52        "AGG"."OVERDUE_FEES$0" "OVERDUE_FEES",
53        SYSDATE "VALUE",
54        "AGG"."CUSTOMER_NUMBER$0" "CUSTOMER_NUMBER"
55      FROM  (SELECT
56          SUM("MOVIE_RENTAL_RECORD_1"."MOVIE_RENTAL_RATE") "RENTAL_SALES$0",
57          SUM("MOVIE_RENTAL_RECORD_1"."OVERDUE_CHARGE") "OVERDUE_FEES$0",
58          "MOVIE_RENTAL_RECORD_1"."CUSTOMER_NUMBER" "CUSTOMER_NUMBER$0",
59          "CUSTOMER"."CUSTOMER_ADDRESS" "CUSTOMER_ADDRESS$0",
60          "CUSTOMER"."CUSTOMER_ZIP_CODE" "CUSTOMER_ZIP_CODE$0",
61          "CUSTOMER"."CUSTOMER_LAST_NAME" "CUSTOMER_LAST_NAME$0",
Line 39 Column 10   Read Only   [PL/SQL] Set based   Windows: CR/LF
```

No manual steps are required to maintain and generate the new code. Also, you can transparently monitor and maintain their ETL operations. Warehouse Builder enables them to perform lineage and impact analysis on their mappings and the Runtime Audit Browser enables them to track and log errors when running the mappings.

# Using Functions In Non-Oracle Platforms

Starting with Oracle Warehouse Builder 11*g* Release 2 (11.2), you can create, import, and use predefined functions within non-Oracle platforms as well. This release extends the usage of functions to the following platforms:

- DB2
- SQL Server

Like with Oracle modules, you can create a new function in DB2 and SQL Server modules. Similarly, you can also import existing functions from a DB2 or SQL Server database. Warehouse Builder also provides predefined functions in the Globals Navigator.

---

**Note:** You cannot copy and paste functions across platforms. For example, you cannot copy a function from an Oracle module and paste it into a SQL Server module.

---

## Creating IBM DB2 and SQL Server Functions

When you define functions using the Custom node under the Public Transformations node of the Globals Navigator, they can be used only when you deploy objects to an Oracle Database location. However, you can use Warehouse Builder to load data into SQL Server and DB2 data objects also. In these cases, you may need to create user-defined functions to transform data for these platforms. Use the Databases node in the Projects Navigator to define functions for these databases.

Once you define functions for a DB2 or SQL Server database, you can use these functions in mappings and process flows with the help of the Transformation operator and Transformation activity, respectively. You can also publish these functions as Web Services.

### Defining IBM DB2 and SQL Server Functions

**To define an IBM DB2 or a SQL Server function:**

1. In the Projects Navigator, expand the Databases node.
2. Depending on whether you are creating a function in DB2 or SQL Server, expand the DB2 or SQL Server node.
3. Expand the Transformations node and right-click the Functions node and select **New Function**.

   The Create Function dialog box is displayed.
4. Enter a name and optional description for the function and click **OK**.

   The Function Editor is displayed.
5. On the Parameters tab, provide the following information:
   - **Return Type:** Select the data type for the return type of the global function.
   - **Parameters:** Each function parameter is represented by a row in the table below the Return Type field. To create a parameter, enter a name on a blank cell and provide details, such as the data type and default value, for the parameter. The list in the Data Type column is populated depending on the platform you choose in the Platform field.

> **Note:** For SQL Server functions, @ is automatically prefixed to each parameter name.

6. On the Implementation tab, enter the code that will be used to implement the function on the platform that you selected in the Parameters tab.
7. From the View menu, select **Code Templates**.

   The Code Templates tab is displayed in the Log window.
8. In the Code Templates tab, select the Function CT that will be used to generate code for the function.

   Warehouse Builder provides prebuilt Function CTs to generate code for DB2 and SQL Server databases. These Function CTs are located in the Globals Navigator under the BUILT_IN_CT node of the Public Code Templates folder. For DB2, you can use DB2_FCT and for SQL Server, you can use SQLSERVER_FCT.

### Importing a Function

You can import existing functions from DB2 and SQL Server databases. This is similar to importing Oracle functions.

**To import DB2 or SQL Server functions**:

1. Right-click an existing DB2 or SQL Server module, and select **Import**, **Database Object**.

   The Import Metadata Wizard is displayed.
2. In the Filter Information page, select **Transformations** from Object Type.
3. In the Object Selection page, select the required functions and move them from the Available field to the Selected field.
4. Verify the information in the Summary page and click **Finish** to begin the import.

The imported functions are now visible under the module in the Projects Navigator.

> **Note:** Warehouse Builder allows you to import overloaded DB2 functions. However you cannot import overloaded SQL Server functions.

### Predefined Generic Heterogeneous Functions

The Heterogeneous node under the Public Transformations node of the Globals Navigator contains predefined functions that you can use for Oracle, SQL Server, and DB2 platforms.

Generic heterogeneous functions are categorized as follows:

- Character
- Conversion
- Date
- Numeric
- Other

To view the function definition and the platform for which it is defined, double-click a function in any of the above categories. The Function Editor containing the Name,

Parameter, and Expression tabs is displayed. Click the Parameter tab to view the platform for which the function is defined, the function parameters, and the function return type. Click the Expression tab to view the expression used for the SQL function and the platform for which the function is defined.

For details about the semantics of the expression provided, the parameters, and parameter data types, refer to the documentation for the particular platform.

#### Using the Functions in Mappings

At the time of creating DB2 or SQL Server functions, you associate a code template with the function. Therefore, you can only use these functions in mappings that are created under the Template Mappings node in Projects Navigator. These mappings are different from the normal mappings as they are used in conjunction with code templates.

## Configuring Functions

After you define a function, you can configure it by setting configuration parameters using the Configuration panel.

The following sections list the configuration parameters supported by Warehouse Builder for the Oracle platform.

### Configuring Oracle Functions

You can set the following configuration parameters for functions defined on the Oracle platform.

#### AUTHID

Use this parameter to specify the privileges with which the function is executed. Select one of the following options:

- CURRENT_USER: Indicates that the function will be executed with the privileges of the current user, in the current user's schema. This limits the scope for name resolution. Oracle Database will look for the function by name in the current user's schema.
- DEFINER: Indicates that the function will be executed with the privileges assigned to the owner of the schema that the function resides in. All external names are to be resolved within the same schema.

#### Deterministic

Select this option to indicate that the function is deterministic. Deterministic functions return the same results for a given set of arguments every time that the function is executed.

Setting this option helps to avoid redundant function calls. If a stored function was called previously with the same arguments, then the previous result can be used. The function result should not depend on the state of session variables or schema objects. Otherwise, results might vary across calls. Only DETERMINISTIC functions can be called from a function-based index or a materialized view that has query-rewrite enabled.

### Parallel Enable

This parameter is an optimization hint. Select this parameter to indicate to the Oracle Database that the function should be executed in parallel whenever called from within a SQL query. The processing will be split between parallel processes (UNIX), or threads (Windows). Setting this option results in a speed improvement on multiprocessor systems.

### Pragma Autonomous Transaction

Selecting this option causes the PL/SQL compiler to mark the function as independent. This allows the function to suspend the main transaction (the one from which the function was invoked), and roll back or commit its own SQL operations.

# 10

# Understanding Performance and Advanced ETL Concepts

Use this chapter as a guide for creating ETL logic that meets your performance expectations.

This chapter contains the following topics:


- Best Practices for Designing PL/SQL Mappings
- Best Practices for Designing SQL*Loader Mappings
- Improved Performance through Partition Exchange Loading
- High Performance Data Extraction from Remote Sources


## Best Practices for Designing PL/SQL Mappings

This section addresses PL/SQL mapping design and includes:


- Set-Based Versus Row-Based Operating Modes
- About Committing Data in Warehouse Builder
- Committing Data Based on Mapping Design
- Committing Data Independently of Mapping Design
- Running Multiple Mappings Before Committing Data
- Ensuring Referential Integrity in PL/SQL Mappings

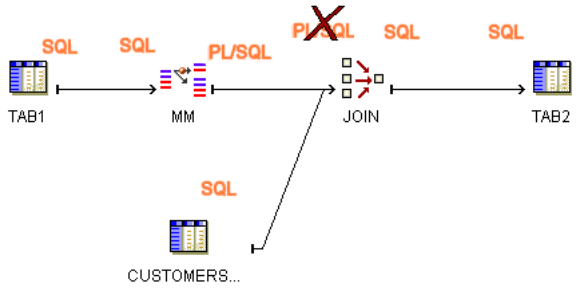

Oracle Warehouse Builder generates code for PL/SQL mappings that meet the following criteria:

- The output code of each operator satisfies the input code requirement of its next downstream operator.
- If the mapping contains an operator that generates only PL/SQL output, all downstream data flow operators must also be implemented by PL/SQL. You can use SQL operators in such a mapping only after loading the PL/SQL output to a target.

As you design a mapping, you can evaluate its validity by examining the input and output code types for each operator in the mapping.

For example, you can see that the mapping in Figure 10–1 is invalid because the Match Merge operator `MM` generates PL/SQL output, but the subsequent Joiner operator accepts SQL input only.

***Figure 10–1 Mapping Violates Input Requirement for Joiner Operator***




To achieve the desired results for the mapping, consider joining the source tables before performing the Match Merge or loading the results from the Match Merge to a staging table before performing the join.

Figure 10–2 displays a mapping in which source tables are joined before the match-merge operation. Figure 10–3 displays a mapping in which the results from the Match Merge operator are loaded into a staging table before performing the join.

***Figure 10–2 Valid Mapping Design with Sources Joined Before Match Merge***

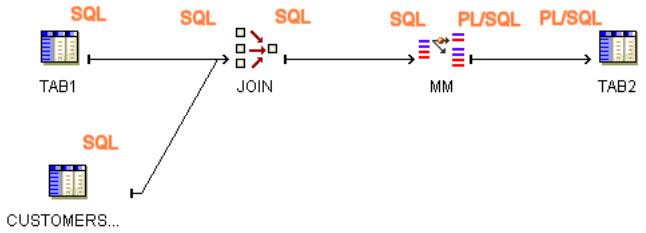



***Figure 10–3 Valid Mapping Design with Staging Table***

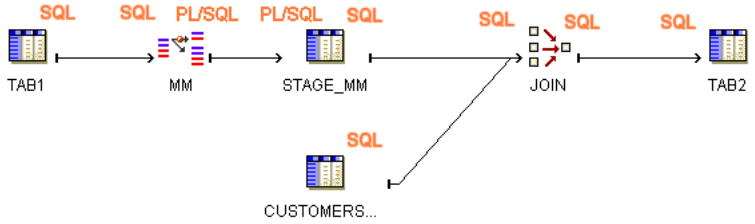



Table 10–1 and Table 10–2 list the implementation types for each Oracle Warehouse Builder operator. These tables also indicate whether or not PL/SQL code includes the operation associated with the operator in the cursor. This information is relevant in determining which operating modes are valid for a given mapping design. It also determines what auditing details are available during error handling.

*Table 10–1 Source-Target Operators Implementation in PL/SQL Mappings*

| Operator | Implementation Types | Valid in Set-Based Mode | Valid in Row-Based Mode | Valid in Row-Based (Target Only) |
|---|---|---|---|---|
| Source Operators: Tables, Dimensions, Cubes, Views, External Tables | SQL | Yes | Yes | Yes. Part of cursor. |
| Target Operators: Tables, Dimensions, Cubes, Views | SQL<br>PL/SQL | Yes, except when loading= UPDATE and database is not 10*g* or higher. | Yes | Yes. Not part of cursor. |
| Flat File as source | For PL/SQL, create an external table. | Yes | Yes | Yes. Part of the cursor. |
| Flat File as target | SQL | Yes, except when loading = DELETE or loading= UPDATE and database is not 10*g* or higher. | Yes | Yes. Not part of cursor. |
| Sequence as source | SQL | Yes | Yes | Yes, part of cursor. |

*Table 10–2 Data Flow Operator Implementation in PL/SQL Mappings*

| Operator Name | Implementation Types | Valid in Set-Based Mode | Valid in Row-Based Mode | Valid in Row-Based (Target Only) Mode |
|---|---|---|---|---|
| Aggregator | SQL | Yes | Yes, only if part of the cursor. | Yes, only if part of the cursor. |
| Constant Operator | PL/SQL<br>SQL | Yes | Yes | Yes |
| Data Generator | SQL*Loader Only | N/A | N/A | N/A |
| Deduplicator | SQL | Yes | Yes, only if part of the cursor | Yes, only if part of the cursor. |
| Expression | SQL<br>PL/SQL | Yes | Yes | Yes |
| Filter | SQL<br>PL/SQL | Yes | Yes | Yes |
| Joiner | SQL | Yes | Yes, only if part of the cursor. | Yes, only if part of the cursor. |
| Lookup | SQL<br>PL/SQL | Yes | Yes, except when the All Rows option is selected on the Multiple Match Rows page of the Lookup operator. | Yes, except when the All Rows option is selected on the Multiple Match Rows page of the Lookup operator. |
| Mapping Input Parameter | SQL<br>PL/SQL | Yes | Yes | Yes |
| Mapping Output Parameter | SQL<br>PL/SQL | Yes | Yes | Yes |

*Table 10–2 (Cont.) Data Flow Operator Implementation in PL/SQL Mappings*

| Operator Name | Implementation Types | Valid in Set-Based Mode | Valid in Row-Based Mode | Valid in Row-Based (Target Only) Mode |
|---|---|---|---|---|
| Match Merge | SQL input<br>PL/SQL output<br>(PL/SQL input from XREF group only) | No | Yes | Yes. Not part of cursor. |
| Name and Address | PL/SQL | No | Yes | Yes. Not part of cursor. |
| Pivot | SQL<br>PL/SQL | Yes | Yes | Yes |
| Post-Mapping Process | Irrelevant | Yes, independent of data flow | Yes | Yes |
| Pre-Mapping Process | Irrelevant | Yes, independent of data flow | Yes | Yes |
| Set | SQL | Yes | Yes, only if part of the cursor. | Yes, only if part of the cursor. |
| Sorter | SQL | Yes | Yes, only if part of the cursor. | Yes, as part of the cursor. |
| Splitter | SQL<br>PL/SQL | Yes | Yes | Yes |
| Table Function | SQL or PL/SQL input<br>SQL output only | Yes | Yes | Yes |
| Transformation as a procedure | PL/SQL | No | Yes | Yes. Not part of cursor. |
| Transformation as a function that does not perform DML | SQL<br>PL/SQL | Yes | Yes | Yes, included in the cursor. |

## Set-Based Versus Row-Based Operating Modes

For mappings with a PL/SQL implementation, select one of the following operating modes:

- Set-Based Mode
- Row-Based Mode
- Row-Based (Target Only) Mode
- Set-based fail over to row-based
- Set-based fail over to row-based (target only)

The default operating mode that you select depends upon the performance that you expect, the amount of auditing data that you require, and how you design the mapping. Mappings have at least one and as many as three valid operating modes, excluding the options for failing over to row-based modes. During code generation, Warehouse Builder generates code for the specified default operating mode as well as for the deselected modes. Therefore, at runtime, you can select to run in the default operating mode or any one of the other valid operating modes.

The types of operators in the mapping may limit the operating modes that you can select. As a general rule, mappings run in set-based mode can include any of the operators except for Match Merge, Name and Address, and Transformations used as procedures. Although you can include any of the operators in row-based and row-based (target only) modes, there are important restrictions on how you use SQL based operators such as Aggregators and Joins. To use SQL-based operators in either of the row-based modes, ensure that the operation associated with the operator can be included in the cursor.

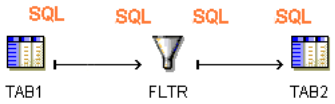
These general rules are explained in the following sections.

### Set-Based Mode

In set-based mode, Warehouse Builder generates a single SQL statement that processes all data and performs all operations. Although processing data as a set improves performance, the auditing information available is limited. Runtime auditing is limited to reporting of the execution error only. With set-based mode, you cannot identify the rows that contain errors.

Figure 10–4 shows a simple mapping and the associated logic that Warehouse Builder uses to generate code for the mapping when run in set-based operating mode. TAB1, FLTR, and TAB2 are processed as a set using SQL.

***Figure 10–4 Simple Mapping Run in Set-Based Mode***




To correctly design a mapping for the set-based mode, avoid operators that require row- by-row processing such as Match Merge and Name and Address operators. If you include an operator in the data flow that cannot be performed in SQL, then Warehouse Builder does not generate set-based code and displays an error when you execute the package in set-based mode.

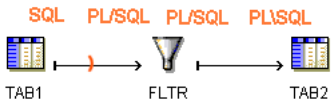
For target operators in a mapping, the loading types INSERT/UPDATE and UPDATE/INSERT are always valid for set-based mode. Warehouse Builder supports UPDATE loading in set-based mode only with Oracle Database is 10*g* or later. Warehouse Builder also supports the loading type DELETE in set-based mode. For a complete listing of how Warehouse Builder handles operators in set-based mappings, see Table 10–2 on page 10-3.

### Row-Based Mode

In row-based mode, Warehouse Builder generates statements that process data row by row. The select statement is in a SQL cursor. All subsequent statements are PL/SQL. You can access full runtime auditing information for all operators performed in PL/SQL and only limited information for operations performed in the cursor.

Figure 10–5 shows a simple mapping and the associated logic that Warehouse Builder uses to generate code for the mapping when run in row-based operating mode. TAB1 is included in the cursor and processed as a set using SQL. FLTR and TAB2 are processed row by row using PL/SQL.

***Figure 10–5 Simple Mapping Run in Row-Based Mode***




If the mapping includes any SQL-based operators that cannot be performed in PL/SQL, Warehouse Builder attempts to generate code with those operations in the cursor. To generate valid row-based code, design your mapping such that if you include any of the following SQL-based operators, Warehouse Builder can include the operations in the cursor:

- Aggregation
- Deduplicator
- Joiner
- Lookup
- Sequence
- Set
- Sorter

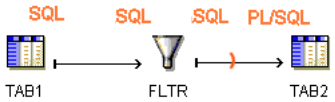
For the preceding operators to be included in the cursor, do not directly precede it by an operator that generates PL/SQL code. In other words, you cannot run the mapping in row-based mode if it contains a Transformation implemented as a procedure, a Flat File used as a source, a Match Merge, or Name and Address operator directly followed by any of the seven SQL-based operators. For the design to be valid, include a staging table between the PL/SQL generating operator and the SQL-based operator.

### Row-Based (Target Only) Mode

In row-based (target only) mode, Warehouse Builder generates a cursor select statement and attempts to include as many operations as possible in the cursor. For each target, Warehouse Builder inserts each row into the target separately. You can access full runtime auditing information for all operators performed in PL/SQL and only limited information for operations performed in the cursor. Use this mode when you expect fast set-based operations to extract and transform the data but need extended auditing for loading the data, which is where errors are likely to occur.

Figure 10–6 shows a simple mapping and the associated logic that Warehouse Builder uses to generate code for the mapping when run in row-based (target only) operating mode. TAB1 and FLTR are included in the cursor and processed as a set using SQL. TAB2 is processed row by row.

***Figure 10–6 Simple Mapping Run in Row-Based (Target Only) Mode***



Row-based (target only) mode places the same restrictions on SQL-based operators as the row-based operating mode. Additionally, for mappings with multiple targets, Warehouse Builder generates code with a cursor for each target.

## About Committing Data in Warehouse Builder

There are two major approaches to committing data in Warehouse Builder. You can commit or rollback data based on the mapping design. To do this, use one of the commit control methods described in "Committing Data Based on Mapping Design" on page 10-7.

Alternatively, for PL/SQL mappings, you can commit or rollback data independently of the mapping design. Use a process flow to commit the data or establish your own method as described in "Committing Data Independently of Mapping Design" on page 10-10.

## Committing Data Based on Mapping Design

By default, Warehouse Builder loads and then automatically commits data based on the mapping design. For PL/SQL mappings you can override the default setting and control when and how Warehouse Builder commits data. You have the following options for committing data in mappings:

**Automatic:** This is the default setting and is valid for all mapping types. Warehouse Builder loads and then automatically commits data based on the mapping design. If the mapping has multiple targets, then Warehouse Builder commits and rolls back each target separately and independently of other targets. Use the automatic commit when the consequences of multiple targets being loaded unequally are not great or are irrelevant.

**Automatic correlated:** Automatic correlated commit is a specialized type of automatic commit that applies to PL/SQL mappings with multiple targets only. Warehouse Builder considers all targets collectively and commits or rolls back data uniformly across all targets. Use the correlated commit when it is important to ensure that every row in the source affects all affected targets uniformly. For more information about automatic correlated commit, see "Committing Data from a Single Source to Multiple Targets" on page 10-7.

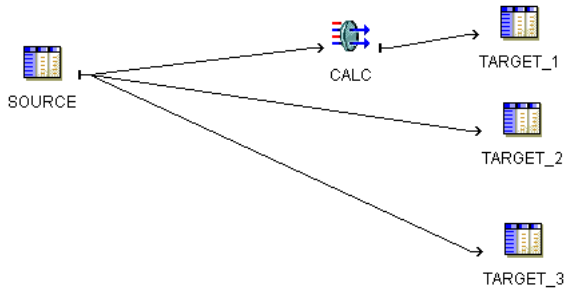
**Manual:** Select manual commit control for PL/SQL mappings when you want to interject complex business logic, perform validations, or run other mappings before committing data. For examples, see "Embedding Commit Logic into the Mapping" on page 10-9 and "Committing Data Independently of Mapping Design" on page 10-10.

### Committing Data from a Single Source to Multiple Targets

If you want to populate multiple targets based on a common source, you may also want to ensure that every row from the source affects all targets uniformly.

Figure 10–7 shows a PL/SQL mapping that illustrates this case. The target tables all depend upon the source table. If a row from SOURCE causes changes in multiple targets (for instance TARGET_1 and TARGET_2), then Warehouse Builder should commit the appropriate data to both affected targets at the same time. If this relationship is not maintained when you run the mapping again, then the data can become inaccurate and possibly unusable.

*Figure 10–7 Mapping with Multiple Targets Dependent on One Source*




If the number of rows from the source table is relatively small, maintaining the three targets may not be difficult. Manually maintaining targets dependent on a common source, however, becomes more tedious as you increase the number of rows from the source, or as you design more complex mappings with more targets and transformations.

To ensure that every row in the source properly affects every target, configure the mapping to use the correlated commit strategy.

### Using the Automatic Correlated Commit Strategy

In set-based mode, correlated commit may impact the size of your rollback segments. Space for rollback segments may be a concern when you merge data (insert/update or update/insert).

Correlated commit operates transparently with PL/SQL bulk processing code.

The correlated commit strategy is not available for mappings run in any mode that are configured for Partition Exchange Loading or that include a Queue, Match Merge, or Table Function operator.

## Automatic Commit versus Automatic Correlated Commit

The combination of the commit strategy and operating mode determines mapping behavior. Table 10–3 shows the valid combinations that you can select.

*Table 10–3 Valid Commit Strategies for Operating Modes*

| Operating Mode | Automatic Correlated Commit | Automatic Commit |
|---|---|---|
| Set-based | Valid | Valid |
| Row-based | Valid | Valid |
| Row-based (target only) | Not Applicable | Valid |

Correlated commit is not applicable for row-based (target only). By definition, this operating mode places the cursor as close to the target as possible. In most cases, this results in only one target for each select statement and negates the purpose of committing data to multiple targets. If you design a mapping with the row-based (target only) and correlated commit combination, then Warehouse Builder runs the mapping but does not perform the correlated commit.

To understand the effects each operating mode and commit strategy combination has on a mapping, consider the mapping from Figure 10–7 on page 10-8. Assume the data from source table equates to 1,000 new rows. When the mapping runs successfully, Warehouse Builder loads 1,000 rows to each of the targets. If the mapping fails to load

the 100th new row to Target_2, then you can expect the following results, ignoring the influence from other configuration settings such as Commit Frequency and Number of Maximum Errors:

- **Set-based/ Correlated Commit:** A single error anywhere in the mapping triggers the rollback of all data. When Warehouse Builder encounters the error inserting into Target_2, it reports an error for the table and does not load the row. Warehouse Builder rolls back all the rows inserted into Target_1 and does not attempt to load rows to Target_3. No rows are added to any of the target tables. For error details, Warehouse Builder reports only that it encountered an error loading to Target_2.
- **Row-based/ Correlated Commit:** Beginning with the first row, Warehouse Builder evaluates each row separately and loads it to all three targets. Loading continues in this way until Warehouse Builder encounters an error loading row 100 to Target_2. Warehouse Builder reports the error and does not load the row. It rolls back the row 100 previously inserted into Target_1 and does not attempt to load row 100 to Target_3. Next, Warehouse Builder continues loading the remaining rows, resuming with loading row 101 to Target_1. Assuming Warehouse Builder encounters no other errors, the mapping completes with 999 new rows inserted into each target. The source rows are accurately represented in the targets.
- **Set-based/ Automatic Commit:** When Warehouse Builder encounters the error inserting into Target_2, it does not load any rows and reports an error for the table. It does, however, continue to insert rows into Target_3 and does not roll back the rows from Target_1. Assuming Warehouse Builder encounters no other errors, the mapping completes with one error message for Target_2, no rows inserted into Target_2, and 1,000 rows inserted into Target_1 and Target_3. The source rows are not accurately represented in the targets.
- **Row-based/Automatic Commit:** Beginning with the first row, Warehouse Builder evaluates each row separately for loading into the targets. Loading continues in this way until Warehouse Builder encounters an error loading row 100 to Target_2 and reports the error. Warehouse Builder does not roll back row 100 from Target_1, does insert it into Target_3, and continues to load the remaining rows. Assuming Warehouse Builder encounters no other errors, the mapping completes with 999 rows inserted into Target_2 and 1,000 rows inserted into each of the other targets. The source rows are not accurately represented in the targets.

### Embedding Commit Logic into the Mapping

For PL/SQL mappings only, you can embed commit logic into the mapping design by adding a Pre-Mapping Process or Post-Mapping Process operator with SQL statements to commit and rollback data. When you run the mapping, Warehouse Builder commits or rollback data based solely on the SQL statements you provide in the Pre-Mapping Process or Post-Mapping Process operator.

Use these instructions to implement a business rule that is tedious or impossible to design given existing Warehouse Builder mapping operators. For example, you may want to verify the existence of a single row in a target. Write the required logic in SQL and introduce that logic to the mapping through a pre or post mapping operator.

**To include commit logic in the mapping design:**

1. Design the mapping to include a Pre-Mapping Process or Post-Mapping Process operator. Use one of these operators to introduce commit and rollback SQL statements.
2. Configure the mapping with **Commit Control** set to Manual.

In the Projects Navigator, right-click the mapping and select **Configure.** Under **Code Generation Options,** select **Commit Control** to Manual.

To understand the implications of selecting to commit data manually, see "About Manual Commit Control" on page 10-10.

3. Deploy the mapping.
4. Run the mapping.

   Warehouse Builder executes the mapping but does not commit data until processing the commit logic you wrote in the Pre-Mapping Process or Post-Mapping Process operator.

## Committing Data Independently of Mapping Design

You may want to commit data independently of the mapping design for any of the following reasons:

- **Running Multiple Mappings Before Committing Data**: You may want to run multiple mappings without committing data until successfully running and validating all mappings. This can be the case when you have separate mappings for loading dimensions and cubes.
- **Maintaining targets more efficiently:** If incorrect data is loaded and committed to a very large target, it can be difficult and time consuming to repair the damage. To avoid this, first check the data and then decide whether to issue a commit or rollback command.

The first step to achieve these goals is to configure the mapping with commit control set to Manual.

### About Manual Commit Control

Manual commit control enables you to specify when Warehouse Builder commits data regardless of the mapping design. Manual commit control does not affect auditing statistics. This means that you can view the number of rows inserted and other auditing information before issuing the commit or rollback command.

When using manual commit, be aware that this option may have performance implications. Mappings that you intend to run in parallel maybe be executed serially if the design requires a target to be read after being loaded. This occurs when moving data from a remote source or loading to two targets bound to the same table.

When you enable manual commit control, Warehouse Builder runs the mapping with PEL switched off.

## Running Multiple Mappings Before Committing Data

This section provides two sets of instructions for committing data independent of the mapping design. The first set describes how to run mappings and then commit data in a SQL*Plus session. Use these instructions to test and debug your strategy of running multiple mappings and then committing the data. Then, use the second set of instructions to automate the strategy.

Both sets of instructions rely upon the use of the main procedure generated for each PL/SQL mapping.

### Main Procedure

The main procedure is a procedure that exposes the logic for starting mappings in Warehouse Builder. You can employ this procedure in PL/SQL scripts or use it in interactive SQL*Plus sessions.

When you use the main procedure, you must specify one required parameter, *p_status*. And you can optionally specify other parameters relevant to the execution of the mapping as described in Table 10–4. Warehouse Builder uses the default setting for any optional parameters that you do not specify.

***Table 10–4 Parameter for the Main Procedure***

| Parameter Name | Description |
|---|---|
| p_status | Use this required parameter to write the status of the mapping upon completion. It operates in conjunction with the predefined variable called *status*.<br><br>The status variable is defined such that OK indicates the mapping completed without errors. OK_WITH_WARNINGS indicates the mapping completed with user errors. FAILURE indicates the mapping encountered a fatal error. |
| p_operating_mode | Use this optional parameter to pass in the default operating mode such as SET_BASED. |
| p_bulk_size | Use this optional parameter to pass in the bulk size. |
| p_audit_level | Use this optional parameter to pass in the default audit level such as COMPLETE. |
| p_max_no_of_errors | Use this optional parameter to pass in the permitted maximum number of errors. |
| p_commit_frequency | Use this optional parameter to pass in the commit frequency. |

## Committing Data at Runtime

For PL/SQL mappings alone, you can run mappings and issue commit and rollback commands from the SQL*Plus session. Based on your knowledge of SQL*Plus and the Main Procedure, you can manually run and validate multiple mappings before committing data.

**To commit data manually at runtime:**

1. Design the PL/SQL mappings. For instance, create one mapping to load dimensions and a separate mapping to load cubes.

   These instructions are not valid for SQL*Loader and ABAP mappings.

2. Configure both mappings with the Commit Control parameter set to Manual.

   In the Projects Navigator, right-click the mapping and select **Configure.** Under the Code Generation Options**,** set the Commit Control parameter to Manual.

3. Generate each mapping.

4. From a SQL*Plus session, issue the following command to execute the first mapping called *map1* in this example:

   ```
   var status VARCHAR2(30);
   execute map1.main(:status);
   ```

   The first line declares the predefined status variable described in Table 10–4. In the second line, *p_status* is set to the status variable. When *map1* completes, SQL*Plus displays the mapping status such as *OK.*

5. Execute the second mapping, in this example, the cubes mapping called *map2.*

   You can run the second in the same way you ran the previous map. Or, you can supply additional parameters listed in Table 10–4 to dictate how to run the *map2* in this example:

```
map2.main (p_status => :status,              \
           p_operating_mode => 'SET_BASED',   \
           p_audit_level => 'COMPLETE');
```

6. Verify the results from the execution of the two mappings and send either the commit or rollback command.

7. Automate your commit strategy as described in "Committing Mappings through the Process Flow Editor" on page 10-12.

### Committing Mappings through the Process Flow Editor

For PL/SQL mappings alone, you can commit or rollback mappings together. Based on your knowledge of the SQL*PLUS activity, the Main Procedure, and writing PL/SQL scripts, you can use process flows to automate logic that commits data after all mappings complete successfully or rollback the data if any mapping fails.

**To commit multiple mappings through a process flow:**

1. Design the PL/SQL mappings.

   These instructions are not valid for SQL*Loader and ABAP mappings.

2. Ensure each mapping is deployed to the same schema.

   All mappings must have their locations pointing to the same schema. You can achieve this by designing the mappings under the same target module. Or, for multiple target modules, ensure that the locations point to the same schema.

3. Configure each mapping with the Commit Control parameter set to Manual.

   In the Projects Navigator, right-click the mapping and select **Configure.** Under Code Generation Options**,** set the Commit Control parameter to Manual.

4. Design a process flow using a SQL*PLUS activity instead of multiple mapping activities.

   In typical process flows, you add a Mapping activity for each mapping and the process flow executes an implicit commit after each Mapping activity. However, in this design, do not add mapping activities. Instead, add a single SQL*PLUS activity.

5. Write a PL/SQL script that uses the main procedure to execute each mapping. The following script demonstrates how to run the next mapping only if the initial mapping succeeds.

```
declare
    status VARCHAR2(30);
begin
    map1.main(status);
    if status != 'OK' then
       rollback;
    else
       map2.main(status);
       if status != 'OK' then
           rollback;
       else
           commit;
```

```
        end if;
    end if;
end;
```

6. Paste your PL/SQL script into the SQL*PLUS activity.

   In the editor explorer, select **SCRIPT** under the SQL*PLUS activity and then double-click **Value** in the object inspector.

7. Optionally apply a schedule to the process flow as described in "Defining Schedules" on page 11-2.

8. Deploy the mappings, process flow, and schedule if you defined one.

### Ensuring Referential Integrity in PL/SQL Mappings

When you design mappings with multiple targets, you may want to ensure that Warehouse Builder loads the targets in a specific order. This is the case when a column in one target derives its data from another target.

To ensure referential integrity in PL/SQL mappings:

1. Design a PL/SQL mapping with multiple targets.

2. (Optional) Define a parent/child relationship between two of the targets by specifying a foreign key.

   A foreign key in the child table must refer to a primary key in the parent table. If the parent does not have a column defined as a primary key, then you must add a column and set it as the primary key. For an example of how to do this, see "Using Conventional Loading to Ensure Referential Integrity in SQL*Loader Mappings" on page 10-13.

3. In the mapping properties, view the Target Load Order property by clicking the Ellipsis button to the right of this property.

   If you defined a foreign key relationship in the previous step, Warehouse Builder calculates a default loading order that loads parent targets before children. If you did not define a foreign key, then use the Target Load Order dialog box to define the loading order.

   For more information, see "Specifying the Order in Which Target Objects in a Mapping Are Loaded" on page 5-26.

4. Ensure that the Use Target Load Ordering configuration parameter is set to its default value of true.

## Best Practices for Designing SQL*Loader Mappings

This section includes the following topics:

- Using Conventional Loading to Ensure Referential Integrity in SQL*Loader Mappings
- Using Direct Path Loading to Ensure Referential Integrity in SQL*Loader Mappings

### Using Conventional Loading to Ensure Referential Integrity in SQL*Loader Mappings

If you are extracting data from a multiple-record-type file with a master-detail structure and mapping to tables, add a Sequence operator to the mapping to retain the relationship between the master and detail records through a surrogate primary key or

foreign key relationship. A master-detail file structure is one where a master record is followed by its detail records. In Example 10–1, records beginning with "E" are master records with Employee information and records beginning with "P" are detail records with Payroll information for the corresponding employee.

***Example 10–1 A Multiple-Record-Type Flat File with a Master-Detail Structure***

```
E 003715 4 153 09061987 014000000 "IRENE HIRSH" 1 08500
P 01152000 01162000 00101 000500000 000700000
P 02152000 02162000 00102 000300000 000800000
E 003941 2 165 03111959 016700000 "ANNE FAHEY" 1 09900
P 03152000 03162000 00107 000300000 001000000
E 001939 2 265 09281988 021300000 "EMILY WELLMET" 1 07700
P 01152000 01162000 00108 000300000 001000000
P 02152000 02162000 00109 000300000 001000000
```

In Example 10–1, the relationship between the master and detail records is inherent only in the physical record order: payroll records correspond to the employee record they follow. However, if this is the only means of relating detail records to their masters, this relationship is lost when Warehouse Builder loads each record into its target table.

### Maintaining Relationships Between Master and Detail Records

You can maintain the relationship between master and detail records if both types of records share a common field. If Example 10–1 contains a field Employee ID in both Employee and Payroll records, then you can use it as the primary key for the Employee table and as the foreign key in the Payroll table, thus associating Payroll records to the correct Employee record.

However, if your file does not have a common field that can be used to join master and detail records, you must add a sequence column to both the master and detail targets (see Table 10–5 and Table 10–6) to maintain the relationship between the master and detail records. Use the Sequence operator to generate this additional value.

Table 10–5 represents the target table containing the master records from the file in Example 10–1 on page 10-14. The target table for the master records in this case contains employee information. Columns E1-E10 contain data extracted from the flat file. Column E11 is the additional column added to store the master sequence number. Notice that the number increments by one for each employee.

***Table 10–5 Target Table Containing Master Records***

| E1 | E2 | E3 | E4 | E5 | E6 | E7 | E8 | E9 | E10 | E11 |
|---|---|---|---|---|---|---|---|---|---|---|
| E | 003715 | 4 | 153 | 09061987 | 014000000 | "IRENE | HIRSH" | 1 | 08500 | 1 |
| E | 003941 | 2 | 165 | 03111959 | 016700000 | "ANNE | FAHEY" | 1 | 09900 | 2 |
| E | 001939 | 2 | 265 | 09281988 | 021300000 | "EMILY | WELSH" | 1 | 07700 | 3 |

Table 10–6 represents the target table containing the detail records from the file in Example 10–1 on page 10-14. The target table for the detail records in this case contains payroll information, with one or more payroll records for each employee. Columns P1-P6 contain data extracted from the flat file. Column P7 is the additional column added to store the detail sequence number. Notice that the number for each payroll record matches the corresponding employee record in Table 10–5.

*Table 10–6 Target Table Containing Detail Records*

| P1 | P2 | P3 | P4 | P5 | P6 | P7 |
|---|---|---|---|---|---|---|
| P | 01152000 | 01162000 | 00101 | 000500000 | 000700000 | 1 |
| P | 02152000 | 02162000 | 00102 | 000300000 | 000800000 | 1 |
| P | 03152000 | 03162000 | 00107 | 000300000 | 001000000 | 2 |
| P | 01152000 | 01162000 | 00108 | 000300000 | 001000000 | 3 |
| P | 02152000 | 02162000 | 00109 | 000300000 | 001000000 | 3 |

## Extracting and Loading Master-Detail Records

This section contains instructions on creating a mapping that extracts records from a master-detail flat file and loads those records into two different tables. One target table stores master records and the other target table stores detail records from the flat file. The Mapping Sequence is used to maintain the master-detail relationship between the two tables.

> **Note:** These instructions are for conventional path loading. For instructions on using direct path loading for master-detail records, see "Using Direct Path Loading to Ensure Referential Integrity in SQL*Loader Mappings" on page 10-18.

This procedure outlines general steps for building such a mapping. Additional detailed instructions are available at:

- "Flat File Operator" on page 25-32
- "Using the Add Operator Dialog Box to Add Operators" on page 5-15
- "Sequence Operator" on page 25-25
- "Configuring Mappings Reference" on page 24-1
- *Oracle Warehouse Builder Sources and Targets Guide*

**To extract from a master-detail flat file and maintain master-detail relationships, use the following steps:**

1. Import and sample the flat file source that consists of master and detail records.

   When naming the record types as you sample the file, assign descriptive names to the master and detail records. This makes it easier to identify those records in the future.

   In this example, for multi-record-type flat files, the Flat File Sample Wizard contains department and employee information. The master record type (for employee records) is called EmployeeMaster, while the detail record type (for payroll information) is called PayrollDetail.

2. Drop a Flat File operator onto the Mapping Editor canvas and specify the master-detail file from which you want to extract data.

3. Drop a Sequence operator onto the mapping canvas.

4. Drop a Table operator for the master records onto the mapping canvas.

   You can either select an existing workspace table that you created earlier or create a new unbound Table operator with no attributes. You can then map or copy all required fields from the master record of the Flat File operator to the master Table

operator (creating columns) and perform an outbound reconciliation to define the table later.

The table must contain all the columns required for the master fields you want to load plus an additional numeric column for loading sequence values.

5. Drop a Table operator for the detail records onto the mapping canvas.

   You can either select an existing workspace table that you created earlier or create a new unbound Table operator with no attributes. You can then map or copy all required fields from the master record of the Flat File operator to the master Table operator (creating columns) and perform an outbound synchronize to define the table later.

   The table must contain all the columns required for the detail fields you want to load plus an additional numeric column for loading sequence values.

6. Map all of the necessary flat file master fields to the master table and detail fields to the detail table.

   Figure 10–8 displays the mapping of the fields.

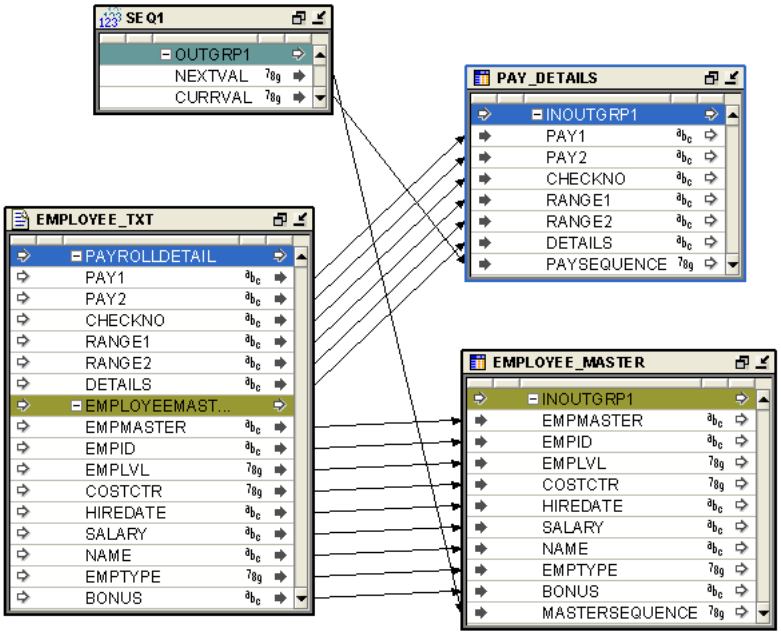
7. Map the Sequence `NEXTVAL` attribute to the additional sequence column in the master table.

   Figure 10–8 displays the mapping from the `NEXTVAL` attribute of the Sequence operator to the master table.

8. Map the Sequence `CURRVAL` attribute to the additional sequence column in the detail table.

   Figure 10–8 shows a completed mapping with the flat file master fields mapped to the master target table, the detail fields mapped to the detail target table, and the `NEXTVAL` and `CURRVAL` attributes from the Mapping Sequence mapped to the master and detail target tables, respectively.

***Figure 10–8 Completed Mapping from Master-Detail Flat File to Two Target Tables***



9. Configure the mapping that loads the source data into the target tables with the following parameters:

   **Direct Mode:** Not selected

   **Errors Allowed:** 0

   **Row:** 1

   **Trailing Nullcols:** True (for all tables)

### Error Handling Suggestions

This section contains error handling recommendations for files with varying numbers of errors.

**If your data file almost never contains errors:**

1. Create a mapping with a Sequence Operator.
2. Configure a mapping with the following parameters:

   Direct Mode= Not selected

   ROW=1

   ERROR ALLOWED = 0
3. Generate the code and run an SQL*Loader script.

   If the data file has errors, then the loading stops when the first error occurs.
4. Fix the data file and run the control file again with the following configuration values:

   CONTINUE_LOAD=TRUE

   SKIP=*number of records already loaded*

**If your data file is likely to contain a moderate number of errors:**

1. Create a primary key (PK) for the master record based on the `seq_nextval` column.
2. Create a foreign key (FK) for the detail record based on the `seq_currval` column which references the master table PK.

   In this case, master records with errors will be rejected with all their detail records. You can recover these records by following these steps.
3. Delete all failed detail records that have no master records.
4. Fix the errors in the bad file and reload only those records.
5. If there are very few errors, you may choose to load the remaining records and manually update the table with correct sequence numbers.
6. In the log file, you can identify records that failed with errors because those errors violate the integrity constraint. The following is an example of a log file record with errors:

```
Record 9: Rejected - Error on table "MASTER_T", column "C3".
ORA-01722: invalid number
Record 10: Rejected - Error on table "DETAIL1_T".
ORA-02291: integrity constraint (SCOTT.FK_SEQ) violated - parent key not found
Record 11: Rejected - Error on table "DETAIL1_T".
ORA-02291: integrity constraint (SCOTT.FK_SEQ) violated - parent key not found
Record 21: Rejected - Error on table "DETAIL2_T".
```

```
ORA-02291: invalid number
```

**If your data file always contains many errors:**

1. Load all records without using the Sequence operator.

   Load the records into independent tables. You can load the data in Direct Mode, with the following parameters that increase loading speed:

   ROW>1

   ERRORS ALLOWED=MAX

2. Correct all rejected records.
3. Reload the file again with a Sequence Operator.

### Subsequent Operations

After the initial loading of the master and detail tables, you can use the loaded sequence values to further transform, update, or merge master table data with detail table data. For example, if your master records have a column that acts as a unique identifier, such as an Employee ID, and you want to use it as the key to join master and detail rows (instead of the sequence field you added for that purpose), you can update the detail tables to use this unique column. You can then drop the sequence column you created for the initial load. Operators such as the Aggregator, Filter, or Match Merge operator can help you with these subsequent transformations.

## Using Direct Path Loading to Ensure Referential Integrity in SQL*Loader Mappings

If you are using a master-detail flat file where the master record has a unique field (or if the concatenation of several fields can result in a unique identifier), you can use Direct Path Load as an option for faster loading.

For direct path loading, the record number (`RECNUM`) of each record is stored in the master and detail tables. A post-load procedure uses the `RECNUM` to update each detail row with the unique identifier of the corresponding master row.

This procedure outlines general steps for building such a mapping. Additional detailed instructions are available:

- For additional information about importing flat file sources, see *Oracle Warehouse Builder Sources and Targets Guide*.
- For additional information about using flat files as a source, "Flat File Operator" on page 25-32.
- For additional information about using Table operators, see "Using the Add Operator Dialog Box to Add Operators" on page 5-15.
- For additional information about using the Data Generator operator, see "Data Generator Operator" on page 25-12.
- For additional information about using the Constant operator, see "Constant Operator" on page 25-9.
- For additional information about configuring mappings, see "Configuring Mappings Reference" on page 24-1.

**To extract from a master-detail flat file using direct path load to maintain master-detail relationships:**

1. Import and sample a flat file source that consists of master and detail records.

When naming the record types as you sample the file, assign descriptive names to the master and detail records. This will help identify those records in the future.

2. Create a mapping that you will use to load data from the flat file source.
3. Drop a Flat File operator onto the mapping canvas and specify the master-detail file from which you want to extract data.
4. Drop a Data Generator and a Constant operator onto the mapping canvas.
5. Drop a Table operator for the master records onto the mapping canvas.

   You can either select an existing workspace table that you created earlier, or create a new unbound Table operator with no attributes and perform an outbound synchronize to define the table later.

   The table must contain all the columns required for the master fields you plan to load plus an additional numeric column for loading the `RECNUM` value.
6. Drop a Table operator for the detail records onto the mapping canvas.

   You can either select an existing workspace table that you created earlier, or create a new unbound Table operator with no attributes and perform an outbound synchronize to define the table later.

   The table must contain all the columns required for the detail fields you plan to load plus an additional numeric column for loading a `RECNUM` value, and a column that will be updated with the unique identifier of the corresponding master table row.
7. Map all of the necessary flat file master fields to the master table and detail fields to the detail table.

   Figure 10–9 displays this mapping of master and detail fields.
8. Map the Data Generator operator's `RECNUM` attribute to the `RECNUM` columns in the master and detail tables.

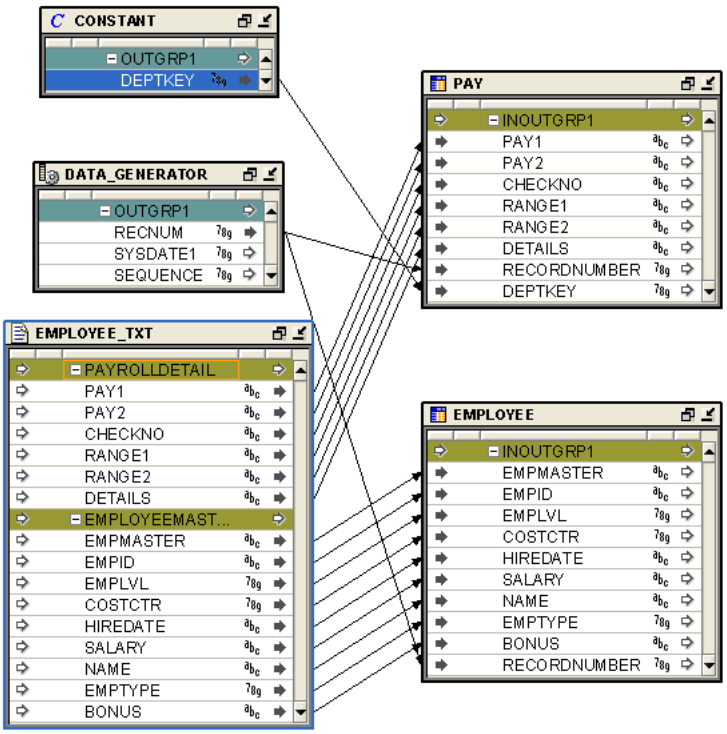
   Figure 10–9 displays the mapping in which the `RECNUM` attribute of the Data Generator operator is mapped to the `RECORDNUMBER` table attribute.
9. Add a constant attribute in the Constant operator.

   If the master row unique identifier column is of `CHAR` data type, in the Property Inspector of the constant attribute, set the Data type property to `CHAR` and the Expression property to asterisk (`*`).

   If the master row unique identifier column is a number, in the Property Inspector of the constant attribute, set the Data type property to `NUMBER` and the Expression property to zero. This marks all data rows as "just loaded".
10. Map the constant attribute from the Constant operator to the detail table column that will later store the unique identifier for the corresponding master table record.

    Figure 10–9 shows a completed mapping with the flat file's master fields mapped to the master target table, the detail fields mapped to the detail target table, the `RECNUM` attributes from the Data Generator operator mapped to the master and detail target tables, respectively, and the constant attribute mapped to the detail target table.

*Figure 10–9 Completed Mapping from Master-Detail Flat File with a Direct Path Load*




11. Configure the mapping with the following parameters:

    **Direct Mode:** True

    **Errors Allowed:** 0

    **Trailing Nullcols:** True (for each table)

12. After you validate the mapping and generate the SQL*Loader script, create a post-update PL/SQL procedure and add it to the Warehouse Builder library.

13. Run the SQL*Loader script.

14. Execute an UPDATE SQL statement by running a PL/SQL post-update procedure or manually executing a script.

The following is an example of the generated SQL*Loader control file script:

```
OPTIONS ( DIRECT=TRUE,PARALLEL=FALSE, ERRORS=0, BINDSIZE=50000, ROWS=200,
READSIZE=65536)
LOAD DATA
CHARACTERSET WE8MSWIN1252
  INFILE 'g:\FFAS\DMR2.dat'
  READBUFFERS 4
  INTO TABLE "MATER_TABLE"
  APPEND
  REENABLE DISABLED_CONSTRAINTS
       WHEN
      "REC_TYPE"='P'
  FIELDS
    TERMINATED BY ','
    OPTIONALLY ENCLOSED BY '"'
    TRAILING NULLCOLS

  (
```

```
  "REC_TYPE" POSITION (1) CHAR ,
  "EMP_ID" CHAR ,
  "ENAME" CHAR ,
  "REC_NUM" RECNUM
  )

INTO TABLE "DETAIL_TABLE"
  APPEND
  REENABLE DISABLED_CONSTRAINTS
       WHEN
      "REC_TYPE"='E'
  FIELDS
    TERMINATED BY ','
    OPTIONALLY ENCLOSED BY '"'
    TRAILING NULLCOLS
     (
  "REC_TYPE" POSITION (1) CHAR ,
  "C1" CHAR ,
  "C2" CHAR ,
  "C3" CHAR ,
  "EMP_ID" CONSTANT '*',
 "REC_NUM" RECNUM
```

The following is an example of the post-update PL/SQL procedure:

```
  create or replace procedure wb_md_post_update(
     master_table varchar2
    ,master_recnum_column varchar2
    ,master_unique_column varchar2
    ,detail_table varchar2
    ,detail_recnum_column varchar2
    ,detail_masterunique_column varchar2
    ,detail_just_load_condition varchar2)
  IS
     v_SqlStmt VARCHAR2(1000);
  BEGIN
     v_SqlStmt := 'UPDATE '||detail_table||' l '||
                  ' SET l.'||detail_masterunique_column||' = (select i.'||master_
unique_column||
                  ' from '||master_table||' i '||
                  ' WHERE i.'||master_recnum_column||' IN '||
                  ' (select max(ii.'||master_recnum_column||') '||
                  ' from '||master_table||' ii '||
                  ' WHERE ii.'||master_recnum_column||' < l.'||detail_recnum_
column||') '||
                  ' ) '||
                  ' WHERE l.'||detail_masterunique_column||' = '||''''||detail_
just_load_condition||'''';
     dbms_output.put_line(v_sqlStmt);
     EXECUTE IMMEDIATE  v_SqlStmt;
  END;
  /
```

## Improved Performance through Partition Exchange Loading

Data partitioning can improve performance when loading or purging data in a target system. This practice is known as Partition Exchange Loading (PEL).

PEL is recommended when loading a relatively small amount of data into a target containing a much larger volume of historical data. The target can be a table, a dimension, or a cube in a data warehouse.

This section includes the following topics:

- About Partition Exchange Loading
- Configuring a Mapping for PEL
- Direct and Indirect PEL
- Using PEL Effectively
- Configuring Targets in a Mapping
- Restrictions for Using PEL in Warehouse Builder

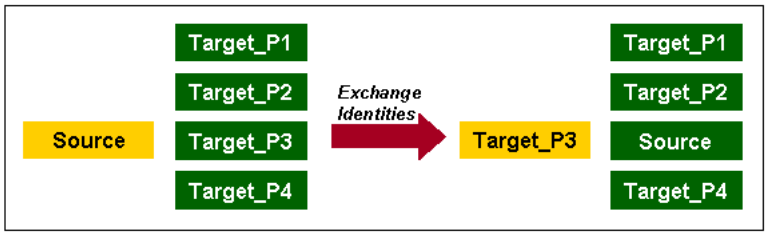
## About Partition Exchange Loading

By manipulating partitions in your target system, you can use Partition Exchange Loading (PEL) to instantly add or delete data. When a table is exchanged with an empty partition, new data is added.

You can use PEL to load new data by exchanging it into a target table as a partition. For example, a table that holds the new data assumes the identity of a partition from the target table and this partition assumes the identity of the source table. This exchange process is a DDL operation with no actual data movement.

Figure 10–10 illustrates an example of PEL. Data from a source table `Source` is inserted into a target table consisting of four partitions (`Target_P1`, `Target_P2`, `Target_P3`, and `Target_P4`). If the new data needs to be loaded into `Target_P3`, then the partition exchange operation only exchanges the names on the data objects without moving the actual data. After the exchange, the formerly labeled `Source` is renamed to `Target_P3`, and the former `Target_P3` is now labeled as `Source`. The target table still contains four partitions: `Target_P1`, `Target_P2`, `Target_P3`, and `Target_P4`. The partition exchange operation available in Oracle 9*i* completes the loading process without data movement.

***Figure 10–10 Overview of Partition Exchange Loading***




## Configuring a Mapping for PEL

**To configure a mapping for partition exchange loading, complete the following steps:**

1. In the Projects Navigator, right-click a mapping and select **Configure.**

   Warehouse Builder displays the Configuration tab for the mapping.

2. By default, PEL is disabled for all mappings. Select **PEL Enabled** to use Partition Exchange Loading.

3. Use **Data Collection Frequency** to specify the amount of new data to be collected for each run of the mapping. Set this parameter to specify if you want the data collected by Year, Quarter, Month, Day, Hour, or Minute. This determines the number of partitions.
4. Select **Direct** if you want to create a temporary table to stage the collected data before performing the partition exchange. If you do not select this parameter, then Warehouse Builder directly swaps the source table into the target table as a partition without creating a temporary table. For more information, see "Direct and Indirect PEL" on page 10-23.
5. If you select **Replace Data**, Warehouse Builder replaces the existing data in the target partition with the newly collected data. If you do not select it, then Warehouse Builder preserves the existing data in the target partition. The new data is inserted into a non-empty partition. This parameter affects the local partition and can be used to remove or swap a partition out of a target table. At the table level, you can set Truncate/Insert properties.

## Direct and Indirect PEL

When you use Warehouse Builder to load a target by exchanging partitions, you can load the target indirectly or directly.

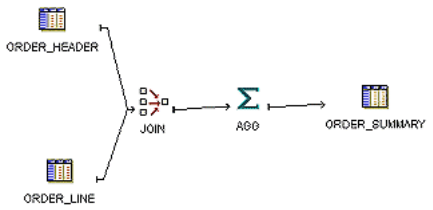
- **Indirect PEL:** By default, Warehouse Builder creates and maintains a temporary table that stages the source data before initiating the partition exchange process. For example, use Indirect PEL when the mapping includes a remote source or a join of multiple sources.
- **Direct PEL:** You design the source for the mapping to match the target structure. For example, use Direct PEL in a mapping to instantaneously publish fact tables that you loaded in a previously executed mapping.

### Using Indirect PEL

If you design a mapping using PEL and it includes remote sources or a join of multiple sources, Warehouse Builder must perform source processing and stage the data before partition exchange can proceed. Therefore, configure such mappings with Direct PEL set to False. Warehouse Builder transparently creates and maintains a temporary table that stores the results from source processing. After performing the PEL, Warehouse Builder drops the table.

Figure 10–11 shows a mapping that joins two sources and performs an aggregation. If all new data loaded into the `ORDER_SUMMARY` table is always loaded into same partition, then you can use Indirect PEL on this mapping to improve load performance. In this case, Warehouse Builder transparently creates a temporary table after the Aggregator and before `ORDER_SUMMARY`.

***Figure 10–11 Mapping with Multiple Sources***



Warehouse Builder creates the temporary table using the same structure as the target table with the same columns, indexes, and constraints. For the fastest performance, Warehouse Builder loads the temporary table using parallel direct-path loading INSERT. After the INSERT, Warehouse Builder indexes and constrains the temporary table in parallel.

### Example: Using Direct PEL to Publish Fact Tables

Use Direct PEL when the source table is local and the data is of good quality. You must design the mapping such that the source and target are in the same database and have exactly the same structure. The source and target must have the same indexes and constraints, the same number of columns, and the same column types and lengths.

For example, assume that you have the same mapping from Figure 10–11 but would like greater control on when data is loaded into the target. Depending on the amount of data, it could take hours to load and you would not know precisely when the target table would be updated.

**To instantly load data to a target using Direct PEL:**

1. Design one mapping to join source data, if necessary, transform data, ensure data validity, and load it to a staging table. Do not configure this mapping to use PEL.

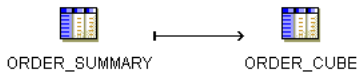
   Design the staging table to exactly match the structure of the final target that you will load in a separate mapping.

   For example, the staging table in Figure 10–11 is `ORDER_SUMMARY` and should be of the same structure as the final target, `ORDER_CUBE` in Figure 10–12.

2. Create a second mapping that loads data from the staging table to the final target. Configure this mapping to use Direct PEL.

   Figure 10–12 displays the mapping that loads data from the staging table to the final target.

***Figure 10–12 Publish_Sales_Summary Mapping***




3. Use either the Warehouse Builder Process Flow Editor or Oracle Workflow to start the second mapping after the completion of the first.

## Using PEL Effectively

You can use PEL effectively for scalable loading performance if the following conditions are true:

- **Table partitioning and tablespace:** The target table must be Range partitioned by one `DATE` column. All partitions must be created in the same tablespace. All tables are created in the same tablespace.
- **Existing historical data:** The target table must contain a huge amount of historical data. An example use for PEL is for a click stream application where the target collects data every day from an OLTP database or Web log files. New data is transformed and loaded into the target that already contains historical data.

- **New data:** All new data must to be loaded into the same partition in a target table. For example, if the target table is partitioned by day, then the daily data should be loaded into one partition.
- **Loading Frequency:** The loading frequency should be equal to or less than the data collection frequency.
- **No global indexes:** There must be no global indexes on the target table.

## Configuring Targets in a Mapping

To configure targets in a mapping for PEL:

- Step 1: Create All Partitions
- Step 2: Create All Indexes Using the LOCAL Option
- Step 3: Primary/Unique Keys Use "USING INDEX" Option

### Step 1: Create All Partitions

Warehouse Builder does not automatically create partitions during runtime. Before you can use PEL, you must create all partitions as described in "Defining Partitions" on page 2-25.

For example, if you select Month as the frequency of new data collection, you must create all the required partitions for each month of new data. Use the object editors to create partitions for a table, dimension, or cube.

To use PEL, all partition names must follow a naming convention. For example, for a partition that will hold data for May 2002, the partition name must be in the format Y2002_Q2_M05.

For PEL to recognize a partition, its name must fit one of the following formats:

**Y**dddd

**Y**dddd_**Q**d

**Y**dddd_**Q**d_**M**dd

**Y**dddd_**Q**d_**M**dd_**D**dd

**Y**dddd_**Q**d_**M**dd_**D**dd_**H**dd

**Y**dddd_**Q**d_**M**dd_**D**dd_**H**dd_**M**dd

Where d represents a decimal digit. All the letters must be in upper case. Lower case is not recognized.

If you correctly name each partition, Warehouse Builder automatically computes the Value Less Than property for each partition. Otherwise, you must manually configure Value Less Than for each partition for Warehouse Builder to generate a DDL statement. The following is an example of a DDL statement generated by Warehouse Builder:

```
. . .
PARTITION A_PARTITION_NAME
      VALUES LESS THAN (TO_DATE('01-06-2002','DD-MM-YYYY')),
. . .
```

### Step 2: Create All Indexes Using the LOCAL Option

Add an index (ORDER_SUMMARY_PK_IDX) to the ORDER_SUMMARY table. This index has two columns, ORDER_DATE and ITEM_ID. Set the following on the Indexes tab of the Table Editor:

- Select UNIQUE in the Type column.
- Select LOCAL in the Scope column.

Now Warehouse Builder can generate a DDL statement for a unique local index on table ORDER_SUMMARY.

Using local indexes provides the most important PEL performance benefit. Local indexes require all indexes to be partitioned in the same way as the table. When the temporary table is swapped into the target table using PEL, so are the identities of the index segments.

If an index is created as a local index, the Oracle server requires that the partition key column must be the leading column of the index. In the preceding example, the partition key is ORDER_DATE and it is the leading column in the index ORDER_SUMMARY_PK_IDX.

### Step 3: Primary/Unique Keys Use "USING INDEX" Option

In this step you must specify that all primary key and unique key constraints are created with the USING INDEX option. In the Projects Navigator, right-click the table and select **Configure**. The Configuration tab for the table is displayed. Select the primary or unique key in the left panel and select **Using Index** in the right panel.

With the USING INDEX option, a constraint will not trigger automatic index creation when it is added to the table. The server will search existing indexes for an index with same column list as that of the constraint. Thus, each primary or unique key constraint must be backed by a user-defined unique local index. The index required by the constraint ORDER_SUMMARY_PK is ORDER_SUMMARY_PK_IDX which was created in "Step 2: Create All Indexes Using the LOCAL Option" on page 10-26.

## Restrictions for Using PEL in Warehouse Builder

These are the restrictions for using PEL in Warehouse Builder:

- **Only One Date Partition Key:** Only one partition key column of DATE data type is allowed. Numeric partition keys are not supported in Warehouse Builder.
- **Only Natural Calendar System:** The current PEL method supports only the natural calendar system adopted worldwide. Specific business calendar systems with user-defined fiscal and quarter endings are currently not supported.
- **All Data Partitions Must Be In the Same Tablespace:** All partitions of a target (table, dimension, or cube) must be created in the same tablespace.
- **All Index Partitions Must Be In the Same Tablespace:** All indexes of a target (table, dimension, or cube) must be created in the same tablespace. However, the index tablespace can be different from the data tablespace.

# High Performance Data Extraction from Remote Sources

Although you can design mappings to access remote sources through database links, performance is likely to be slow when you move large volumes of data. For mappings that move large volumes of data between sources and targets of the same Oracle

Database version, you have an option for dramatically improving performance through the use of transportable modules.

> **See Also:** "Moving Large Volumes of Data Using Transportable Modules" on page 10-1 for instructions on using transportable modules

You can also efficiently extract data from remote Oracle or other heterogeneous database sources using Code Template (CT) mappings. Warehouse Builder provides a set of predefined Code Templates that you can use in CT mappings for different data movement options.

> **See Also:** ■
>
> ■ "Creating Code Template (CT) Mappings" on page 7-12 for information about defining and using CT mappings
>
> ■ "About Prebuilt Code Templates Shipped with Warehouse Builder" on page 7-13 for a description of the Code Templates shipped with Warehouse Builder

# 11

# Scheduling ETL Jobs

This chapter contains the following topics:


- Overview of Schedules
- Defining Schedules
- Applying Schedules to ETL Objects
- Scheduling ETL Jobs in Oracle Enterprise Manager


## Overview of Schedules

Use schedules to plan when and how often to execute operations that you designed within Oracle Warehouse Builder. You can apply schedules to mappings and process flows that you want to execute in Oracle Database 10*g* or later.

You can define schedules to execute once or to execute repeatedly based on an interval you define in the user interface. For each schedule you define, Warehouse Builder generates codes that follows the iCal calendaring standards, which can be deployed to a scheduler such as Oracle Scheduler or Oracle Concurrent Manager.

When you are in the development phase of using Warehouse Builder, you may not want to schedule mappings and process flows but rather start and stop them immediately from a Control Center as described in "Deploying Objects" on page 12-6.

### Overview of Defining Schedules

Schedules are defined in the context of projects and contained in Calendar modules under the Schedules node on the Projects Navigator. Calendar modules are used to group a set of related schedules that are deployed to the same location.

A schedule definition contains details about when and how often operations you define using Warehouse Builder are executed. When you define a schedule, specify the following details:

- Start date and time

  The date and time when the schedule begins to execute

- Schedule frequency

  The frequency at which the schedule is run (for example, daily, weekly, and so on)

- Repeat interval

  The time intervals at which the schedule is repeated when it is active. The repeat interval determines how often the schedule repeats. For example, you define a

schedule whose frequency is weekly. For the repeat interval of this schedule, you specify that this schedule should repeat every two weeks, on Tuesday.

- End date and time

  The date and time at which the schedule stops execution

### Overview of Using Schedules

Once you define a schedule, you can associate it with ETL objects so that the objects are executed according to the details defined in the schedule. You can associate schedules with mappings, Code Template (CT) mappings, process flows, and data auditors.

For example, you create a schedule that starts on 12-mar-08 and that ends on 11-mar-09. You define the schedule frequency as weekly and the repeat interval as every week on Friday. You then associate this schedule with a mapping. Your mapping will be executed according to the details specified in the schedule. The mapping will execute every Friday between 12-mar-08 and 11-mar-09.

### Overview of Deploying Schedules

The location to which you deploy schedules depends on the ETL objects with which the schedule is associated. Scheduled jobs must be deployed to an Oracle Database location. Process flow packages must be deployed either to an Oracle Workflow location or to a Concurrent Manager location.

Scheduled jobs may reference an executable object, such as a process flow or a mapping. If a job references a process flow, then you must deploy the process flow to Oracle Workflow and deploy the scheduled job to either a database location or a Concurrent Manager location.

## Defining Schedules

Use the following steps to define a schedule.

1. Expand the project node under which you want to create your schedule.
2. If you have not already done so, create a Calendar module to contain your schedule.

   To create a Calendar module, right-click the Schedules node and select **New Calendar module**. The Create Module Wizard is displayed. Use this wizard to create the Calendar module. Ensure that the location associated with this module is the location to which you want your schedules to be deployed.
3. Right-click the Calendar module in which you want to create your schedule, and select **New Calendar**.

   The Create Schedule Wizard is displayed.
4. On the Welcome page, click **Next**.
5. On the Name and Description page, provide a name and an optional description for the schedule and click **Next**.

   Ensure that the schedule name is unique within the Calendar module in which it is displayed. See "Naming Conventions for Data Objects" on page 2-8 for more information.
6. On the Choose Start and End Time page, select the time zone, start time, and end time for the schedule. Click **Next**.

For more information about setting these values, see "Start and End Dates and Times" on page 11-4.

> **Note:** The Create Schedule Wizard does not contain all the available scheduling capabilities. To access the full range of scheduling capabilities, after you define a schedule using the wizard, use the Schedule Editor to specify additional scheduling capabilities.

7. On the Choose the Frequency and Repeat Interval page, specify the frequency and repeat interval for the schedule. Click **Next**.

   For more information about setting the frequency and repeat interval, see "Defining Schedules To Repeat" on page 11-4.

   > **See Also:** "Example Schedules" on page 11-8 for examples of schedules

8. On the Summary page, review the options that you selected for the schedule. Click **Finish** to define the schedule.

   The schedule is created and added to the Projects Navigator under the Calendar module.

Defining a schedule creates the metadata for the schedule in the workspace. Before you can use this schedule to execute ETL jobs at the times defined in the schedule, you must deploy the schedule. When you deploy a schedule to an Oracle Database location, the schedule is created in the database scheduler.

> **See Also:**
>
> - About Deploying Schedules on page 12-3
> - Deploying Objects on page 12-6

## Editing Schedules

Use the Schedule Editor to edit schedules.

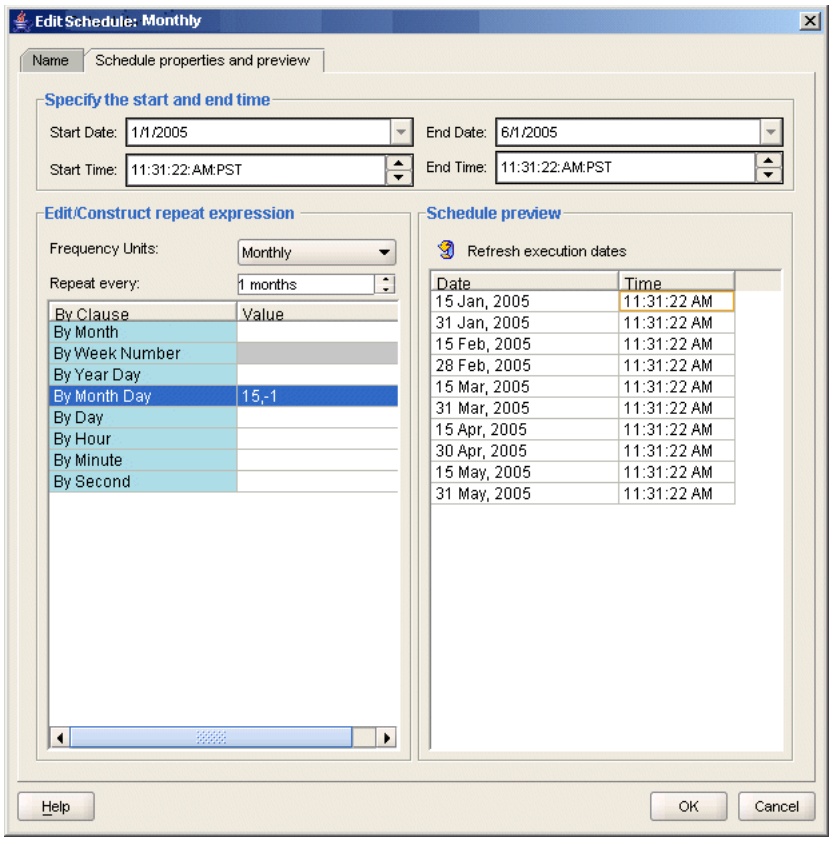
Figure 11–1 shows the Schedule Editor with the Start and End Dates and Times at the top of the editor.

The repeat expression appears in the lower-left panel of the editor. Use the repeat expression to specify the Frequency Unit, Repeat Every, and one or more By Clauses.

The schedule preview appears in the lower-right panel. The preview refreshes each time you press the Enter key or navigate to a new cell on the Schedule Editor. If you specify an invalid schedule, then the preview displays an error message.

For examples of schedules that you can define, see Example Schedules on page 11-8.

***Figure 11–1 The Schedule Editor***




### Start and End Dates and Times

The start and end dates and times define the duration for which the schedule is valid.

Begin by specifying the time zone. You can accept the default start date or specify a time in the future. Be sure to change the default end date, which is the same as the default start date.

When working in the wizard, click **Next** to view the next page.

When working in the Schedule Editor, the start date and time become the defaults for the By Clauses in the Repeat Expression. The execution time in Schedule Preview corresponds to the Start Time.

### Defining Schedules To Repeat

The repeat expression determines how often the schedule is executed. Define the repeat expression by specifying the Frequency Unit, the Repeat Every value, and one or more By Clauses values.

When you are working in the wizard, By Clauses are not available. After you complete the wizard, you can open the schedule and set the By Clauses using the Schedule Editor.

#### Frequency Unit

The Frequency Unit determines the type of recurrence. The possible values are YEARLY, MONTHLY, WEEKLY, DAILY, HOURLY, MINUTELY, and SECONDLY.

Also, you can define schedules to run One Time.

### Repeat Every

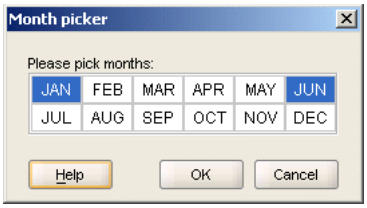
The Repeat Every value specifies how often the recurrence repeats. The default value is 1 and the maximum value is 999. If you select YEARLY for the Frequency Unit and leave the Repeat Every value at 1, then the schedule is evaluated for every year included in the date range that you specify in Start and End Dates and Times. For the same schedule, if you change Repeat Every to 2, the schedule is evaluated only every other year within the specified date range.

### By Clauses

By Clauses enable you to define repeat expressions for complex schedules, such as a schedule to run the first Friday of any month that contains 5 weeks. For each clause, you can either enter values or click the Ellipsis button to view a selector dialog box. If your goal is to know how to quickly type in values, then first use the selector dialog box to learn what values are valid, and then see "Example Schedules" on page 11-8.

Figure 11–2 displays the selector dialog box to pick months in a year.

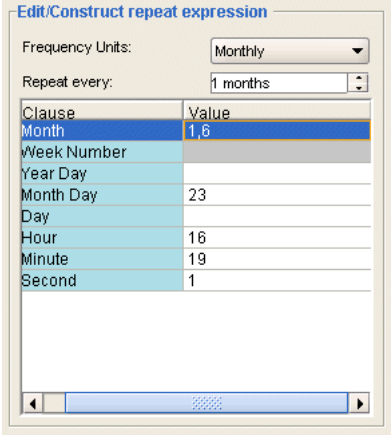
***Figure 11–2 Selector dialog box for Picking Months in a Year***




When you use the selector dialog box and click **OK,** the results are displayed in the Schedule Editor. In this way, you can use the selector dialog box to learn what values are valid.

Figure 11–3 displays the Schedule Editor with the results.

***Figure 11–3 Month Clause for January and June***




You can define the following by clauses:

- By Month
- By Week Number

- By Year Day
- By Month Day
- By Day
- By Hour
- By Minute
- By Second
- By Set Position

#### By Month

This specifies in which month or months the schedule is valid. If you type in values, then use numbers such as 1 for January and 3 for March, or use three-letter abbreviations such as FEB for February and JUL for July.

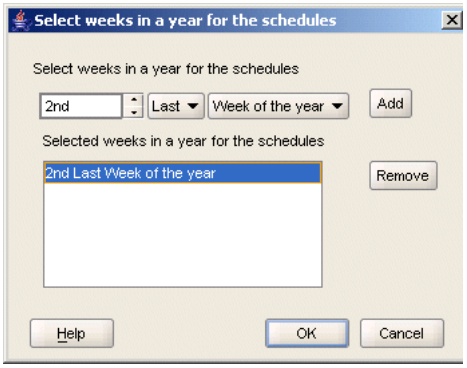
#### By Week Number

Only when you select Yearly for the Frequency Unit can you schedule by the week number in the year.

You can either type in values or click the Ellipsis button to view the selector dialog box. If you type in values, then valid values include positive and negative integers from 1 to 52 or 53, depending on the year. For example, to set a schedule to run on the second to last week of the year, you can either type -2 or fill in the selector dialog box.

Figure 11–4 displays the Selector dialog box with the schedule that runs on the second to last week of every year.

***Figure 11–4 By Week Number Clause Set to Second To Last Week of the Year***




The By Week Number clause follows the ISO-8601 standard, which defines the week as starting with Monday and ending with Sunday. Also, the first week of a year is defined as the week containing the first Thursday of the Gregorian year and containing January 4th.

Using this standard, a calendar year can have 52 or 53 weeks. Part of week 1 may be in the previous calendar year. Part of week 52 may be in the following calendar year. If a year has a week 53, then part of it must be in the following calendar year.

As an example, in the year 1998, week 1 began on Monday, December 29th, 1997. The last week, week 53, ended on Sunday, January 3rd, 1999. Therefore, December 29th, 1997, is in the first week of 1998 and January 1st, 1999, is in the 53rd week of 1998.

#### By Year Day

Use this clause to specify the day of the year as a number. A value of 1 equates to January 1st and 35 is February 4th. Valid values are 1 to 366 and -366 to -1.

The negative values are useful for identifying the same dates year after year despite the occurrence of leap years. For example, the 60th day of the year is March 1st except for leap years when it is February 29th. To calculate the appropriate negative value, count backward from the last day of the year. Therefore, the By Year Day for December 31st is -1. December 30th is -2. To define a schedule for every March 1st, despite leap years, set By Year Day to -306.

#### By Month Day

This clause specifies the day of the month as a number. Valid values are 1 to 31 and -1 to -31. An example is 10, which means the 10th day of the selected month. Use the minus sign (-) to count backward from the last day. For example, if you set the By Month Day clause to -1, the schedule runs on the last day of every month. A value of -2 runs the schedule on the next to last day of every month.

#### By Day

This clause specifies the day of the week from Monday to Sunday in the form MON, TUE, and so on.

You can prefix the By Day values with positive and negative numbers. The numeric prefix that you can use depends on the value that you select for the Frequency Unit.

If you select Yearly as the frequency, you can prefix By Day with values that represent the weeks in a year, 1 to 53 and -53 to -1. Therefore, By Day set to 26Fri equates to the 26th Friday of the year. An entry of -1Mon when the frequency equals Yearly, equates to the last Monday of the year.

If you select Monthly as the frequency, you can prefix By Day with values that represent the weeks in a month, 1 to 5 and -5 to -1. In this case, an entry of -1Mon with frequency set to Monthly results in the last Monday of every month.

#### By Hour

This clause enables you to schedule by the hour. Valid values are 0 to 23 where 0 is midnight, 5 is 5 a.m., 13 is 1 p.m., and 23 is 11 p.m.

#### By Minute

Use this clause to schedule by the minute. Valid values are 0 to 59. As an example, 45 means 45 minutes past the hour.

#### By Second

Use this clause to schedule by the second. Valid values are 0 to 59. As an example, 30 means 30 seconds past the minute.

#### By Set Position

If your Oracle Database version is 10*g* Release 2 or later, you can use this clause to schedule based on the position of items in a previously evaluated list of timestamps. Use other By clauses to return a list of timestamps. Then add the By Set Position clause to select one or more items from that list. This clause is useful for requirements such as running a job on the last workday of the month.

Valid values are 1 through 9999. A negative number selects an item from the end of the list (-1 is the last item, -2 is the next to last item, and so on) and a positive number

selects from the front of the list. This clause is always evaluated last and only once per frequency. The supported frequencies are MONTHLY and YEARLY.

## Example Schedules

Use Table 11–1 as a guide for defining schedules.

***Table 11–1 Example Repeat Expressions for Schedules***

| Schedule Description | Frequency Units | Repeat Every | By Clause |
|---|---|---|---|
| Every Friday | weekly | 1 week | By Day = FRI |
| Every other Friday | weekly | 2 weeks | By Day = FRI |
| Last day of every month. | monthly | 1 month | By Month Day = -1 |
| Second-to-last day of every month | monthly | 1 month | By Month Day = -2 |
| First Friday of any month containing 5 weeks | monthly | 1 month | By Day = -5FRI |
| Last workday of every month | monthly | 1 month | By Day = MON,TUE,WED,THU,FRI;<br>By Set Pos = -1 |
| On March 10th | yearly | 1 year | By Month = MAR<br>By Month Day = 10 |
| Every 12 days | daily | 12 days | Not applicable |
| Every day at 8 a.m. and 5 p.m. | daily | 1 day | By Hour = 8,17 |
| On the second Wednesday of every month | monthly | 1 month | By Day = 2 WED |
| Every hour for the first 3 days of every month | hourly | 1 hour | By Month Day = 1,2,3 |

# Applying Schedules to ETL Objects

After you define a schedule, you associate the schedule with an ETL object. This enables you to execute the ETL object at the times specified in the schedule. You can associate schedules with mappings, Code Template (CT) mappings, process flows, and data auditors.

### Steps to Apply Schedules to ETL Objects

1. In the Projects Navigator, right-click the ETL object to which you want to apply a schedule, and select **Configure.**

   A Configuration tab containing the configuration parameters for the selected object is displayed in the Document window.

2. Click the Ellipsis button on the Referred Calendar field.

   The Referred Calendar dialog box containing the list of available schedules is displayed.

For any mapping or process flow that you want to schedule, the physical name must be 25 characters or fewer and the business name must be 1995 characters or fewer. This restriction enables Warehouse Builder to append to the mapping name the suffix *_job* and other internal characters required for deployment and execution.

3. Select the schedule that you want to apply to the selected object and click **OK**.
4. Deploy the schedule and any associated schedules.

   When properly deployed with its associated objects, the target schema executes the ETL object based on the schedule that you created.

## Scheduling ETL Jobs in Oracle Enterprise Manager

After you successfully deployed a mapping or a process flow, you can schedule it to run in Oracle Enterprise Manager.

---

**Note:** For more information about creating jobs and schedules, refer to the *Oracle Enterprise Manager Concepts* and the Oracle Enterprise Manager Help system.

---

**To schedule a mapping or process flow in Oracle Enterprise Manager:**

1. Successfully deploy the mapping or process flow in Warehouse Builder.
2. Connect to Enterprise Manager as a Warehouse Builder repository owner or repository user.
3. Create a scheduler job that uses the `WB_RT_API_EXEC.RUN_TASK` function in a PL/SQL block.

   For more information about this function, see "The WB_RT_API_EXEC.RUN_TASK Function" on page 11-10.
4. Create a schedule for running the job.

### The SQLPLUS_EXEC_TEMPLATE SQL Script

This script enables you to start the ETL process from SQL*Plus, and to use scheduling tools such as cron, AT, Autosys, and Tivoli.

The `sqlplus_exec_template.sql` script is located in the following directory: *OWB_HOME*`/owb/rtp/sql`.

#### Return Values

1 = Success
2 = Warning
3 = Error

#### Syntax

```
SQLPLUS_EXEC_TEMPLATE rt_owner location task_type task_name
       system_params custom_params
```

#### Arguments

Provide a value for each of the following arguments.

- **rt_owner:** The repository owner.

- **location:** For PL/SQL mappings and process flows, specify the location you used for deployment.

  For SQL*Loader and SAP mappings, set this parameter to `PlatformSchema`. This is a case-sensitive variable.

- **task_type:** Enter the appropriate task type for the mapping or the process flow.
  - `PLSQLMAP`: PL/SQL mapping
  - `SQLLOADERCONTROLFILE`: SQL*Loader mapping
  - `PROCESSFLOW`: Process flow
  - `ABAPFILE`: SAP mapping
  - `DATAAUDITOR`: Data Auditor mapping
  - `SCHEDULEDJOB`: Warehouse Builder scheduled job
- **task_name:** The physical name of the mapping or the process flow
- **system_params:** Values of system parameters for this task type. These values override the default values. Enter the parameters in the form name=value. Separate multiple parameters with commas, and enclose the entire string in double quotes. A backslash (\) is the escape character, when you must include commas or double quotes as literal text.

  The following examples are correct:

  ```
  ","
  "this_param=true"
  "this_param=true, that_param=2"
  ```

- **custom_params:** Values of a custom parameter defined for this task. Refer to system_params for the syntax.

### Examples

In each of the following examples, you may need to provide the path to sqlplus.exe and to `sqplus_exec_template.sql`.

```
sqlplus user/password@tns_name @sqlplus_exec_template MY_RUNTIME MY_WAREHOUSE
PLSQL MY_MAPPING "," ","

sqlplus user/password@tns_name @sqlplus_exec_template MY_RUNTIME PlatformSchema
SQL_LOADER MY_LOAD "," ","

sqlplus user/password@tns_name @sqlplus_exec_template MY_RUNTIME MY_WORKFLOW
PROCESS MY_PROCESS "," ","

sqlplus user/password@tns_name @sqlplus_exec_template MY_RUNTIME PlatformSchema
ABAP MY_SAP "," ","
```

## The WB_RT_API_EXEC.RUN_TASK Function

The `RUN_TASK` function of the `WB_RT_API_EXEC` PL/SQL package enables you to schedule and run the ETL process from Warehouse Builder.

### Return Value

The return value is affected by the parameters of the function.

When `background=0` and `oem_friendly=0`:

1 = Success
2 = Warning
3 = Error

When `background=0` and `oem_friendly=1`:

0 = Success or Warning
3 = Error

When `background=1`:

0 = Task successfully submitted for execution
1 = Task not successfully submitted

### Syntax

```
RUN_TASK
     ( location        IN   VARCHAR2,
       task_type       IN   VARCHAR2,
       task_name       IN   VARCHAR2,
       custom_params   IN   VARCHAR2   DEFAULT NULL,
       system_params   IN   VARCHAR2   DEFAULT NULL,
       oem_friendly    IN   NUMBER     DEFAULT 0,
       background      IN   NUMBER     DEFAULT 0
     )
     RETURN NUMBER;
```

Provide a value for each of the following parameters:

- **location:** For PL/SQL mappings and process flows, specify the location you used for deployment.

  For SQL*Loader and SAP mappings, set this parameter to PlatformSchema. This is a case-sensitive variable.

- **task_type:** Enter the appropriate task type for the mapping or the process flow.
  - `PLSQLMAP` or `PLSQL`: PL/SQL mapping
  - `SQLLoader` or `SQLLoaderControlFile` or `SQLLoaderMap`: SQL*Loader mapping
  - `Process` or `ProcessFlow`: Process flow
  - `ABAPFile` or `SAPMap` or SAP: SAP mapping
  - `DataAuditor`: Data auditor mapping
  - `ScheduledJob`: Warehouse Builder schedule object
  - `AppsCMScheduler`: Concurrent Manager schedule job
  - `DBMSScheduler`: Database schedule job

> **Note:** Previously, you could specify `task_type` using both, a numeric value as well as a literal value. For example, you could use 3 to specify `ProcessFlow`, or 4 to specify `SAP` and so on. But starting with 11g Release 2 (11.2), numeric values are no longer valid. It is mandatory to use the literal value. The value can be in uppercase, lowercase, or in mixed case. For example, `PROCESSFLOW`, `ProcessFlow`, and `processflow` are all valid

- **task_name:** The name of the mapping or the process flow.
- **custom_params:** Values of a custom parameter defined for this task. Refer to system_params for the syntax.
- **system_params:** Values of system parameters for this task type. These values override the default values. Enter the parameters in the form name=value.

  Separate multiple parameters with commas, and enclose the entire string in double quotes. A backslash (\) is the escape character, when you must include commas or double quotes as literal text

  The following examples are correct:

  ```
  ","
  "this_param=true"
  "this_param=true, that_param=2"
  ```

- **oem_friendly:** Controls the return values. Set to 1 for execution in Enterprise Manager, or set to 0 for other environments
- **background:** Controls execution of the task. Set to 1 for background, or set to 0 for foreground.

### Example

The following example displays the return value of the function, which runs a mapping named CUSTOMER_MAP in SALES_TARGET_LOCATION.

```
BEGIN
DBMS_OUTPUT.PUT_LINE('Result: ' || TO_CHAR(gccrep.wb_rt_api_exec.run_task(
'SALES_TARGET_LOCATION','PLSQLMAP','CUSTOMER_MAP', null, null, 1)));
END;
```

# 12

# Deploying to Target Schemas and Executing ETL Logic

Oracle Warehouse Builder provides functionality that supports a single logical model and multiple physical models. This enables you to design your data warehouse once and implement this design on multiple target systems. In addition, Warehouse Builder supports multiple physically different implementations of the same object definitions.

This chapter describes the implementation environment in Warehouse Builder. It also describes how to deploy objects and execute ETL logic.

This chapter contains the following topics:


- Overview of Deployment and Execution in Warehouse Builder
- Steps in the Deployment and Execution Process
- Deploying Objects
- Starting ETL Jobs
- Starting ETL Jobs in SQL*Plus
- Managing Jobs Using SQL Scripts
- Example: Updating a Target Schema


## Overview of Deployment and Execution in Warehouse Builder

After you design your data warehouse, you must implement this design in the target schema by deploying and executing design objects. The Control Center Manager offers a comprehensive deployment console that enables you to view and manage all aspects of deployment and execution. It provides access to the information stored in the active Control Center.

### About Deployment

Deployment is the process of creating physical objects in a target location according to the logical objects defined in a Warehouse Builder workspace. The data objects created when you designed the target schema and defined ETL objects are logical definitions. Warehouse Builder stores the metadata for these data objects in the workspace. To create these objects physically in the target schema, you must deploy these objects. For example, when you create a table using the Design Center, the metadata for this table is stored in the workspace. To physically create this table in the target schema, you must deploy this table to the target schema.

As part of the deployment process, Warehouse Builder validates and generates the scripts for the object, transfers the scripts to the Control Center, and then invokes the scripts against the deployment action associated with the object. You can deploy an object from the Projects Navigator or using the Control Center Manager.

> **Note:** Whenever you deploy an object, Warehouse Builder automatically saves all changes to all design objects to the workspace. You can display a warning message by selecting **Prompt for commit** on the Preferences dialog box.

You can deploy only those objects for which you have the `COMPILE` privilege. By default, you have this privilege on all objects in the workspace. However, the workspace owner may have instituted a different security policy.

> **Note:** Always maintain objects using Warehouse Builder. Do not modify the deployed, physical objects manually in SQL. Otherwise, the logical objects and the physical objects will not be synchronized, which may cause unpredictable results.

### About Deployment Actions

As soon as you define a new object in the Design Center, the object is listed in the Control Center Manager under its deployment location. Each object has a default deployment action, which you can display. The default deployment action for an object is based on a comparison of its current design status to its current deployment status. For example, a table that has not been previously deployed will have a default deployment action of Create. A table that was previously deployed will have a default action of Upgrade. You can override the default by choosing a different deployment action in the Control Center Manager.

The default is set by the previous action and varies depending on the type of object.

These are the deployment actions:

- **Create**: Creates the object in the target location. If an object with that name already exists, then an error may result. For example, this may happen if the object has not been previously deployed from Warehouse Builder.
- **Upgrade**: Modifies the object without losing data, if possible. You cannot undo or redo an upgrade action. This action is not available for some object types, such as schedules.

  > **Note:** When you use the Control Center to upgrade a table that contains a `ROW MOVEMENT` clause in its DDL script, the upgrade fails.
  >
  > To solve this problem, before you deploy the table using an Upgrade action, set the Row Movement configuration parameter of the table to NULL and then deploy the table.

- **Drop**: Deletes the object from the target location.
- **Replace**: Deletes and re-creates the object. This action is quicker than Upgrade, but it deletes all data.

### About Deployment Status

After you deploy an object, Warehouse Builder assigns a deployment status to it. The status represents the result of the deployment. You can view the deployment status in the Control Center Manager.

The deployment status can be one of the following:

- **Not Deployed:** Indicates that the object has not yet been deployed to the target schema.
- **Success:** Indicates that the object has been successfully deployed to the target schema.
- **Warning:** Indicates that some warnings were generated during the deployment of the object.
- **Failed:** Indicates that deployment of the object failed.

### About Deploying Dimensional Objects

To deploy MOLAP dimensional objects, ensure that the version of the location and the the PL/SQL Generation Mode configuration parameter of the Oracle module are set. The location version must be at least 11gR1, and the configuration parameter PL/SQL Generation Mode must be either default, 11gR1 or higher to generate 11g AWs that support cube organized MVs.

### About Deploying Mappings and Process Flows

ETL objects include mappings and process flows. Deploying a mapping or process flow includes these steps:

- Generate the PL/SQL, SQL*Loader, or ABAP script, if necessary.
- Register the required locations and deploy any required connectors. This ensures that the details of the physical locations and their connectors are available at runtime.
- Transfer the PL/SQL, XPDL, SQL*Loader, or ABAP scripts from the Design Center to the Control Center.

To successfully deploy Warehouse Builder process flows to Oracle Workflow, ensure access to the correct version of Oracle Workflow as described in the *Oracle Warehouse Builder Installation and Administration Guide for Windows and UNIX*.

### About Deploying Code Template (CT) Mappings and Web Services

Before you deploy CT mappings or Web services, you must start the Control Center Agent as described in "Starting the Control Center Agent (CCA)" on page 7-23.

Deploying Code Template mappings or Web services includes the following steps:

1. Generate the `.ear` files.
2. Transfer the .ear files to the OC4J server associated with the Control Center Agent.

### About Deploying Schedules

Depending on the ETL objects that are associated with the schedule, you can deploy schedules to Oracle Database or Concurrent Manager.

> **See Also:** "Overview of Deploying Schedules" on page 11-2 for more information about deploying schedules

For remote Oracle Workflow locations and remote Oracle Warehouse Builder 10*g* locations to which schedules are deployed, ensure that the target location has the `CREATE SYNONYM` system privilege. If the evaluation location is specified or the deployment location references a different database instance from the Control Center schema, then the deployment location must have the `CREATE DATABASE LINK` system privilege.

## About Execution

For objects that contain ETL logic (such as mappings, process flows, transformations, Code Template mappings, and Web services) there is an additional step of execution. Execution is the process of executing the ETL logic defined in the deployed objects.

Typically, objects are deployed once and they can be executed multiple times. When there are changes in the object definition, you must redeploy the objects. For example, you deploy a mapping after it is defined. The mapping can be executed once or scheduled to be executed at specific intervals (daily or weekly). If the mapping definition changes, then you must redeploy the mapping.

For example, you define a mapping that sources data from a table, performs transformations on the source data, and loads it into the target table. When you deploy this mapping, the PL/SQL code generated for this mapping is stored in the target schema. When you execute this mapping, the ETL logic is executed and the data is picked up from the source table, transformed, and loaded into the target table.

Another example is of defining a Web service that checks if the data in a table conforms with the data rules defined for the table. When you deploy a Web service, the `.ear` file generated for the Web service is transferred to the Control Center Agent. When you execute the Web service, the ETL logic defined in the Web service is executed and a check is performed to verify that the data in the table does not violate any data rules defined on the table.

## About Configurations

Warehouse Builder separates the logical design of the objects from the physical details of the deployment. It creates this separation by storing the physical details in configuration parameters. An object called a *named configuration* stores all of the configuration settings. Use named configurations to implement different physical parameters for the same design on different systems (for example, development, production, testing). This enables you to easily move Warehouse Builder applications from the development to the test environment and then into production. For example, on the development system, you can specify the parallel settings as NOPARALLEL. On the production system, you can specify the parallel setting as PARALLEL with a degree of 16.

You can create a different named configuration for each deployment location, with different settings for the object parameters in each one. Each named configuration is associated with only one control center.

> **See Also:** *Oracle Warehouse Builder Installation and Administration Guide for Windows and UNIX* for more information about creating multiple configurations

## About Viewing and Setting Configuration Properties for Different Configurations

When you create multiple configurations in a project, you can set different configuration parameters for an object in each configuration. At any time, only one of this configurations is activated, and is called the active configuration. For example,

you have three configurations, PROD_CONFIG, DEV_CONFIG, and QA_CONFIG. For a table, SALES_TAB, you can set different configuration parameters in each of the three configurations.

> **See Also:** *Oracle Warehouse Builder Installation and Administration Guide for Windows and UNIX* for more information about multiple configurations

When you right-click and object and select **Configure**, the configuration parameters for the object in the active configuration, are displayed in a new Configure tab. If you change the active configuration by selecting a different configuration using the Configuration list on the toolbar, then the configuration parameters for the newly-activated configuration are listed to the right of the existing configuration parameters.

You can display the configuration parameters for other configurations defined in your project by clicking **Manage Configuration Columns** on the toolbar displayed at the top of the Configure tab. The Select Configurations dialog box is displayed. This dialog box lists all the configurations defined in the project. Select the configurations for which you want to display configuration parameters in the Configure tab and click **OK**. The parameters for the selected configurations are displayed, adjacent to the current configuration parameters.

You can compare the configuration property settings for different configurations by clicking **Highlight Differences** on the toolbar displayed at the top of the Configure tab. Warehouse Builder highlights the configuration parameters that have different settings in the various selected configurations. This enables you to compare the configuration settings an object in each configuration.

To delete the configuration parameters for a particular configuration from the Configure tab, click **Manage Configuration Columns**. In the Select Configurations dialog box that is displayed, deselect the configuration whose settings you want to remove from the Configure tab, and click **OK**.

The set of configuration parameters for the active configuration are always displayed in the Configure tab and you cannot delete this set.

## Steps in the Deployment and Execution Process

During the lifecycle of a data system, you typically will take these steps in the deployment process to create your system and the execution process to move data into your system:

1. Select a named configuration, from the list of configurations on the toolbar, with the object settings and the Control Center that you want to use.
2. Deploy data objects and ETL objects to the target location. You can deploy them individually, in stages, or all at once.

   For information about deploying objects, see "Deploying Objects" on page 12-6.
3. Review the results of the deployment. If an object fails to deploy, then fix the problem and try again.

   For more information about deployment results, see "Reviewing Deployment Results" on page 12-8.
4. For executable objects such as mappings or process flows, you can either start the ETL process immediately or schedule its execution for a later date.

For information about starting the ETL process, see "Starting ETL Jobs" on page 12-9.

For information about scheduling ETL objects, see "Scheduling ETL Jobs" on page 11-1.

5. Revise the design of target objects to accommodate user requests, changes to the source data, and so forth.
6. Set the deployment action on the modified objects to Upgrade or Replace.

   For more information about deployment actions, see "About Deployment Actions" on page 12-2.
7. Repeat steps 1 to 4.

---

**Note:** Warehouse Builder automatically saves all changes to the workspace before deployment.

---

# Deploying Objects

You can deploy objects using the Projects Navigator or Control Center Manager. Deployment from the Projects Navigator is restricted to the default action, which may be set to Create, Replace, Drop, or Update. The default action is determined by changes to the object design since it was last deployed. To override the default action, use the Control Center Manager, which provides full control over the deployment process.

After deploying an ETL object, you must explicitly start the scripts that perform the ETL operations defined in the ETL object, as described in "Starting ETL Jobs" on page 12-9.

## Deploying Objects Using the Control Center Manager

When you use the Control Center Manager to deploy objects, Warehouse Builder automatically deploys all dependent objects of the objects being deployed.

---

**Note:** Numerous settings on the Preferences dialog box control the behavior of Control Center Manager. Additional settings control the actual deployment process.

From the Tools menu, click **Preferences**. The settings are listed under Control Center Monitor and Deployment. Click **Help** for descriptions of the settings.

---

### Steps to Deploy Objects Using the Control Center Manager

1. In the Projects Navigator, open the project containing the object that is to be deployed.
2. Select **Control Center Manager** from the Tools menu.

   The Control Center Manager that provides access to the control center for the active configuration of the project is displayed. If this menu choice is not available, then check that the appropriate named configuration and Control Center are active.

> **See Also:** *Oracle Warehouse Builder Installation and Administration Guide for Windows and UNIX* for information about configurations

3. If you are deploying relational or ROLAP dimensional objects, ensure that the implementation details of these objects are specified. You can do this by performing binding.

   For more information, see "Relational Implementation of Dimensional Objects" on page 3-9 and "ROLAP Implementation of Dimensional Objects" on page 3-12.

4. (Optional) If you are deploying Code Template mappings or Web services, start the Control Center Agent as described in "Starting the Control Center Agent (CCA)" on page 7-23.

5. In the Control Center Manager navigation tree, expand the location node containing the object to be deployed.

6. Select the objects to be deployed.

   Select multiple objects by holding down the Ctrl key while selecting the objects.

   You can deploy all the objects contained under a particular node by selecting the node. For example, to deploy all tables in a particular module, select the Tables node under that module. To deploy all objects in a module, select the module.

7. In the Details tab of the Object Details panel, set the deployment action for the selected objects.

   To deploy the selected objects using the default deployment action, click **Default Actions** on the Object Details tab.

   > **See Also:** "About Deployment Actions" on page 12-2 for more information about deployment actions

8. Deploy the selected objects.

   To deploy objects to the Control Center Manager, click **Deploy** on the toolbar.

   To deploy objects to a local file, from the File menu, select **Deploy** and then **To File**. Choose the directory in which you want to save the file and provide a name for the file.

   > **Note:** Deploying to a local file option creates a file in the specified directory. The file permissions on this file are set to the Oracle owner. You may need to change the permissions on the file in order to read it.

## Deploying Objects Using the Projects Navigator

When you deploy objects using the Projects Navigator, the deployment action used is the default action set by Warehouse Builder. For more information about the default deployment actions, see "About Deployment Actions" on page 12-2.

Before you deploy an object, ensure that the object is generated successfully. Generation creates the code required to create the object in the target schema.

### Steps to Deploy Objects Using the Projects Navigator

1. In the Projects Navigator, expand the project and then module that contain the object you want to deploy.

2. Ensure that you successfully deploy all dependent objects of the object being deployed.

   For example, before deploying a relational dimension, ensure that all the tables that store the implementation details of the dimension and the sequence used to generate the surrogate identifier are deployed. While deploying a process flow, ensure that all mappings or transformations used in the process flow are successfully deployed.

3. If you are deploying relational or ROLAP dimensional objects, ensure that the implementation details of these objects are specified. You can do this by performing binding.

   For more information, see "Relational Implementation of Dimensional Objects" on page 3-9 and "ROLAP Implementation of Dimensional Objects" on page 3-12.

4. If you are deploying Code Template mappings or Web services, start the Control Center Agent as described in "Starting the Control Center Agent (CCA)" on page 7-23.

5. Select the objects to be deployed and click **Deploy** on the toolbar.

   *or*

   Select the objects to be deployed and then choose **Deploy** from the menu.

   To select multiple objects, hold down the Ctrl key while selecting objects.

After the deployment is complete, a new tab is opened in the Log window to display the details of each deployment.

## Deploying Target Systems to a Remote System

You must perform the following additional steps to deploy design objects to a target schema that is different from the one in which you define the design objects.

1. Install Oracle Warehouse Builder on the target system.
2. Use the Repository Assistant to create a workspace and a repository user on the target system.
3. In the Globals Navigator, create a new configuration and new Control Center that uses the configuration. The Control Center should correspond to the workspace you created on the target system.
4. Set the newly created configuration as the default configuration.
5. Create a location corresponding to the remote target schema.
6. Deploy the design and ETL objects.

## Reviewing Deployment Results

After you deploy objects, you can review deployment results and check the deployment status. If the objects were not deployed successfully, then you can view the error messages generated for the deployment.

### Viewing Deployment Results for Objects Deployed Using the Control Center Manager

When you use the Control Center Manager to deploy objects, you can monitor the progress of a job using the Status column in the Control Center Jobs panel. When the

job is complete, the new deployment status of the object appears in the Details tab. You can review the results and view the scripts.

**To view deployed scripts:**

1. In the Control Center Jobs panel of the Control Center Manager, double-click the job related to the deployment for which you want to view deployed scripts.

   The Jobs Details window is displayed for the selected job. This window displays details of any errors that occurred during the deployment.

2. In the Job Details window, select the object in the navigation tree.

3. On the Script tab, select a script and click **View Code**, or just double-click the script name.

### Viewing Deployment Results for Objects Deployed Using the Projects Navigator

When you use the Projects Navigator to deploy objects, a new tab is displayed in the Log window for each deployment. This tab contains a node tree with the object name and deployment status. Expand the node to view the following nodes:

- Validation: Expand this node to display the validation messages for the object.
- Scripts: Expand this node to view the scripts that are generated for the object.
- Deployment: Expand this node to view the deployment status. Details of errors that occurred during deployment are listed here. While deploying PL/SQL mappings, information about whether the error occurred in the package body or the package specification is also included.

**To view deployed scripts:**

1. In the Log window, select the tab related to the deployment for which you want to view deployed scripts.

2. Expand the node displaying the object name, and then the Scripts node.

   The scripts used for the deployment are listed under the Scripts node.

3. Double-click the script that you want to view.

   A new tab is opened in the Document window containing the generated code. This tab contains the Source, Spec, and Body subtabs. The Spec subtab contains the package specification, and the Body subtab contains the package body.

   If the object deployment fails, you can use the Spec and Body subtabs to view the code in which the error occurred.

## Starting ETL Jobs

ETL is the process of extracting data from its source location, transforming it as defined in a mapping, and loading it into target objects. When you start execution of a mapping, process flow, or data auditor, you submit it as a job to the Warehouse Builder job queue. The job can start immediately or at a scheduled time, if you create and use schedules. For more information about schedules, see "Defining Schedules" on page 11-2.

Like deployment, you can execute a mapping, process flow, or data auditor from the Projects Navigator or the Control Center Manager. You can also start these jobs by using tools outside of Warehouse Builder that execute SQL scripts.

Starting a mapping, process flow, or data auditor involves the following steps:

1. Deploying the objects, as needed.

For more information about deploying objects, see "Deploying Objects" on page 12-6.

2. Executing the object by using the Projects Navigator or Control Center Manager as described in the following sections.

**To start ETL from the Projects Navigator:**

Select the mapping or process flow, then select **Start** from the File menu.

For every ETL object that you execute (start), a new tab containing an execution log is displayed in the Log window. The tab title is the object name followed by the job ID. Use this tab to monitor the status of the execution and to view execution results.

When you execute objects that use the Control Center Agent, such as Code Template mappings or Web services, the job log files are located in the `OWB_ORACLE_HOME/owb/jrt/log/jrt/job`*`job_id`*`/log.xml` directory path. In this directory path, *`job_id`* is the ID of the executed job. Use this log file to troubleshoot any errors that may occur during the deployment and execution of Code Template mappings or Web services.

**To start ETL from the Control Center Manager:**

Select the mapping or process flow, then click **Start** on the toolbar.

Alternatively, you can select the mapping or process flow, then select **Start** from the File menu.

For information about executing Web services, see "Executing Web Services" on page 16-11.

> **See Also:** "Starting ETL Jobs in SQL*Plus" on page 12-12 for information about using SQL*Plus to start ETL jobs

## Viewing Execution Results for ETL Jobs

After executing ETL objects, Warehouse Builder displays the execution results in a tab in the Log window. A separate tab is used to display the execution results of each ETL job.

The execution results provides detailed information about the ETL job. The information is displayed using the following columns in the execution results tab:

- **Rows Selected:** Represents the number of rows selected from the source objects during the ETL job.
- **Rows Inserted:** Represents the number of rows inserted into the target objects during the ETL job.
- **Rows Updated:** Represents the number of rows in the target objects that were updated as part of the ETL job.
- **Rows Deleted:** Represents the number of rows deleted from the target objects during the ETL job.
- **Errors:** Represents the number of errors encountered during the ETL job execution.
- **Warnings:** Represents the number of warnings encountered during the ETL job execution.
- **Start time:** Represents the time at which the ETL job was started.
- **Elapsed time:** Represents the time taken to complete the ETL job.

### Execution Results Tab Icons

Icons at the top of the execution results tab enable you to select the information that should be displayed on the tab. Table 12–1 displays the icons and their uses.

***Table 12–1 Execution Results Tab Icons***

| Icon | Name | Description |
|---|---|---|
| | Show Warnings | Lists any warnings that occur during the ETL job execution in the Execution Results tab. This is a toggle switch. |
| | Show Errors | Lists any errors that occur during the ETL job execution in the Execution Results tab. This is a toggle switch. |
| | Show Parameters | Lists the values of parameters that are used during the ETL job execution. The parameters are listed under the Parameters node. |
| | Stop Job | Stops the execution of the ETL job currently being executed. This icon is disabled when the execution is complete or has not started. |
| | Go To Source | Displays the editor for the ETL object that is currently being executed. |
| | Show Details | Displays the Execution Details dialog box. |

### Return Status of ETL Jobs

Every ETL job that is completed should have one of the following values as its return status.

- SUCCESS - Mapping completes successfully with no errors.
- WARNING - Mapping completes with errors but does not exceed the Maximum Number of Errors parameter.
- ERROR - Mapping does not complete, or mapping has more errors than the Maximum Number of Errors parameter.

## Viewing the Data

After ETL is completed, you can easily check any data object in Warehouse Builder to verify that the results are as you expected.

**To view the data stored in a data object:**

In the Projects Navigator, right-click the object and select **Data**. The Data Viewer will open with the contents of the object.

## Scheduling ETL Jobs

You can use any of the following methods to schedule ETL:

- Use the scheduler.

  See "Defining Schedules" on page 11-2.

- Use the Oracle Database Scheduler by means of the PL/SQL package `DBMS_SCHEDULER` PL/SQL.

  See *Oracle Database PL/SQL Packages and Types Reference* for details about the `DBMS_SCHEDULER` package

- Use a third-party scheduling tool.

## Starting ETL Jobs in SQL*Plus

In addition to executing objects using the Control Center Manager, you can use SQL*Plus. To do this, use a script provided with Warehouse Builder named `sqlplus_exec_template`. Alternatively, you can use `sqlplus_exec_background_template` to run a job in the background.

Take these steps to run the `SQLPLUS_EXEC_TEMPLATE` script in SQL*Plus:

1. From the Tools menu of the Design Center, select SQL*Plus.

   The SQL*Plus panel is displayed.

2. Connect as a Warehouse Builder user, not as a repository owner.

3. Start the script, using syntax such as the following:

```
@%ORACLE_HOME%\owb\rtp\sql\sqlplus_exec_template MY_RUNTIME MY_WAREHOUSE PLSQL
MY_MAPPING "," ","
```

**See Also:** "The SQLPLUS_EXEC_TEMPLATE SQL Script" on page 11-9 for a complete description of the syntax

### Managing Jobs Using SQL Scripts

Numerous SQL scripts are installed with Warehouse Builder so that you can manage deployment jobs, execution jobs, and the Control Center using SQL scripts. The scripts are located in *ORACLE_HOME*/`owb/rtp/sql` directory. Comments in these scripts explain how to use them.

**See Also:** *Oracle Warehouse Builder Installation and Administration Guide for Windows and UNIX* for information about using these scripts.

## Example: Updating a Target Schema

#### Scenario

You are in charge of managing a data warehouse that has been in production for a few months. The data warehouse was originally created by using two source schemas, Human Resources (`HR`) and Order Entry (`OE`) and was loaded into the Warehouse (`WH`) target schema. Recently you learned of two changes to tables in the `HR` and `OE` schemas. The `WH` schema must be updated to reflect these changes.

- Change #1: The first change was made to the `HR` schema. The length of the `REGION_NAME` column in the `REGIONS` table was changed from 25 to 100 characters.

  Figure 12–1 displays a representation of the changes to the `REGIONS` table.

*Figure 12–1 Changed REGIONS Table*

Original REGIONS Table

| Column Name | Data Type | Length | Precision | Scale |
|---|---|---|---|---|
| REGION_ID | Number | | 0 | 0 |
| REGION_NAME | Varchar2 | 25 | | |

Length of REGION_NAME → Changed from 25 to 100

Updated REGIONS Table

| Column Name | Data Type | Length | Precision | Scale |
|---|---|---|---|---|
| REGION_ID | Number | | 0 | 0 |
| REGION_NAME | Varchar2 | 100 | | |

- Change #2: The second change was made to the `OE` schema. A row called `LOT_SIZE_NUMBER` was added to the `ORDER_ITEMS` table with a data type of NUMBER, precision of 8, and scale of 0.

  Figure 12–2 displays a representation of the changed ORDER_ITEMS table.

*Figure 12–2 Changed ORDER_ITEMS Table*

Original ORDER_ITEMS Table

| Column Name | Data Type | Length | Precision | Scale |
|---|---|---|---|---|
| ORDER_ID | Number | | 12 | 0 |
| LINE_ITEM_ID | Number | | 3 | 0 |
| PRODUCT_ID | Number | | 6 | 0 |
| UNIT_PRICE | Number | | 8 | 2 |
| QUANTITY | Number | | 8 | 0 |

Column LOT_SIZE_NUMBER → Added with Precision 8 and Scale 0

Updated ORDER_ITEMS Table

| Column Name | Data Type | Length | Precision | Scale |
|---|---|---|---|---|
| ORDER_ID | Number | | 12 | 0 |
| LINE_ITEM_ID | Number | | 3 | 0 |
| PRODUCT_ID | Number | | 6 | 0 |
| UNIT_PRICE | Number | | 8 | 2 |
| QUANTITY | Number | | 8 | 0 |
| LOT_SIZE_NUMBER | Number | | 8 | 0 |

### Solution

To update the `WH` schema, you must first determine the impact of these changes, and then create and execute a plan for updating the target schema. The following steps provide an outline for what you must do:

Step 1: Identify Changed Source Objects

Step 2: Determine the Impact of the Changes

Step 3: Reimport Changed Objects

Step 4: Update Objects in the Data Flow

Step 5: Redesign your Target Schema

Step 6: Redeploy Scripts

Step 7: Test the New ETL Logic

Step 8: Update Your Discoverer EUL

Step 9: Execute the ETL Logic

### Case Study

#### Step 1: Identify Changed Source Objects

The first step in rolling out changes to your data warehouse is to identify the changes in source objects. To do this, you must have a procedure or system in place that can notify you when changes are made to source objects.

In this scenario, the group managing the HR and OE schemas informed you that some objects had been changed. The first change was made to the HR schema. The REGION_NAME column was extended from 25 to 100 characters to accommodate longer names. The second change was made to the OE schema. The LOT_SIZE_NUMBER column was added and must be integrated into the WH schema.

#### Step 2: Determine the Impact of the Changes

After you have identified the changes, you must determine their impact on your target schema.

For Change #1, made to the HR schema, you must update any dependent objects. This entails reimporting the REGIONS table and then updating any objects that use the REGION_NAME column. To identify dependent objects, you can use the Impact Analysis diagram. You also must update any mappings that use this table.

For Change #2, made to the OE schema, in addition to reimporting the table and updating mappings, you must find a way to integrate the new column into the WH schema. Because the column was added to keep track of the number of parts or items in one unit of sales, add a measure called NUMBER_OF_IND_UNITS to the SALES cube in the WH schema and have this measure for each order. Then you must connect this new column to the SALES cube.

#### Step 3: Reimport Changed Objects

Because two source objects have changed, you must reimport their metadata definitions into your workspace. Select both the REGIONS table in the HR schema and the ORDER_ITEMS table in the OE schema from the navigation tree and use the Metadata Import Wizard to reimport their definitions.

Warehouse Builder automatically detects that this is an update and proceeds by only updating changed definitions. The Import Results dialog box that appears at the end of the import process shows the details of the synchronization. Click **OK** to continue the import and commit your changes to the workspace. If you do not want to continue with the import, then click Undo.

#### Step 4: Update Objects in the Data Flow

If the change in the source object altered only existing objects and attributes, such as Change #1 in the HR schema, use Impact Analysis diagrams to identify objects that must be reconciled.

In this scenario, you must reconcile the column length in all objects that depend on the REGIONS table to ensure that the data continues to load properly.

**To update objects in the data flow:**

1. Select the REGIONS table in the HR schema from the navigation tree. Select **View** and then click **Impact**.

A new tab is displayed in the Document Editor containing the Impact Analysis diagram. This reveals that the CUSTOMER dimension in the WH schema is the only object affected by the REGIONS table.

This step requires that you have already set up the Repository Browser. For more information about setting this up, see *Oracle Warehouse Builder Installation and Administration Guide for Windows and UNIX*.

2. Open the CUSTOMER dimension in the Dimension Editor and update the Region Name level attribute to the 100-character length.
3. Open the MAP_CUSTOMER mapping that connects the source to the target. For both the REGIONS Table operator and the CUSTOMER Dimension operator, perform an inbound synchronization from data object to mapping operator.

   The mapping operators must be synchronized with the mapping objects that they represent to generate code based on the updated objects.

You have now completed updating the metadata associated with Change #1.

Because Change #2 introduced a new column, you need not update the data flow as you did for Change #1. Ensure that you perform an inbound synchronization on all the mappings that use an ORDER_ITEMS Table operator. From the Impact Analysis diagram for the ORDER_ITEMS table, you can see that only the mapping MAP_SALES is affected.

### Step 5: Redesign your Target Schema

Because Change #2 introduced the new LOT_SIZE_NUMBER column to the ORDER_ITEMS table, you must redesign your WH target schema to incorporate this new data into your cube. You can do this by adding a new measure called NUMBER_OF_IND_UNITS to your SALES cube.

**To redesign the target schema:**

1. Add the measure NUMBER_OF_IND_UNITS with the NUMBER data type, precision of 8, and scale of 0 to the SALES cube.
2. View the lineage diagram for the SALES cube to determine which mappings contain the SALES cube. Perform an inbound synchronization on all SALES cube mapping operators.
3. Open the mapping MAP_SALES and ensure that the table ORDER_ITEMS is synchronized inbound.
4. Connect the LOT_SIZE_NUMBER column in the ORDER_ITEMS table to the Joiner, and then to the Set Operation, and then add it to the Aggregator operator. Ensure that you are doing a Sum operation in the Aggregator operator.
5. Finally, connect the LOT_SIZE_NUMBER output attribute of the Aggregator operator to the NUMBER_OF_IND_UNITS input attribute of the SALES cube.

### Step 6: Redeploy Scripts

After the mappings have been debugged, use the Design Center to regenerate and redeploy scripts. Use the Control Center Manager to discover the default deployment action. Warehouse Builder detects the type of deployment to run.

### Step 7: Test the New ETL Logic

After you have reconciled all objects and ensured that the WH target schema has been updated to reflect all changes, test the ETL logic that is be generated from the

mappings. Use the Mapping Debugger to complete this task. If you find any errors, then resolve them and redeploy the scripts.

### Step 8: Update Your Discoverer EUL

If you are using Oracle Discoverer as your reporting tool, proceed by updating your EUL.

**To update your Oracle Discoverer EUL:**

1. Identify the objects that must be updated in the End User Layer (EUL) because of changes made to their structure or data. In this case, the changed objects are the `REGIONS` and `SALES_ITEMS` tables and the `SALES` cube.
2. In the Projects Navigator, select all the objects identified in Step 1, right-click and select **Derive**.

   The Perform Derivation Wizard displays and updates these object definitions in the Business Definition Module that contains these objects.
3. Expand the Item Folders node in the Business Definition Module that contains these changed objects.
4. Select the objects identified in Step 1, right-click and select **Deploy**.

   The changes to the objects are updated in the Oracle Discover EUL.

### Step 9: Execute the ETL Logic

After the mappings have been deployed, execute and load data into the target.

# 13

# Auditing Deployments and Executions

Auditing deployment and execution information can provide valuable insights into how your target is being loaded and how you can further optimize mapping and process flow settings and parameters. It also provides immediate feedback on deployment information that enables you to retrieve the deployment history of an object. The Repository Browser in Oracle Warehouse Builder provides the auditing and deployment information.

This chapter contains the following topics:


- About Auditing Deployment and Executions
- Opening the Repository Browser
- Design Reports
- Control Center Reports
- Common Repository Browser Tasks


## About Auditing Deployment and Executions

When you deploy and execute objects, Warehouse Builder generates and stores audit information related to the deployments and executions. You can use one of the following methods to view and manage audit information:

- Control Center Manager

  When you perform deployment or execution using the Control Center Manager, the Control Center Jobs panel displays the results of the deployment or execution. Double-click the row representing a deployment or execution job to display the Job Details window. For execution, the Job Details window displays the number of rows selected, inserted, updated, and deleted.

- Repository Browser or Heterogeneous Repository Browser

  The sections in this chapter describe how to use the Repository Browser and Heterogeneous Repository Browser to access audit information

- Public views that expose Warehouse Builder repository audit metadata

  For more information about the public view, see *Oracle Warehouse Builder API and Scripting Reference*.

## About the Repository Browser

The Repository Browser is a browser-based tool that generates reports from data stored in Oracle Warehouse Builder repositories. Using the Repository Browser, you can view:

- Detailed information about the design of a workspace. Reports are generated from data stored in the workspaces.
- Reports that provide access to both high-level and detailed ETL runtime information. This information includes the following:
  - Timings for each mapping and process flow
  - Details of activities for each process flow
  - Error details
  - Deployment information to manage separate target environments

As an alternative to using the Repository Browser, you can access the same information through the public views. Start a SQL*Plus session and query the public views. See *Oracle Warehouse Builder API and Scripting Reference* for a list of public views.

You can use Warehouse Builder to deploy data to non-Oracle databases such as Microsoft SQL Server and IBM DB2 UDB. However, in these cases, you will not be able to view design-time object data and deployment audit data. You can only view reports that contain execution audit data.

For more information about reports that will be generated for the Repository Browser in a heterogeneous environment, see "List of Heterogeneous Repository Browser Reports" on page 13-4.

## About the Heterogeneous Repository Browser (HRAB)

The Heterogeneous Repository Browser (HRAB) is a browser-based tool that generates reports about objects deployed to OC4J servers or heterogeneous databases such as DB2 and SQL Server. Heterogeneous Repository Browser enables you to access auditing information from systems that do not have Oracle Warehouse Builder installed.

Using the Heterogeneous Repository Browser, you can view reports that provide:

- Execution information about ETL objects deployed to heterogeneous databases such as SQL Server and DB2.
- Execution information for objects deployed to an OC4J server, such as Web services

Before you access auditing information from heterogeneous databases or OC4J servers, you must create data stores that represent the database or OC4J server. Your administrator will create the data store and assign it to the Repository Browser or OC4J server.

### Differences Between Repository Browser and Heterogeneous Repository Browser

- The Heterogeneous Repository Browser reports provide information about object execution only, not design and deployment.
- Heterogeneous Repository Browser reports contain only a subset of the information provided by Repository Browser reports.

For example, Repository Browser reports contain trace information generated while debugging mappings. Heterogeneous Repository Browser reports do not contain this information.

### Installing the Heterogeneous Repository Browser on Heterogeneous Databases and OC4J Servers

To enable access auditing information for deployments to heterogeneous databases and OC4J servers (that are not part of an Oracle Warehouse Builder installation), you must install the audit tables and views in the target database.

The scripts to install audit tables are stored in the *OWB_ORACLE_HOME*/owb/rtasst/jrtaudit directory. This directory contains subdirectories called db2, sqlserver, and oracle. Depending on your target database, select the subdirectory from which to execute the installation scripts.

For example, to install audit tables and views on DB2, execute the install.sql file located in the *OWB_ORACLE_HOME*/owb/rtasst/jrtaudit/db2 directory.

### Creating Data Stores

Data stores enable you to access auditing information about deployments to heterogeneous databases such as DB2 or SQL Server, and to OC4J servers that are not part of the Warehouse Builder installation.

Your administrator will create the data store in the Heterogeneous Repository Browser of the database (SQL Server or DB2) or the OC4J server. The data store should be named AuditDS.

Data stores can be created using the admin_client.jar command-line utility. The following is an example of creating a data store testDataStore.

```
java -jar admin_client.jar deployer:oc4j:localhost oc4jadmin
  -addManagedDataSource -applicationName default -dataSourceName testDataSource
  -jndiLocation jdbc/testDataSource -connectionPoolName scottConnectionPool
```

You are prompted to enter the password for the oc4jadmin user. Enter the password and press the Enter key.

Data stores are also created implicitly. If a Code Template (CT) mapping references an object stored in a database location, then, when the CT mapping is deployed, a suitable data source is created.

## Types of Auditing

Auditing information displayed by the Repository Browser can be classified as follows:

- PL/SQL Runtime Auditing

  This refers to the auditing information for the traditional Warehouse Builder runtime auditing of objects deployed using PL/SQL scripts.

  Data objects, mappings, and process flows use PL/SQL deployments.

- Heterogeneous Runtime Auditing

  This refers to auditing information about objects that are deployed to an OC4J server or to a heterogeneous database such as DB2 or SQL Server. Includes auditing from more than ne Control Center Agents (CCA).

This includes information about code templates, Web services, and Code Template mappings.

Table 13–1 describes the types of auditing available when you deploy Warehouse Builder objects to different target locations.

***Table 13–1 Types of Auditing for Different Deployment Location***

| Deployment Location | PL/SQL Runtime Auditing | Heterogeneous Runtime Auditing |
|---|---|---|
| Oracle Database, with a Warehouse Builder installation | Yes | Yes |
| Oracle Database, without Warehouse Builder installation | No | Yes |
| Heterogeneous database | No | Yes |
| OC4J server | No | Yes |

## List of Heterogeneous Repository Browser Reports

The content of the reports generated by Heterogeneous Repository Browser is based on the corresponding reports produced by the Repository Browser. The reports are a subset of the reports produced for an Oracle Database environment. The terms used in the Heterogeneous Repository Browser reports are related to heterogeneous databases. For example, the term Process is replaced by Task and Map by Step.

However, Heterogeneous Repository Browser reports contain lesser information than the Repository Browser reports for an Oracle Database. The following Repository Browser reports are available for the Heterogeneous Repository Browser:

- Execution Schedule Report
- Execution Summary Report
- Map Execution Report
- Map Start Report
- Map Run Execution Report
- Map Run Trace Report
- Map Run File Report
- Run Error Diagnostic Report
- Management Reports

The following functionality is available through Heterogeneous Repository Browser reports:

- Purge execution audit data for a job
- Purge error and trace data for a job

## Viewing Audit Reports

Audit reports provide information about deployment and ETL jobs. Each time you deploy an object or start a job, the details of these activities are stored in the workspace. You can access this information from the following environments:

- Control Center Manager
- Repository Browser

The Repository Browser provides the information in the form of predefined reports. The reports are displayed in your default Web browser. Note that the Repository Browser Listener must be running.

# Opening the Repository Browser

To access auditing data for objects deployed to heterogeneous databases or OC4J servers, ensure that you install the required audit tables on the heterogeneous database or OC4J server.

See "Installing the Heterogeneous Repository Browser on Heterogeneous Databases and OC4J Servers" on page 13-3 for details about creating the audit tables.

Opening the Repository Browser is a multistep process:

1. Before you can open the Repository Browser, the Repository Browser Listener must be started as described in "Managing the Repository Browser Listener" on page 13-5.
2. When the Repository Browser Listener is running, you can start the Repository Browser in a number of ways as described in "Accessing the Repository Browser" on page 13-6.
3. The Repository Browser opens to the Login page where you log in to a workspace as described in "Logging in to a Workspace" on page 13-6.

---

**Note:** To open the Repository Browser, you must have the ACCESS_PUBLICVIEW_BROWSER system privilege. You automatically have this privilege if you are the owner of the workspace that you want to browse. If you are not the owner of the workspace, then contact your database administrator to obtain this privilege.

---

## Managing the Repository Browser Listener

Before you can open the Repository Browser, you must start the Repository Browser Listener.

### Starting the Repository Browser Listener in Windows

Open a command prompt window and run `startOwbbInst.bat` located in the *`OWB_ORACLE_HOME`*`\owb\bin\win32` directory. You are prompted to set the password for the OC4J administrator. Enter a password and press the Enter key. When you are prompted to confirm the password, enter the same password again.

### Stopping the Repository Browser Listener in Windows

Open a command prompt window and run `stopOwbbInst.bat` located in the *`OWB_ORACLE_HOME`*`\owb\bin\win32` directory. You are prompted to provide the OC4J administrator password that you set while starting the Repository Browser listener. Type the password and press the Enter key.

### Starting the Repository Browser Listener in UNIX

Run `./startOwbbInst.sh` located in the *`OWB_ORACLE_HOME`*`/owb/bin/unix` directory. You are prompted to set the password for the OC4J administrator. Enter a password and press the Enter key. When you are prompted to confirm the password, enter the same password again.

### Stopping the Repository Browser Listener in UNIX

Run `./stopOWBBInst.sh` located in the *OWB_ORACLE_HOME*`/owb/bin/unix` directory. You are prompted to provide the OC4J administrator password that you set while starting the Repository Browser listener. Type the password and press the Enter key.

## Accessing the Repository Browser

Once the Listener is running, you can start the Repository Browser in any one of the following ways:

- From the **Start** menu, select **Programs,** then the Warehouse Builder folder**,** and then **Warehouse Builder, Repository Browser.**
- From the **Tools** menu of the Design Center, select **Repository Browser**.
- From within any web browser, type the location of the Repository Connection page. For example, if the Repository Browser Listener is running on a computer named `owb_server`, then typing the following address will start the Repository Browser:

  `https://owb_server:8999/owbb/RABLogin.uix?mode=design`

  or

  `https://owb_server:8999/owbb/RABLogin.uix?mode=runtime`

Regardless of which approach you take, once you start the Repository Browser, the browser opens the Repository Connection page from which you log in to the Repository Browser.

## Logging in to a Workspace

To log in to a workspace, specify the connection information for the workspace that you would like to access. If you do not know the connection information, then contact your database administrator.

You can connect to the Control Center either as the workspace owner or as a workspace user. When you connect as the workspace owner, you can purge audit details, validate locations, unregister locations, change service node settings, view user data values, and start or stop execution jobs. When you connect as a workspace user with administrative privileges, you can purge audit details, unregister locations, change service node settings, view user data values related to your execution jobs, and start or stop your execution jobs.

Once you log in to the Repository Browser, you can view any of the following reports:

- Deployment
- Execution
- Management

> **Note:** If you log in to a heterogeneous database or an OC4J server using a data store, you can only view Execution reports.

Depending on the location of the auditing information, see the following sections:

- Connecting to an Oracle Database
- Connecting to a Heterogeneous Database or OC4J Server

If you have access to more than one workspace, the Repository Browser displays a list containing these workspaces. Select the workspace to which you want to connect.

Select **Design Center** to display the Repository Navigator that provides access to reports about object design. Select **Control Center** to display the Control Center Reports page that lists all the types of reports available through the Repository Browser.

### Connecting to an Oracle Database

**User Name:** Provide the name of the workspace owner or workspace user used to connect to the workspace.

**Password:** Provide the password of the user specified in the User Name field.

**Select the host address, port number, and service name:** Choose this option to connect to the workspace by providing the host name, port number, and service name of the Oracle Database that contains the workspace.

**Select the net service name:** Choose this option to connect to the workspace by providing the net service name associated with the Oracle Database that contains the Warehouse Builder repository.

### Connecting to a Heterogeneous Database or OC4J Server

Warehouse Builder enables you to access auditing information about deployments to heterogeneous databases (such as SQL Server and DB2) and to external OC4J servers. You use a data store to connect to the heterogeneous database or OC4J server. Contact your database administrator for the user credentials to be used for the data store and provide these in the User Name and Password fields.

# Design Reports

The Repository Browser provides reports about the design of Oracle Warehouse Builder workspaces. These reports include object summary reports, lineage reports, and impact analysis.

Following are the list of design reports provided by the Repository Browser:

- Object Properties
- Object Reports
- Object Lineage
- Object Impact

## Repository Navigator

Use the Repository Navigator page to search the metadata of a workspace and to access metadata main properties, lineage, impact analysis, and a list of related reports and Control Center reports for the workspace. The Repository Navigator contains the following sections: Search, All, and Related Links.

### Search

Search by the object type, or name, or both.

- To search by object type, select the type of object that you want to search from the **Search By Type** list. The search result is a list of all objects of this type.

- To search by name, enter the name of the object in the Search field. You can search for just the first character of the name, in which case the search result is a list of objects whose names begin with that character.

Click **Go** to start your search.

The Repository Browser displays the results of the search in a new page called the Workspace Objects Search page. You can also search for new objects from this page.

### All

Contains a navigator tree for the workspace.

The use of the columns is described in Table 13–2.

***Table 13–2 Column Description for the Table in the All Section***

| Column Head | Description: |
|---|---|
| Focus | Click an icon in this column to change the focus of the tree. |
| Name | The name of an item in the tree.<br>Click the plus (+) or minus (-) sign next to an item in the tree to expand or collapse the tree.<br>Click the Name of the object to open the Object Properties page for that object. |
| Reports | Click an icon in this column to open the Object Reports page for the related item.<br>The reports listed in the Objects Report page depend on the object for which the reports are displayed. For example, clicking the Reports link for a workspace displays summary reports regarding the workspace, connectors, the Control Center, and locations. Clicking the Reports link for a project displays the Detailed Project Report and Summary Reports. |
| Lineage | Click an icon in this column to open the Object Lineage page for the related item. |
| Impact | Click an icon in this column to open the Object Impact page for the related item. |

### Related Links

Click **Control Center: Reports** to open the Control Center Reports page from which you can select a deployment, execution, or management report.

### Refresh

Click **Refresh** to refresh your view of the workspace.

The tree collapses when you refresh the data. If you had navigated to or focused to a specific area before refreshing, then you can navigate to or focus on the desired node in the tree.

## Object Properties

The Object Properties page displays the properties of the object that you selected in the Repository Navigator. The properties include the object name, business name, validation status, creation timestamp, update timestamp, and the name of the user who created and updated the object.

From this page you can go to the Object Reports, Object Lineage, or Object Impact pages by clicking on the corresponding link on the left side of the page.

**See Also:**

- Summary Reports on page 13-9
- Detailed Reports on page 13-10

## Object Reports

The Object Reports page provides access to the predefined Design Center reports for the object that you selected in the Repository Navigator. Use these reports to examine your metadata.

Click a Report name to display a report.

The following types of reports are available:

- Summary Reports
- Detailed Reports
- Implementation Reports
- Impact Analysis Reports

### Summary Reports

The type of information displayed in a summary report is determined by the object selected. For example, a Table Summary Report lists all tables in the module. A Materialized View Summary Report lists all materialized views in the module. Header information that identifies the module is also displayed. Selecting the name of an item displays the detailed report for that item.

Summary reports are available for the following objects:

Advanced queue
Collection
Connector
Control Center
Cube
Data Profile
Dimension
External Table
Flat File
Function
Location
Mapping
Materialized view
Nested table
Object Type
Package
Pluggable mapping
Procedure
Process flow
Sequence
Table function
Table
Varray
View

### Detailed Reports

The type of information displayed in a detailed report is determined by the object selected. Detailed reports provide comprehensive information about an item. For example, a Detailed Table Report lists information about the table columns, keys, foreign keys, and physical configuration parameters.

Detailed reports are available for the following objects:

Advanced Queue
Alternative Sort Order
Application Server
Business Area
Collection
Configuration Template
Cube
Customer Application
DB2
DRDA
Data Profile
Data Rule
Dimension Drill Path
Dimension
Drill Path
Drill to Detail
Expert Module
Expert
Expert Task
External Table
File Module
File
Function
Gateway
Informix
Installation
Item Folder
Control Code Template
Function Code Template
Code Template Folder
Integration Code Template
Journal Code Template
Load Code Template
Oracle Target Code Template
Code Template Task
List of Values
MIV
Mapping Module
Mapping
Materialized View
Module
Nested Table
ODBC
Object Type
Oracle Business Intelligence Module
Oracle Discoverer Module
Package
Peoplesoft Application

Pluggable Mapping
Procedure
Process Flow Module
Process Flow
Project
RDB Module
Record
Registered Function
SQL Server
Sequence
Siebel Application
Source Module
Sybase Module
Table Function
Table
Target Module
Teradata Module
Varray
View
Web Service Package

### Implementation Reports

Implementation Reports can be run on dimensions and cubes. They provide information about how physical objects are used to implement logical objects.

### Impact Analysis Reports

Impact Analysis Reports list all items belonging to the subject of the report. The name of the mapping and the name of the item that it is mapped to are also displayed. The report provides a one-step impact analysis for all items related to the selected item.

For example, if you want a list of all the columns in a table that are used as sources in any mappings, use this report.

For more information about Impact Analysis Reports, see "Object Impact" on page 13-12.

### Lineage Reports

Lineage Reports list items that are used as targets in a mapping. For more information about lineage reports, see "Object Lineage" on page 13-11.

## Object Lineage

The Object Lineage page displays the lineage diagram for the object that you selected in Repository Navigator. A Lineage Diagram graphically displays all the objects and transformations that are used to form the subject of the Diagram.

Lineage can be performed at either the object level or the item level. At the object level, the diagram can contain tables, views, materialized views, dimensions, cubes, records, and operators. At the item level the diagram can contain columns, measures, fields, operator parameters, and level attributes. The Lineage Diagram is displayed with the subject on the right side of the screen.

From the Object Lineage page you can go to the Object Properties, Object Reports, or Object Impact pages by clicking on the corresponding link on the left side of the page.

## Object Impact

The Object Impact page displays the Impact Analysis diagram for the object that you selected in the Repository Navigator. The Impact Analysis diagram is a graphical representation of the objects on which the definition of the selected object depends. It represents the potential impact of a change in the definition of the selected object. The subject is displayed on the left side of the screen.

From this page you can go to the Object Reports, Object Properties, or Object Lineage pages by clicking on the corresponding link on the left side of the page.

Lineage and Impact Analysis diagrams are created based on a Dependency Index. For the data displayed in the diagram to be current, the index must be refreshed.

When you click **Impact Analysis** for an object, the impact_*objectID*.jpeg file, where *objectID* represents the object ID of the selected object, is generated and displayed on the Browser page. Similarly, when you click **Lineage** for an object, the lineage_*objectID*.jpeg file is generated. For both lineage and impact analysis, in addition to the .jpeg file, an .svg file with the same name is generated. If you need to use these files, then they are available in the following locations:

On Unix, these files are stored in the folder
*OWB_ORACLE_HOME*/owb/j2ee/applications/owbb/generated_images/.

On Windows, these files are stored in the folder
*OWB_ORACLE_HOME*\owb\j2ee\applications\owbb\generated_images\

# Control Center Reports

The Control Center section of the Repository Browser provides the following types of reports: Deployment Reports, Execution Reports, and Management Reports.

> **Note:** You can access the Design Object Reports from any Control Center reports by clicking the **Design Repository: Navigator** link on the report page.

### Deployment Reports

Top-level deployment reports are:

- Deployment Schedule Reports that show basic attributes and display a node tree giving details of all deployments in time order
- Object Summary Reports that show basic attributes, and list deployed objects (processes, maps and data objects) in type, name order with details of their latest deployment
- Locations Reports that show all the locations into which objects have been deployed

From these top-level deployment reports, you can access Deployment Error Detail Reports and Deployment Reports that supply details about the deployment of a specific process flow, mapping, or data object.

### Execution Reports

Top-level execution reports are:

- Execution Schedule Reports that show basic attributes and display a node tree giving details of all Process Runs (and top-level Map Runs) in time order

- Execution Summary Reports that show basic attributes and lists executed processes (and top-level maps) in type, name order

From these top-level execution reports, you can access other reports that allow you to:

- Monitor jobs using Execution Reports (sometimes called Execution Detail Reports) that show the execution job details of a given Process Run or Map Run; Execution Job Reports that show details of logical errors for a given target detected during the execution of Map Runs; and Job Error Diagnostic Reports that show basic details of runtime errors and target details, and, when possible, list source and target column details.
- Display diagnostics using Error Table Execution Reports that show details of logical errors for a given target detected during the execution of Map Run; Trace Reports that show details of source and target values plus data errors detected during the execution of Map Runs; and Job File Reports that show basic Process or Map attributes, list log and data files associated with the Process or Map Run, and display the contents of the selected file.
- Rerun jobs using Job Start Reports that show Process or Map identification properties (including that latest deployment and latest execution dates), list all execution parameters for the Process as specified by the latest deployment, and assign parameter default values from the latest deployment specification.

### Management Reports

The main Repository Browser management report is the Service Node Report that enables you to manage service node information for the Oracle Real Application Clusters (Oracle RAC) system.

Also, from a Locations Report (a top-level deployment report) you can access the Location Validation Report that shows basic Location attributes, current Control Center connection details, and current Location connection details.

## Deployment Reports

Top-level deployment reports are Deployment Schedule Report, Object Summary Report, and Locations Report. From these top-level deployment reports, you can access Deployment Error Detail Reports and Deployment Reports that supply details about the deployment of a specific process, map, or data object.

### Deployment Schedule Report

The Deployment Schedule report is a top-level Control Center report that shows basic attributes and displays a node tree giving details of all deployments in time order.

Use Deployment Schedule reports to view run details, and access Data Object, Map, and Process Deployment reports.

The Deployment Schedule Report contains a node tree with the columns displayed in Table 13–3.

***Table 13–3 Deployment Schedule Report Contents***

| Column Name | Description |
|---|---|
| Select | Click to select this node in the deployment tree. This functionality is used in conjunction with the purge facility on the Deployment Schedule Report. |
| Focus | Click the icon in this column to change the focus of the tree to this node. |

*Table 13–3 (Cont.) Deployment Schedule Report Contents*

| Column Name | Description |
|---|---|
| Name | A tree that represents all of the items in this deployment report. To expand a node, click its plus icon (+). To collapse a node, click its minus icon (-).<br>Click a location name to display the Location Deployment Schedule Report. Click an object name to display the Deployment Report for the object. |
| Dep | A number that identifies a deployment run |
| Type | The type of item |
| Obj Status | The status of the object |
| Date | The date of deployment |
| Dep Status | The status of the deployment. These include the following:<br>▪ Complete: Indicates that the deployment is complete.<br>▪ Busy: Indicates that the deployment is in progress.<br>▪ Ready: Indicates that the job has just started or is about to end.<br>**Note:** If the deployment status shows a Ready status for a very long time, use the script `deactivate_deployment.sql` located in the *`OWB_ORACLE_HOME`*`/owb/rtp/sql` folder to deactivate the status of the deployment. |
| Related Information | Other related information including a link to a related Deployment Error Detail Report, if appropriate |

With a Deployment Schedule Report, you can also:

- Expand deployments to show run details.
- Filter deployments based on date range.
- Set a date range for which you want to view deployments.
- Refresh reports to show up-to-date deployment details.
- If you have sufficient privileges, you can purge selected deployment audit details.

### Location Deployment Schedule Report

A Location Deployment Schedule Report is similar in appearance to a Deployment Schedule Report except that it only shows the deployments for a specific location and does not offer you the opportunity to purge audit details.

Expand the node tree for the location to view the objects deployed to the location. Click an object name to display the Deployment Report for the object. For objects whose deployment failed, use the link in the Related Information column to access the Deployment Error Detail Report.

## Locations Report

This deployment report shows all Locations into which objects have been deployed.

Within this report, you can:

- Sort Locations on name and latest deployment time.
- If you have sufficient privileges, you can unregister selected Locations.

- If you have sufficient privileges and a link appears in the Validation column, you can open a related Location Validation Report to test and update connection details for a Location.

The Locations Report contains the following three sections: Control Center Connection, Logical Details, and Physical Details.

**Control Center Connection** This section displays the service description for the Control Center.

**Logical Details** This section displays details for all locations such as name, type, service description, date of the last deployment to this location, and the user name used for deployment. Click a location name to display the Object Summary Report for that location.

To unregister locations, select the locations by clicking the Select column to the right of the locations and then click **Unregister Selected Location**s.

**Physical Details** This section contains a node tree that displays the host name, port, and service name associated with the Control Center, and the list of locations belonging to the Control Center.

### Object Summary Report

An Object Summary Report shows basic attributes and lists deployed objects (process flows, mappings, and data objects) in type/name order with details of their latest deployment.

Click an object name to display the Deployment Report for the selected object. Click a location name to display the Location Object Summary Report for the location.

Within this report, you can:

- Sort execution runs on name, type, location, latest deployment time, object status
- Filter objects on type and status

### Location Object Summary Report

A Location Object Summary Report lists all the objects (process flows, mappings, and data objects) deployed to the selected location. The details provided include the object name, object type, latest deployment date, and object status.

This report is similar to an Object Summary Report except that it also includes a Location parameters section. The Location Parameters section displays location details such as host name, port number, service name, schema name, FTP user name, and database link name. When you have sufficient privileges, you can update the Web Server Base setting, if applicable.

Click an object name to display the Deployment Report for the selected object.

### Deployment Report

This deployment report supplies details about the deployment of a specific process flow, mapping, or data object.

When the item is a Process Flow, this report shows basic process flow attributes and lists all deployments of the process flows and its subprocesses in time order. When the item is a Mapping, this report shows basic mapping attributes and lists all deployments in time order. When the item is a Data Object, this report shows basic

data object attributes and lists all deployments of the data object and its second-class objects in time order.

Click the link in the Deployment Message section to view details of errors that occurred during deployment.

Within this report you can:

- Sort deployments on deployment time
- Filter deployments on deployment status

### Deployment Error Detail Report

A Deployment Error Detail Report shows details of a specific deployment error and lists all the messages for the deployment error.

Within this report, you can:

- Sort the error messages by message number
- Filter the error messages by severity

## Execution Reports

The top-level execution reports are Execution Schedule Reports and Execution Summary Reports.

From these top-level execution reports, you can access Error Table Execution Reports, Job Error Diagnostic Reports, Trace Reports, Execution Job Reports, Job File Reports, Job Start Reports, and Execution Reports.

### Execution Schedule Report

This execution report shows basic attributes and displays a node tree giving details of all Process Flow executions and top-level Mapping executions in time order. The details for each execution include the name, job ID, type, location name, execution date, and execution status.

Click an object name to display the Execution Report for the object. The Related Information column provides a link to the Execution Job Report. If you have sufficient privileges, then you can purge execution audit details for objects, by selecting the objects and clicking **Purge Selected Audit Details**.

Within this report, you can:

- Focus on details for one Process Run.
- Expand the Process Run to show activity run details.
- Filter Process Runs on execution name, execution status and date range (for example, to display only runs with "busy" status).
- Use the Calendar icon for date picker available to set the start and end of date range.
- Refresh the report to show up-to-date execution run details.
- If you have sufficient privileges, you can purge selected Process Run execution audit details.

### Execution Summary Report

This execution report lists all the Process Flow and Mapping executions in type, name order. You can access the Execution Job Report by clicking the link in the Related Information column.

Within the Execution Summary Report, you can:

- Sort execution runs on name, type, latest execution time, and execution status
- Filter Processes (and Maps) on type and execution status

### Execution Report

An execution report (sometimes called an Execution Detail Report) shows all of the execution run details of a given Process, a given Map, or all of the Map Runs for which a given Data Object is a source or target.

When the Execution Report is for a Process or Map, the report shows basic Process or Map attributes and lists the Process or Map Runs in time order.

Click the Start link at the top of this page to display the Job Start Report.

Within an Execution Report, you can:

- Sort the Process or Map Runs on execution start time
- Hide or show a Process or Map Run to show activity run details
- Filter Process or Map Runs on execution status and execution severity

### Error Table Execution Report

This execution report shows details of logical errors for a given target detected during the execution of Map Runs.

Within this report, you can:

- Sort logical errors on map type, map name, execution start time, rule type, and rule usage.
- Filter logical errors on map name, rule type and rule usage.
- If you have sufficient privileges, you can use the Purge Error Table to remove selected logical errors.

### Execution Job Report

An Execution Job Report shows detailed information about the execution of either a Process Run or a Map Run.

Execution Job reports contain the following sections: Execution Details, Execution Parameters, Step Details, Error Messages, Logical Errors, and Audit Details.

#### Execution Job Report for a Process Run

When the Execution Job Report is for a Process Run, it shows basic Process Run execution details, lists execution parameters, lists activity (Map and Subprocess) details in time order, and lists error messages.

Within an Execution Job Report for a Process Run, you can:

- Hide or show activity details to show Map Run details
- Refresh the report to show up-to-date execution run details
- Terminate the Process Run

### Execution Job Report for a Map Run

When the Execution Job Report is for a Map Run, it shows basic Map Run execution details, including source and target Data Objects, lists execution parameters, lists map step details in time order, lists error messages, lists logical error details, and displays the contents of the SQL Loader log file (if applicable).

Within an Execution Job Report for a Map Run, you can:

- Hide or show map step details, including source and target Data Objects
- Refresh the report to show up-to-date execution run details
- Sort logical errors on error table, map step, rule type, rule usage
- Terminate the Map Run
- If your role has sufficient privileges, you can purge Error and Trace audit details for the Map Run and purge the Error Table to remove selected logical errors

## Trace Report

This execution report, also called the Map Run Trace Report, shows details of source and target values plus data errors detected during the execution of Map Runs.

Within this report, you can:

- Sort files on rowkey, table name.
- Filter diagnostic trace on execution severity and source or target.

> **Note:** Trace diagnostics are available when the Map Run is executed with a particular setting of the Audit Level runtime parameter. Use this trace facility only if required, because it can generate a large volume of audit data.

## Job File Report

This execution report shows basic Process Flow or Mapping attributes, lists log and data files created by external processes during the Process Flow or Map execution, and displays the contents of the selected file.

Within a Job File Report, you can:

- Sort files on file type, creation time.
- View the contents of any selected file.

## Job Start Report

This execution report shows Process Flow or executable Map identification properties, (including latest deployment and latest execution dates), lists all execution parameters for the Process or executable Map as specified by the latest deployment, and assigns parameter default values from the latest deployment specification.

Within a Job Start Report, you can:

- Sort execution parameters on name, category
- Change values of any input parameter where permitted
- Change the default Execution Name as necessary
- Reset all parameter settings to their default values

- Apply basic validation to parameter values
- Start the Process or Map Run, so that it is scheduled for execution immediately
- Navigate to the Deployment Report for latest deployment details of Process or Map
- Navigate to the Execution Run Report for the latest execution of current Process or Map

### Job Error Diagnostic Report

This execution report (also called the Run Error Diagnostic Report) displays diagnostic information gathered for a given runtime error during a process flow or mapping execution. It also shows runtime error details, target details, and lists source and target column details, where possible. Note that some column values are displayed only if your role has the appropriate privilege.

Within this report, you can sort column details on source or target category, source or target name, rowkey, and column name.

## Management Reports

The top-level management report is the Service Node Report. From this report you can open a Location Validation Report.

### Service Node Report

This management report displays and enables you to manage service node information for the Oracle Real Application Clusters (Oracle RAC) system. Specifically, it performs the following functions:

- Shows the basic attributes
- Lists details and status of all service nodes currently used in the Oracle RAC system, generated from the underlying system tables
- Lists service nodes available to the Oracle RAC system that are currently not in use
- Shows the net service name to be used to access the runtime repository

Within a Service Node Report, you can:

- Sort service nodes on instance number, instance name, or runtime version
- Update an instance number when the node is not enabled or active
- Set or unset an enabled setting. (Note that you can never change an active setting as it is maintained by the Oracle RAC system.
- Remove selected service nodes that are not enabled or active from being used by the Oracle RAC system
- Add a node to the service, from the list of available nodes
- Set the runtime repository net service name
- Refresh report to show up-to-date service node details

(Note that you can add, remove or update node details only if you have sufficient privileges.)

### Location Validation Report

This management report shows basic Location attributes such as name and type, current Control Center connection details, and current Location connection details.

Within a Location Validation Report, you can:

- Test the Location connection by clicking **Test Connection**
- Update Location connection details by clicking **Update Details**

## Common Repository Browser Tasks

The following scenarios are examples of some typical actions performed using the Repository Browser:

- Identifying Recently-Run Processes
- Identifying Why a Process Run Failed
- Comparing Process Runs
- Discovering Why a Map Run Gave Unexpected Results
- Identifying Recently-Made Deployments
- Identifying the Data Objects That Are Deployed to a Specific Location
- Identifying the Data Objects That Are Deployed to a Specific Location
- Identifying the Map Runs that Use a Specific Deployed Data Object
- Discovering the Default Deployment Time Settings of a Deployed Process
- Rerunning a Process
- Monitoring a Process Run
- Terminating a Process Run
- Removing the Execution Audit Details for a Process
- Removing Old Deployment Audit details
- Viewing Error Tables Created as a Result of Data Auditor Execution
- Unregistering a Location
- Updating Location Connection Details for a Changed Database Environment
- Updating Service Node Details in a Changing Oracle RAC Environment

### Identifying Recently-Run Processes

1. Open the Execution Schedule Report to see the latest Process runs.
2. Filter the displayed information by using the execution name, execution status, and date range, as required.
3. Note any Process runs that are reported as having errors or not having completed.
4. Expand the tree structure for any Process run identified in Step 3 to see details of its activities (that is, any of its subprocesses and maps).

### Identifying Why a Process Run Failed

1. Open the Execution Schedule Report and note the Process run that is marked as having errors.

2. Click the **Run Execution Report** link, which opens a Execution Report that provides details of the Process run.
3. Note any Map runs that are reported as having errors or not having completed.
4. Click the **Run Execution Report** link, which opens an Execution Report that provides details of any Map run identified in Step 3.
5. For any process-level or map-level error messages, click the **Run Error Diagnostic Report** link, which opens a Job Error Diagnostic Report that displays more details of the error, including source data values.

## Comparing Process Runs

1. Open the Execution Summary Report to see a list of all Processes.
2. Click the Process name to see its Execution Report.
3. Compare the results of previous runs, using the hide/show feature to reveal further details as required.

## Discovering Why a Map Run Gave Unexpected Results

1. Open the Execution Schedule Report and note the Process Run that contains the required Map Run.
2. Click the **Run Execution Report** link that opens an Execution Report that provides details of the Process Run.
3. Click the **Run Execution Report** link that opens an Execution Report that provides details of the Map Run.
4. If the Map Run had the `Audit Level` runtime parameter set to `Complete`, select the Trace tab link to see its Trace Report.
5. Filter trace lines by error and source or target as required, and note any unexpected source or target actions.
6. For error messages, click the **Run Error Diagnostic Report** link that opens a Job Error Diagnostic Report that displays more details of the error, including source data values.
7. Click **Purge Error** and **Trace Lines** to remove all details or errors and trace for this Map Run, if they are no longer required.

   If purging, a confirmation screen will be shown, requesting that the action be confirmed.

## Identifying Recently-Made Deployments

1. Open the Deployment Schedule Report to see the latest deployments.
2. Filter the displayed information by using the date range, as required.

   Note any deployments that are reported as having errors or not having completed.
3. Expand the tree structure for any deployment to see details of its components (that is, units and deployed objects).
4. For error messages, click the **Deployment Error Detail Report** link to display the related Deployment Error Detail Report.

## Identifying the Data Objects That Are Deployed to a Specific Location

1. Open the Locations Report to see the registered Locations.
2. Click the Location name link to see its Object Summary Report.
3. Filter the displayed information by using object type and object status, as required.
4. Click the **Name** link for a Data Object to see its Deployment Report.

   Details are shown for all deployments of this Data Object and its second-class Data Objects.

## Identifying the Map Runs that Use a Specific Deployed Data Object

1. Open the Object Summary Report to see a list of all deployed objects.
2. Filter the information shown by using object type and object status as required.
3. In the Name column, click the link representing an object to view the Deployment Report for this object.
4. In the Available Reports section at the top right, click **Execution** to display the Execution Report for the object. This report contains a summary of how and when the object was used in a Map Run.
5. In the Execution Details section of the Execution report, select Execution Job Report to display the Execution Job Report that contains details of the mapping or process flow execution.

## Discovering the Default Deployment Time Settings of a Deployed Process

1. Open the Object Summary Report to see all deployed Processes.
2. In the Name column, click a process flow to display its Deployment Report.
3. In the Available Reports section, click **Start** to display the Job Start Report for the process.

   The execution parameters have the default deployment time settings.
4. In the Execution Parameters section, change any of the input parameter values, as required.
5. Click **Start Execution** to execute the new process run.

## Rerunning a Process

1. Open the Execution Schedule Report to see list of all Process Runs.
2. In the Execution Details section, click the mapping or process flow name in the Name column to display the Execution Report for the selected object.
3. In the Available Reports section, click **Start** to display the Job Start Report for the selected object.

   The execution parameters have the default deployment-time settings.
4. Change any of the input parameter values, as required.
5. Click **Start Execution** to execute a new Process Run.

## Monitoring a Process Run

1. Open the Execution Schedule Report to see the executing Process Runs.

2. If necessary, use the Execution Status filter to display only currently executing Process Runs.
3. Click **Refresh** as required, to follow the progress of the Process Runs.
4. In the Name column, click a process flow name to the display the Execution Report containing the details of a given Process Run.
5. Click **Refresh** as required, to follow the progress of the Process Run.
6. For Process Runs known to the Workflow system, click the **Related information** link to switch across to the Oracle Workflow Monitor and follow its progress in a graphical display. Use the browser's Back button to return to the current report page.

## Terminating a Process Run

1. Open the Execution Schedule Report to see the executing Process Runs.
2. In the Name column, click a process flow name to display the Execution Report for the process run.
3. Click **Stop** to terminate the given Process Run.
4. Click **Refresh** as required, to follow the progress of the Process Run as its execution is terminated.

## Removing the Execution Audit Details for a Process

1. Open the Execution Schedule Report to see the latest Process Runs.
2. Filter the displayed information by using an execution name.
3. Select all the executions that are to be removed, and click **Purge Selected Audit Details**.

   A confirmation screen is shown, requesting you to confirm the action.

## Removing Old Deployment Audit details

1. Open the Deployment Schedule Report to see the latest deployments.
2. Filter the displayed information by using the date range.
3. Select all the deployments that are to be removed, and click **Purge Selected Audit Details**.

   A confirmation screen is shown, requesting you to confirm the action.

## Viewing Error Tables Created as a Result of Data Auditor Execution

1. Grant privileges to the Repository Browser on the error table.

   See "Granting Privileges on Error Tables" on page 20-9.
2. Open the Execution Summary Report or Execution Schedule Report to see a list of all the Processes.
3. Click the link in the Related Information column corresponding to the process flow that contained the data auditor to display the Execution Job Report for the process flow.
4. Click the link in the Related Information column corresponding to the data auditor activity to display the Execution Job Report for the activity.

5. Data that violated the data auditor constraints is listed in the Logical Errors table. Click the link in the Error Table column to display the Execution Report for the error table.

If you have sufficient privileges, you can use the Execution Report for an error table to purge data from that error table.

## Unregistering a Location

1. Open the Locations Report to see the registered Locations.
2. Select the Location that is to be unregistered, and click **Unregister Selected Locations**.

   A confirmation screen is shown, requesting you to confirm the action.

## Updating Location Connection Details for a Changed Database Environment

1. Open the Locations Report to see the Locations.
2. Select the location that is to be validated by clicking the **Select** column to the right of that location. Click the link in the Validation column corresponding to the selected location.

   The Location Validation Report is displayed, showing the connection details of the Location and the Control Center
3. Change the service description values, as necessary, and click **Update Details**.
4. In the Location Status section, click **Get Status** to validate the current connection settings for the Location.

   Note that the results of Location connection tests are not maintained beyond the current session.

## Updating Service Node Details in a Changing Oracle RAC Environment

1. Open the Service Node Report to see the settings that currently describe the Oracle RAC system.
2. Update details and usage of the Service Nodes, then click **Update Node Details** for the requested changes to be made.
3. Add or remove Service Nodes, as required.
4. Click **Refresh** to see the current settings of the Oracle RAC system.
5. Set the Net Service Name by which the Control Center may be accessed, as necessary.

# 14

# Managing Metadata Dependencies

The Metadata Dependency Manager enables you to detect and resolve the impact of the changes made to the object definitions or the metadata in the workspace.

This chapter contains the following topics:


- About the Metadata Dependency Manager
- Opening an LIA Diagram
- Managing and Exploring Objects in an LIA Diagram
- Making Changes to Design Metadata Using Automatic Change Propagation


## About the Metadata Dependency Manager

The Metadata Dependency Manager provides the interface through which you can explore dependencies among data objects, as represented by the metadata in your Oracle Warehouse Builder repository. The Metadata Dependency Manager presents dependencies in the form of interactive lineage and impact diagrams. A lineage diagram traces the data flows for an object back to the sources and displays all objects along those paths. An impact diagram identifies all the objects that are derived from the selected object.

This type of information can help you in many circumstances such as the following:

- Starting from a target object, such as a dimension, cube, or business intelligence tool report, identify the columns in each data source that are used in computing the results in the target.
- Assess the impact of design changes in an object such as a source table or a pluggable mapping that is used throughout a project.
- Propagate a design change, such as a change to the data type of a source table column, to downstream objects in the design.

Using end-to-end data lineage and impact analysis reduces project risk by allowing better planning for design changes, faster identification of unanticipated impacts when source systems change, and enabling more effective auditing of your business intelligence results, master data or other data integration processes.

### Example: Lineage and Impact Analysis (LIA)

The metadata from sources such as files, databases, and applications can change even after the design and implementation of a data integration system. A change in the source metadata implies a corresponding impact in your Warehouse Builder implementation. Warehouse Builder enables you to reimport the modified source

definitions into the workspace. However, your original warehouse design may no longer remain valid with the reimported definitions, and you may need to make changes to the design and fix the inconsistencies.

You need to first find out how the warehouse design is affected by the changes in the source and then determine all the design objects that are dependent upon the sources. Next, you must synchronize the metadata so that all the affected design objects are updated to reflect the changes in the source. After this process is complete, you can redeploy the updated design to rebuild your data warehouse and synchronize the data.

For example, a company retrieves source data for a data warehouse from a flat file named `CUSTOMERS`. The `CUSTOMERS` flat file and a downstream staging table are referenced from multiple mappings in the ETL design, and ultimately loaded into a cube and an instance of Oracle Discoverer used for reporting. Records in `CUSTOMERS` include a numeric `CUSTOMER_ID` column, a `REGION` column which can take values `US`, `EMEA`, `APAC` or `OTHER`, Records from `CUSTOMERS` are also fed to a customer master database.

Over time, the following issues arise:

- The owners of the customer master data discover that a required column `GROUP_ID` sometimes contains a `NULL` value.
- The marketing department questions the breakdown of customers and sales by region as shown in reports generated by Oracle Discoverer.
- The `CUSTOMER_ID` column which previously only contained numeric values now may include letters and numbers.
- The existing `CUSTOMER_NAME` column, which was previously 50 characters, is expanded to 100 characters.
- A pluggable mapping referenced in several other mappings in the design is updated to reject rows with NULL values for the `GROUP_ID` column.

Changes to the definition of `CUSTOMERS` can potentially affect all of the objects downstream of that flat file. Errors in the customer master data could be originating in source data or could be caused by a bug in a ETL mapping. The questions about the Oracle Discoverer reports can be resolved if the origin of the region information for customers can be documented.

Without data lineage and impact analysis based on Warehouse Builder repository metadata, developers responsible for the changes must manually review the entire data integration design, including staging tables, ODS tables, dimensions, cubes, Discoverer objects and the master data. Design changes in the source data require manual update of any ETL mappings used to load those objects. Marketing's confidence in the BI reports depends upon the developers' thorough manual review of the design.

The metadata dependency management, data lineage and impact analysis features of Warehouse Builder simplifies these tasks:

- The Metadata Dependency Manager automates tracing the lineage of the bad columns in the customer master data.
- The Discoverer design can be validated because the origin of the region data used in the reports is documented and can be proven.
- ETL mappings and targets that may be affected by the addition of non-numeric values to the `CUSTOMER_ID` column can be identified automatically.

- Target objects and ETL mappings throughout the design that are affected by the change to the CUSTOMER_NAME column definition can be automatically identified and even updated.

## About Lineage and Impact Analysis and Metadata Dependency Diagrams

Metadata dependency diagrams show the relationships among objects managed by Warehouse Builder. The diagrams show both relationships of structure (for example, a primary key and foreign key relationship between columns in two tables) and data relationships (for example, the flow of data from the CUSTOMERS flat file to the CUSTOMERS_STAGE staging table).

Diagrams can be read to discover data lineage and impact analysis information. A diagram can be read from left to right to discover impact analysis information (that is, which objects are affected by a given object or column) or from right to left to discover data lineage information (that is, to identify the source of data in an output object).

For example, you might have a mapping that extracts data from a file and loads it into a table by way of an external table. This is the relationship:

```
flat_file > external_table > table
```

Figure 14–1 shows a lineage diagram of an external table named ADDRESS_EXT_TAB. ADDRESS_CSV is a flat file, and it is part of the lineage of ADDRESS_EXT_TAB. Thus, any change to ADDRESS_CSV will impact ADDRESS_EXT_TAB.

***Figure 14–1 Lineage Analysis Diagram for ADDRESS_EXT_TABLE***

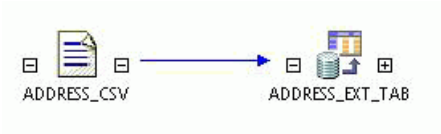
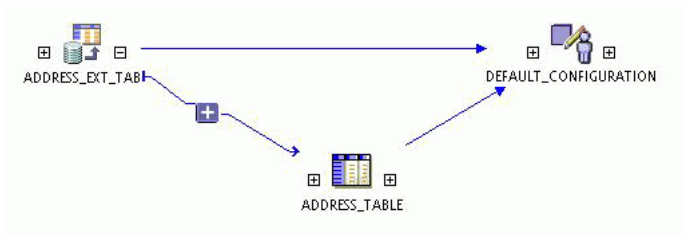



Figure 14–2 shows an impact diagram of ADDRESS_EXT_TAB, which includes the ADDRESS_TABLE. Any change to ADDRESS_EXT_TAB will impact ADDRESS_TABLE. ADDRESS_EXT_TAB is part of the lineage of ADDRESS_TABLE.

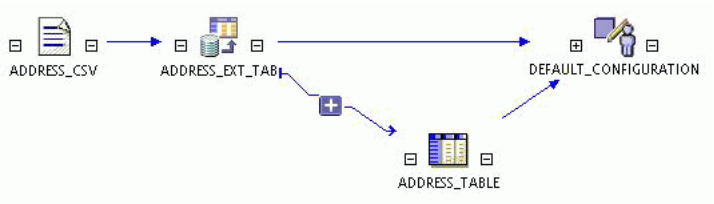
***Figure 14–2 Impact Analysis Diagram for ADDRESS_EXT_TABLE***




**Note:** The diagrams include the DEFAULT_CONFIGURATION object as well as the source and target data objects, because the selected configuration can affect how data is moved from sources to targets.

You can expand the diagram to include the lineage and the impact of an object by clicking the plus signs (+) on either side of the object icon in the diagram, as shown in Figure 14–3.

*Figure 14–3 Lineage and Impact Analysis Diagram*




# Opening an LIA Diagram

You can generate an LIA diagram from the Projects Navigator in the Design Center.

**To generate an LIA diagram from the Design Center:**

1. Expand the Projects Navigator until you see the object that you want to analyze.
2. Right-click the object and select **Lineage** or **Impact**.

   The Lineage or Impact tab is displayed, showing either the lineage of the object (one level of objects to the left of the selected object) or its impacts (one level of objects to the right of the selected object).

The Lineage and Impact commands are also available from the View menu.

# Managing and Exploring Objects in an LIA Diagram

Your initial selection of an object and a diagram type simply determine the initial starting point and the direction that the diagram branches from that object. You can modify an LIA diagram in the following ways:

- Drag-and-drop another object onto the diagram to view its dependencies along with the other objects.
- Click the plus (+) and minus (-) signs next to an object icon to expand or collapse a branch of the diagram.
- Remove the selected objects from the canvas. From the Graph menu, select **Hide Selected Object** to remove objects from the canvas. To restore the hidden objects, select Refresh from the Graph menu.
- Use the grouping tool to collapse a section of the diagram into a single icon, as described in "Using Groups in an LIA Diagram" on page 14-5.
- Double-click an object to display its attributes, as described in "Displaying an Object's Attributes" on page 14-6.
- Right-click an object to display the following menu options: Open Editor, Show Full Lineage, and Show Full Impact.

  The Open Editor option is described in "Making Changes to Design Metadata Using Automatic Change Propagation" on page 14-7.

  The Show Full Lineage and Show Full Impact options are described "Exploring Object Lineage and Impact in an LIA Diagram" on page 14-4.

## Exploring Object Lineage and Impact in an LIA Diagram

Use the following options in the Graph menu to explore the lineage and impact analysis information in LIA diagrams:

- **Show Full Impact**: Expands nodes to show all impacts in the diagram of the selected object.
- **Show Full Lineage**: Generates the full lineage diagram of the selected object.
- **Show Lineage**: Displays the next level of objects in the lineage diagram of the selected object.
- **Hide Lineage**: Hides the lineage of the selected object.
- **Show Impact**: Displays the next level of objects in the impact diagram of the selected object.
- **Hide Impact**: Hides all impacts of the selected object.

## Using Find to Search for Objects in an LIA Diagram

You can search for objects in the lineage and impact analysis diagram, as with other editors within Warehouse Builder. Searching forward follows the impact analysis from left to right, and searching backwards follows the data lineage from right to left. The diagram moves so that the current matching node is at the center of the diagram. You can also highlight all matches for a search in the diagram.

**To search within a Lineage and Impact Analysis (LIA) diagram:**

1. From the Search menu, select **Find**.

   The Find dialog box is displayed.
2. In the **Find** field, enter the name of the object you want to find.
3. To use additional options while searching for objects, click **Show Advanced**.

   The advanced search options are displayed in the Find dialog box.
4. (Optional) Specify advanced search options as described in the following sections:
   - "Specifying the Search Criteria" on page 5-48
   - "Match Options" on page 5-48
   - "Find Options" on page 5-49
   - "Scope" on page 5-49
   - "Direction" on page 5-49

   These sections discuss performing advanced search for mappings and pluggable mappings. However, the functionality and the processes are the same when you perform advanced search in an LIA diagram.
5. Click **OK**.

## Using Groups in an LIA Diagram

Groups enable you to organize the objects in a complex diagram so that they are easier to locate and edit. By reducing the number of objects in a diagram, you can more easily focus on the objects currently of interest.

**To create a group:**

1. Select a group of objects by dragging and dropping a box around them.
2. Click **Group Selected Objects** on the toolbar.

   The Group Selected Data Objects dialog box is displayed.

3. Enter a name for the group.

   The selected objects are collapsed into a single folder icon.

To display the individual objects in a group, double-click the folder icon. You can work on these objects in the same way as ungrouped objects.

To ungroup the objects, select the group and click **Ungroup Selected Object** on the toolbar.

### Managing Groups in an LIA Diagram

The following operations are available for working with groups in an LIA diagram.

- **Group Selected Objects**: Creates a group containing the selected objects on the canvas. A folder icon represents all objects in the group. Grouping enables you to reduce clutter on the canvas when there are many objects. Double-click the icon to display the individual objects in the group.
- **Ungroup Selected Objects**: Eliminates the selected group so that all objects are represented individually. Select the folder icon you want to ungroup and click **Ungroup Selected Folders**.
- **Group By Module**: Automatically groups all objects by module. A folder icon represents the module and all objects in the module. To group by module, select **Group By Module** from the Graph menu.

  Double-click the icon to display the individual objects in the group.
- **Ungroup Modules**: Eliminates the module groups so that all objects are represented individually. To ungroup modules, select **Ungroup Modules** from the Graph menu.

## Displaying an Object's Attributes

You can expand an object icon in a diagram so that you can examine its attributes. To expand an icon, double-click it. To reduce it to an icon, click the down arrow in the upper-right corner.

#### To generate an LIA diagram for an attribute:

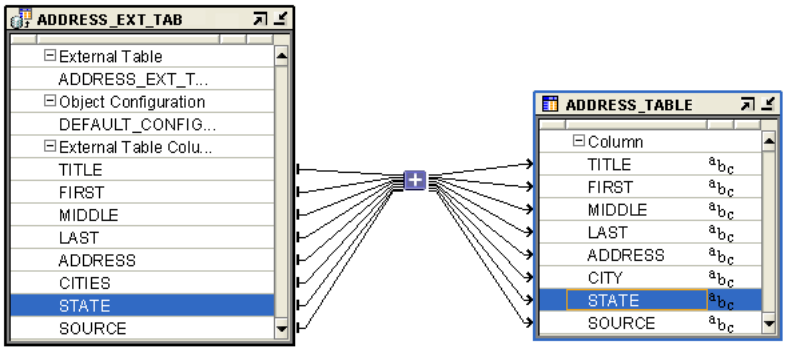
1. Generate an LIA diagram for an object.
2. Double-click the icons to display their attributes.
3. Right-click an attribute and select **Show Lineage** or **Show Impact**.

   The attributes along the lineage or impact path for the selected attribute are highlighted in a different color.

You can use this detailed information for auditing or when planning to propagate changes.

Figure 14–4 shows two expanded icons whose column attributes are connected by a mapping.

*Figure 14–4 Expanded Icons in an LIA Diagram*




## Exporting and Printing LIA Diagrams

LIA diagrams can be exported to SVG or JPEG formats, or printed using commands under the File menu.

- **Export Diagram**: Exports the active diagram to the local file system as an SVG or JPEG file. To export an LIA diagram, select **Export**, then **Diagram** from the File menu.
- **Print Options**: Provides Print Setup, Preview, and Print options for printing the diagram.

# Making Changes to Design Metadata Using Automatic Change Propagation

The LIA diagrams identify all of the objects that may be invalidated by a change to one or more objects. With this knowledge, you can examine the affected objects and modify them as necessary. Many changes can automatically be propagated to downstream objects in the Metadata Dependency Manager.

### To manually modify objects:

1. In the Dependency Manager, navigate to the first object to be changed. For example, navigate to a source table.
2. Right-click the object icon in a diagram and select **Open Editor**.

   Warehouse Builder opens the editing tool for the object. For example, if you selected a table, then Warehouse Builder opens the Data Object Editor.
3. Make the necessary changes in the editing tool and then save your changes.
4. Repeat these steps for all objects identified in the LIA diagram as needing change.

In the case that only a few objects are affected by a change, then you may prefer to modify the object manually. However, if many objects are affected, you can use automated change propagation to save time.

### To propagate metadata changes using Dependency Manager:

1. Double-click the object icon in a diagram.

   For example, double-click on the icon for a source table.
2. Right-click the metadata that you want to change, and select **Propagate Change**.

For example, right-click a column in the table.

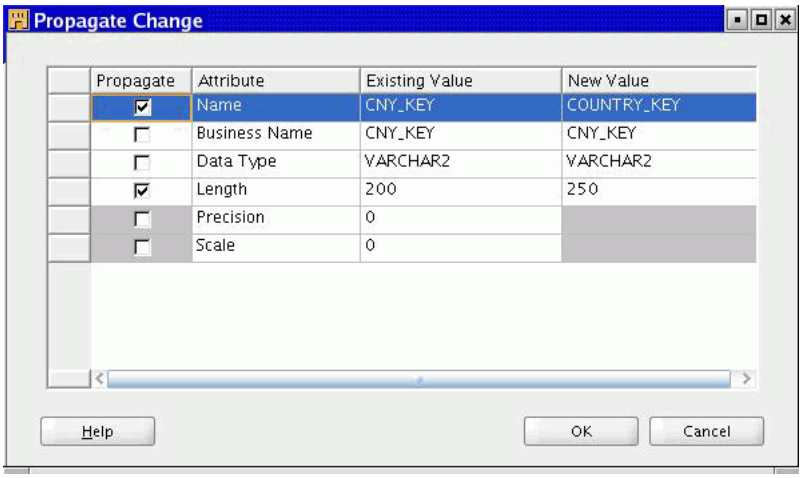
3. Change the attributes as described in "Automated Change Propagation in the Dependency Manager" on page 14-8.

## Automated Change Propagation in the Dependency Manager

In the Propagate Change dialog box, you specify metadata changes which Warehouse Builder propagates to all dependent objects, as indicated in the Lineage Impact Analysis diagram. This dialog box displays metadata details under the following columns: Propagate, Attribute, Existing Value, and New Value.

Figure 14–5 displays the Propagate change dialog box.

***Figure 14–5 Propagate Change Dialog Box***




The Attribute column lists the metadata attribute, such as name, business name, and data type. The Existing Value column lists the current value of the attribute and the New Value lists the changed value discovered by the Metadata Dependency Manager.

Select **Propagate** for each attribute that you want to change. For example, if you want to change the data type and length for a column, ensure that you select Propagate to the left of these attributes.

In the **New Value** field, you can enter the desired values for each attribute. Click **OK** after selecting the changes to propagate.

# 15

# Troubleshooting and Error Handling for ETL Designs

This chapter discusses troubleshooting ETL and describes the error logs in Oracle Warehouse Builder. It also discusses error handling techniques for ETL such as DML error logging.

This chapter contains the following topics:

- Inspecting Error Logs in Oracle Warehouse Builder
- Determining the Operators that Caused Errors in Mappings
- Using DML Error Logging
- Troubleshooting the ETL Process

## Inspecting Error Logs in Oracle Warehouse Builder

While working with Oracle Warehouse Builder, the designers need to access log files and check on different types of errors. This section outlines all the different types of error messages that are logged by Warehouse Builder and how to access them.

Warehouse Builder logs the following types of errors when you perform different operations:

- Troubleshooting Validation Errors on page 15-1
- Troubleshooting Generation Errors on page 15-2
- Troubleshooting Deployment and Execution Errors on page 15-3
- Troubleshooting Name and Address Server Errors on page 15-4

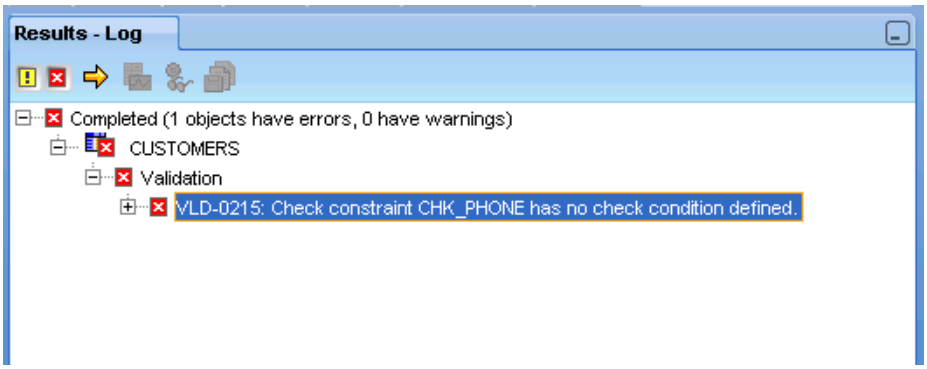
This section shows you how to retrieve error logs after performing different operations in Warehouse Builder.

### Troubleshooting Validation Errors

In Warehouse Builder, you can validate all objects by selecting the objects from the Projects Navigator and then selecting **Validate** from the File menu. After the validation is complete, the validation messages are displayed in the Log window.

Figure 15–1 displays the validation messages in a new tab of the Message Log window.

***Figure 15–1 Validation Error Messages***




You can also validate mappings from the Mapping Editor by selecting **Mapping,** then **Validate.** The validation messages and errors are displayed in the Validation Results window.

In the validation results, expand the node displaying the object name and then the Validation node. The validation errors, if any are displayed. Double-click a validation message to display the detailed error message in a message editor window.

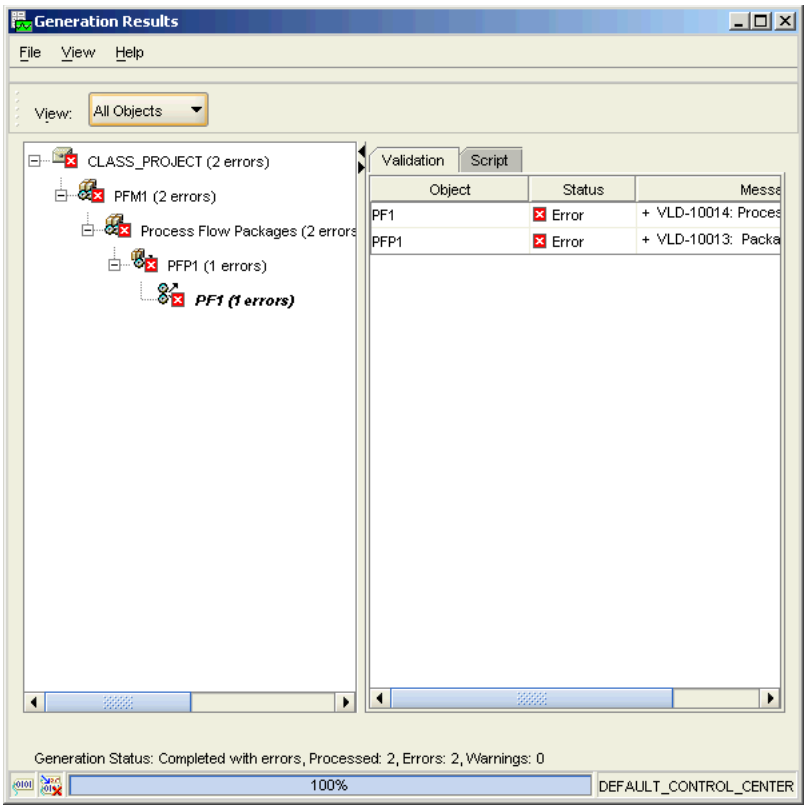
Warehouse Builder saves the last validation messages for each previously validated object. You can access these messages at any time by selecting the object from the console tree in the Projects Navigator, selecting **View** from the menu bar, and then clicking **Validation Messages.** The messages are displayed in the Validation Results window.

## Troubleshooting Generation Errors

After you generate scripts for Warehouse Builder objects, the Log window displays the generation results and errors. Double-click an error message in the Log window to display a message editor that enables you to save the errors to your local system.

Figure 15–2 displays the Generation Results window.

*Figure 15–2 Generation Results Window*




## Troubleshooting Deployment and Execution Errors

You can store execution or deployment error and warning message logs on your local system by specifying a location for them. In the Projects Navigator, select the project. Then from the Tools menu, select **Preferences**. In the object tree on the left of the Preferences dialog box, expand the OWB node, and click the Logging option. Use the properties listed on the right to set the log file path, file name, and maximum file size. You can also select the types of logs that you want to store.

You can view this log of deployment and error messages from the Warehouse Builder console by selecting **View** from the menu bar, and then **Log.** This Message Log panel is read-only.

### Runtime Audit Browser

If an error occurs while transforming or loading data, the audit routines report the errors into the runtime tables. You can easily access these error reports using the Runtime Audit Browser. The Runtime Audit Browser provides detailed information about past deployments and executions. These reports are generated from data stored in the runtime repositories. Click the Execution tab in the Execution reports to view error messages and audit details.

### Determining the Operators that Caused Errors in Mappings

When you encounter errors while deploying mappings, use the line number provided in the error message to determine where the error occurred. The generated code contains comments for each operator in the mapping. This enables you to determine which operator in the mapping caused the error.

The following example displays the code generated, in set-based mode, for a Filter operator. Notice that the comments enclosed between /* and */ list the operator for which a particular part of the statement is executed.

```
INSERT INTO "FLTR_TGT"
      ("CHANNEL_ID", "CHANNEL_DESC")
      (SELECT
/*+ NO_MERGE */
/* CHANNELS.INOUTGRP1, FILTER.INOUTGRP1 */
  "CHANNELS"."CHANNEL_ID" "CHANNEL_ID",
  "CHANNELS"."CHANNEL_DESC" "CHANNEL_DESC"
FROM
  "SH"."CHANNELS"@"ORA11@SH_SRC_LOCATION"  "CHANNELS"
  WHERE
  ( "CHANNELS"."CHANNEL_ID"  < 5/* OPERATOR FILTER: FILTER CONDITION */ )
      )
    ;
```

### Troubleshooting Name and Address Server Errors

If you are using the Name and Address cleansing service provided by Warehouse Builder, you may encounter related errors.

Name and Address server start up and execution errors can be located at:

*OWB_ORACLE_HOME*/owb/bin/admin/NASvr.log

If your Name and Address server is enabled in:

*OWB_ORACLE_HOME*/owb/bin/admin/NameAddr.properties:TraceLevel=1,

then it produces the log file NASvrTrace.log in the same directory.

## Using DML Error Logging

Error logging enables the processing of DML statements to continue despite errors during the statement execution. The details of the error such as the error code, and the associated error message, are stored in an error table. After the DML operation completes, you can check the error table to correct rows with errors. DML error logging is supported for SQL statements such as INSERT, UPDATE, MERGE, and multitable insert. It is useful in long-running, bulk DML statements.

Warehouse Builder provides error logging for the following data objects used in set-based PL/SQL mappings: tables, views, materialized views, dimensions, and cubes. DML error logging is supported only for target schemas created on Oracle Database 10*g* Release 2 or later versions.

### About DML Error Tables

DML error tables store details of errors encountered while performing DML operations using a mapping. You can define error tables for tables, views, and materialized views only.

Use the **DML Error Table Name** property to log DML errors for a particular data object. In the mapping that uses the data object as a target, set the DML Error Table Name property of the operator that is bound to the target object to the name of the DML error table that will store DML errors.

You can create your own tables to store DML errors or allow Warehouse Builder to generate the DML error table. While deploying a mapping in which the DML Error

Table Name property is set for target operators, if a table with the name specified by the DML Error Table property does not already exist in the target schema, it is created.

When DML error tables are created along with the mapping, dropping the mapping causes the DML error tables to be dropped, too.

In addition to the source target object columns, DML error tables contain the columns listed in Table 15–1. If you use your own tables to log DML errors, then ensure that your table contains these columns.

***Table 15–1 DML Error Columns in Error Tables***

| Column Name | Description |
|---|---|
| `ORA_ERR_NUMBER$` | Oracle error number |
| `ORA_ERR_MESG$` | Oracle error message text |
| `ORA_ERR_ROWID$` | ROWID of the row in the error (for update and delete) |
| `ORA_ERR_OPTYPE$` | Type of operation: insert (I), update (U), delete (D) |
| `ORA_ERR_TAG$` | Step or detail audit ID from the runtime audit data. This is the `STEP_ID` column in the runtime view `ALL_RT_AUDIT_STEP_RUNS`. |

## Enabling DML Error Logging

DML error logging is generated for set-based PL/SQL mappings if the following conditions are satisfied:

- In the mapping that loads the table, view, materialized view, dimension, or cube, the **DML Error Table Name** property is set for the operator representing the target object.
- The PL/SQL Generation Mode configuration parameter of the module that contains the mapping is set to Oracle 10gR2, Oracle 11gR1, Oracle 11gR2, or Default.

  If the value is set to Default, ensure that location associated with this module has the Version property set to 10.2, 11.1, or 11.2.

### DML Error Logging and ETL

The execution of mappings that contain data objects for which DML error logging is enabled fails if any of the following conditions occur:

- The number of errors generated exceeds the specified maximum number of errors for the mapping.

  The default set for this value is 50. You can modify this value by setting the Maximum number of errors configuration parameter of the mapping. In the Projects Navigator, right-click the mapping and select **Configure**. In the Maximum number of errors configuration parameter, specify the number of errors that can generated before the mapping execution is terminated.
- Errors occur due to functionality that is not supported.

  See "DML Error Logging Limitations" on page 15-6.

You can truncate the DML error table and delete error details generated during a previous load. This helps in housekeeping of the error tables. To truncate an error table before the map is executed, select the Truncate Error Table property of the operator bound to the data object that has DML error logging enabled.

The properties Roll up Errors and Select only errors from this property are not used for DML error logging.

### DML Error Logging Limitations

- DML error logging is not supported for nonscalar data types.
- Each DML statement has specific limitations, which are listed in the documentation related to that statement.

  **See Also:** *Oracle Database SQL Language Reference* for limitations on DML error logging for each DML statement

- If you have a DML error table defined for a data object, you cannot upgrade the data object using the Upgrade option in the Control Center Manager.
- Depending on your error logging needs, you can configure the Table operator in a mapping to use the APPEND or NOAPPEND hint. For example, direct-path insert does not support error logging for unique key violations. To log unique key violations, use the NOAPPEND hint.

## Troubleshooting the ETL Process

This section contains troubleshooting tips for errors that you may encounter while performing ETL.

### ORA-04063 While Running Hybrid Maps

While executing a hybrid mapping that contains a PL/SQL Oracle Target CT, you may encounter the following error:

```
ORA-04063: package body "DEMO.ORACLE_SQL_POWER_MTI" has errors
```

This indicates that there are compilation errors in `ORACLE_SQL_POWER_MTI`, the PL/SQL package generated to implement the mapping.

To determine the cause of this error, start SQL*Plus, connect as the target user, and execute the following commands:

```
ALTER PACKAGE ORACLE_SQL_POWER_MTI COMPILE BODY;
SHOW ERRORS;
```

For example, a table not found error may occur if permissions on source tables or views are not granted to the target user. Resolve any errors, recompile the package, and then execute the hybrid mapping.

### Agent Log Files

The agent log files enable you to debug deployment and execution errors in Code Template mappings. On Windows, the agent logs are displayed in the Design Center console.

The following are the agent logs on UNIX:

- The `jrt.log` file located in *`OWB_ORACLE_HOME`*`/owb/bin/admin` contains output from the Agent process and audit setup errors.

- The Code Template mapping execution logs are stored in the *OWB_ORACLE_HOME*/owb/jrt/log/owb folder. Each job execution is represented by a separate directory that contains XML log files with the audit trail of the job execution.

## Error Starting the Control Center Agent (CCA)

While starting the Control Center Agent, you may encounter the following error:

```
Error initializing server: Application: system is in failed state as
initialization failed.
```

To resolve this error, delete all the subdirectories in the folder *OWB_ORACLE_HOME*/owb/jrt/applications and then start the Control Center Agent.

## Error Executing Web Services from the Secure Web Site

Sometimes you may encounter the following error when you execute a Web service from a secure Web site:

SSL Error: unable to find valid certification path to requested target.

Use the following steps to overcome this error.

1. Export the certificate used for the SSL channel from the OC4J server side. This is in the *$J2EE_HOME*/config directory.

   *$JAVA_HOME*/bin/keytool -export -storepass welcome -file server.cer -keystore OWB_ORACLE_HOME.owb/jrt/config/serverkeystore.jks

   server.cer is the file to which the certificate is exported, serverkeystore.jks is the key store used in the OC4J server embedded in Warehouse Builder. If you use an OC4J instance other than the one embedded in Warehouse Builder, serverkeystore.jks is the key store file you created when you setup the SSL with OC4J.

2. Copy the exported server.cer from step 1 to the OC4J server side $JAVA_HOME/jre/lib/security directory.

3. Import the certificate to the java trusted certification store at the OC4J server side.

   $JAVA_HOME/bin/keytool -import -v -trustcacerts -file

   $JAVA_HOME/jre/lib/security/server.cer -keystore

   $JAVA_HOME/jre/lib/security/cacerts

   where cacerts is the file used to store the trusted certificates, server.cer is the file copied from step 2.

   You will be prompted for the cacerts password.

## REP-01012 While Deploying Mappings to a Target Schema

When you deploy mappings to a target schema, you may encounter the following error:

```
REP-01012: Cannot deploy PL/SQL maps to the target schema because it is not owned
by the Control Center
```

### Cause

This error occurs when you attempt to deploy mappings to a target schema into which objects were previously deployed using a different Control Center.

A target schema can be associated with only one Control Center. Audit data regarding deployments to this target schema is written to audit tables in the Control Center repository. The Control Center creates various objects (primarily synonyms) in the target schema that provide information about the Control Center to which audit data should be written. When you attempt to use a different Control Center to deploy a mapping to the same target schema, you encounter the REP-01012 error.

> **Note:** You can use a local Control Center to deploy data objects such as tables, view, dimensions, and so on to a remote target. This is because Warehouse Builder does not maintain auditing information for these objects.

However, Warehouse Builder allows you to deploy mappings to a target schema.

Typically, when you use a Control Center for deployments, you deploy mappings to a target schema by using the Control Center installed on the database containing the target schema. You can deploy mappings to a target schema by using a Control Center installed on a different database than the one that contains the target schema. In the Locations Navigator, create a Control Center that points to the Control Center installed on the remote host containing the target schema. Thus you can deploy mappings to the remote control center.

#### Solution

1. Drop the mappings that have been deployed to the target schema using the original Control Center.
2. Unregister the location from the original Control Center.
3. Delete the synonyms that provide the association between the target schema and its control center from the target schema.
4. Register the location using the new Control Center.
5. Deploy mappings using the new Control Center.

## Unable to Delete a Location

Before you delete a location, do the following:

- Reconfigure any modules that use the location to use a different location

  Edit the modules and remove the location from the set of possible data locations for the module.
- Unregister the location

Log in as the `OWBSYS` user and execute the following query to determine if the location is still associated with a Control Center.

```
SELECT s.name owner, r.name referenced, c.name connector,
       c.REGISTERED, c.STRONGTYPENAME
FROM cmplogicalconnector_v c, cmplocation_v s, cmplocation_v r
WHERE c.owninglocation = s.elementid AND c.referencedlocation = r.elementid;
```

# 16

# Creating and Consuming Web Services in Warehouse Builder

Web services are the basis of the widely used Service-Oriented Architecture (SOA) approach to integrating enterprise applications. They enable easy access to remote content and application functionality using industry-standard mechanisms, without any dependency on the provider's platform, the location, or the service implementation.

Oracle Warehouse Builder-based solutions can participate fully in SOA-based architectures. You can publish certain Warehouse Builder objects as Web services, thus allowing other developers to use industry standards to leverage functionalities defined in these objects.

This chapter contains the following topics:


- Introduction to Web Services
- Publishing Warehouse Builder Objects as Web Services
- Creating Web Services Based on a URL
- Executing Web Services
- Using Web Services as Activities in Process Flows
- Using Web Services in Mappings
- Using Secure Sockets Layer (SSL) to Access Web Services Securely
- Case Study: Using Web Services for Data Integration


## Introduction to Web Services

A Web service is a software system designed to provide a standard, vendor-neutral method of accessing computing resources or services over a network. It uses open, XML-based standards and transport protocols to exchange data with calling clients.

A Web service generally consists of:

- a published interface definition for the Web service, that describes messages that clients can send to the service and responses the Web service will return to the caller.
- an implementation that provides the functionality exposed through the interface.

The caller of a Web service does not depend upon underlying implementation details such as the choice of programming language, application server technology or

database. The published interface describes all information required for the caller to consume the Web service.

Because implementation differences are not exposed to the caller, Web services enable easy integration of software components from different technology vendors. The approach of building solutions based on Web services is commonly called Service-Oriented Architecture (SOA).

While there are several common approaches to building Web services, in enterprise computing, Web services generally interact using the XML-based standards Simple Object Access Protocol (SOAP) and Web Services Description Language (WSDL).

### Simple Object Access Protocol (SOAP)

**Simple Object Access Protocol (SOAP)** is a protocol for exchanging XML-based messages over a computer network, normally using HTTP. SOAP forms the foundation layer of the Web services stack, providing a basic messaging framework that more abstract layers can build on. It is used to send Web service requests and to receive Web service responses.

### Web Services Description Language (WSDL)

The **Web Services Description Language (WSDL)** is an XML-based service description of how to communicate using the Web service. It includes the following information:

- Purpose and location (on a remote computer) of the Web service
- Operations that you can perform on the Web service
- Input parameters and return values for each operation
- Protocol bindings and message formats required to interact with the Web services

The supported operations and messages are described abstractly, and then bound to a concrete network protocol and message format.

## Advantages of Web Services

Web services provide the following advantages:

- Support Service Oriented Architecture (SOA)
- Enable sharing of application functionality between application developers
- Enable you to build services that invoke data integration processes
- Enable you to build applications faster because you can reuse application logic created by others

## About Web Services in Oracle Warehouse Builder

Warehouse Builder supports Web services integration using the SOAP and WSDL standards, and thus can be fully integrated into SOA-based enterprise architectures. Developers experienced with Warehouse Builder can create and leverage Web services-based solutions in their data integration designs using their existing tools, code, and skill set. For example, you can integrate your ETL design into larger solutions based on products such as Oracle BPEL Process Manager.

> **See Also:** *Oracle Warehouse Builder Sources and Targets Guide* for more information about Web services concepts

Warehouse Builder supports the following Web service-related functionality:

- Publishing Warehouse Builder ETL jobs as SOAP-based Web services, which can then be invoked or consumed by other systems.

  See "About Publishing Web Services" on page 16-3.

- Calling Web services that expose functionality created outside of your Warehouse Builder ETL design.

  See "About Consuming Web Services" on page 16-3.

These two areas of functionality enable Warehouse Builder-based designs to participate fully in SOA-based solutions.

### About Defining Web Services

Web services are defined within an application server module in the Design Center. An Application Server module is associated with the location to which the Web services are deployed. It contains Web services and Web service packages. Web service packages are primarily used to group related Web services and contain a set of Web services.

Use the Application Servers node in the Projects Navigator to define Web services based on existing Warehouse Builder objects. Use the Public Application Servers node in the Globals Navigator to define public Web services.

### About Publishing Web Services

The process of making ETL processes designed using Warehouse Builder available to other application developers in the form of Web services is referred to as publishing Web services. To publish a Web service, you must create a WSDL file that contains information about your Web service and make this WSDL file available for remote access.

When you use Warehouse Builder to publish Web services, you only need to select the object whose functionality you want to publish as a Web service. The code generator generates the required WSDL file.

> **See Also:** "Publishing Warehouse Builder Objects as Web Services" on page 16-4

### About Consuming Web Services

The process of using Web services that are made available remotely by other application developers in your ETL designs is called consuming Web services. To consume a Web service, you must know the location of the WSDL file of the Web service. You can then make a request to the Web service to perform the required task.

Before you consume remote Web services in your ETL designs, you must import the Web service into Warehouse Builder.

> **See Also:**
>
> - "Using Web Services in Mappings" on page 16-17
> - "Using Web Services as Activities in Process Flows" on page 16-16

## About Public Web Services

Public Web services are accessible across the workspace in which they are defined and are not limited to a particular project. Public Web services are created under the Public Application Servers node of the Globals Navigator.

Public Web services can only be based on URLs, not on Warehouse Builder objects. Creating public Web services enables you to create Web services based on existing WSDL files.

> **See Also:** "Creating Web Services Based on a URL" on page 16-10 for more information about creating public Web services

# Publishing Warehouse Builder Objects as Web Services

You can make ETL processes defined using certain Warehouse Builder objects available to other application developers by publishing these objects as Web services. After you publish a Warehouse Builder object as a Web service, other developers can remotely access the Web service and use the functionality defined in this object. Because Web services use open, industry-standard mechanisms, the developers need not install Oracle Warehouse Builder or be familiar with how it works.

You can create Web services based on the following Warehouse Builder objects:

- Mappings, including Code Template (CT) mappings
- Process flows
- Transformations
- Data auditors
- Table or module for Change Data Capture

### Methods of Creating Web Services Based on Warehouse Builder Objects

Use one of the following methods to create Web services based on Warehouse Builder objects.

- Publish Warehouse Builder objects as Web services.

  See "Steps to Publish Warehouse Builder Objects as Web Services" on page 16-5.

- Define a Web service using the Create Web Service Wizard. Then generate and deploy the Web service.

  See "Steps to Create Web Services Based on Warehouse Builder Objects" on page 16-5 for information about creating Web services.

You can also publish Web services that are based on URLs as described in "Steps to Publish Web Services Based on a URL" on page 16-5.

---

> **Note:** During the lifetime of the data warehouse, the definitions of the object on which a Web service is based can change. To propagate these changes to the Web service, redeploy the Web service using the steps described in "Deploying Web Services" on page 16-9.
>
> When a Web service that is used in a process flow is modified and redeployed, ensure that you synchronize the Web service as described in "Synchronizing Web Service Activities with Their Referenced Web Services" on page 16-17.

---

### Supported Versions for Web Services

Table 16–1 lists the versions of standards and products supported by Warehouse Builder.

***Table 16–1 Supported Versions of Standards and Products for Web Services***

| Standard or Product | Version |
|---|---|
| WSDL | 1.1 |
| SOAP | 1.1, 1.2 |
| OC4J standalone | 10*g* and later |
| Oracle Application Server | 10*g* and later |

### Steps to Publish Warehouse Builder Objects as Web Services

You can quickly publish a Warehouse Builder object as a Web service from the Projects Navigator. Right-click the object (such as mapping, process flow, data auditor, or transformation) that you want to publish as a Web service and select **Publish as Web Service**. The Select Application Server or Web Service Package dialog box is displayed containing the existing application server modules and Web service packages. Select the application server module or Web service package to which the object should be published as a Web service. Warehouse Builder creates the Web service based on the selected object under the application server module or Web service package, and then publishes the Web service (deploys the Web service to the application server).

> **See Also:** "Example: Publishing Mappings as Web Services" on page 16-21 for an example of publishing Warehouse Builder objects as Web services

### Steps to Create Web Services Based on Warehouse Builder Objects

Use the following steps to publish Web services that are based on Warehouse Builder objects.

1. If you have not already done so, in the Projects Navigator, create an Application Server module and its associated location. An Application Server module is a container for a set of Web services and Web service packages.

   > **See Also:** *Oracle Warehouse Builder Sources and Targets Guide* for more information about creating Application Server modules

2. Create a Web service as described in "Creating Web Services Based on Warehouse Builder Objects" on page 16-6.
3. Validate the Web service as described in "Validating Web Services" on page 16-8.
4. Generate the Web service as described in "Generating Web Services" on page 16-8.
5. Deploy the Web service as described in "Deploying Web Services" on page 16-9.

You can use the functionality defined in the Web service by executing the Web service as described in "Executing Web Services" on page 16-11.

### Steps to Publish Web Services Based on a URL

Use the following steps to publish Web services that are based on a URL.

1. If you have not already done so, in the Globals Navigator, create a public application server module, under the Public Application Servers node, and its

associated location. A public application server module is a container for a set of Web services.

> **See Also:** *Oracle Warehouse Builder Sources and Targets Guide* for more information about creating public application server modules

2. Create a Web service as described in "Creating Web Services Based on a URL" on page 16-10.
3. Validate the Web service as described in "Validating Web Services" on page 16-8.
4. Generate the Web service as described in "Generating Web Services" on page 16-8.
5. Deploy the Web service as described in "Deploying Web Services" on page 16-9.

## Creating Web Service Packages

A Web service package is a container for a set of Web services. Use Web service packages to group a set of related Web services. A Web service package does not have any location associated with it and uses the same location details of the Application Server module that contains it.

You can create Web service packages only in the Projects Navigator and not in the Globals Navigator.

**To create a Web service package:**

1. Expand the project node under which you want to create a Web service package.
2. If you have not already done so, create an application server module to contain the Web service package.

   > **See Also:** *Oracle Warehouse Builder Sources and Targets Guide* for more information about creating application server modules

3. Expand the application server node under which you want to create the Web service package, right-click Web Service Packages, and then select **New Web Service Package**.

   The Create Web Service Package dialog box is displayed.
4. Enter the following details in the Create Web Service Package dialog box.
   - **Name:** The name of the Web service package
   - **Description:** An optional description for the Web service package

## Creating Web Services Based on Warehouse Builder Objects

Use the Projects Navigator to create Web services based on Warehouse Builder objects.

**To create a Web service based on a Warehouse Builder object:**

1. In the Projects Navigator, expand the project node and then the application server node under which you want to create a Web service.
2. Right-click the Web Services node and select **New Web Service**.

   To create a Web service under a Web service package, in the Projects Navigator, right-click the Web service package and select **New Web Service**.

   The Create Web Service Wizard is displayed.
3. On the Welcome page of the wizard, click **Next**.

4. On the Name and Description page, provide details as described in "Naming the Web Service" on page 16-7 and click **Next**.
5. On the Implementation page, provide details as described in "Defining the Web Service Implementation" on page 16-7 and click **Next**.
6. On the Review Specification page, review the details that you entered in the wizard. To modify any values, click **Back**. To complete the definition of the Web service, click **Finish**.

   Click **View Source** to view the WSDL code that will be generated by Warehouse Builder to implement this Web service.

The Web service is created and added to the navigator tree.

Alternatively, you can quickly create a Web service by right-clicking the object based on which you want to create a Web service and selecting **Create Web Service**. The Select Application Server or Web Service Package dialog box is displayed. Select the and click **OK**. The Web service is created and added under the application server node you selected.

### Example: WSDL File for a Web Service

The following is an example of a WSDL file created for a Web service.

```
<definitions
     name="HttpSoap11"
     targetNamespace="http://dbWebService.packaging.sdk.jrt.wh.oracle/"
     xmlns="http://schemas.xmlsoap.org/wsdl/"
     xmlns:tns="http://dbWebService.packaging.sdk.jrt.wh.oracle/"
     xmlns:mime="http://http://schemas.xmlsoap.org/wsdl/mime/"
     xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    >
    <types>
    </types>
    <message name="FUNCTION_TARGETInput">
        <part name="P1" type="xsd:string"/>
        <part name="P2" type="xsd:string"/>
    </message>
...
...
...
```

## Naming the Web Service

Use the Name and Description page to describe the Web service. Provide the following information on this page:

**Name:** The name of the Web service. The name should conform to the Warehouse Builder naming standards.

**Description:** An optional description for the Web service.

## Defining the Web Service Implementation

Use the Implementation page to select the Warehouse Builder object on which the Web service should be based.

Select one of the following options to create a Web service:

- Create from a mapping
- Create from a process flow
- Create from a transformation
- Create from a data auditor
- Create from a table or module used for Change Data Capture

Based on the option that you select, the text area below the options displays the available objects on which you can base the Web service. Select the Warehouse Builder object from this text area.

When you create a Web service based on a CT mapping, if the agent associated with the Web service is different from the agent associated with the Code Template mapping, a warning is displayed during validation. You can still generate, deploy, and execute such a Web service. However, in some cases, the execution may fail.

While creating a Web service based on a CT mapping, it is recommended that the Web service use the same agent as the Code Template mapping.

## Validating Web Services

When you validate a Web service, Warehouse Builder verifies the metadata definitions and configuration parameters to ensure that they are valid according to the rules defined by Warehouse Builder. When you validate a Web service, the WSDL file associated with the Web service is validated against the W3C WSDL schema. Successful validation ensures that code can be generated to deploy the Web service.

To validate a Web service, select the Web service in the Design Center and click **Validate** on the toolbar. Or, right-click the Web service in the Design Center, and select **Validate**.

You can also validate an application server module or a Web service package. This validates all the Web services contained in the application server module or Web service package.

## Generating Web Services

Generating Web services creates the code required to deploy the Web service to the associated OC4J or Oracle Application Server location. When you generate a Web service, Warehouse Builder creates a corresponding WSDL file for each Web service. For Web services based on Warehouse Builder objects, an `.ear` file is also generated. For Web service packages, one `.ear` file is generated for each Web service.

The generated files are stored in a default location on the file system on which the Design client is installed. You cannot view or edit these generated files.

To generate a Web service, select the Web service and click **Generate** on the toolbar. Or right-click the Web service and select **Generate**. You can also generate an application server module or a Web service package, to generate code concurrently for all the Web services contained in the application server module or Web service package at once.

---

**Note:** You cannot generate a public Web service. However, you can validate a public Web service.

---

## Deploying Web Services

Deploying Web services uses the scripts created during generation to create the Web service in the agent location associated with the application server module containing the Web service.

When you deploy a Web service, the `.ear` file corresponding to this Web service is located in the *OWB_ORACLE_HOME*`/owb/jrt/applications` directory. This directory also contains a separate folder for each Web service deployed to the OC4J instance that is embedded in Warehouse Builder.

To deploy Web services, you need a Control Center Agent (CCA) and an OC4J instance.

You can deploy Web services to:

- OC4J standalone instance

  When you install Warehouse Builder, an OC4J instance is installed with it. You can deploy Web services to this OC4J instance or to an OC4J instance that is part of an Oracle Application Server instance.

- Oracle Application Server

  You can deploy Web services to any Oracle Application Server. Before you do so, create a URI location that points to this Oracle Application Server instance. For more information about creating a location related to an Oracle Application Server instance, see *Oracle Warehouse Builder Sources and Targets Guide*.

You can deploy Web services either from the Design Center or from the Control Center Manager.

**Note:** You cannot deploy a public Web service.

### Prerequisites for Deploying Web Services

Before you deploy a Web service based on a Warehouse Builder object, ensure that you:

- Deploy the Warehouse Builder objects on which the Web service is based
- Start the Control Center Agent

**See Also:** "Starting the Control Center Agent (CCA)" on page 7-23 for information about starting the Control Center Agent

### Deploying Web Services Using the Control Center Manager

From the Design Center, open the Control Center by selecting **Control Center Manager** from the Tools menu. In the Control Center Manager, expand the node that represents the location that contains the Web service. Select the Web service, set the Default Actions to Create, and click **Deploy** on the toolbar.

### Deploying Web Services Using the Design Center

From the Projects Navigator, right-click the Web service and select **Deploy**. Or select the Web service and click **Deploy** on the toolbar.

# Creating Web Services Based on a URL

Warehouse Builder allows you to create public Web services from a URL. Public Web services are ones that you define under the Public Application Servers node of the Globals Navigator. You use public Web services primarily to leverage existing functionality, available as Web services, in ETL objects. Before you use existing Web services (local or remote) in your ETL designs, you must create a public Web service in Warehouse Builder that represents the functionality in the Web service.

To create a Web service from a URL, you need a URL pointing to the valid WSDL file. The URL can point to a WSDL file on the local file system or in a remote location. If the WSDL file is on a remote computer, then you must specify the proxy settings used to access the remote location.

### Proxy Settings for Creating Web Services Based on External URLs

When you create a Web service based on an external URL, you must specify the proxy settings that should be used to access the external URL. Use the following steps to set the proxy:

1. In the Design Center, select **Preferences** from the Tools menu.

   The Preferences dialog box is displayed.

2. In the navigator tree on the left, select **Web Browser and Proxy**.

   The Web Browser and Proxy preferences are displayed on the right of the Preferences dialog box.

3. Select **Use HTTP Proxy Server** and specify values for the following fields:

   - **Host Name:** Represents the name of the proxy server
   - **Port Number:** Represents the port number of the proxy server
   - **Exceptions:** Represents the addresses for which the proxy server is bypassed. Use an asterisk (*) as a wildcard and separate multiple entries using a vertical bar (|).

4. If your proxy server requires authentication, select **Proxy Server Requires Authentication**. Enter the credentials using the User Name and Password fields.

5. Click **Test Proxy** to test your proxy settings.

### Steps to Create a Web Services Based on a URL

Use the following steps to create a Web Service based on a URL.

1. In the Globals Navigator of the Design Center, expand the application server node under which you want to create a Web service.

2. Right-click the application server node and select **New Web Service**.

   The Create Web Service Wizard is displayed.

3. On the Welcome page of the wizard, click **Next**.

4. On the Name and Description page, provide details as described in "Naming and Describing a Public Web Service" on page 16-11 and click **Next**.

5. On the Review Specification page, review the details that you entered in the wizard. To modify any values, click **Back**. To complete the definition of the Web service, click **Finish**.

   Click **View Source** to view the WSDL code used to implement this Web service.

The Web service is created and added to the navigator tree.

## Naming and Describing a Public Web Service

Use the Name and Description page to specify the location of the WSDL file that will act as a basis for the Web service. This page contains the following fields:

- **Name:** Represents the name of the Web service. The name is derived automatically from the WSDL file and you can edit it, if required.
- **WSDL File Location:** Click **Browse** to specify the path of the WSDL file. The URL address can be the path of a local file or the URL address of a remote accessible across a network. The WSDL file contains the definitions of the existing Web service.

  This property is displayed only when you use the Globals Navigator to create a Web service based on a URL.
- **Description:** An optional description for the Web service.

# Executing Web Services

Executing a Web service enables you to run the functionality defined in the Web service. You can execute a Web service either from the Control Center Manager or from a Web browser.

### Prerequisites for Executing Web Services

To execute Web services, you must use a J2EE user who is granted the `OWB_J2EE_EXECUTOR` role. When you use the OC4J server embedded within Warehouse Builder, this role is already created for you.

---

**Note:** For Web services that contain CT mappings, if the Web service execution is likely to take more than a day, it is recommended that you split the job into smaller ones. The default transaction timeout for the OC4J is set to one day. If your job execution takes more than a day, then the execution will time out and unexpected errors may be encountered.

---

Use one of the following methods to assign this role to a J2EE user:

- Manage J2EE User Accounts option in the Repository Assistant
- J2EE User Management option under the Tools menu of the Design Center

**See Also:**

- "Using Secure Sockets Layer (SSL) to Access Web Services Securely" on page 16-19 for more information about J2EE roles
- *Oracle Warehouse Builder Installation and Administration Guide for Windows and UNIX* for more information about managing J2EE users

## Using the Control Center Manager to Execute Web Services

**To execute a Web service using the Control Center Manager:**

1. In the Control Center Manager, expand the location node that contains the Web service, select the Web service and click **Start** on the toolbar. Or right-click the Web service and select **Start**.

   The Select Operations dialog box is displayed. The Operations list contains the list of operations that you can perform using the Web service.

   Note that if your Web service contains only one operation, the Select Operations dialog box is not displayed.

2. In the Operations list, select the operation that you want to perform and click **OK**.

   The Input Parameters dialog box that is used to provide the execution parameters for the Web service is displayed.

3. In the Input Parameters dialog box, enter values for the displayed parameters.

   The parameters listed depend on the type of object on which the Web service is based. For example, Web services based on PL/SQL mappings, process flows, and data auditors have the CUSTOM_PARAMS and SYSTEM_PARAMS parameters. Web services based on Code CT mappings contain the parameters OWB_PARAMS.

   - **CUSTOM_PARAMS:** Represents the values for the mapping input parameters used in the Web service.

     For example, when you create a Web service based on a mapping and the mapping requires input parameters, use the CUSTOM_PARAMS field to enter values for these input parameters. Use commas to separate multiple values.

   - **SYSTEM_PARAMS:** Represents the values for mapping execution parameters, if any, such as Bulk Size, Audit Level, Operating Mode, Maximum Number of Errors, and Commit Frequency. When you have multiple system parameters, use a comma to separate each parameter.

     For example, OPERATING_MODE=SET_BASED,AUDIT_LEVEL=NONE.

   - **OWB_PARAMS:** Represents the parameters of the CT mapping on which the Web service is based.

4. Click **OK**.

The Web service is executed and the results of the execution are displayed in a new log window in the Design Center. The details displayed include the number of rows selected, inserted, updated, or deleted and any errors or warnings that occurred.

## Using a Browser to Execute Web Services

Use any browser to execute Web services that were deployed to either the OC4J server embedded in Warehouse Builderor to other OC4J servers.

When you use the OC4J server embedded in Warehouse Builder to access Web services, all the prerequisites for Web service security are provided. You must provide basic authentication before you can execute the Web service.

**To execute Web services using a browser:**

1. Ensure that the prerequisites, as described in "Prerequisites for Executing Web Services" on page 16-11, are satisfied.

2. (Optional) While executing Web services that were deployed to an OC4J server other than the one installed with Warehouse Builder, perform the steps listed in "Setting Up Secure Access on External OC4J Servers" on page 16-19.

> **Note:** You may encounter errors while executing Web services that were not deployed to the OC4J server embedded in Warehouse Builder. See "Error Executing Web Services from the Secure Web Site" on page 15-7 for information about resolving these errors.

3. Open a Web browser, specify the following URL in the address bar, and press the Enter key.

   `http://host_name:8888/jndi_name/webservice`

   To execute the Web service securely, use the following URL:

   `https://host_name:4443/jndi_name/webservice`

   The endpoint page for the Web service is displayed. If you use the AGENTWEBSERVICE Web service provided by Warehouse Builder, then the AgentWebService Endpoint page is displayed.

   Here, *host_name* represents the host name of the computer on which the Web service is stored, and *jndi_name* is the name of the `.ear` file generated for the Web service. To execute Web services using the AGENTWEBSERVICE installed with Warehouse Builder, use jrt as the *jndi_name*.

   The default port numbers used for the Web service are 8888 and 4443. You can use different port numbers.

4. Follow the steps listed in "Performing Operations on Web Services Using a Browser" on page 16-13 to execute the Web service.

## Performing Operations on Web Services Using a Browser

The Web service AGENTWEBSERVICE, under the AGENT_SERVER node of the Globals Navigator, is an embedded Web service that exposes the Agent server installed with Warehouse Builder as a Web service. The steps listed in this section are performed using the AgentWebService Endpoint page, which is the interface corresponding to the AGENTWEBSERVICE. This Web service is started automatically when the Control Center Agent (CCA) is started. However, you can use other Web service endpoints to perform operations defined by Web services.

1. Select the operation that you want to perform on the Web service and provide the information required to perform the operation.

   Following are some of the operations that you can select when you use AGENTWEBSERVICE.

   - isDeployed

     See "Determining If a Web Service or Application Was Deployed to an OC4J Server" on page 16-14

   - runCCJob

     See "Executing a Control Center Job" on page 16-14

   - abortJob

     See "Terminating an Execution Job" on page 16-15

   - invokeEAR

     See "Running Deployed Applications" on page 16-15

2. Expand the Show Transport Info node and select **Enable** to the right of the label HTTP Authentication.
3. In the Transport Info section, enter details in the following fields:
   - **Username:** Name of a J2EE user, with the `OWB_J2EE_OPERATOR` role, that is used to execute the Web service
   - **Password:** Password for the J2EE user that you specified in the Username field
4. Click **Invoke**.

   The Test Result page containing the results of the operation is displayed.

   **Note:** Sometimes, after you enter the credentials and click **Invoke**, you may be prompted for credentials. Reenter the credentials of the J2EE that you specified in the Transport Info section.

### Determining If a Web Service or Application Was Deployed to an OC4J Server

The isDeployed operation enables you to determine if a Web service or application is deployed to an OC4J server.

Use the following steps to determine if a Web service was deployed to an OC4J Server.

1. On the AgentWebService Endpoint page, select **isDeployed** in the Operation field.
2. In the jndiName field, enter the name of the application.

   Use the jndi_name or the fully qualified application name. To determine this name, check the *OWB_ORACLE_HOME*`/owb/jrt/applications` directory.

If the Web service was deployed to the OC4J instance, the XML code on the Test Result page displays True.

### Executing a Control Center Job

The runCCJob operation enables you to execute a Control Center job. Jobs include Web services defined using Warehouse Builder, mappings, and process flows.

Use the following steps to execute a Control Center job.

1. On the AgentWebService Endpoint page, select runCCJob in the Operation field.
2. Provide information in the following fields:
   - **username:** Represents the name of the workspace user executing the Web service.
   - **password:** Represents the password of the user specified in the username field.
   - **workspace:** Represents the name of the workspace in which the Web service execution job should be run. If the user executing the Web service is not the workspace owner, then prefix the workspace name with the username (for example, test_user.my_workspace.)
   - **location:** Represents the physical name of the location to which the task is deployed.
   - **task_type:** Represents the type of task. Use the following values:

     PLSQL - for PL/SQL mappings

     SQL_LOADER - for SQL*Loader mappings

PROCESS - for process flows

SAP - for SAP mappings

DATA_AUDITOR - for Warehouse Builder data auditor mappings

- **task_name:** Represents the physical name of the deployed object. For example, MY_MAPPING. For process flows, qualify the process flow name with the name of the process flow package to which it belongs (for example, `MY_PROCESS_FLOW_PACK. MY_PROCESS_FLOW`).
- **connection_string:** Represents the connection information of the computer that has the Control Center Manager.
- **system_params:** Represents the mapping execution parameters of the mapping, if any, such as Bulk Size or Commit Frequency. When you have multiple system parameters, use a comma to separate each parameter.

  For example, OPERATING_MODE=SET_BASED,AUDIT_LEVEL=NONE.
- **custom_params:** Represents the input parameters for the mapping.

The runCCJob operation returns 1 if the execution was successful, 2 if there were warnings, and 3 if there were errors in the execution.

### Terminating an Execution Job

The abortJob operation enables you to terminate a particular job that was submitted to the Control Center Manager.

Use the following steps to terminate a particular job.

1. On the AgentWebService Endpoint page, select abortJob in the Operation field.
2. In the jobID field, enter the Job ID of the Control Center job that you want to terminate.
3. In the timeOut field, enter the value for the time out in milliseconds. Entering a zero in this field indicates that there is no timeout.

A return value of true in the Test Results page indicates that the terminate message was sent to the Control Center Manager.

### Running Deployed Applications

The invokeEAR operation enables you to run deployed applications such as Web services and CT mappings.

Use the following steps to execute a Web service or CT mapping.

1. On the AgentWebService Endpoint page, select invokeEAR in the Operation field.
2. In the jndiName field, enter the JNDI name of the Web service.
3. In the soa_params field, enter the values of the mapping execution parameters. Separate each value using a comma.
4. In the owb_params field, enter the values of the input parameters.

The Test Result page contains the Job ID of the Web service execution.

# Using Web Services as Activities in Process Flows

You can use the functionality defined in a Web service as part of a process flow. The Web service can either be created or imported into Warehouse Builder. To use Web services in a process flow, use the Web Service activity.

For an example of using Web services in process flows, see "Case Study: Using Web Services for Data Integration" on page 16-21.

## Rules for Using Web Services in Process Flows

In process flows, Warehouse Builder only supports Web services that conform to the following rules:

- Only Web services described through an accessible WSDL file are supported.
- If the WSDL file contains more than one service, then one service must be nominated.
- Web services that have basic authentications need a URI location to provide the credentials. You must set the Deployed Location property of the Web Service activity to this URI location.
- If the Web service contains more than one port then the port must be nominated.
- The port must use the http transport.

## Steps to Use Web Services in Process Flows

**To use a Web service in a process flow:**

1. In the Projects Navigator, create a process flow.

   For more information about creating process flows, see "Steps for Defining Process Flows" on page 8-5.

2. Add all the activities, except the Web service activity, that are part of the process flow and establish data flows between them.

3. If you are using an external Web service, import the Web service into Warehouse Builder by creating a public Web service in the Globals Navigator.

   **See Also:** "Creating Web Services Based on a URL" on page 16-10 for more information about importing Web services

4. Drag and drop the Web service, either from the Projects Navigator or the Globals Navigator, onto the Process Flow Editor canvas.

   *or*

   From the Graph menu, select **Available Objects**. The Add Available Objects dialog box is displayed. Select the required Web service and click **OK**.

   The Web Service Operation dialog box is displayed.

5. Select an operation from the available operations of the Web service and click **OK**. The selected operation is used in the process flow.

   The Web service is added to the Process Flow Editor and its properties are listed in the Structure panel.

   The operation in a synchronous Web service has both input and output messages. They will be mapped to input or output parameters of the Web Service activity in the process flow.

6. Provide the required input values for the Web service activity properties. Select the property in the Structure panel, and use the Property Inspector to set values.

   If the Web service needs authentication, create a URI location and set the Deployed Location property of the Web Service activity to this URI location.
7. Establish data flows to and from the Web Service activity.
8. Generate the process flow and resolve any errors that may occur.
9. Ensure that all Web services that you added to the process flow in the form of Web Service activities are deployed.
10. Deploy the process flow package containing the process flow created in Step 1. Use the Control Center Manager or right-click the process flow package in the Projects Navigator and select **Deploy**.
11. Execute the process flow. Right-click the process flow in the Projects Navigator and select **Start**.

### Synchronizing Web Service Activities with Their Referenced Web Services

When the definition of a Web service is modified, you must propagate these changes to all the process flows that consume this Web service.

**To synchronize Web services used in process flows:**

1. Right-click the process flow that uses the Web service and select **Open**.

   The Process Flow Editor for this process flow is displayed.
2. Select the Web Service activity that represents the Web service that has changed. From the Edit menu, select **Synchronize**.

   The Synchronize dialog box is displayed. The object with which the Web service should be synchronized is selected, and you cannot modify this.
3. Specify the Matching Strategy by selecting one of the following matching options: Match By Object Id, Match By Object Position, or Match by Object Name.

   For details about these options, click **Help**.
4. Specify the Synchronize Strategy by selecting Replace or Merge.
5. Click **OK** to synchronize the Web service with the object on which it is based.

The changes made to the Web service are propagated to the Web Service activity that is based on the Web service.

## Using Web Services in Mappings

Because Web services are essentially functions, Warehouse Builder leverages the Web services support provided by Oracle Database to enable you to use Web services in mappings (which are PL/SQL packages). Thus, you can leverage functionality present in existing Web services.

Use one of the following methods to create a mapping that uses Web services as sources or targets:

- Use the `UTL_HTTP` package.
- Use the JPublisher utility to interface SQL to the Web service.

  See "Steps to Consume a Web Service in a Mapping Using JPublisher" on page 16-18.

- Use the `UTL_DBWS` package to consume Web services.

  Depending on the version of Oracle Database, you may need to download and install the DBWS utility to perform these tasks.

- Use the script `use_webservice_in_mapping.tcl` located in the *`OWB_ORACLE_HOME`*`/owb/misc/mappingWS` directory to create an expert that you can use to consume Web services in mapping.

### Steps to Consume a Web Service in a Mapping Using JPublisher

The JPublisher utility translates your object types (which can be Oracle objects, Varrays, nested tables, REFs, or object types) to Java classes and generates accessor methods for each of the object's attributes. JPublisher creates the mapping between object types and Java classes, and between object attribute types and their corresponding Java types.

For PL/SQL packages, JPublisher creates a class containing a wrapper method for each subprogram in the package. Like object methods, the wrapper methods generated for each subprogram are always instance methods, even when the original method is static. The wrapper methods generated by JPublisher provide a convenient way to invoke PL/SQL stored procedures from Java code or to invoke a Java stored procedure from a client Java program.

1. Use JPublisher to generate table function proxies for the WSDL file and publish PL/SQL wrapper and proxy code.

   For example, the following command generates proxies for the Web service available at the URL http://99.22.32.21:9762/services/test_ws?wsdl.

   ```
   jpub -user wh_tgt/wh_tgt_pswd -sysuser system/oracle -dir=test_ws
        -proxywsdl=http://90.22.32.21:9762/services/test_ws?wsdl
   ```

   Here, `wh_tgt` and `wh_tgt_pswd` are the database credentials for the Warehouse Builder location where the mapping is deployed.

   JPublisher generates Java classes and PL/SQL wrappers and loads them into the specified schema (`WH_TGT`).

2. (Optional) Define PL/SQL code to call the Web service.

   This step verifies that the callout to the table function works at the Oracle Database level, without involving Warehouse Builder.

   The following example verifies that the callout to the function called `my_func` that is part of the package `my_pack` works correctly:

   ```
   SELECT * FROM TABLE(WH_TGT.MY_PROC.MY_FUNC)
   ```

3. Import the Web service metadata, for which you generated table function proxies, into Warehouse Builder using the Import Metadata Wizard.

   The metadata includes user-defined types and PL/SQL packages.

4. Open the Warehouse Builder mapping in which you want to consume the Web service and add a Table Function operator to call the Web service. Perform the following tasks:

   - In the Table Function Name property of the Table Function operator, enter the name of the generated table function (from Step 1) that you want to add to the mapping.
   - In the INGRP1 group of the Table Function operator, select the type of input accepted by the table function using the Input Parameter Type field.

5. Define the source rows that represent the input to the Web service.

   Map the operator that represents the Web service input to the inout group of the Table Function operator.

6. Capture the output of the Web service in a table in the mapping.

   Because the Table Function operator returns a collection type as output, use the Expand Object operator, if required, to map the individual out rows to the target table.

7. (Optional) If you need to set a Web proxy (for example because you are running behind a corporate firewall), use a Pre-Mapping Process operator that uses the procedure INITIALIZE_PROXY to configure the HTTP proxy.

8. Generate and execute the mapping.

# Using Secure Sockets Layer (SSL) to Access Web Services Securely

Where security is a primary concern, Warehouse Builder enables you to access Web services in a secure way using the Secure Sockets Layer (SSL). This ensures that messages exchanged between the OC4J server and the Web service are secured.

You can access Web services deployed to both the OC4J server embedded in Warehouse Builder and to other external OC4J servers securely.

**See Also:** "Using a Browser to Execute Web Services" on page 16-12

## J2EE Roles for Control Center Agent Security

Warehouse Builder provides the following three roles to facilitate Warehouse Builder Control Center Agent security.

- `OWB_J2EE_EXECUTOR`: Enables grantees to execute mappings in the Control Center Agent (CCA).
- `OWB_J2EE_OPERATOR`: Includes the `OWB_J2EE_EXECUTOR` role and enables grantees to access and manipulate audit information.
- `OWB_J2EE_ADMINISTRATOR`: Includes the `OWB_J2EE_EXECUTOR` role and enables grantees to administer OC4J and deploy Warehouse Builder objects to the Control Center Agent.

## Setting Up Secure Access on External OC4J Servers

You can securely access Web services located on other OC4J servers (that are not embedded in Warehouse Builder). Before you do so, you must set up security on the OC4J server.

**Use the following steps to set up secure access on other OC4J servers:**

1. Create a key store with an RSA private/public key pair using the keytool utility.

   The following example uses the RSA key pair generation algorithm to generate a key store that resides in a file named `mykeystore.jks` and which has a password of 123456.

   ```
   %keytool -genkey -keylag RSA -keystore mykeystore.jks -storepass 123456
   ```

   The `keystore` option sets the file name where the keys are stored. The `storepass` option sets the password for protecting the key store. If you omit the `storepass` option, then you will be prompted for the password.

2. You are prompted to enter a key entry password. In OC4J 10.1.3.x implementations, the key store password must be the same as the key entry password.

   The `mykeystore.jks` file is created in the current directory. The default alias of the key is `mykey`.

3. If you do not have a `secure-web-site.xml` file, create one in the following location: *ORACLE_HOME*`/j2ee/home/config`.

   To start, copy whatever content you need from `default-web-site.xml`. This typically includes the following subelements under the <web-site> element:

   - <web-app> (for each Web application that you want to secure)
   - <access-log> (for logging; confirm that this specifies an appropriate log file)
   - <default-web-app>

4. Update `secure-web-site.xml` with the following elements:

   - Update the web-site element to add secure="true" and to set the port number to some available port. For standalone OC4J, use the HTTP protocol, which is the default setting. To use the default of 443, you must be a super user.

     When you set protocol="http" and secure="true", the HTTPS protocol is used.

     The following is an example of a <web-site> element.

     ```
     <web-site port="4443" secure="true" protocol="http"
     display-name="Default Oracle OAS Containers for J2EE Web Site">
     ...
     ...
     </web-site>
     ```

   - Add an entry under the web-site element to define the key store and its password as follows:

     <ssl-config keystore="*your_keystore*" keystore-password="*your_password*" />

     Here, *your_keystore* is the path to the key store—either absolute, or relative to *ORACLE_HOME*`/j2ee/home/config` (where the Web site XML file is located)—and `your_password` is the key store password.

5. Save the changes to `secure-web-site.xml`.

6. Enable the secure Web site by adding the secure Web site to the `server.xml` file located in *OWB_ORACLE_HOME*`/owb/jrt/config` directory.

7. Restart the OC4J server to ensure that the previous changes are applied.

8. If they are not yet created, create the `OWB_J2EE_EXECUTOR`, `OWB_J2EE_OPERATOR`, and `OWB_J2EE_ADMINISTRATOR` roles.

   See the `file system-jazn-data.xml` file located in the *OWB_ORACLE_HOME*`/owb/jrt/config` folder.

9. Create the J2EE user used to execute Web services and grant the `OWB_J2EE_EXECUTOR` role to this user.

### Updating the Key Store Password

For the OC4J server that is embedded in Warehouse Builder, you are provided with a secure key store for using SSL with Web services. This key store is available in the `serverkeystore.jks` file in the *OWB_ORACLE_HOME*/`owb/jrt/config` folder.

The default password for this key store is `welcome`. Use the *JAVA_HOME*/`bin/keytool` to change the password. Or just replace the key store with a newly created key store using *JAVA_HOME*/`bin/keytool`.

## Case Study: Using Web Services for Data Integration

Company A and Company B have just been merged. Company A is located in San Francisco, USA, and Company B is located in Shanghai, China. Currently, they are still following their own separate business processes. There is a need to develop a plan to integrate their business processes.

Company A uses Oracle Database to store their data and Oracle Warehouse Builder for data integration and ETL. A mapping is used to determine the total sales for a specified period.

Company B uses a SQL Server database to store data and Oracle Warehouse Builder for data integration and ETL. Because the source tables are in SQL Server, a CT mapping is used to determine the total sales over a specified period.

### Example: Publishing Mappings as Web Services

Company A stores the sales details in a table called `ORDERS`. The `PRODUCTS` table stores details about products. The mapping `LOAD_TOTAL_SALES_MAP` transforms source data and loads the details of total sales into the target table `TOTAL_SALES`. As part of the data integration requirement, the business processes of Company A and Company B must be integrated.

Publishing the mapping `LOAD_TOTAL_SALES_MAP` as a Web service will enable the functionality defined in the mapping to be accessed remotely, without dependency on the location, data format, or provider's platform.

Company A uses the following tables to store data:

- `ORDERS`: contains the columns `order_id`, `order_date`, `product_id`, `quantity`, and `customer_id`
- `PRODUCTS`: contains the columns `product_id`, `product_name`, `product_desc`, and `product_price`
- `CUSTOMERS`: contains the columns `customer_id`, `first_name`, `last_name`, `cust_address`, and `cust_city`

Use the following steps to publish the `LOAD_TOTAL_SALES_MAP` as a Web service.

1. In the Projects Navigator, create an application server module called `INTEGRATION_AS_MOD` that will contain the Web service you are creating. Ensure that the location details of this module are set to the agent location to which the Web service will be deployed.
2. Expand the Oracle module that contains the `LOAD_TOTAL_SALES_MAP` mapping.
3. Right-click the `LOAD_TOTAL_SALES_MAP` mapping and select **Publish as Web Service**.

   The Select Application Server or Web Service Package dialog box is displayed.

4. Select INTEGRATION_AS_MOD and click **OK**.

The Web service called WS_LOAD_TOTAL_SALES_MAP is created and deployed to the agent location associated with the application server module INTEGRATION_AS_MOD.

## Example: Consuming Web Services in Process Flows

Company B uses a CT mapping called LOAD_TOT_SALES_CT_MAP to load aggregate sales during a specified period to the target table TOT_SALES. This mapping is similar to the mapping used by Company A. However, a CT mapping is used because the source tables ORDERS, PRODUCTS, and CUSTOMERS are stored in a SQL Server database.

Because Company B is located in China, the source tables store the sales figures in Chinese Yuan. However, because Company A and Company B have been merged, the sales head wants to see the consolidated sales of both companies in a common currency, U.S. Dollar. You must now convert the sales figures of Company B to U.S. Dollar.

To determine the conversion rate, you can use an external Currency Converter Web Service. This Web service takes two input parameters, From Currency and To Currency. Its output is the multiple that must be used to convert the From Currency to the To Currency. Because this is an external Web service, you must first import this Web service into Warehouse Builder.

### Steps to Consume a Web Service in a Process Flow

1. Modify the LOAD_TOT_SALES_CT_MAP Code Template (CT) Mapping.
2. Import the Currency Converter Web Service.
3. Create a Process Flow That Consumes the Currency Converter Web Service.

### Modify the LOAD_TOT_SALES_CT_MAP Code Template (CT) Mapping

Edit the CT mapping LOAD_TOT_SALES_CT_MAP and add a Mapping Input operator and an Expression operator. The Mapping Input operator is used to provide the currency conversion value. The Expression operator is used to compute the total sales in U.S. Dollar by multiplying the total sales in Chinese Yuan with the conversion value and then loading the converted sales figures into the TOT_SALES table.

### Import the Currency Converter Web Service

The Currency Converter Web service is an external Web service that is available at:

http://www.webservicex.net/CurrencyConvertor.asmx?WSDL

Before you can consume this Web service in a process flow, you must import this Web service into Warehouse Builder, using the following steps:

1. In the Globals Navigator, create an application server module called PUBLIC_AS_MOD.

   To import a Web service that is based on a URL, you must create a public Web service in the Globals Navigator.

2. In the Globals Navigator, right-click PUBLIC_AS_MOD and select **New Web Service** to create a Web service based on a URL. This Web service is called WS_CURR_CONVERT.

   Use the URL to the currency converter Web service to specify the WSDL file location.

For more details about creating a Web service based on a URL, see "Creating Web Services Based on a URL" on page 16-10.

### Create a Process Flow That Consumes the Currency Converter Web Service

Use a process flow to establish the order in which objects are executed and to use the output of an object as input to another. The process flow loads the target table with sales figures for Company B in U.S. Dollar.

Figure 16–1 displays the process flow that loads the `TOT_SALES` table with the sales figures for Company B in U.S. Dollar.

The Web service `CONVERTSERVICE_CONVERSIONRATE` is executed first and its output is the conversion value that should be multiplied to a value in Chinese Yuan to convert it to U.S. Dollar. This value is provided as the input to the Mapping Input Parameter operator in the `LOAD_TOT_SALES_CT_MAP` CT mapping, represented by the Mapping activity CMAP1, that represents the conversion value.

***Figure 16–1 Process Flow that Consumes a Web Service***

## Example: Integrating Warehouse Builder Web Services with Oracle BPEL Process Manager

You can integrate ETL functionality developed using Warehouse Builder with products such as Oracle BPEL Process Manager. This is achieved by publishing ETL objects as Web services that can be consumed by Oracle BPEL Process Manager.

Oracle BPEL Process Manager provides a comprehensive and easy-to-use solution for designing, deploying, and managing BPEL Processes.

### Scenario

The sales head of Company A wants to evaluate the sales performance of both Company A and Company B over a specified time period. Both company A and Company B have their own processes to determine sales performance over a specified period. The need is to integrate these processes so that you have an easy way to determine the total sales, during a period, for the combined company.

### Before You Integrate ETL Functionality with Oracle BPEL Process Manager

Company A uses the mapping `LOAD_TOTAL_SALES_MAP`, which is published as the Web service `WS_LOAD_TOTAL_SALES_MAP`. Company B uses the Code Template mapping `LOAD_TOT_SALES_CT_MAP` and a process flow to convert the sales in Chinese Yuan to U.S. Dollar. Ensure that you publish this process flow as a Web service. Also ensure that you publish any objects associated with these Web services.

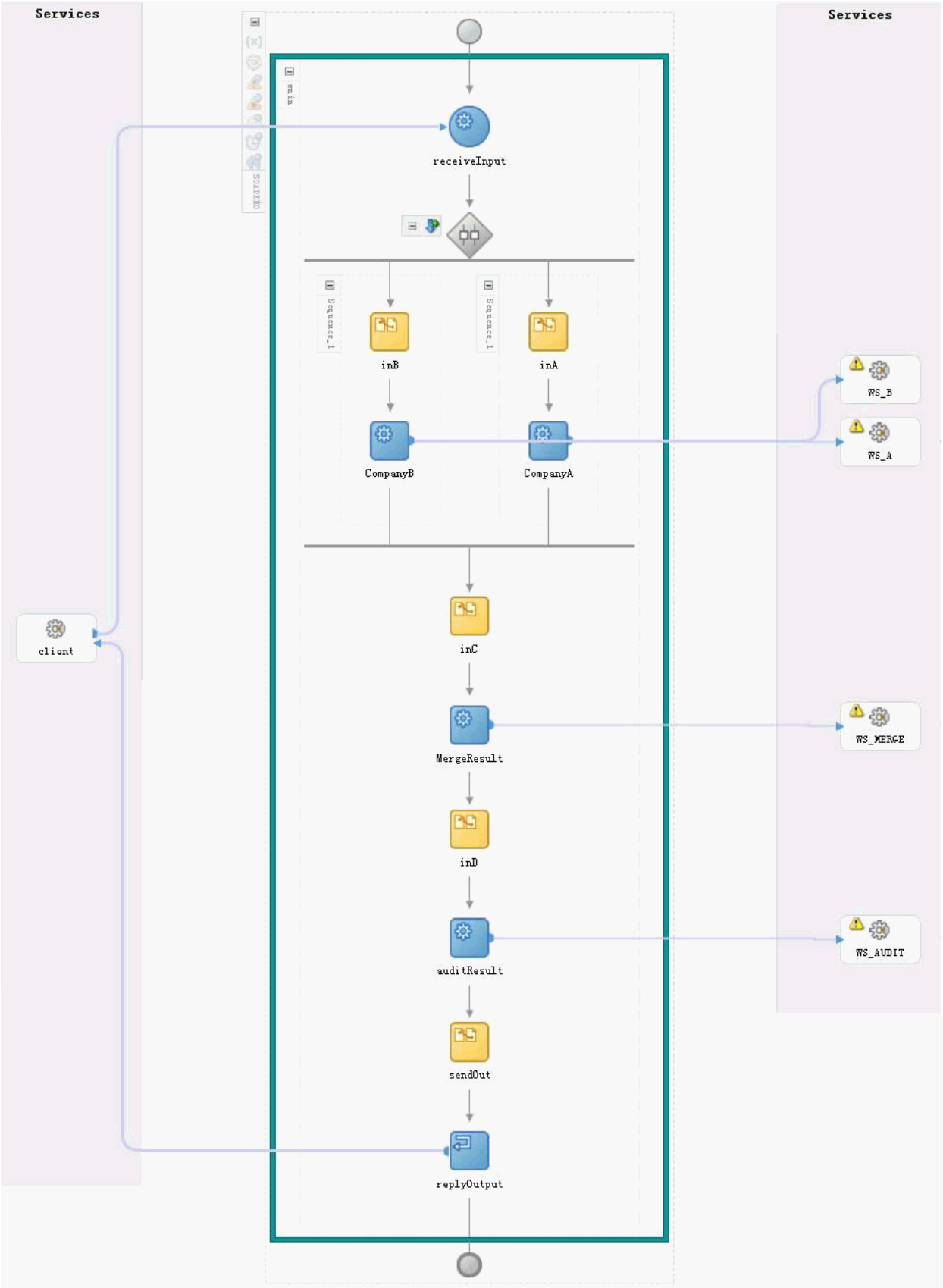
### Steps to Integrate With BPEL

1. Start the Oracle SOA Suite.
2. Start JDeveloper BPEL Designer

3. Create a BPEL process called BPEL_INTEG. This process uses the Web services created by Company A and Company B.

   Figure 16–2 displays the process BPEL_INTEG.

***Figure 16–2 BPEL Process that Uses Web Services***



4. Deploy the BPEL process by right-clicking the BPEL process in the Applications Navigator, selecting **Deploy**, and choosing the deployment location.

   You can use the Log panel to view the deployment results.

5. Login to BPEL Manager using a web browser.

6. On the Dashboard tab, select BPEL_INTEG in the Deployed BPEL Processes section.

7. Select the Initiate tab.

8. Enter values for the input parameters START_DATE and END_DATE using the Input field and click **Post XML Message** to run the BPEL process.

   For example, enter the following value in the Input field: START_DATA=2007-01-01,END_DATA=2008-12-31.

9. View the audit trail for the Business Process Execution. Select the Instances tab of the BPEL Console and click the Flow link.

# 17

# Moving Large Volumes of Data Using Transportable Modules

Oracle Warehouse Builder enables you to build and publish an enterprise data warehouse in stages. You can improve the performance and manageability of the data warehouse.

Warehouse Builder mappings access remote data through database links. Processing overhead and network delays make this data access process slower than local data access by the mappings. You can use one of the following strategies to speed up data access:

- Create a transportable module to copy remote objects (tables, views, materialized views, and so on) from a source database into a target database. The mappings in the target data warehouse can then access data locally.
- Data can be partially processed in the source database and then the preprocessed data can be copied, using a transportable module, from source to target database for final loading into the data warehouse.

A transportable module functions like a shipping service that moves a package of objects from one site to another at the fastest possible speed.

**Note:** To utilize transportable modules, ensure that your organization has licensed the Warehouse Builder Enterprise ETL Option.

The chapter contains the following topics:


- About Transportable Modules
- Benefits of Using Transportable Modules
- Instructions for Using Transportable Modules
- Editing Transportable Modules


## About Transportable Modules

Transportable modules enables you to rapidly copy a group of related database objects from one database to another.

Using the Design Center, you first create a transportable module, and specify the source database location and the target database location. Then, you select the database objects to be included in the transportable module. The metadata of the selected objects are imported from the source database into the transportable module. The metadata is stored in the workspace. To physically move the data and metadata from source into target, you must configure and deploy the transportable module to

the target location. During deployment, both data and metadata are extracted from the source database and created in the target database.

A combination of the following technologies enables the movement of data and metadata:

- Oracle Data Pump
- Transportable Tablespace
- DBMS_FILE_TRANSFER
- Binary FTP
- Local file copy
- code generation and deployment

You can configure transportable modules to influence which technologies are used.

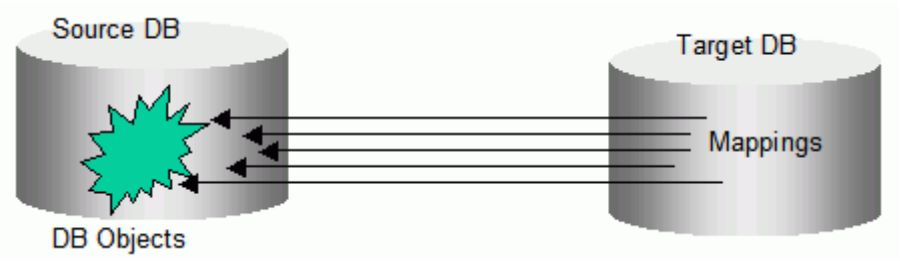
You can add the following source objects to transportable modules:

- Tablespaces
- Schemas
- Tables
- Views
- Sequences
- Materialized Views
- PL/SQL Functions, Procedures, and Packages
- Object Types
- Varying Array Types (Varrays)
- Nested Table Types

The traditional Extract, Transform, and Load (ETL) process extracts data from remote databases through multiple remote accesses using database links.

Figure 17–1 displays the traditional extraction of data from remote databases.

***Figure 17–1 Extraction Of Data From Remote Databases Through Multiple Remote Accesses Using Database Links***

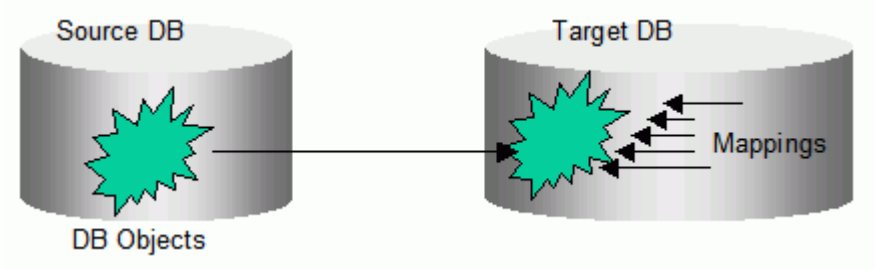



During remote accesses using database links, significant performance degradation occurs due to serial queries and serial DMLs, and network latencies. The performance degradation will appear more if the same source tables are accessed multiple times.

In the transportable module architecture, all the source objects needed by the mappings are bundled together and moved to the target during a deployment. The transportable modules deployment uses Oracle Data Pump, FTP, and Oracle

transportable table space to achieve very high transportation performance. This transportation absorbs the cost of the network delays just once. After deployment, mappings access data locally, which can easily benefit from parallel queries and parallel DMLs. Repeated accesses to the same data increases the performance benefit of transportable modules.

***Figure 17–2 Transportable Modules Deployment***




Using transportable modules, data warehouse loadings become more manageable. The source database needs to be shut down only for a short period of time for the transportable module to complete the deployment. Users of the source database do not have to wait until the entire data is loaded into the data warehouse. For example, if you are using the transportable tablespace implementation, transportable modules can copy a tablespace of 20 GB in about five minutes, resulting in a down time of five minutes in the source database.

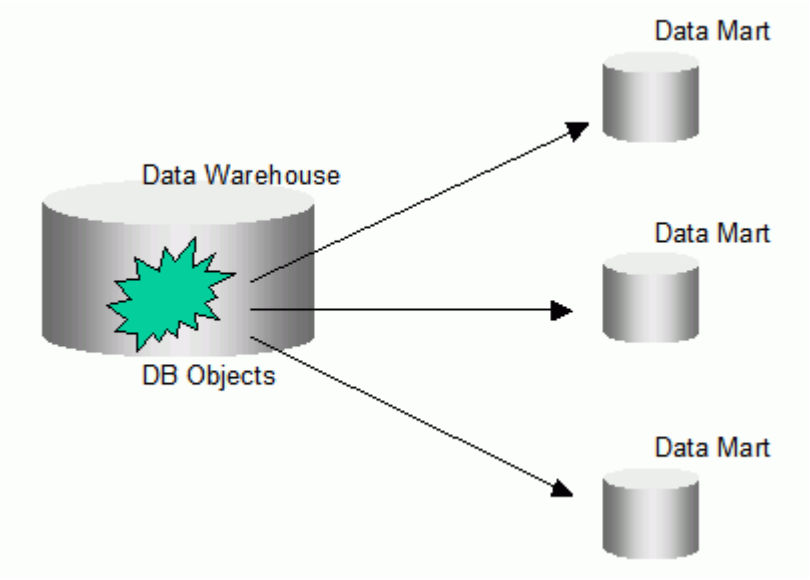
Data copied into the target database is a snapshot of the information present in the source database. This can be used to create a data versioning system. Advanced users can create streams on the transported tables for capturing real-time changes from the source. The transported tables can also be copied into larger tables as a partition.

In a multidepartmental enterprise environment, the target database may actually be an operational data store that is used for intermediate reporting and updating purposes. This target database could in turn, serve as a source to the next stage of data collection. You can use the transportable modules at multiple stages, along the path on which the data is moved before it is stored in the data warehouse.

Transportable modules can also be used for publishing data marts. A data mart is normally a portion of a larger data warehouse for single or departmental access. At times, creating a data mart amounts to copying what has already been collected and processed in the data warehouse. A transportable module can be created to perform this task. You can also use the same transportable module to deploy a data mart to multiple locations.

Figure 17–3 displays a transportable module used for publishing for data marts.

*Figure 17–3 Data Marts in a Data Warehouse*




Because a transportable module deploys a snapshot of the source database objects, the deployment time can be used to track the version of the data marts.

## About Transportable Modules and Oracle Database Technology

Transportable modules work by leveraging technology in Warehouse Builder plus technology in the Oracle Database. A transportable module replicates parts of a source database into a target database. The parts of the source database that can be replicated include tablespaces, tables, indexes, constraints, and other relational objects.

Depending on the database version, the Oracle Database replicates the tablespace. When you transport data between two releases of 8*i* databases or between two releases of 9*i* databases, the database calls the Oracle transportable tablespaces functionality. When you transport data between two Oracle 10*g* databases, the database calls the Oracle Data Pump functionality.

In the case of Oracle Database 10*g* and Oracle Data Pump, you can transport tables without transporting their tablespaces. For example, if your table is 100 KB and its tablespace size is 10MB, then you can deploy the table without deploying the entire tablespace. Only Oracle Data Pump provides the option to copy an entire schema. For Oracle 10*g* release database, you specify either data pump or transportable tablespaces during configuration as described in "Configuring a Transportable Module" on page 17-12.

---

**Note:** For more information about transportable tablespace and Data Pump, see the Oracle Database 11*g* documentation.

---

# Benefits of Using Transportable Modules

Before the introduction of transportable modules, the most scalable data transportation method relied on moving flat files containing raw data. This method required data to be unloaded or exported into files from the source database, and then these files were loaded or imported into the target database. The transportable

modules method entirely bypasses the unload and reload steps and gives you access to the Oracle Database technologies Transportable Tablespaces and Data Pump.

### High Performance Data Extraction

Transportable modules reduce the need for mappings to access data remotely. If you have large volumes of data on remote computers, then use transportable modules to quickly replicate the source onto the Oracle target database. Warehouse Builder mappings can then directly access a local copy of the data. In addition, because the source becomes part of the target, you can perform the ETL operations directly on the source data.

### Distribute and Archive Data Marts

A central data warehouse handles ETL processing while dependent data marts are read-only. You can use transportable modules to copy from a read-only data mart to multiple departmental databases. In this way, you can use your central data warehouse to periodically publish new data marts and then replace old data marts by dropping the old tablespace and importing a new one. Because duplication and distribution takes relatively less time, you can publish and distribute a data mart for daily analytical or business operations.

### Archive Sources

You can set the source tablespaces to read-only mode and then export them to a target. All the data files are copied, creating a consistent snapshot of the source database at a given time. This copy can then be archived. The archived data can be restored in the source and target databases.

## Instructions for Using Transportable Modules

### Before You Begin

Ensure that you can connect to source and target databases as a user with the necessary roles and privileges as described in Verifying the Requirements for Using Transportable Modules on page 17-6.

Ensure that your organization has licensed the Warehouse Builder Enterprise ETL Option.

**To use transportable modules, refer to the following sections:**

**Note to Database Administrators:** Step 1 of these instructions requires some powerful database roles and privileges. Step 3 requires knowledge of schema passwords. Depending on security considerations, you can allow developers to perform Step 3 or restrict it to database administrators only.

1. Specifying Locations for Transportable Modules on page 17-7

   Ensure to successfully test these connections before proceeding to the next step.

2. Creating a Transportable Module on page 17-8
3. Configuring a Transportable Module on page 17-12
4. Generating and Deploying a Transportable Module on page 17-15
5. Designing Mappings that Access Data through Transportable Modules on page 17-17
6. Editing Transportable Modules on page 17-17

## Verifying the Requirements for Using Transportable Modules

When creating a Transportable Module source location, the source location user must possess specific roles and/or privileges depending on the version of the source database.

- If the source database is earlier than Oracle 10*g*, then the SYSDBA privilege is required for the source location user.
- If the source database is Oracle 10*g*, then the SYSDBA privilege is not required, but the following must be assigned to the source location user.
  - CONNECT role
  - EXP_FULL_DATABASE role
  - ALTER TABLESPACE privilege

When creating a Transportable Module target location, the target location user must possess specific roles and/or privileges depending on the version of the target database.

- If the target database is earlier than Oracle 10*g*, then the SYSDBA privilege is required for the target location user.
- If the target database is Oracle 10*g*, then the SYSDBA privilege is not required but the following must be assigned to the target location user.
  - CONNECT role with admin option
  - RESOURCE role with admin option
  - IMP_FULL_DATABASE role
  - ALTER TABLESPACE privilege
  - EXECUTE_CATALOG_ROLE with admin option
  - CREATE MATERIALIZED VIEW privilege with admin option
  - CREATE ANY DIRECTORY privilege

---

**Note:** Transportable Module source and target location users must be assigned many powerful roles and privileges in order for the transportable modules to read objects from the source database and for creating objects in the target database. In a production environment, if necessary, the DBA may choose to create the transportable module source and target locations (using the Locations Navigator) for the data warehouse developers, and conceal the passwords.

---

The following is a SQL script for the DBA to assign source location users the required roles and privileges in the source database:

```
grant connect to <TM src location user>;
grant exp_full_database,alter tablespace to <TM src location user>;
```

The following is a SQL script for the DBA to assign target location users the required roles and privileges in the target database:

```
grant connect,resource to <TM tgt location user> with admin option;
grant imp_full_database,alter tablespace to <TM tgt location user>;
grant execute_catalog_role to <TM tgt location user> with admin option;
```

```
grant create materialized view to <TM tgt location user> with admin option;
grant create any directory to <TM tgt location user>;
```

## Specifying Locations for Transportable Modules

Before you create a transportable module, first define its source and target locations in the Location Navigator. Each transportable module can have only one source and one target location.

**To specify a transportable module location:**

1. In the Locations Navigator, expand the **Locations** node.
2. Expand the **Databases** node.
3. Right-click either the **Transportable Modules Source Locations** or **Transportable Modules Target Locations** node and then select **New.**

   Warehouse Builder displays a dialog box for specifying the connection information for the source or target location.
4. The instructions for defining source and target locations are the same except that you do not specify optional FTP connection details for targets. Follow the instructions in "Transportable Module Source Location Information" to specify the connection information and then test the connection.

### Transportable Module Source Location Information

Warehouse Builder first uses this connection information to import metadata for the transportable module from the source computer into the workspace. During deployment, the connection information is used to move data from the source to the target.

#### Name

A name for the location of the source or target database.

#### Description

An optional description for the location.

#### User Name/Password

Warehouse Builder uses the database user name and password to retrieve the metadata of the source objects you want to include in the transportable module. Warehouse Builder also uses this information during deployment to perform transportable tablespace or data pump operations.

To access databases for use with transportable modules, you must ensure that the user has the necessary database roles and privileges as described in Verifying the Requirements for Using Transportable Modules on page 17-6.

#### Host

Host name of the computer on which the database is installed.

#### Port

Port number of the computer on which the database is installed.

#### Service

Service name of the computer on which the database is installed.

#### Version

Choose the Oracle Database release number from the list.

#### FTP User Name/Password (Optional)

Specify FTP account credentials if you intend to use Oracle Transportable Tablespace as the method for transporting data. FTP credentials are not required if you do not plan to configure the Transportable Tablespace method.

You can leave the FTP account credentials blank, if you configure to use the Transportable Tablespace, but both source and target databases are located in the same computer, or both source and target can access shared disk volumes. Without the FTP credentials, an attempt is made to perform a plain copy of the source files from the source directory to target directory.

#### Test Connection

Click **Test Connection** to validate the connection information. Warehouse Builder attempts to connect to the source database and, if applicable, to the FTP service on the source computer. A success message is displayed only after both credentials are validated.

## Creating a Transportable Module

**To create a transportable module:**

1. From the Projects Navigator, expand the **Databases** node.
2. Right-click the **Transportable Modules** node and select **New**.

   The Welcome page of the Create Transportable Module Wizard is displayed.
3. The wizard guides you through the following tasks:

   Describing the Transportable Module

   Selecting the Source Location

   Selecting the Target Location

   Selecting Tablespaces and Schema Objects to Import

   Reviewing the Transportable Module Definitions

### Describing the Transportable Module

On the Name and Description page, type a name and optional description for the transportable module.

### Selecting the Source Location

Although you can create a new source location from the wizard page, it is recommended that you define locations for transportable modules before starting the wizard as described in "Transportable Module Source Location Information" on page 17-7.

When you select an existing location, the wizard tests the connection and does not allow you to proceed until you specify a location with a valid connection.

### Selecting the Target Location

Select a target location from the list. If no target locations are displayed, then click **New** and define a target location as described in "Transportable Module Source Location Information" on page 17-7.

### Selecting Tablespaces and Schema Objects to Import

Use the Define Contents page to select tablespaces and schema objects to include in the transportable module. On the left pane, Available Database Objects lists all source tablespaces, schemas, and available schema objects. On the right pane, Selected Database Objects displays the objects after you select and move the objects.

Expand the tablespaces to display the schemas in each tablespace and the objects in each schema. Non-tablespace schema objects such as views and sequences are also listed under their respective schema owners, even though these objects are not stored in the tablespace. To select multiple objects at the same time, hold down the Ctrl key while selecting them. You can include the following types of objects in transportable modules:

- Tables
- Views
- Materialized Views
- Sequences
- PL/SQL Functions, Procedures, and Packages
- Object Types, Varray Types, and Nested Tables Types

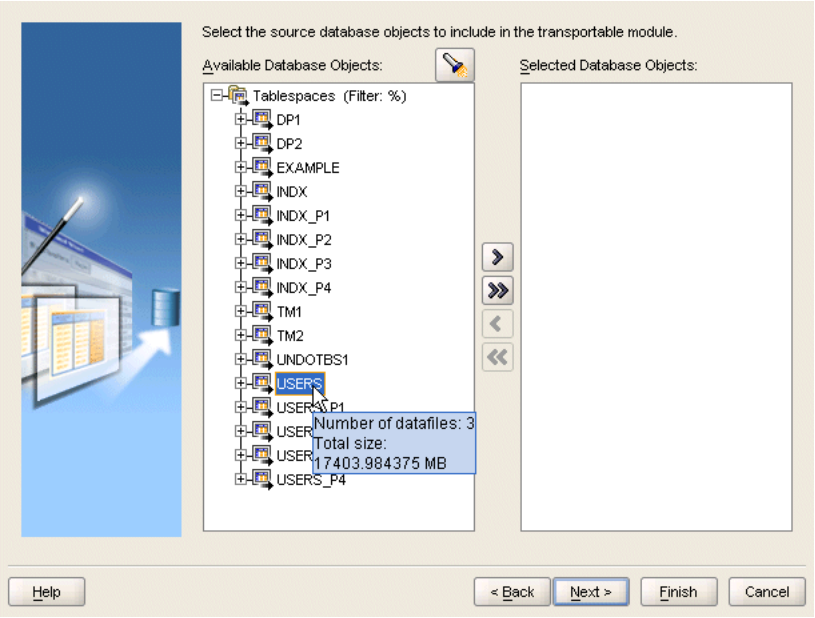
Select the tablespaces and schema objects from the Available Database Objects field and click the arrow buttons in the center to move the objects to the Selected Database Objects field.

### Available Database Objects

You can view the number of data files and their total size by placing your mouse over a node. The wizard displays the information in a tooltip.

Figure 17–4 displays the wizard with the tooltip.

*Figure 17–4 Viewing the Number of Data Files and Total Size*




**Finding Objects in the Available Database Object List:** Click the flashlight icon to find source data objects by type or name. In the **Object** field, type a name or pattern by which to filter your search, using the % character as a wildcard. From the **Type** list, indicate the object type you are searching. Check the required box to perform the search by name or by description.

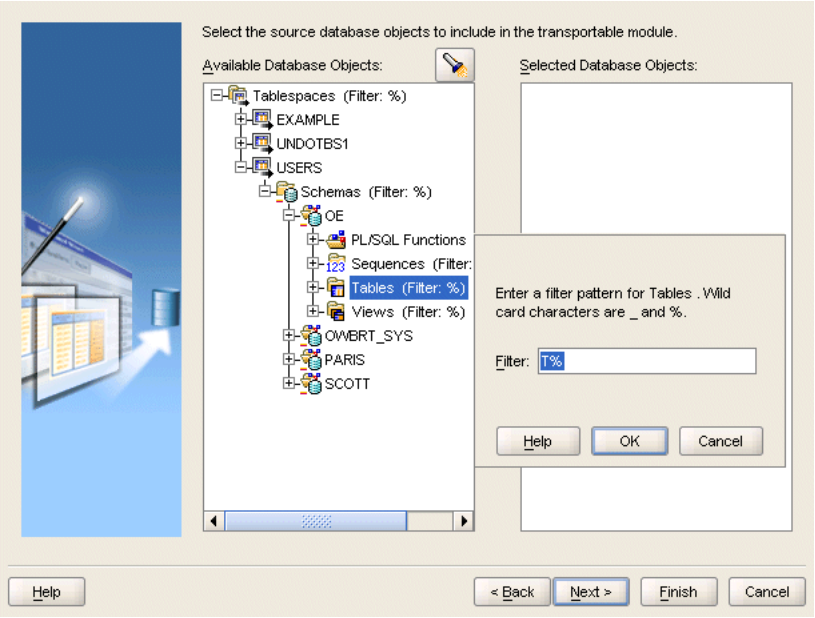
For example, type 'T%' in the Object field, select tablespaces from the Type field, and click **Find Next.** The cursor on the Available Database Objects navigation tree selects the name of the first tablespace that starts with a 'T.' If that is not the tablespace you want to select, then click **Find Next** to find the next tablespace. During this searching process, the navigation tree expands all the schema names and displays all the tablespaces.

> **Tip:** When searching for schema level objects such as tables, it is recommended that you select a tablespace or schema from the navigation tree before launching the search. This prevents a search over all tablespaces and significantly reduces the search time.

**Filtering the Available Database Objects List:** You can double-click a schema node or any of the nodes in the schema to type in a filter pattern. For example, if you type T% and click **OK,** the navigation tree displays only those objects that start with the letter T. The filter criteria will be displayed with the object name in the navigation tree, providing a helpful hint of which object types have filters applied.

Figure 17–5 displays the Define Contents page with a schema selected.

*Figure 17–5 Schema Node Selected on Define Contents Page*

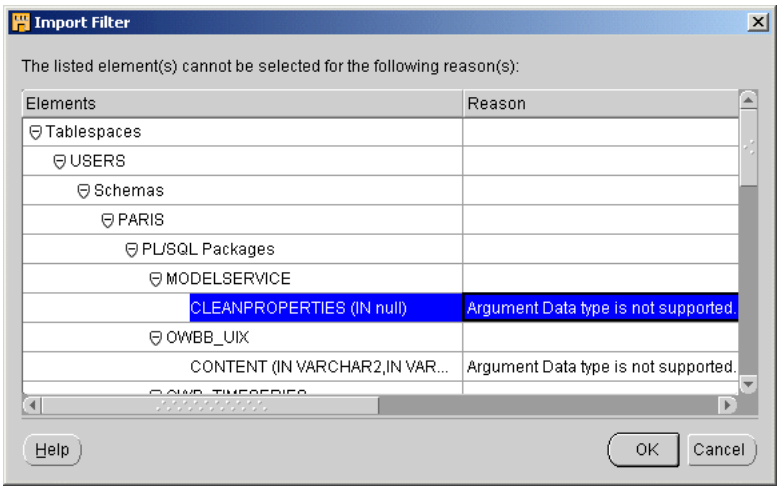



### Objects Not Available for Inclusion in Transportable Modules

If you select items that cannot be included in a transportable module, then a dialog box is displayed listing items that cannot be included and describing why.

Figure 17–6 displays the Import Filter dialog box.

*Figure 17–6 Import Filter Dialog Box*




## Reviewing the Transportable Module Definitions

Review the summary information and click **Finish** to import the metadata of the selected tablespace and schema objects.

After the transportable module is created in your workspace, you can locate it on the Projects Navigator under the Transportable Modules node. Expand the tree to display the imported definitions.

Warehouse Builder creates separate modules for separate schemas. The schema names on the Projects Navigator mirror the schema names in your source database.

Because the objects contained in a transportable module mirror the source database, you cannot edit these objects using the user interface. If the source database changes, then you can reimport the objects. If you want to delete objects from the transportable module, then right-click the object and select **Delete.** This action deletes the object from the definition of the transportable module but does not effect the underlying source database.

## Configuring a Transportable Module

In the Projects Navigator, right-click a transportable module and select **Configure** to configure it for deployment to the target database. You set configuration parameters at the following levels:

- Transportable Module Configuration Properties
- Schema Configuration Properties
- Target DataFile Configuration Properties
- Tablespace Configuration Properties

For most used cases, you can accept the default settings for all the configuration parameters with the exception of the Password setting. You must specify a password for each target schema. If the schema already exists in the target, then specify an existing password. If the schema does not already exist, then the schema can be created with the password you provide.

Depending on your company security polices, knowledge of schema passwords may be restricted to database administrators only. In that case, the database administrator must specify the password for each schema. Alternatively, developers can define new passwords for new schemas if the target has no existing schemas that match source schemas.

### Transportable Module Configuration Properties

Set the following runtime parameters for the transportable module:

#### Target OS Type

Select the type of operating system for the target. For versions earlier than Oracle Database 10*g*, the type of operating system on the target computer must be the same as the source computer. For versions Oracle Database 10*g* or higher, you can deploy to any operating system from any operating system.

#### Work Directory

You should create a directory on the target computer dedicated to the deployment of transportable modules. This dedicated directory stores files generated at run time including temporary files, scripts, log files, and transportable tablespace data files. If you do not create a dedicated directory and type its full path as the **Work Directory,** then the generated files are saved under the runtime home directory.

#### What to Deploy

Warehouse Builder enables you to select whether you want to deploy only the tables in your transportable module or all the related catalog objects, such as views and sequences, as well. Select the **TABLES_ONLY** if you want to deploy only tables. Otherwise, select the **ALL_OBJECTS**.

Use the **TABLES_ONLY** option to refresh the data in a transportable module. If you had previously deployed a transportable module with the **ALL_OBJECTS** option and want to replace only the tablespace from the same source, then redeploy the transportable module with the **TABLES_ONLY** option. The deployment drops the existing tablespace in the target, inserts the new one, and then recompiles the previously deployed metadata.

Similarly, if you previously deployed the transportable module using Data Pump, then the redeployment will only modify the tables in the transportable module.

### Transport Tablespace

By default, this setting is enabled and the tablespaces are transported. If you enable this setting, then also specify the settings under Target DataFile Configuration Properties.

If both the source and target databases are Oracle 10*g* or higher, then consider disabling this setting. For example, if your table is 100 KB and its tablespace size is 10 MB, then you can deploy the table without deploying the entire tablespace. When you disable **Transport Tablespace,** Oracle Data Pump is used to deploy the table and you can specify the Table Exists Action setting.

> **Note:** If source or target location is not Oracle 10*g*, the Transport Tablespace option is selected by default. In that case, Transportable Tablespace is the only implementation method for data movement. If both source and target locations are Oracle 10*g*, then you can deselect Transport Tablespace and use Data Pump.

If Transport Tablespace is selected, then there are further restrictions, depending on the versions of the source and target locations, as described in Table 17–1. When planning for data replications, take these restrictions into consideration. In general, Oracle 10*g*, particularly Oracle10*g* release 2, is the preferred target database.

***Table 17–1 Requirements for Replicating Data Between Database Versions***

| Source location | Target location |
|---|---|
| 10*g* | Targeting another Oracle 10*g* location requires that both databases must have the same character set and the same national character set.<br>Targeting an Oracle 8*i* or 9*i* location is not possible. |
| 9*i* | Targeting an Oracle 9*i* or 10*g* location requires that both databases must have the same character set, the same national character set, and both databases must be on the same operating system platform.<br>Targeting an Oracle 8*i* or 9*i* location is not possible. |

*Table 17–1 (Cont.) Requirements for Replicating Data Between Database Versions*

| Source location | Target location |
|---|---|
| 8*i* | Targeting an Oracle 8*i*, 9*i*, or 10*g* requires all of the following:<br>▪ Both source and target databases must have the same character set.<br>▪ Both source and target databases must have the same national character set.<br>▪ Both source and target databases must be on the same operating system platform.<br>▪ Both source and target databases must have the same block size.<br>▪ Cannot change schema names during transporting tablespaces.<br>▪ Cannot change tablespace names during transporting tablespaces. |

## Schema Configuration Properties

Set the following schema parameters for the transportable module:

### Target Schema Name

This property enables you to change the name of the source schema when it is deployed to the target. Select the Default or click the Ellipsis button to type the new name for your schema in the target and click **OK.** For example, you can change SCOTT to SCOTT1.

### Password

For existing schemas, type a valid password for the schema. For schemas to be created, Warehouse Builder creates the schema with the password you provide.

### Default Tablespace

Specify the default tablespace to be used when creating the target schema. If you leave this setting blank, then the default specified by the target is used.

### Schema Exists Action

Specify what action should be taken if the schema already exists in the target. The default value is Skip.

### Schema Does Not Exist Action

Specify what action should be taken if the schema does not already exist in the target. The default value is Create.

### Table Exists Action

When Transport Tablespace is disabled, use this property to specify what action should be taken if the table already exists in the target. The default value is Skip.

### Copy Source Schema

When you use Oracle Data Pump by deselecting Transport Tablespace, you can select this option to copy the entire source schema into the target.

#### Parallel

When you use Oracle Data Pump by deselecting Transport Tablespace, specify the maximum number of processes for the Oracle Database to use for carrying out the transfer of data.

### Target DataFile Configuration Properties

You must set the following data file parameters for the transportable module:

#### Directory

Indicate the directory where you want the data file to be stored on your target computer. If you leave the directory unspecified, then the data file is stored in the Work Directory.

#### File Name

Specify the name of the data file to be created in the target computer. You can use this parameter to rename the data file. Accept the DEFAULT to persist the data file name from the source database or click the Ellipsis button to type a new name for the data file, and click **OK.**

#### Overwrite

If this parameter is selected, then the existing data file is overwritten. Otherwise, the deployment is terminated if an existing data file is found.

### Tablespace Configuration Properties

When you enable Transport Tablespace, set the following tablespace parameters for the transportable module:

#### Tablespace Name

If you are using a database prior to 10*g*, then the target tablespace name must be the same as your source tablespace name. For such cases, this field is read-only. If a tablespace with the same name already exists in your target database, then the runtime operation will first drop the existing tablespace and replace it with the new one.

If you are using Oracle Database 10*g* or higher, then you can change the target tablespace name.

#### Drop Existing Tablespace

If this setting is selected, the existing tablespace is dropped and recreated in the target. By default, this setting is not selected and prevents you from deleting the tablespace in the target in the event that the tablespace with the same name already exists. In this case, the deployment process stops with an error.

## Generating and Deploying a Transportable Module

When you deploy a transportable module, the Control Center displays the transportable module as including all the tables while the other catalog objects such as views are displayed separately. When you select a deploy action for the transportable module, the Control Center sets the associated catalog objects to the same deploy action.

During deployment of a transportable module, there are two ways for users to monitor the deployment progress. The first way is by the use of the Job Details

window. The status line is instantly refreshed with the most up-to-date status. The message box immediately above the status line shows all the messages logged so far.

Another way of observing the progress is by viewing the log file that the transportable module deployment process generates. The transportable module log file is created in the Work Directory that the user has configured. The name of the file is always <The TM Name>.log, for example TM1.log if the name of the transportable module is TM1. This file is a plain text file containing the same messages that you can see in the message box in the Job Details window. Example 17–1 shows the contents of a transportable module log file.

Currently, there are a total of 16 steps to view the log files. Some steps may be skipped depending on the user configurations, and some steps may contain error messages that transportable module considers ignorable, such as failures in creating referential constraints due to referenced tables not found errors. This log file contains important information. It must be carefully examined during and after the transportable module deployment completes.

***Example 17–1 Log file containing important information***

```
step1 begin: making connection to target db ...
step1 end: connected to target
Target ORACLE_HOME = /data/oracle/ora1010
step2 begin: making connection to source db...
step2 end: skipped.
step3 begin: making source tablespaces read only...
step3 end: skipped.
step4 begin: exporting tts...
step4 end: skipped.
step 5 begin: checking for existing datafiles on target...
step5 end: skipped.
step 6 begin: drop existing tablespaces
step6 end: skipped.
step7 begin: transporting datafiles...
step7 end: skipped.
step8 begin: managing schemas/users ...
step8 end: completed setting up target schemas
step9 begin: drop non-table schema objects...
step9 end: nothing to drop.
step10 begin: converting datafiles...
step10 end: skipped.
step 11 begin: importing tts ...
find or create a useable dblink to source.
step11 end: importing tts is not requested by user.
step 11 end: import tts is successful
step 12 begin: restore source tablespaces original status ...
step12 end: skipped.
step13 end: skipped.
step14 begin: non-tts import ...

Import: Release 10.1.0.4.0 - Production on Tuesday, 04 April, 2006 10:43
Copyright (c) 2003, Oracle.  All rights reserved.

Username:
Connected to: Oracle Database 10g Enterprise Edition Release 10.1.0.4.0 -
Production
With the Partitioning, OLAP and Data Mining options
Starting "TMTGT_U"."SYS_IMPORT_TABLE_02":  TMTGT_
U/********@(DESCRIPTION=(ADDRESS=(HOST=LOCALHOST)(PROTOCOL=tcp)(PORT=1521))(CONNEC
T_DATA=(SERVICE_NAME=ORA1010.US.ORACLE.COM))) parfile=/home/ygong/tmdir/TM1_
```

```
imptts.par
Estimate in progress using BLOCKS method...
Processing object type TABLE_EXPORT/TABLE/TBL_TABLE_DATA/TABLE/TABLE_DATA
Total estimation using BLOCKS method: 64 KB
Processing object type TABLE_EXPORT/TABLE/TABLE
. . imported "TMU1"."TA"                                    2 rows
Processing object type TABLE_EXPORT/TABLE/STATISTICS/TABLE_STATISTICS
Processing object type TABLE_EXPORT/TABLE/CONSTRAINT/REF_CONSTRAINT
ORA-39083: Object type REF_CONSTRAINT failed to create with error:
ORA-00942: table or view does not exist
Failing sql is:
ALTER TABLE "TMU1"."TA" ADD CONSTRAINT "TA_T1_FK" FOREIGN KEY ("C") REFERENCES
"TMU1"."T1" ("C") ENABLE

Job "TMTGT_U"."SYS_IMPORT_TABLE_02" completed with 1 error(s) at 10:44
step14: import has failures.
step14 end: non-tts import completed with warnings
step15 end: create flat file directories skipped.
step16 end: transporting flat files skipped.
```

### Designing Mappings that Access Data through Transportable Modules

After you successfully deploy a transportable module, you can use the objects in the transportable module in ETL designs. When you add source and target operators to a mapping, you can select objects from the transportable module folder.

## Editing Transportable Modules

A transportable module is located under the transportable modules node within the Databases node on the Projects Navigator.

You can edit a transportable module by right-clicking the name of the transportable module from the Projects Navigator and selecting **Open**. Warehouse Builder displays the Edit Transportable Module dialog box containing four tabs.

### Name

From the Name tab, you can edit the name and description of the transportable module.

### Source Location

Warehouse Builder uses this connection information to access the source computer and import the metadata into its workspace. Warehouse Builder also uses this information during runtime to move the tablespace data from the source to the target.

The Source Database tab is read-only. Once you have imported tablespace definitions from a source computer, you cannot change the location information.

### Tablespaces

The Tablespaces tab displays the tablespaces to be transported and their size. This tab is read-only. You can also view the tablespace size for individual data files in a tablespace. For details, see "Viewing Tablespace Properties" on page 17-18.

## Target Locations

Displays the available and selected target locations. You can move a location from Available Locations to Selected Locations, or configure a new location.

## Viewing Tablespace Properties

You can view the properties of a tablespace by right-clicking the name of the tablespace from the Projects Navigator and selecting **Open**. Warehouse Builder opens the Edit Tablespace dialog box. This property sheet displays the size of individual data files in a tablespace. It has two tabs, Name and Source Datafiles.

## Reimporting Metadata into a Transportable Module

If your source data has changed since you last created a transportable module, then you can reimport the metadata to update your workspace definitions. When you open the Reimport dialog box, the source location you specified while creating the transportable module is stored and the source objects are displayed.

**To reimport transportable module definitions:**

1. From the Projects Navigator, right-click the **Transportable Modules** name and select **Reimport**.

   The Re-create Transportable Module dialog box is displayed.

2. From the Available Database Objects column, select the objects you want to reimport.

   The database objects that have been previously imported into the workspace are listed in bold. You can also choose to import new definitions.

3. Use the arrow buttons to move the objects to the Selected Database Objects column, and click **OK.**

Warehouse Builder reimports existing definitions and creates new ones. The transportable module reflects the changes and updates after the reimport is completed.

# Part III

## Data Profiling and Data Quality

Oracle Warehouse Builder provides data quality functionality that can be a part of your ETL process. It also enables you to perform data profiling and check for data compliance.

> **See Also:** *Oracle Warehouse Builder Concepts* for an overview of data quality and data profiling

This part contains the following chapters:


- Chapter 18, "Performing Data Profiling"
- Chapter 19, "Designing and Deriving Data Rules"
- Chapter 20, "Monitoring Quality with Data Auditors and Data Rules"
- Chapter 21, "Data Cleansing and Correction with Data Rules"
- Chapter 22, "Name and Address Cleansing"
- Chapter 23, "Matching, Merging, and Deduplication"

# 18

# Performing Data Profiling

This chapter describes the data profiling features of Oracle Warehouse Builder and how to use them, with Warehouse Builder ETL or with other ETL tools.

This chapter contains the following topics:


- Overview of Data Profiling
- Performing Data Profiling
- Tuning the Data Profiling Process for Better Profiling Performance
- Data Watch and Repair (DWR) for Oracle Master Data Management (MDM)


## Overview of Data Profiling

Data profiling enables you to assess the quality of your source data before you use it in data warehousing or other data integration scenarios.

Data profiling analyzes the content, structure, and relationships within data to uncover patterns and rules, inconsistencies, anomalies, and redundancies.

Data profiling can be usefully applied to any source in a data integration or warehousing scenario, and to master data stores in MDM scenarios. It is also useful for any scenario involving a new source, because it enables the discovery of information beyond the basic metadata defined in the data dictionary.

### Sources Supported by Warehouse Builder for Data Profiling

Warehouse Builder data profiling can support the following source types:

- Oracle databases
- Data sources accessed through Oracle gateways or ODBC
- Flat file sources

  To profile flat files, you must import them into Warehouse Builder, create external tables based on the flat files, and then profile the external tables.
- SAP R/3 and other ERP application sources

---

**Note:** Data profiling does not support sources accessed through JDBC.

---

## Using Warehouse Builder Data Profiling with Warehouse Builder ETL

Warehouse Builder data profiling and the rest of the data quality features also derive value from and add value to Warehouse Builder ETL in the following ways:

- Warehouse Builder data quality can automatically generate data cleansing logic based on data rules. The cleansing processes are implemented by automatically created ETL mappings, with the same functionality as other ETL mappings. Deployment, execution and scheduling for data cleansing processes is identical to other mappings. Developers familiar with Warehouse Builder ETL features can, for example, tune performance on data cleansing processes, look at the generated code, and so on, as with any ETL mappings. Where custom data cleansing and correction logic is required, it can be implemented in PL/SQL.
- Metadata about data profiling results, represented as data rules, can be bound to the profiled data objects, and those rules are then available in any context in which the profiled objects are used in ETL.

  For example, Warehouse Builder ETL can use data rules in ETL logic to implement separate data flows for compliant and noncompliant rows in a table. Noncompliant rows can be routed through any data cleansing, correction or augmentation logic available in the database, transformed using ordinary mapping operators, or simply logged in the error table. All required logic for the different options for handling noncompliant rows is automatically generated, for any Warehouse Builder mapping in which a data rule is used.

The data profiling features of Oracle Oracle Warehouse Builder also use the infrastructure of Warehouse Builder ETL to connect to data sources, access the data to be profiled, and move intermediate profiling results into a scratch area called a profiling workspace.

## Using Warehouse Builder Data Profiling with Other ETL Solutions

Warehouse Builder data profiling and data quality can be used alongside any third-party ETL solution or hand-coded ETL solution. In such scenarios, the usage model is:

- Leave your existing ETL solution in place
- In Warehouse Builder, create a workspace and locations so that you can connect to your data sources.
- Create a data profile, add the objects to be profiled, and set any profiling parameters.
- Run your profiling jobs
- Explore the results in the Data Profile Editor
- Derive data rules based on the profiling results, in order to better understand and document patterns in your data.

See "Performing Data Profiling" on page 18-4 for details on how to perform these tasks.

You can also use the data cleansing and correction features of Warehouse Builder alongside third party ETL solutions. See Chapter 21, "Data Cleansing and Correction with Data Rules" for details on data cleansing and correction, and Chapter 20, "Monitoring Quality with Data Auditors and Data Rules" for details on data auditing.

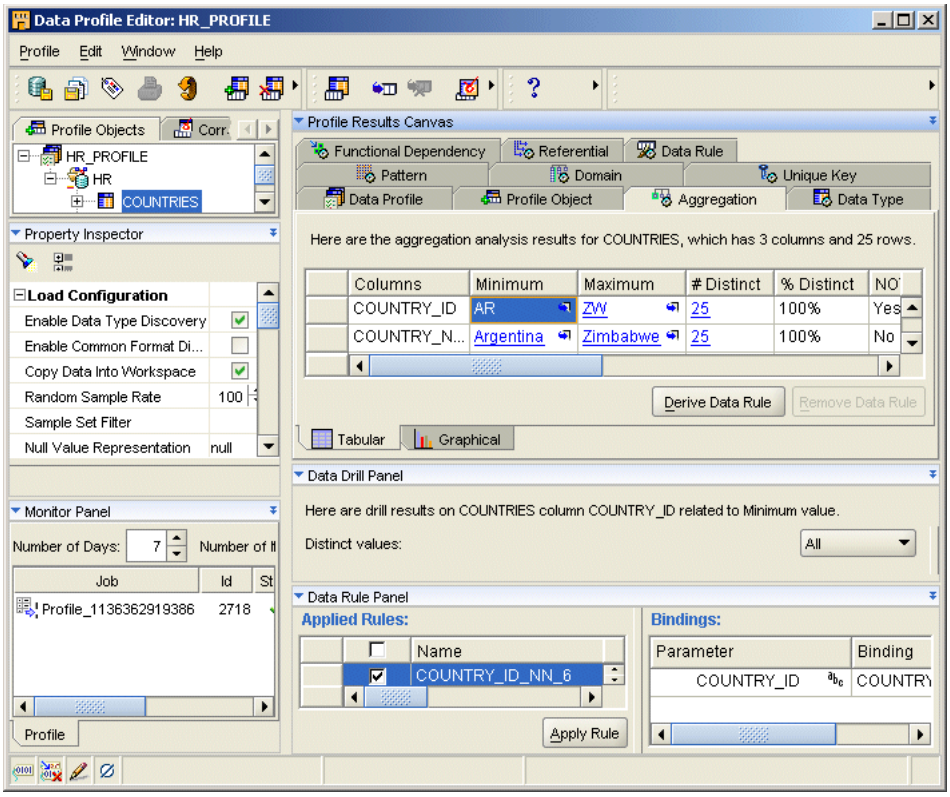
## About the Data Profile Editor

The Data Profile Editor provides the primary user interface for most of the data profiling, data rules and data cleansing functionality of Warehouse Builder. From the Data Profile Editor, you can:

- Set up and run data profiling, that is, attribute analysis and structural analysis of selected objects.
- Generate a new target schema based on the profile analysis and source table rules, where all of the data will be compliant with the data rules applied to objects in the profile.
- Automatically generate mappings and transformations that perform data correction based on your data rules and selected cleansing strategies.

Figure 18–1 displays the Data Profile Editor.

***Figure 18–1 Data Profile Editor***




The Data Profile Editor consists of the following:

- Menu Bar
- Toolbars
- Object Tree
- Property Inspector
- Monitor Panel
- Profile Results Canvas
- Data Drill Panel

- Data Rule Panel

Refer to the online help for detailed information about the contents of each panel.

## Performing Data Profiling

Data profiling is resource intensive and profile execution times can be quite long, especially on large data volumes. Profiling the entire contents of one or all of your source systems is not likely to be the most efficient way to produce useful profiling results. An iterative approach, in which initial limited data profiling produces discoveries that are used in deciding what to profile next, is more likely to be effective when faced with a large and complex set of sources to profile.

The following guidelines can increase the usefulness of your data profiling results while reducing profiling time and effort:

- Profile sources that impact numerous downstream objects, or where the objects affected are the most important outputs of your system. For example, if a source table contains data that affect the contents of numerous downstream business intelligence reports, then consider profiling that table. If only certain columns are used in downstream targets, then consider using attribute sets to limit profiling to those columns.

  **Tip:** Use the data lineage and impact analysis features of Warehouse Builder to identify sources with wide impact in your system. For more information, see *Oracle Warehouse Builder Installation and Administration Guide for Windows and UNIX*.

- Profile sources that are more likely to contain erroneous data. For example, customer and order data that are entered manually are more likely to contain errors than product data downloaded from suppliers.
- Profile any new source before integrating it into any existing system, especially sources where initial data quality is unknown.
- If you have documentation for a source, you may want to define data rules based on that documentation, then specifically profile for compliance with those rules. If this effort reveals noncompliant data, then consider more complete profiling.
- Disable types of profiling that are not likely to provide significant results. For example, you may want to perform an initial profiling pass limited to domain discovery or before more advanced and computationally intensive profiling types such as discovering functional dependencies.
- Initially, limit profiling to a random sampling of data, rather than profiling the entire contents of a source. After identifying an initial set of data rules, you can profile all rows for compliance with those rules.

For example, consider a data source that contains five tables: `CUSTOMERS`, `REGIONS`, `ORDERS`, `PRODUCTS`, and `PROMOTIONS`. You have the following information:

- The `CUSTOMERS` table is known to contain many duplicate and erroneous entries and is used in creating marketing campaigns.
- The `ORDERS` table includes order data transcribed from handwritten forms and telephone orders
- The `PRODUCTS` table contains data downloaded from a trusted supplier, and has a `VARCHAR2(4000)` column `DESCRIPTION` that contains free-form text.

- The documentation for the source system says that there is a foreign key relationship between the `ORDERS` and `REGIONS` tables, but this documentation is several years old and the source systems may have changed.

In such a case, you can limit your initial profiling process as follows:

- Profile the `CUSTOMERS` table before the others, because it is known to contain errors
- Exclude the `PRODUCTS.DESCRIPTION` column from all profiling, because free-form text is not likely to produce useful information in profiling
- Use random sampling to profile a limited number of rows from all of the tables
- Profile the `ORDERS` and `REGIONS` tables together to test the foreign key relationship

Later, you can do more profiling based on discoveries in the first passes.

> **See Also:** "Tuning the Data Profiling Process for Better Profiling Performance" on page 18-22 for information about how to tune the data profiling process

## Data Profiling Restrictions

- You can only profile data stored in Oracle databases, data accessible through Oracle Gateways, and data in SAP systems. If the data you need to profile is stored in a flat file, then create an external table based on this flat file.
- You cannot directly profile data that is accessed through JDBC-based connectivity. You must first stage this data in an Oracle Database and then profile it.
- Data profiling always uses the default configuration. Customers using multiple configurations should ensure that the default configuration has appropriate settings for profiling.
- The profiling workspace location must be an Oracle 10*g* database or higher.
- You cannot profile a source table that contains complex data types if the source module and the data profile are located on different database instances.
- Data profiling cannot analyze more than 165 columns of data in each table, view or materialized view at a time. If you need to profile an object with more than 165 columns, then you must create an attribute set to select a subset of columns to be profiled together. See "Using Attribute Sets to Profile a Subset of Columns from a Data Object" on page 18-20 for details.

## Prerequisites for Data Profiling

- Before profiling data objects, ensure that your project contains the correct metadata for the source objects on which you are performing data profiling.
- The objects to be profiled must already contain source data. For example, if you are profiling tables stored in a staging area, you must load the staging tables from their sources before executing the profile.
- Because data profiling uses mappings to run the profiling, you must ensure that all locations that you are using are registered. Data profiling attempts to register your locations. If, for some reason, data profiling cannot register your locations, you must explicitly register the locations before you begin profiling.

**See Also:** *Oracle Warehouse Builder Sources and Targets Guide* for more information about importing metadata.

## Steps to Perform Data Profiling

To prepare for data profiling, you must create one or more data profile objects in your project. Each data profile object is a metadata object in a project. A data profile object stores the following:

- The set of objects to be profiled together
- The types of profiling to perform on this set of objects
- Settings such as thresholds that control the profiling operations
- Results returned by the most recent execution of data profiling using these settings
- Metadata about data corrections generated from profiling results

After you have decided which objects to profile, use the following steps to guide you through the profiling process:

1. In the Projects Navigator, expand a project node and import all objects that you want to profile into this project.

   See "Prerequisites for Data Profiling" on page 18-5.

2. Under the project into which you imported objects, create a data profile object that will contain the objects to be profiled.

   See "Creating Data Profiles" on page 18-6.

3. Configure the data profile to specify the types of analysis that you want to perform on the data being profiled.

   See "Configuring Data Profiles" on page 18-7.

4. Profile the data to perform the types of analysis specified in the previous step.

   See "Profiling Data" on page 18-10.

5. View and analyze the data profiling results.

   See "Viewing Profile Results" on page 18-11.

Based on the profiling results, you can decide to generate schema and data corrections. These corrections are automatically generated by Warehouse Builder. For more information about automatically generating schema and data corrections, see "Generating Correction Mappings from Data Profiling Results" on page 21-2.

You can also derive data rules based on the data profiling results. For more information about deriving data rules based on profiling results, see "Deriving Data Rules From Data Profiling Results" on page 19-4.

## Creating Data Profiles

Once your locations are prepared and the data is available, you must create a data profile object in Design Center. A data profile is a metadata object in the workspace that specifies the set of data objects to profile, the settings controlling the profiling operations, the results returned after you profile the data, and correction information (if you decide to use these corrections).

**To create a data profile:**

1. From the Projects Navigator, expand the project node in which you want to create a data profile.
2. Right-click Data Profiles and select **New Data Profile**.

   The Welcome page of the Data Profile Wizard is displayed.
3. On the Welcome page, click **Next**.
4. On the Name and Description page, enter a name and an optional description for the data profile. Click **Next**.
5. On the Select Objects page, select the objects that you want to include in the data profile and use the arrows to move them to the Selected list. Click **Next**.

   To select multiple objects, hold down the **Ctrl** key while selecting objects. You can include tables, views, materialized views, external tables, dimensions, and cubes in your data profile.
6. If you selected tables, views, or materialized views that contain attribute sets, the Choose Attribute Set dialog box is displayed. The list at the bottom of this dialog box displays the attribute sets defined on the data object.
   - To profile only the attributes defined in the attribute set, select the attribute set from the list.
   - To profile all columns in the data object, select **<all columns>** from the list.
7. If you selected dimensional objects on the Select Objects page, a warning is displayed informing you that the relational objects bound to these dimensional objects will also be added to the profile. Click **Yes** to proceed.
8. On the Summary page, review the choices that you made on the previous wizard pages. Click **Back** to change any selected values. Click **Finish** to create the data profile.

   The Warehouse Builder note dialog is displayed. Click **OK** to display the Data Profile Editor for the newly created data profile.

   The new data profile is added to the Data Profiles node in the navigation tree.

## Configuring Data Profiles

You can, and should, configure a data profile before running it if there are specific types of analysis that you do, or do not, want to run.

You can configure a data profile at one of the following levels:

- The entire profile (all the objects contained in the profile)
- An individual object in the data profile

  For example, the data profile contains three tables. To perform certain types of analysis on one table, configure this table only.
- An attribute within an object

  For example, you know that there is only one problematic column in a table and you already know that most of the records should conform to values within a certain domain. You can focus your profiling resources on domain discovery and analysis. By narrowing down the type of profiling necessary, you use fewer resources and obtain the results faster.

### Steps to Configure Data Profiles

1. In the Projects Navigator, right-click the data profile and select **Open**.

   The Data Profile Editor for the data profile is displayed.

2. Select the level at which you want to set configuration parameters.

   - To configure the entire data profile, on the Profile Objects tab, select the data profile.
   - To configure a particular object in the data profile, on the Profile Objects tab, expand the node that represents the data profile. Select the data object by clicking on the data object name.
   - To configure an attribute within an object in a data profile, on the Profile Objects tab, expand the node that represents the data profile. Expand the data object that contains the attribute and then select the required attribute by clicking the attribute name.

3. Set the required configuration parameters using the Property Inspector panel.

   Configuration parameters are categorized into the following types: Load, Aggregation, Pattern Discovery, Domain Discovery, Relationship Attribute Count, Unique Key Discovery, Functional Dependency Discovery, Row Relationship Discovery, Redundant Column Discovery, and Data Rule Profiling. The following sections provide descriptions for the parameters in each category.

### Load Configuration Parameters

Table 18–1 describes the parameters in this category.

***Table 18–1 Description of Load Configuration Parameters***

| Configuration Parameter Name | Description |
|---|---|
| Enable Data Type Discovery | Set this parameter to true to enable data type discovery for the selected table. |
| Enable Common Format Discovery | Set this parameter to true to enable common format discovery for the selected table. |
| Copy Data Into Workspace | Set this parameter to true to enable copying of data from the source to the profile workspace. |
| Random Sample Rate | This value represents the percent of total rows that will be randomly selected during loading. |
| Always Force a Profile | Set this parameter to true to force the data profiler to reload and reprofile the data objects. |
| Sample Set Filter | This represents the `WHERE` clause that will be applied on the source when loading data into the profile workspace. Click the Ellipsis button on this field to display the Expression Builder. Use the Expression Builder to define the `WHERE` clause. |
| Null Value Representation | This value will be considered as null value during profiling. You must enclose the value in single quotation marks. The default value is null, which is considered as a database null. |

### Aggregation Configuration Parameters

This category contains the **Not Null Recommendation Percentage** parameter. If the percentage of null values in a column is less than this threshold percent, then that column will be discovered as a possible Not Null column.

### Pattern Discovery Configuration Parameters

- **Enable Pattern Discovery:** Set this parameter to true to enable pattern discovery.
- **Maximum Number of Patterns:** This represents the maximum number of patterns that the profiler will get for the attribute. For example, when you set this parameter to 10, it means that the profiler will get the top 10 patterns for the attribute.

### Domain Discovery Configuration Parameters

Table 18–2 describes the parameters in this category.

*Table 18–2 Description of Domain Discovery Configuration Parameters*

| Configuration Parameter Name | Description |
|---|---|
| Enable Domain Discovery | Set this parameter to true to enable domain discovery. |
| Domain Discovery Max Distinct Values Count | Represents the maximum number of distinct values in a column in order for that column to be discovered as possibly being defined by a domain. Domain discovery of a column occurs if the number of distinct values in that column is at or below the Max Distinct Values Count property, and the number of distinct values as a percentage of total rows is at or below the Max Distinct Values Percent property. |
| Domain Discovery Max Distinct Values Percent | Represents the maximum number of distinct values in a column, expressed as a percentage of the total number of rows in the table, in order for that column to be discovered as possibly being defined by a domain. Domain Discovery of a column occurs if the number of distinct values in that column is at or below the Max Distinct Values Count property, and the number of distinct values as a percentage of total rows is at or below the Max Distinct Values Percent property. |
| Domain Value Compliance Min Rows Count | Represents the minimum number of rows for the given distinct value in order for that distinct value to be considered as compliant with the domain. Domain Value Compliance for a value occurs if the number of rows with that value is at or above the Min Rows Count property, and the number of rows with that value as a percentage of total rows is at or above the Min Rows Percent property. |
| Domain Value Compliance Min Rows Percent | Represents the minimum number of rows, expressed as a percentage of the total number of rows, for the given distinct value in order for that distinct value to be considered as compliant with the domain. Domain Value Compliance for a value occurs if the number of rows with that value is at or above the Min Rows Count property, and the number of rows with that value as a percentage of total rows is at or above the Min Rows Percent property. |

### Relationship Attribute Count Configuration Parameters

This category contains the **Maximum Attribute Count** parameter that represents the maximum number of attributes for unique key, foreign key, and functional dependency profiling.

### Unique Key Discovery Configuration Parameters

- **Enable Unique Key Discovery:** Set this parameter to true to enable unique key discovery.
- **Minimum Uniqueness Percentage:** This is the minimum percentage of rows that must satisfy a unique key relationship.

### Functional Dependency Discovery Configuration Parameters

- **Enable Functional Dependency Discovery:** Set this parameter to true to enable functional dependency discovery.

- **Minimum Functional Dependency Percentage:** This is the minimum percentage of rows that must satisfy a functional dependency relationship.

### Row Relationship Discovery Configuration Parameters

- **Enable Relationship Discovery:** Set this parameter to true to enable foreign key discovery.
- **Enable Soundex Relationship Discovery:** Set this parameter to true to enable soundex relationship discovery for columns with string data types. Note that you must ensure that these attributes are part of relationship discovery.
- **Minimum Relationship Percentage:** This is the minimum percentage of rows that must satisfy a foreign key relationship.
- **Minimum Soundex Relationship Percentage:** This is the minimum percentage of rows that need to satisfy a soundex relationship. Values with the same soundex value will be considered the same.

### Redundant Column Discovery Configuration Parameters

- **Enable Redundant Columns Discovery:** Set this parameter to true to enable redundant column discovery with respect to a foreign key-unique key pair.
- **Minimum Redundancy Percentage:** This is the minimum percentage of rows that are redundant.

### Performance Configuration

This category contains the **Enable Materialized View Creation** parameter. Set this parameter to true to create materialized views for each column in every table of the data profile. This enhances query performance during drill down.

### Data Rule Profiling Configuration Parameters

This category contains the **Enable Data Rule Profiling for Table** parameter. Set this parameter to true to enable data rule profiling for the selected table. This setting is only applicable for a table, and not for an individual attribute.

## Profiling Data

Data profiling is achieved by performing deep scans of the selected objects. This can be a time-consuming process, depending on the number of objects and the type of profiling that you are running. However, profiling is run as an asynchronous job, and the Design Center client can be closed during this process. As with ETL jobs, you will see the job running in the job monitor. Warehouse Builder prompts you when the profiling job is complete.

### Steps to Profile Data

After you have created a data profile, you can open it in the Data Profile Editor to profile the data or review profile results from a previous run. The objects that you selected when creating the profile are displayed in the object tree of the Data Profile Editor. You can add objects to the profile by selecting **Profile** and then **Add**.

**To profile data using a data profile:**

1. Expand the Data Profiles node in the Projects Navigator, right-click a data profile, and select **Open**.

   The Data Profile Editor opens the selected data profile.

2. If you have not already done so, configure the data profile as described in "Configuring Data Profiles" on page 18-7.

3. From the **Profile** menu, select **Profile**.

   If this is the first time you are profiling data, the Data Profile Setup dialog box is displayed. Enter the details of the profiling workspace. For more information about the information to be entered, click **Help**.

   Warehouse Builder begins preparing metadata for profiling. The progress window containing the name of the object being created to profile the data is displayed. After the metadata preparation is complete, the Profiling Initiated dialog box is displayed, informing you that the profiling job has started.

4. On the Profiling Initiated dialog box, click **OK**.

   Once the profiling job starts, the data profiling is asynchronous, and you can continue working or even close the Design Center. Your profiling process will continue to run until it is completed.

5. View the status of the profiling job in the Monitor panel of the Data Profile Editor.

   You can continue to monitor the progress of your profiling job in the Monitor panel. After the profiling job is complete, the status displays as complete.

6. After the profiling is complete, the Retrieve Profile Results dialog box is displayed, and you are prompted to refresh the results. Click **Yes** to retrieve the data profiling results and display them in the Data Profile Editor.

   **Note:** Data profiling results are overwritten on subsequent profiling executions.

## Viewing Profile Results

After the profile operation is complete, you can open the data profile in the Data Profile Editor to view and analyze the results. The profiling results contain a variety of analytical and statistical information about the data profiled. You can immediately drill down into anomalies and view the data that caused them. You can then determine what data must be corrected.

**To view the profile results:**

1. Select the data profile in the navigation tree, right-click, and select **Open**.

   The Data Profile Editor opens and displays the data profile.

2. If you have previous data profiling results displayed in the Data Profile Editor, refresh the view when prompted so that the latest results are shown.

   The results of the profiling are displayed in the Profile Results Canvas.

3. Minimize the Data Rule and Monitor panels by clicking on the arrow symbol in the upper-left corner of the panel.

   This maximizes your screen space.

4. Select objects in the Profile Objects tab of the object tree to focus the results on a specific object.

   The profiling results for the selected object are displayed using the following tabs of the Profile Results Canvas.

   - Data Profile

- Profile Object
- Aggregation
- Data Type
- Domain
- Pattern
- Unique Key
- Functional Dependency
- Referential
- Data Rule

You can switch between various objects in the data profile. The tab that you had selected for the previous object remains selected.

### Data Profile

The Data Profile tab contains general information about the data profile. Use this tab to store any notes or information about the data profile.

### Profile Object

The Profile Object tab contains two subtabs: Object Data and Object Notes. The Object Data tab lists the data records in the object you have selected in the Profile Objects tab of the object tree. The number of rows that were used in the sample is listed. You can use the buttons along the top of the tab to execute a query, get more data, or add a `WHERE` clause.

### Aggregation

The Aggregation tab displays all the essential measures for each column, such as minimum and maximum values, number of distinct values, and null values. Some measures are available only for specific data types. These include the average, median, and standard deviation measures. Information can be viewed from either the Tabular or Graphical subtabs.

Table 18–3 describes the various measurement results available in the Aggregation tab.

***Table 18–3 Aggregation Results***

| Measurement | Description |
|---|---|
| Column | Name of the column, within the profiled data object, for which data profiling determined the aggregation measures |
| Minimum | Minimum value with respect to the inherent database ordering for the column |
| Maximum | Maximum value with respect to the inherent database ordering of the column |
| # Distinct | Total number of distinct values for the column |
| % Distinct | Percentage of distinct values in the column over the entire row set |
| Not Null | Indicates if a NOT NULL constraint is defined in the database for the column |
| Recommended NOT NULL | Indicates if data profiling results recommend that the column allow null values. A value of Yes represents a recommendation that this column should not allow null values |

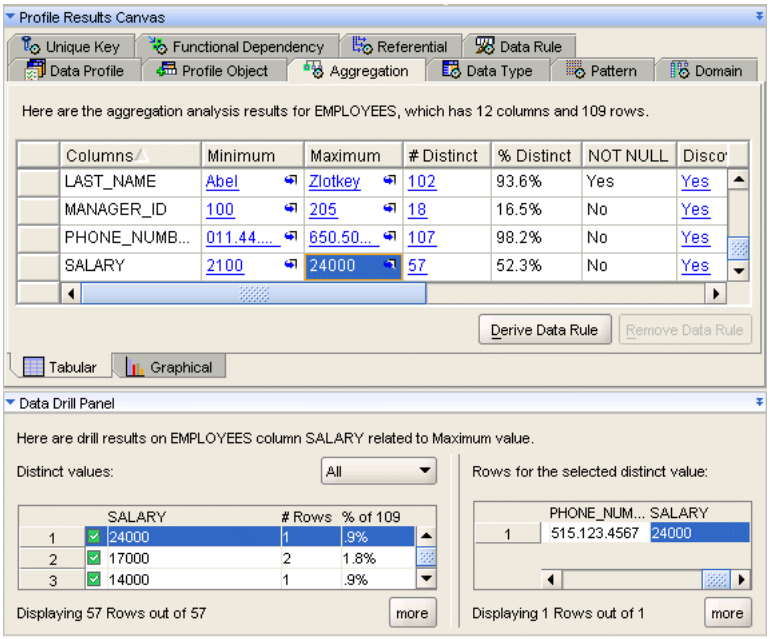
*Table 18–3 (Cont.) Aggregation Results*

| Measurement | Description |
|---|---|
| # Nulls | Total number of null values for the column |
| %Nulls | Percentage of null values, for the column, over the entire row set |
| Six Sigma | Number of null values (defects) to the total number of rows in the table (opportunities) |
| Average | Average value, for the column, in the entire row set |
| Median | Median value, for the column, in the entire row set |
| Std Dev | Standard deviation for the column |

A hyperlinked value in the aggregation results grid indicates that you can click the value to drill down into the data. This enables you to analyze the data in the sample that produced this result.

For example, if you scroll to the SALARY column, shown in Figure 18–2, and click the value in the Maximum cell showing 24000, the Data Drill Panel at the bottom changes to show you all the distinct values in this column with counts on the left. On the right, the Data Drill can zoom into the value that you select from the distinct values and display the full record where these values are found.

*Figure 18–2 Aggregation Tabular Results*




The graphical analysis displays the results in a graphical format. You can use the graphical toolbar to change the display. You can also use the Column and Property menus to change the displayed data object.

## Data Type

The Data Type tab provides profiling results about data types. This includes metrics such as length for character data types and the precision and scale for numeric data types. For each data type that is discovered, the data type is compared to the dominant

data type found in the entire attribute and the percentage of rows that comply with the dominant measure is listed.

One example of data type profiling would be finding a column defined as `VARCHAR` that stores only numeric values. Changing the data type of the column to `NUMBER` would make storage and processing more efficient.

Table 18–4 describes the various measurement results available in the Data Type tab.

***Table 18–4 Data Type Results***

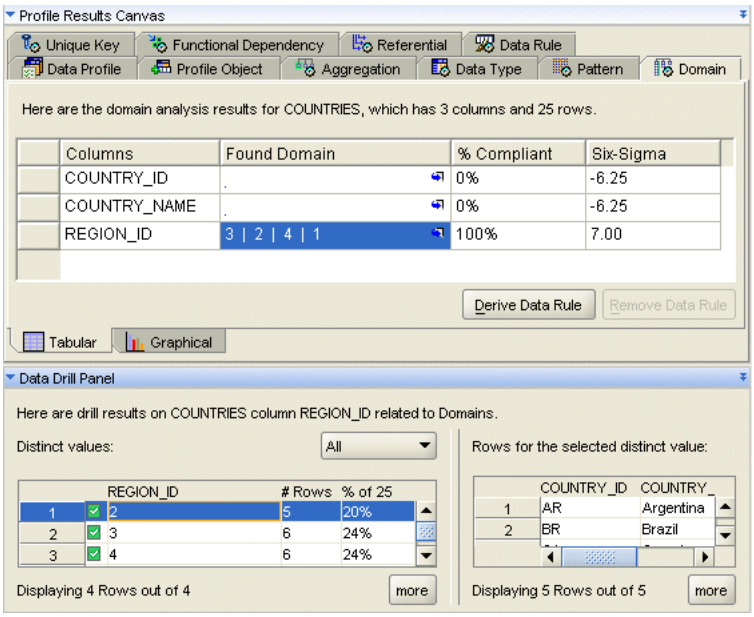
| Measurement | Description |
|---|---|
| Columns | Name of the column, within the data object, for which data type analysis was performed |
| Documented Data Type | Data type of the column in the source object |
| Dominant Data Type | From analyzing the column values, data profiling determines that this is the dominant (most frequent) data type. |
| % Dominant Data Type | Percentage of total number of rows where column value has the dominant data type |
| Documented Length | Length of the data type in the source object |
| Minimum Length | Minimum length of the data stored in the column |
| Maximum Length | Maximum length of the data stored in the column |
| Dominant Length | From analyzing the column values, data profiling determines that this is the dominant (most frequent) length. |
| % Dominant Length | Percentage of total number of rows where column value has the dominant length |
| Documented Precision | Precision of the data type in the source object |
| Minimum Precision | Minimum precision for the column in the source object |
| Maximum Precision | Maximum precision for the column in the source object |
| Dominant Precision | From analyzing the column values, data profiling determines that this is the dominant (most frequent) precision |
| % Dominant Precision | Percentage of total number of rows where column value has the dominant precision |
| Documented Scale | Scale specified for the data type in the source object |
| Minimum Scale | Minimum scale of the data type in the source object |
| Maximum Scale | Maximum scale of the data type in the source object |
| Dominant Scale | From analyzing the column values, data profiling determines that this is the dominant (most frequent) scale. |
| % Dominant Scale | Percentage of total number of rows where column value has the dominant scale |

### Domain

The Domain tab displays results about the possible set of values that exist in a certain attribute. Information can be viewed from either the Tabular or Graphical subtabs.

Figure 18–3 displays the Domain tab of the Data Profile Editor.

***Figure 18–3 Domain Discovery Results***




The process of discovering a domain on a column involves two phases. First, the distinct values in the column are used to determine whether that column might be defined by a domain. Typically, there are few distinct values in a domain. Then, if a potential domain is identified, the count of distinct values is used to determine whether that distinct value is compliant with the domain. The properties that control the threshold for both phases of domain discovery can be set in the Property Inspector.

If you find a result that you want to know more about, drill down and use the Data Drill panel to view details about the cause of the result.

For example, a domain of four values was found for the column REGION_ID: 3,2,4, and 1. To see which records contributed to this finding, select the REGION_ID row and view the details in the Data Drill panel.

Table 18–5 describes the various measurement results available in the Domain tab.

***Table 18–5 Domain Results***

| Measurement | Description |
|---|---|
| Column | Name of the column for which domain discovery was performed |
| Found Domain | The discovered domain values |
| % Compliant | The percentage of all column values that are compliant with the discovered domain values |
| Six Sigma | The Six Sigma value for the domain results |

## Pattern

The Pattern tab displays information discovered about patterns within the attribute. Pattern discovery is the profiler's attempt at generating regular expressions for data that it discovered for a specific attribute. Note that non-English characters are not supported in the pattern discovery process.

Table 18–6 describes the various measurement results available in the Pattern tab.

*Table 18–6 Pattern Results*

| Measurement | Description |
|---|---|
| Columns | Name of the column for which pattern results were discovered |
| Dominant Character Pattern | The most frequently discovered character pattern or consensus pattern |
| % Compliant | The percentage of rows whose data pattern agrees with the dominant character pattern |
| Dominant Word Pattern | The most frequently discovered word character pattern or consensus pattern |
| % Compliant | The percentage of rows whose data pattern agrees with the dominant word pattern |
| Common Format | Name, Address, Date, Boolean, Social Security Number, E-mail, URL. This is the profiler's attempt to add semantic understanding to the data that it sees. Based on patterns and some other techniques, it will try to determine which domain bucket a certain attribute's data belongs to. |
| % Compliant | The percentage of rows whose data pattern agrees with the consensus common format pattern |

## Unique Key

The Unique Key tab provides information about the existing unique keys that were documented in the data dictionary, and possible unique keys or key combinations that were detected by the data profiling operation. The uniqueness % is shown for each. The unique keys that have No in the Documented column are the ones that are discovered by data profiling.

For example, a phone number is unique in 98% of the records. It can be a unique key candidate, and you can then cleanse the noncompliant records. You can also use the drill-down feature to view the cause of the duplicate phone numbers in the Data Drill panel. Table 18–7 describes the various measurement results available in the Unique Key tab.

*Table 18–7 Unique Key Results*

| Measurement | Description |
|---|---|
| Unique Key | The discovered unique key |
| Documented | Indicates if a unique key on the column exists in the data dictionary.<br>A value of Yes indicates that a unique key exists in the data dictionary. A value of No indicates that the unique key was discovered as a result of data profiling. |
| Discovered | From analyzing the column values, data profiling determines whether a unique key should be created on the column listed in the Local Attribute(s) column. |
| Local Attribute(s) | The name of the column in the data object that was profiled |
| # Unique | The number of rows, in the source object, in which the attribute represented by Local Attribute is unique |
| % Unique | The percentage of rows, in the source object, for which the attribute represented by Local Attribute is unique |
| Six Sigma | Number of null values (defects) to the total number of rows in the table (opportunities) |

### Functional Dependency

The Functional Dependency tab displays information about the attribute or attributes that seem to depend on or determine other attributes. Information can be viewed from either the Tabular or Graphical subtabs. You can use the **Show** list to change the focus of the report. Note that unique keys defined in the database are not discovered as functional dependencies during data profiling.

Table 18–8 describes the various measurement results available in the Functional Dependency tab.

***Table 18–8 Functional Dependency Results***

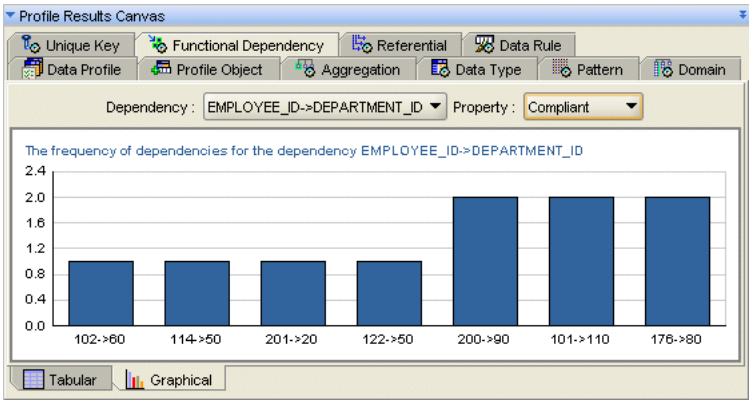
| Measurement | Description |
|---|---|
| Determinant | Name of the attribute that is found to determine the attribute listed under Dependent |
| Dependent | Name of the attribute that is found to be determined by the value of another attribute |
| # Defects | Number of values in the Determinant attribute that were not determined by the Dependent attribute |
| % Compliant | Percentage of values that are compliant with the discovered dependency |
| Six Sigma | Six Sigma value |
| Type | Type of functional dependency. Possible values are unidirectional or bidirectional |

For example, if you select Only 100% dependencies from the **Show** list, the information shown is limited to absolute dependencies. If you have an attribute that is always dependent on another attribute, then it is recommended that it be a candidate for a reference table. Suggestions are shown in the Type column. Removing the attribute into a separate reference table normalizes the schema.

The Functional Dependency tab also has a Graphical subtab so that you can view information graphically. You can select a dependency and property from the lists to view graphical data.

For example, in Figure 18–4, you select the dependency where `EMPLOYEE_ID` seems to determine `DEPARTMENT_ID` (`EMPLOYEE_ID` -> `DEPARTMENT_ID`). Warehouse Builder determines that most `EMPLOYEE_ID` values determine `DEPARMENT_ID`. By switching the Property to Non-Compliant, you can view the exceptions to this discovery.

***Figure 18–4 Graphical Functional Dependency***



### Referential

The Referential tab displays information about foreign keys that were documented in the data dictionary, as well as relationships discovered during profiling. For each relationship, you can see the level of compliance. Information can be viewed from both the Tabular and Graphical subtabs. In addition, two other subtabs are available only in the Referential tab: Joins and Redundant Columns.

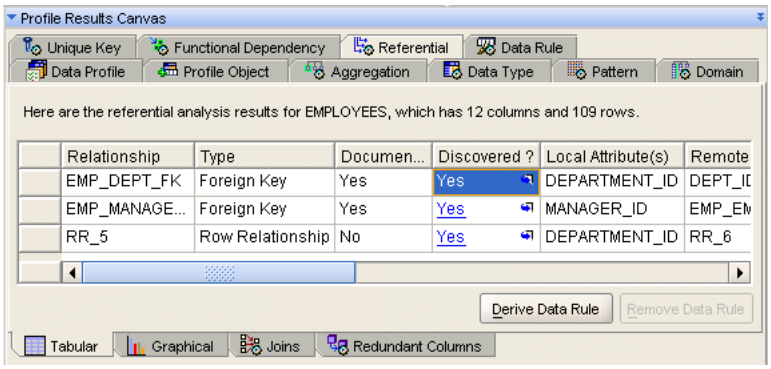
Table 18–9 describes the various measurement results available in the Referential tab.

***Table 18–9 Referential Results***

| Measurement | Description |
|---|---|
| Relationship | Name of the relationship |
| Type | Type of relationship. The possible values are Row Relationship and Foreign Key. |
| Documented | Indicates if a foreign key exists on the column in the data dictionary<br>Yes indicates that a foreign key on the column exists in the data dictionary. No indicates that the foreign key was discovered as a result of data profiling. |
| Discovered | From analyzing the column values, data profiling determines whether a foreign key should be created on the column represented by Local Attribute(s). |
| Local Attribute(s) | Name of the attribute in the source object |
| Remote Key | Name of the key in the referenced object to which the local attribute refers |
| Remote Attribute(s) | Name of the attributes in the referenced object |
| Remote Relation | Name of the object that the source object references |
| Remote Module | Name of the module that contains the referenced object |
| Cardinality Range | Range of the cardinality between two attributes.<br>For example, the `EMP` table contains 5 rows of employee data. There are two employees each in department 10 and 20 and one employee in department 30. The `DEPT` table contains three rows of department data with deptno value 10, 20, and 30.<br>Data profiling will find a row relationship between the `EMP` and `DEPT` tables. The cardinality range will be 1-2:1-1. This is because in `EMP`, the number of rows per distinct value ranges from 1 (for deptno 30) to 2 (deptno 10 and 20). In `DEPT`, there is only one row for each distinct value (10, 20, and 30). |
| # Orphans | Number of orphan rows in the source object |
| % Compliant | Percentage of values that are compliant with the discovered dependency |
| Six Sigma | Number of null values (defects) to the total number of rows in the table (opportunities) |

For example, you are analyzing two tables for referential relationships: the `Employees` table and the `Departments` table. Using the Referential data profiling results shown in Figure 18–5, you discover that the `DEPARTMENT_ID` column in the `Employees` table is related to `DEPARTMENT_ID` column in the `Departments` table 98% of the time. You can then click the hyperlinked Yes in the Discovered column to view the rows that did not comply with the discovered foreign key relationship.

***Figure 18–5 Referential Results***



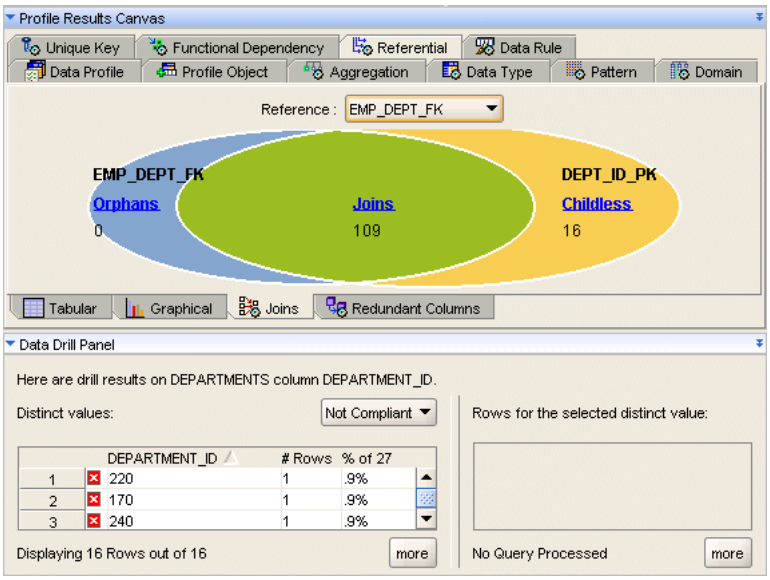

You can also select the Graphical subtab to view information graphically. This view is effective for viewing noncompliant records, such as orphans. To use the Graphical subtab, make a selection from the **Reference** and **Property** lists.

The Joins subtab displays a join analysis on the reference selected in the Reference list. The results show the relative size and exact counts of the three possible outcomes for referential relationships: joins, orphans, and childless objects.

For example, both the `EMPLOYEES` and `DEPARTMENTS` tables contain a `DEPARTMENT_ID` column. There is a one-to-many relationship between the `DEPARTMENT_ID` column in the `DEPARTMENTS` table and the `DEPARTMENT_ID` column in the `EMPLOYEES` table. The Joins represent the values that have values in both tables. Orphans represent values that are only present in the `EMPLOYEES` table and not the `DEPARTMENTS` table. And Childless values are present in the `DEPARTMENTS` table and not the `EMPLOYEES` table. You can drill into values on the diagram to view more details in the Data Drill panel.

Figure 18–6 displays the Joins subtab of the Referential tab.

***Figure 18–6 Join Results***




The Redundant Columns subtab displays information about columns in the child table that are also contained in the primary table. Redundant column results are only available when perfectly unique columns are found during profiling.

For example, consider two tables EMP and DEPT, shown in Table 18–10 and Table 18–11, having the following foreign key relationship: EMP.DEPTNO (fk) = DEPT.DEPTNO (uk).

*Table 18–10 EMP Table*

| Employee Number | Dept. No | Location |
|---|---|---|
| 100 | 1 | CA |
| 200 | 2 | NY |
| 300 | 3 | MN |
| 400 | 1 | CA |
| 500 | 1 | CA |

*Table 18–11 DEPT Table*

| Dept No | Location | Zip |
|---|---|---|
| 1 | CA | 94404 |
| 3 | NY | 10022 |
| 3 | MN | 21122 |

In this example, the Location column in the EMP table is a redundant column, because you can get the same information from the join.

### Data Rule

The Data Rule tab displays the data rules that are defined as a result of data profiling for the table selected in the object tree. The details for each data rule include the following:

- **Rule Name:** Represents the name of the data rule.
- **Rule Type:** Provides a brief description about the type of data rule.
- **Origin:** Represents the origin of the data rule. For example, a value of Derived indicates that the data rule is derived.
- **% Compliant:** Percent of rows that comply with the data rule.
- **# Defects:** Number of rows that do not comply with the data rule.

The data rules on this tab reflect the active data rules in the Data Rule panel. You do not directly create data rules on this tab.

## Using Attribute Sets to Profile a Subset of Columns from a Data Object

Attribute sets enable you to restrict a data profiling operation to a subset of columns from a table, view, or materialized view.

Reasons to use an attribute set include:

- You can decrease profiling time by excluding columns for which you do not need profiling results.
- Data profiling can only profile up to 165 columns from a table, view, or materialized view at a time. You can use an attribute set to select a set of 165 or fewer columns to profile from the object.

Data profiling using attribute sets, consists of the following high-level steps:

1. Defining Attribute Sets
2. Creating a Data Profile That Contains the Attribute Set
3. Performing data profiling

   See "Profiling Data" on page 18-10 for more information about profiling data.
4. Reviewing data profiling results

   See "Viewing Profile Results" on page 18-11 for more information about viewing profiling results.

### Defining Attribute Sets

Use the following steps to define an attribute set in a table, view, or materialized view.

1. In the Projects Navigator, double-click the table, view, or materialized view.

   The editor for the selected object is opened.
2. Select the Attribute Sets tab.
3. In the Attribute Sets section, click a blank area in the Name column, enter the name of the attribute set you want to create, and press the Enter key.
4. Select the name of the attribute set created in Step 3.

   The Attributes of the selected attribute set section displays the attributes in the data object.
5. Select **Include** for all the attributes that you want included in the attribute set.
6. Save your changes by clicking **Save** on the toolbar.

### Creating a Data Profile That Contains the Attribute Set

1. In the Projects Navigator, right-click the Data Profiles node and select **New Data Profile**.

   The Welcome page of the Create Data Profile Wizard is displayed.
2. On the Welcome page, click **Next**.
3. On the Name and Description page, enter a name and an optional description for the data profile. Click **Next**.
4. On the Select Objects page, select the data object that you want to profile and use the shuttle arrows to move the data object to the Selected list.

   When the selected data object contains attribute sets, the Choose Attribute Set dialog box is displayed.
5. On the Choose Attribute Set dialog box, select the attribute set that you want to profile and click **OK**.
6. On the Select Objects page, click **Next**.
7. On the Summary page, review the options you chose on the previous wizard pages and click **Finish**.

   The data profile is created and added to the navigator tree.

## Editing Data Profiles

Once you create a data profile, you can use the Data Profile Editor to modify its definition. You can either modify profile settings or add to and remove from a data profile. To add objects, you can use either the menu bar options or the Select Objects tab of the Edit Data Profile dialog box.

**To edit a data profile:**

1. In the Projects Navigator, right-click the data profile and select **Open**.

   The Data Profile Editor is displayed.

2. From the **Edit** menu, select **Properties**.

   The Edit Data Profile dialog box is displayed.

3. On the Edit Data Profile dialog box, edit any of the following properties of the data profile and click **OK**.

   - To modify the name or description of the data profile, on the Name tab, select the name or description and enter the new value.
   - To add or remove objects, on the Select Objects tab, use the arrows to add objects to or remove objects from the data profile.
   - To change the location that is used as the data profiling staging area, use the Data Locations tab.

     Use the arrows to move the new location to the Selected Locations section. Ensure that you select **New Configuration Default** to set this location as the default profiling location for the data profile.

     **Note:** If you modify the profiling location after you have performed data profiling, the previous profiling results are lost.

### Adding Data Objects to a Data Profile

**To add data objects to a data profile:**

1. Right-click the data profile in the Projects Navigator and select **Open**.

   The Data Profile Editor is displayed.

2. From the **Profile** menu, select **Add**.

   The Add Profile Tables dialog box is displayed.

3. On the Add Profile Tables dialog box, select the objects that you want to add to the data profile. Use the arrows to move them to the Selected section.

   You can select multiple objects by holding down the Ctrl key and selecting objects.

# Tuning the Data Profiling Process for Better Profiling Performance

Data profiling is a processor and I/O intensive process, and the execution time for profiling ranges from a few minutes to a few days. You can achieve the best possible data profiling performance by ensuring that the following conditions are satisfied:

- Your database is set up correctly for data profiling.
- The appropriate data profiling configuration parameters are used when you perform data profiling.

## Tuning the Data Profile for Better Data Profiling Performance

You can configure a data profile to optimize data profiling results. Use the configuration parameters to configure a data profile.

> **See Also:** "Configuring Data Profiles" on page 18-7 for details about configuring data profiles and the configuration parameters that you can set

Use the following guidelines to make your data profiling process faster:

- Perform only the types of analysis that you require.

  If you know that certain types of analysis are not required for the objects that you are profiling, use the configuration parameters to turn off these types of data profiling.

- Limit the amount of data profiled.

  Use the `WHERE` clause and Sample Rate configuration parameters.

If the source data for profiling is stored in an Oracle Database, it is recommended that the source schema should be located on the same database instance as the profile workspace. You can do this by installing the profiling workspace into the same Oracle Database instance as the source schema location. This avoids using a database link to move data from source to profiling workspace.

## Tuning the Oracle Database for Better Data Profiling Performance

To ensure good data profiling performance, the computer that runs Oracle Database must have certain hardware capabilities. In addition to this, you must optimize the Oracle Database instance on which you are performing data profiling.

For efficient data profiling, consider the following resources:

- Multiple Processors
- Memory
- I/O System

### Multiple Processors

The computer that runs Oracle Database needs multiple processors. Data profiling has been designed and tuned to take maximum advantage of the parallelism provided by Oracle Database. While profiling large tables (more than 10 million rows), it is highly recommended that you use a multiple processor computer.

Hints are used in queries required to perform data profiling. It picks up the degree of parallelism from the initialization parameter file of the Oracle Database. The default initialization parameter file contains parameters that take advantage of parallelism.

### Memory

It is important that you ensure a high memory hit ratio during data profiling. You can ensure this by assigning a larger size of the System Global Area. Oracle recommends that the size of the System Global Area be at least 500 megabytes (MB). If possible, configure it to 2 gigabytes (GB) or 3 GB.

For advanced database users, Oracle recommends that you observe the buffer cache hit ratio and library cache hit ratio. Set the buffer cache hit ratio to higher than 95% and the library cache hit ratio to higher than 99%.

### I/O System

The capabilities of the I/O system have a direct impact on the data profiling performance. Data profiling processing frequently performs full table scans and massive joins. Because today's CPUs can easily outdrive the I/O system, you must carefully design and configure the I/O subsystem. Consider the following considerations that aid better I/O performance:

- You need a large number of disk spindles to support uninterrupted CPU and I/O cooperation. If you have only a few disks, then the I/O system is not geared towards a high degree of parallel processing. It is recommended to have a minimum of two disks for each CPU.
- Configure the disks. Oracle recommends that you create logical stripe volumes on the existing disks, each striping across all available disks. Use the following formula to calculate the stripe width:

  ```
  MAX(1,DB_FILE_MULTIBLOCK_READ_COUNT/number_of_disks) X DB_
  BLOCK_SIZE
  ```

  Here, `DB_FILE_MULTIBLOCK_SIZE` and `DB_BLOCK_SIZE` are parameters that you set in your database initialization parameter file. You can also use a stripe width that is a multiple of the value returned by the formula.

  To create and maintain logical volumes, you need volume management software such as Veritas Volume Manager or Sun Storage Manager. If you are using Oracle Database 10*g* or a later and you do not have any volume management software, then you can use the Automatic Storage Management feature of Oracle Database to spread the workload to disks.
- Create different stripe volumes for different tablespaces. It is possible that some of the tablespaces occupy the same set of disks.

  For data profiling, the USERS and the TEMP tablespaces are normally used at the same time. Consider placing these tablespaces on separate disks to reduce I/O contention.

# Data Watch and Repair (DWR) for Oracle Master Data Management (MDM)

Data Watch and Repair (DWR) is a profiling and correction solution designed to assist data governance in Oracle's Master Data Management (MDM) solutions. MDM applications need to provide a single consolidated view of data. To do this, they need to first clean up a system's master data before they can share it with multiple connected entities.

Warehouse Builder provides data profiling and data correction functionality that enables MDM applications to cleanse and consolidate data. You can use DWR for the following MDM applications:

- Customer Data Hub (CDH)
- Product Information Management (PIM)
- Universal Customer Master (UCH)

## Overview of Data Watch and Repair (DWR) for MDM

Data Watch and Repair (DWR) enables you to analyze, cleanse, and consolidates data stored in MDM databases using the following:

- Data profiling

Data profiling data analysis method that enables you to detect and measure defects in your source data.

For more information about data profiling, see "Overview of Data Profiling" on page 18-1.

- Data rules

  Data rules are help ensure data quality by determining the legal data and relationships in the source data. You can import MDM-specific data rules, define your own data rules before you perform data profiling, or derive data rules based on the data profiling results.

  For more information about data rules, see "Overview of Data Rules" on page 19-1.

- Data correction

  Data correction enables you to correct any inconsistencies, redundancies, and inaccuracies in both the data and metadata. You can automatically create correction mappings to cleanse source data.

  For more information about data correction, see "Overview of Data Rules in ETL and Automatic Data Correction" on page 21-1.

DWR enables you to measure crucial business rules regularly. As you discover inconsistencies in the data, you can define and apply new data rules to ensure data quality.

### Predefined Data Rules for MDM

Warehouse Builder provides a set of data rules that are commonly used in MDM applications. These include the following customer data rules that can be used in both Customer Data Hub (CDH) and Universal customer Master (UCM) applications:

- Attribute Completeness
- Contact Completeness
- Data Type
- Data Domain
- Restricted Values
- Unique Key Discovery
- Full Name Standardization
- Common Pattern
- Name Capitalization
- Extended Phone Numbers
- International Phone Numbers
- No Access List by Name Only
- No Access List by Name or SSN
- No Email List

For more details about these data rules, refer to the *Oracle Watch and Repair for MDM User's Guide*.

## Prerequisites for Performing Data Watch and Repair (DWR)

To use Data Watch and Repair (DWR), you need the following software:

- Oracle Database 11g Release 1 (11.1) or later
- Oracle Warehouse Builder 11g Release 1 (11.1.0.7) or later
- One or more of the following Master Data Management (MDM) applications: Customer Data Hub (CDH), Product Information Management (PIM), or Universal Customer Master (UCH)

For MDM applications that run on an Oracle Database, you can directly use DWR. However, for MDM applications that do not run on an Oracle Database, you must set up a gateway with the third-party database.

## Steps to Perform Data Watch and Repair (DWR) Using Warehouse Builder

Use the following steps to perform Data Watch and Repair (DWR).

1. Create a location corresponding to the Master Data Management (MDM) application database.

   Use the Oracle node under the Databases node in the Locations Navigator. Specify the details of the MDM database such as the user name, password, host name, port, service name, and database version.

2. In the Projects Navigator, expand the Applications node to display the nodes for the MDM applications.

   The CDH node represents Customer Data Hub application, the PIM node to the Product Information Management application, and UCM to Universal Customer Master.

3. Right-click the node corresponding to the type of MDM application for which you want to perform DWR and select Create CMI Module.

   Use the Create Module Wizard to create a module that stores your MDM metadata definitions. Ensure that you select the location you created in step 1 while creating the module.

4. Import metadata from your MDM application into the module created in step 3. Right-click the module and select Import.

   The Metadata Import Wizard is displayed that enables you to import MDM metadata.

5. Import data rules specific to MDM as described in "Importing MDM Data Rules" on page 18-27.

6. Apply data rules to the MDM application tables as described in "Applying Data Rules to Data Objects" on page 19-7.

   Applying data rules to tables enables you to determine if your table data complies with the business rules defined using data rules. You can apply data rules you imported in step 5 or other data rules that you created.

   For more information about creating data rules, see "Creating Data Rules Using the Create Data Rule Wizard" on page 19-5.

7. Create a data profile that includes all tables from the MDM application that you want to profile.

   For more information about creating data profiles, see "Creating Data Profiles" on page 18-6.

8. Perform data profiling on the MDM application objects as described in "Profiling Data" on page 18-10.
9. View the data profiling results as described in "Viewing Profile Results" on page 18-11.
10. (Optional) Derive data rules based on data profiling results as described in "Deriving Data Rules From Data Profiling Results" on page 19-4.

    Data rules derived from data profiling results are automatically applied to the table.
11. Create correction mappings as described in "Steps to Create Correction Objects" on page 21-2.
12. Correct data and metadata using the correction mappings generated by Warehouse Builder as described in "Deploying Schema Corrections" on page 21-8 and "Deploying Correction Mappings" on page 21-8.
13. Write the corrected data, stored in the correction objects created in step 12, to the MDM application as described in "Writing Corrected Data and Metadata to the MDM Application" on page 18-27.

### Importing MDM Data Rules

Data rules required for Customer Data Hub (CDH) and Universal Customer Master (UCM) applications are provided in the `OWB_ORACLE_HOME/owb/misc/dwr/customer_data_rules.mdl` file. To import these data rules, from the File menu, select **Import**, then **Warehouse Builder Metadata**. In the Metadata Import dialog box, select the `customer_data_rules.mdl` and click **OK**. For more information about using the Metadata Import dialog, click Help on this page.

The imported data rules are listed in the Globals Navigator, in the MDM Customer Data Rules node under the Public Data Rules node.

### Writing Corrected Data and Metadata to the MDM Application

The cleansed and corrected data is contained in the correction objects created as a result of data profiling.

To be more efficient, you can write back only those rows that must be corrected. You can achieve this by modifying the generated correction mapping. Delete the branch that passes through the compliant rows unchanged (this is the branch that contains the minus filter and the minus set operators). Retain only the corrected rows processing branch in the correction mapping.

Use the following steps to write corrected data to the source MDM application:

1. Create a mapping using the Mapping Editor.
2. Drag and drop the corrected table on to the Mapping Editor. This represents the source table.
3. For UCM, drag and drop the interface table that corresponds to the base table with which you are working.

   Use the MDM application tools and documentation to determine the base table for a particular interface table.
4. Map the columns from the corrected table to the interface table.
5. Deploy and execute the mapping to write corrected data to the source MDM application.

6. Update the base table with changes made to the interface table. You can use Siebel Enterprise Integration Manager (EIM). EIM can be run using the command line or from a Graphical User Interface (GUI).

   For more details about using the EIM, see *Siebel Enterprise Integration Manager Administration Guide*.

# 19

# Designing and Deriving Data Rules

This chapter describes Oracle Warehouse Builder data rules and their applications, and describes how to design data rules and derive them from data profiling results.

This chapter contains the following topics:


- Overview of Data Rules
- Using Data Rules


## Overview of Data Rules

Data rules are definitions for valid data values and relationships that can be created in Warehouse Builder. They can be applied to tables, views, materialized views, and external tables. They determine legal data within a table (or other object) or legal relationships between data in different columns of a table or different tables.

Data rules are central to the data quality features of Warehouse Builder.

- After data profiling, data rules can be automatically generated based upon any discovered data relationships. Data profiling can also test data against specific data rules.
- Data auditors use data rules to test data for compliance, and generate statistics about noncompliant data.
- Automatically generated data cleansing mappings use data rules as the basis of determining which data is noncompliant. Selecting a data rule and the cleansing strategy to apply for noncompliant data determines the content of the cleansing mapping.
- Schema cleansing translates data rules bound to an object into constraints defined on the cleansed schema.
- In ETL mappings, if a table or other object has one or more data rules applied in the context of the mapping, then Warehouse Builder automatically creates and manages error tables and other logic to manage and audit noncompliant rows.

There are two ways to create a data rule:

- Derive one or more rules from data profiling results, as described in "Deriving Data Rules From Data Profiling Results" on page 19-4
- Define a data rule directly, as described in "Creating Data Rules Using the Create Data Rule Wizard" on page 19-5

Every Warehouse Builder workspace also has a certain number of predefined public data rules, accessible in all projects in the workspace, for common conditions such as testing for non-null values, testing for common data formats, and so on.

The metadata for a data rule is stored in the workspace. To use a data rule, you apply the data rule to a data object. For example, you create a data rule called `gender_rule` that specifies that valid values are 'M' and 'F'. You can apply this data rule to the `emp_gender` column of the `Employees` table. Applying the data rule ensures that the values stored for the `emp_gender` column are either 'M' or 'F'. You can view the details of the data rule bindings on the Data Rule tab of the Table Editor for the `Employees` table.

## Types of Data Rules

Table 19–1 describes the types of data rules.

***Table 19–1 Types of Data Rules***

| Data Rule Type | Description | Example |
|---|---|---|
| Domain List | Defines a list of values that an attribute is allowed to have | The Gender attribute can have "M" or "F". |
| Domain Pattern List | Defines a list of patterns that an attribute is allowed to conform to. The patterns are defined in the Oracle Database regular expression syntax. | A pattern for telephone number is:<br>(^[[:space:]]*[0-9]{ 3 }[[:punct:]\|:space:]]?[0-9]{ 4 }[[:space:]]*$) |
| Domain Range | Defines a range of values that an attribute is allowed to have | The value of the Salary attribute can be between 100 and 10000. |
| Common Format | Defines a known common format that an attribute is allowed to conform to<br>This rule type has many subtypes: Telephone Number, IP Address, SSN, URL, E-mail Address. Each type has predefined formats listed. You can add more formats to this list. | An E-mail address should be in the following format:<br>^(mailto:[-_a-z0-9.]+@[-_a-z0-9.]+$) |
| No Nulls | Specifies that the attribute cannot have null values | The `department_id` column for an employee in the `Employees` table cannot be null. |
| Functional Dependency | Defines that the data in the data object may be normalized | The `Dept_name` attribute is dependent on the `Dept_no` attribute. |
| Unique Key | Defines whether an attribute or group of attributes are unique in the given data object | The name of a department must be unique. |
| Referential | Defines the type of relationship (1:n) a value must have with another value | The `department_id` column of the `Departments` table must have a 1:n relationship with the `Department_id` column of the `Employees` table. |
| Name and Address | Uses the Name and Address support to evaluate a group of attributes as a name or address | |

*Table 19–1 (Cont.) Types of Data Rules*

| Data Rule Type | Description | Example |
|---|---|---|
| Custom | Applies a SQL expression that you specify to its input parameters | A custom rule called VALID_DATE has two input parameters, START_DATE and END_DATE. A valid expression for this rule is defined as follows:<br>"THIS"."END_DATE" > "THIS"."START_DATE". |

## Data Rules as Objects and Binding Data Rules

Data rules are objects in each workspace, independent of individual data elements that a rule may govern. For example, the rule "No Nulls" exists independent of any particular column that the rule is applied to, and a Common Format rule such as Email Address is independent of any specific column in any specific table which is subject to that rule.

In the Projects Navigator, the Data Rules node for a project contains Data Rule folders, which group one or more data rules. There are also public data rules defined for all projects in a workspace.

Data rules have input parameters that identify the objects to which the rules are applied in a given instance. To bind a data rule is to associate that data rule with particular data objects that the rule governs, such as one or more columns in one or several tables. Note that a data rule can be bound multiple times, to several columns in a single table or across multiple tables.

For example, the `gender_rule` domain list rule that limits a column to the values M and F can be applied to several different columns in a table or even several different tables, such as `EMPLOYEES.emp_gender`, `EMPLOYEES.manager_gender`, and `CHILDREN.child_gender`. In such a case, if the rule is updated, all places where the rule is bound are affected. The next time you generate and deploy code using that data rule, the updated data rule definition is used. For example, if you changed the gender_rule to accept X to indicate unknown gender, then you can enforce that rule change everywhere by regenerating ETL for the objects to which the gender_rule is bound.

# Using Data Rules

In addition to deriving data rules based on the results of data profiling, you can define your own data rules. You can bind a data rule to multiple tables within the project in which the data rule is defined. An object can contain any number of data rules.

Use the Design Center to create and edit data rules. Once you create a data rule, you can use it in any of the following scenarios.

### Using Data Rules in Data Profiling

When you are using data profiling to analyze tables, you can use data rules to analyze how well data complies with a given rule, and to collect statistics. From the results, you can derive a new data rule. If data profiling determines that the majority of records have a value of red, white, and blue for a particular column, then a new data rule can be derived that defines the color domain (red, white, and blue). This rule can then be reused to profile other tables, or used like other rules in schema correction, data cleansing, and data auditing.

### Using Data Rules in Schema Correction

Data rules can be used to convert a source schema into a new target schema where the structure of the new tables strictly adheres to the data rules. In the new schema, table columns have data types consistent with the data rules, constraints based upon the rules are generated and applied to the tables and enforced, and schemas are normalized.

### Using Data Rules in Data Cleansing

The second way that data rules are used is in a correction mapping that validates the data in a source table against the data rules, to determine which records comply and which do not. The analyzed data set is then corrected (for example, orphan records are removed, domain value inaccuracies are corrected, and so on) and the cleansed data set is loaded into the corrected target schema.

### Using Data Rules in Data Auditing

Data rules are also used in data auditing. Data auditors are processes that validate data against a set of data rules to determine which records comply and which do not. Data auditors gather statistical metrics on how well the data in a system complies with a rule, and they report defective data into auditing and error tables. In that sense they are like data-rule-based correction mappings, which also offer a report-only option for data that does not comply with the data rules.

## Managing Data Rules in Folders

Each data rule belongs to a data rule folder, which is a container object that groups related data rules. Before you can create any data rules, you must first create at least one data rule folder.

To create a data rule folder, in the navigation tree, right-click **Data Rules** and select **New Data Rule Folder**. The Create Data Rule Folder dialog box is displayed. Enter a name for the new data rule folder and click **OK**.

## Deriving Data Rules From Data Profiling Results

Based on the results of data profiling, you can derive data rules that can be used to cleanse your data. Although you can create data rules and apply them manually to your data profile, deriving data rules based on data profiling results enhances productivity and ensures that your rules do reflect the underlying data.

A data rule is an expression that determines the set of legal data that can be stored within a data object. Use data rules to ensure that only values compliant with the data rules are allowed within a data object. Data rules will form the basis for correcting or removing data if you decide to cleanse the data. You can also use data rules to report on noncompliant data.

For example, you have a table called Employees with the following columns: `Employee_Number`, `Gender`, `Employee_Name`. The profiling result shows that 90% of the values in the Employee_Number column are unique, making it a prime candidate for the unique key. The results also show that 85% of the values in the `Gender` column are either 'M' or 'F', making it a good candidate for a domain. You can then derive these rules directly from the Profile Results Canvas.

### Steps to Derive Data Rules

1. Select a data profile in the navigation tree, right-click, and select **Open**.

   The Data Profile Editor is displayed with the profiling results.

2. Review the profiling results and determine the findings from which you want to derive data rules.

   The types of results that warrant data rules vary. Some results commonly derived into data rules include a detected domain, a functional dependency between two attributes, or a unique key.

3. Select the tab that displays the results from which you want to derive a data rule.

   For example, to create a data rule that enforces a unique key rule for the `EMPLOYEE_NUMBER` column, navigate to the Unique Key tab.

4. Select the cell that contains the results from which you want to derive a data rule.

   You can define a data rule on a cell that contains a blue arrow icon. If the cell contains a green arrow icon, then a data rule has already been defined for the column represented in that cell

5. From the Profile menu select **Derive Data Rule**. Or click the **Derive Data Rule** button.

   For example, to create a Unique Key rule on the EMPLOYEE_NUMBER column, select this column and click **Derive Data Rule**.

   The Derive Data Rule Wizard is displayed.

6. On the Welcome page, click **Next**.

7. On the Name and Description page, the Name field displays a default name for the data rule. To specify a new name, select the name, enter the new name, and click **Next**.

8. On the Define Rule page, provide details about the data rule parameters and click **Next**.

   The Type field is automatically populated depending on the type of data being derived. You cannot edit the type of data rule.

   Additional fields in the lower portion of this page define the parameters for the data rule. Some of these fields are populated with values based on the result of data profiling. The number and type of fields depend on the type of data rule.

9. On the Summary page, review the options that you set in the wizard using this page. Click **Back** to change any of the selected values. Click **Finish** to create the data rule.

   The data rule is created and it appears in the Data Rule panel of the Data Profile Editor. The derived data rule is also added to the Derived_Data_Rules node under the Data Rules node in the Projects Navigator. You can reuse this data rule by attaching it to other data objects.

## Creating Data Rules Using the Create Data Rule Wizard

The Data Rules folder in the Projects Navigator contains the data rules. Every data rule must belong to a data rule folder. The subfolder DERIVED_DATA_RULES in each project contains the data rules derived as a result of data profiling. You can create additional data rule folders to contain any data rules that you create.

**To create a data rule:**

1. Right-click the data rule folder in which you want to create a data rule and select **New Data Rule**.

   The Welcome page of the Create Data Rule Wizard is displayed.

2. On the Welcome page, click **Next**.
3. On the Name and Description page, specify a name and an optional description for the data rule. Click **Next**.
4. On the Define Rule page, specify the type of data rule to create. Also specify any additional information required to create the data rule. Click **Next**.

   For example, when you create a Domain Range rule, you must specify the values that represent the valid domain values.

   See "Defining the Data Rule" on page 19-6 for information about defining the data rule.
5. On the Summary page, review the selections that you made in the wizard. Click **Back** to modify any selected values. Click **Finish** to create the data rule.

   The data rule is added to the data rule folder under which you created the data rule.

### Defining the Data Rule

Use the Define Rule page or the Define Rule tab to provide details about the data rule. The top section of the page displays the **Type** list that represents the type of data rule. When you are creating a data rule, expand the Type field to view the types of data rules, and select the type that you want to create. When you edit a data rule, the Type field is disabled, as you cannot change the type of data rule once it is created. For more information about types of data rules, see "Types of Data Rules" on page 19-2.

> **Note:** When you are deriving a data rule, the Type field is automatically populated and you cannot edit this value.

The bottom section of this page specifies additional details about the data rule. The number and names of the fields displayed in this section depend on the type of data rule that you create.

For example, if you select Custom as the type, use the Attributes section to define the attributes required for the rule. Use the Ellipsis button on the Expression field to define a custom expression involving the attributes that you defined in the Attributes section.

If you select Domain Range as the type of data rule, the bottom section of the page provides fields to specify the data type of the range, the minimum value, and the maximum value. When you are deriving a data rule, some of these fields are populated based on the profiling results from which you are deriving the rule. You can edit these values.

## Editing Data Rules

After you create a data rule, you can edit its definition. You can rename the data rule and edit its description. You cannot change the type of data rule. However, you can change the other parameters specified for the data rule. For example, for a Domain Range type of data rule, you can edit the data type of the range, the minimum range value, and the maximum range value.

**To edit a data rule:**

1. In the Projects Navigator, right-click the data rule and select **Open**.

   The Edit Data Rule dialog box is displayed.

2. On the Name tab, you can perform the following tasks:
   - To rename a data rule, select the name and enter the new name.
   - To edit the description for the data rule, select the description and enter the new description.
3. On the Define tab, edit the properties of the data rule.

> **Note:** You cannot change the type of data rule. You can only modify the properties related to that type of data rule, such as the domain bounds, domain list, and number of attributes in a unique key.

## Applying Data Rules to Data Objects

Applying a data rule to an object binds the definition of the data rule to the object. For example, binding a rule to the table `Dept` ensures that the rule is implemented for the specified attribute in the table. You apply a data rule using the object editor. You can also apply a derived data rule from the Data Rule panel of the Data Profile Editor.

The Apply Data Rule Wizard enables you to apply a data rule to a data object. You can apply precreated data rules, or any data rule you created to data objects. The types of data objects to which you can apply data rules are tables, views, materialized views, and external tables.

**To apply a data rule to a data object:**

1. In the Projects Navigator, right-click the object to which you want to apply a data rule and select **Open**.

   The editor for the data object is displayed.
2. On the Data Rules tab, any data rules already bound to the data object are displayed. Click **Apply Rule** to apply a new data rule.

   The Apply Data Rule Wizard is displayed.
3. On the Welcome page, click **Next**.
4. On the Select Rule page, select the data rule that you want to apply to the data object and click **Next**.

   Data rules are grouped under the nodes BUILT_IN, DERIVED_DATA_RULES, and any other data rule folders that you create.

   The BUILT_IN node contains the default data rules defined in the workspace. These include rules such as foreign key, unique key, and not null.

   The DERIVED_DATA_RULES node lists all the data rules that were derived as a result of data profiling.
5. On the Name and Description page, enter a name and an optional description for the data rule and click **Next**.
6. On the Bind Rule Parameters page, use the Binding list to select the column to which the data rule must be applied and click **Next**.
7. On the Summary page, review the selections that you made on the previous wizard pages. Click **Back** to modify selected values. Click **Finish** to apply the data rule.

   The data rule is bound to the data object and is listed on the Data Rules tab.

## Applying Data Rules within ETL Mappings Manually

When you use a data object that has a data rule defined within an ETL mapping, you can decide whether you want to enforce the data rule within the mapping.

**To apply a data rule in an ETL mapping:**

1. Define your ETL mapping using the steps described in "Steps to Perform Extraction, Transformation, and Loading (ETL) Using Mappings" on page 5-9.
2. In the mapping editor, select the data object which has the data rule defined.

   The Property Inspector displays the mapping properties.
3. Under the Data Rules node, click the Ellipsis to the right of Data Rules field.

   The Data Rules dialog box is displayed. On the left, the data rules that are defined on the selected data object are listed.
4. Select the data rule for which you want to specify an action.

   On the right, the Data Rule Information section displays the details about the selected data rule.
5. In the Rule Action column, select one of the following options to indicate the action must be performed when a source record violates the selected data rule:
   - **IGNORE:** The data rule violation is ignored and the record is processed as part of the logic in the ETL mapping.
   - **MOVE TO ERROR:** The row that violates the data rule is moved to the error table.
   - **REPORT:** Rows that violate the selected data rule are added to a report.

# 20

# Monitoring Quality with Data Auditors and Data Rules

This chapter describes the use of data auditors to monitor data quality.

This chapter contains the following topics:

- Overview of Data Auditors
- Monitoring Data Quality Using Data Auditors

## Overview of Data Auditors

Data auditors are processes that validate data against a set of data rules to determine which records comply and which do not. Data auditors gather statistical metrics on how well the data in a system complies with a rule by auditing and marking how many errors are occurring against the audited data. Data auditors ensure that your data complies with the data rules you defined, and thus let you track compliance with your business rules.

Data auditors are an important tool in ensuring, on an ongoing basis, that data quality levels meet business requirements. Identifying sudden increases in bad data also enables you to associate the increase with specific events and then identify possible root causes, such as addition of new data sources or changes to a data source or application that impacts your system.

To monitor data using Warehouse Builder, you must first discover or design data rules, as described in "Performing Data Profiling" on page 18-4, then define data auditors, and schedule them or incorporate them into larger process flows.

> **See Also:** "Monitoring Data Quality Using Data Auditors" on page 20-2

Data auditors can be deployed and executed ad hoc if necessary. More often, they are scheduled for regular execution as part of a process flow, to monitor the quality of the data in an operational environment such as a data warehouse or ERP system, either immediately after updates like data loads, or at regular intervals.

> **See Also:** "Auditing Data Objects Using Data Auditors" on page 20-6

### Data Auditor Thresholds

Data auditors have thresholds that allow you to create logic based on the fact that too many noncompliant records can divert the process flow into an error or notification stream. You can specify a threshold value for each data rule that the data auditor

audits. This value is used to determine if the data in the data object is within the limits that you defined. Based on this threshold, the process can choose actions. In a process flow, you can test this value and branch based upon the result.

For example, you create a data auditor to audit the data in the `Employees` table. This table contains two data rules, `emp_email_unique_rule` and `emp_sal_min_rule`. You specify that the threshold value for both rules is 80%. If less than 80% of the data in the `Employees` table does not comply with the data rules, then the auditing for the table fails.

> **See Also:** "Specifying Actions for Data That Violates Defined Data Rules" on page 20-3 for information about specifying threshold value

#### Audit Results for Data Auditors

In addition to setting thresholds for noncompliant records, you can capture audit results and store them for analysis. When executed, the data auditor sets several output values. One of these values is the audit result. Audit results provide information about the extent of data rule violations that occurred while running the data auditor.

> **See Also:** "Data Auditor Execution Results" on page 20-7

## Monitoring Data Quality Using Data Auditors

Data auditors are objects that you can use to continuously monitor your source schema to ensure that the data adheres to the defined data rules. You can monitor an object only if you have defined data rules for the object.

A data auditor can monitor data quality for more than one data object. You can create data auditors for tables, views, materialized views, and external tables.

> **See Also:** "Overview of Data Auditors" on page 20-1 for more information about data auditors

**To monitor data quality, perform the following steps:**

1. Create a data auditor containing the data objects that you want to monitor.

   See "Creating Data Auditors" on page 20-3.

2. (Optional) Configure the data auditor and set physical deployment parameters for the data auditor. These parameters are used while running the data auditor.

   See "Configuring Data Auditors" on page 20-4.

3. Deploy the data auditor to create the data auditor in the target schema.

   See "Deploying Objects" on page 12-6.

4. Run the data auditor to identify records that do not comply with the data rules defined on the data objects. You can either run data auditors manually or schedule them to run at specified times.

   See "Auditing Data Objects Using Data Auditors" on page 20-6 for information about running data auditors.

5. View the records that were identified as not complying with the defined data rules for the objects that are part of the data auditor.

   See "Viewing Data Auditor Error Tables" on page 20-8.

**Note:** You cannot import metadata for data auditors in Merge mode. For more information about import mode options, see *Oracle Warehouse Builder Installation and Administration Guide for Windows and UNIX*.

## Creating Data Auditors

Use the Create Data Auditor Wizard to create data auditors. Data auditors are part of an Oracle module in a project.

**To create a data auditor:**

1. Expand the Oracle module in which you want to create the data auditor.
2. Right-click **Data Auditors** and select **New Data Auditor**.

   The Create Data Auditor Wizard is displayed.
3. On the Name and Description page, provide the following details and click **Next**.
   - **Name:** Enter the name of the data auditor.
   - **Description:** Enter an optional description for the data auditor.
4. On the Select Objects page, select the data objects that you want to audit and click **Next**.

   The Available section lists the objects available for auditing. The Selected section contains the objects that are selected for auditing. Use the buttons to move objects to the Selected section. To select multiple objects, hold down the Ctrl key while selecting objects.
5. On the Choose Actions page, specify the action to be taken for records that do not comply with the data rules bound to the selected objects and click **Next**.

   See "Specifying Actions for Data That Violates Defined Data Rules" on page 20-3.
6. On the Summary page, review the selections that you made. Click **Back** to modify any selected values or click **Finish** to create the data auditor.

The new data auditor is added to the Data Auditors node. At this stage, only the metadata for the data auditor is stored in your workspace. To use this data auditor to monitor the quality of data in your data objects, you must run the data auditor as described in "Auditing Data Objects Using Data Auditors" on page 20-6.

### Specifying Actions for Data That Violates Defined Data Rules

Use the Choose Actions page of the Create Data Auditor Wizard or the Choose Action tab of the Edit Data Auditor dialog box to specify how records that violate data rules defined on the data objects are handled. You can also specify the level of non-compliance that is permitted for the data.

This page contains two sections: Error threshold mode and Data Rules.

**Error threshold mode**

Use the Error threshold mode to specify the method used to determine compliance of data to data rules.

Select one of the following methods:

- **Percent:** The data auditor will set the audit result based on the percentage of records that do not comply with the data rule. This percentage is specified in the rule's Defect Threshold value.

- **Six Sigma:** The data auditor will set the audit result based on the Six Sigma values for the data rules. If the calculated Six Sigma value for any rule is less than the specified Sigma Threshold value, then the data auditor will set the AUDIT RESULT to 2.

**Data Rules**

The Data Rules section lists the data rules applied to the objects selected on the Select Object page. For each rule, specify the following:

- **Action:** The action to be performed if data in the source object does not comply with the data rule. Select **Report** to ensure that the data rule is audited. Select **Ignore** if you want the data rule to be ignored.
- **Defect Threshold:** The percent of records that should comply with the data rules to ensure successful auditing. Specify a value between 1 and 100. This value is ignored if you select Six Sigma in the Error threshold mode section.
- **Sigma Threshold:** The required success rate. Specify a number between 0 and 7. If you set the value to 7, then no failures are allowed. This value is ignored if you select Percent in the Error threshold mode section.

### Editing Data Auditors

After you create a data auditor, you can edit it and modify any of its properties using the following steps.

1. In the Projects Navigator, right-click the data auditor and select **Open**.

   The Edit Data Auditor dialog box is displayed.
2. On the Name tab, enter a new name or description for the data auditor.
3. On the Select Objects tab, use the buttons to add or remove objects that will be audited as part of the data auditor.
4. On the Choose Actions tab, edit the data correction actions that you specified.

   See "Specifying Actions for Data That Violates Defined Data Rules" on page 20-3.
5. On the Reconcile Objects tab, select the check box to the left of an object to reconcile its definition with the latest repository definition. Click **Reconcile**.
6. Click **OK** to close the Edit Data Auditor dialog box.

## Configuring Data Auditors

During the configuration phase, you assign physical deployment properties to the data auditor that you created by setting the configuration parameters. The Configuration tab for the data auditor enables you to configure the physical properties of the data auditor.

**To configure a data auditor:**

1. From the Projects Navigator, expand the Databases node and then the Oracle node.
2. Right-click the name of the data auditor that you want to configure and select **Configure**.

   The Configuration tab for the data auditor is displayed.
3. Based on your requirement, configure the parameters listed in Run Time Parameters, Data Auditor Parameters, and Code Generation Options.

### Run Time Parameters

Table 20–1 lists the Run Time configuration parameters.

***Table 20–1 Run Time Configuration Parameters for Data Auditors***

| Configuration Parameter Name | Description |
|---|---|
| Default Purge Group | Used when executing the package. Each audit record in the runtime schema is assigned to the purge group specified. |
| Bulk Size | Number of rows to be fetched as a batch while processing cursors |
| Analyze table sample percentage | Percentage of rows to be sampled when the target tables are analyzed. You analyze target tables to gather statistics that you can use to improve performance while loading data into the target tables. |
| Commit frequency | Number of rows processed before a commit is issued |
| Maximum number of errors | Maximum number of errors allowed before the execution of this step is terminated |
| Default Operating mode | Represents the operating mode used<br>The options that you can select are Row based, Row based (target only), Set based, Set based fail over to row based, Set based fail over to row based (target only). |
| Default Audit Level | Indicates the audit level used when executing the package. When the package is run, the amount of audit information captured in the runtime schema depends on the value set for this parameter.<br>The options that you can select are as follows:<br>ERROR DETAILS: At run time, error information and statistical auditing information is recorded.<br>COMPLETE: All auditing information is recorded at run time. This generates a huge amount of diagnostic data which may quickly fill the allocated tablespace.<br>NONE: No auditing information is recorded at run time.<br>STATISTICS: At run time, statistical auditing information is recorded. |

### Data Auditor Parameters

This category uses the same name as the data auditor. Table 20–2 describes the generic data auditor configuration parameters.

***Table 20–2 Data Auditor Configuration Parameters for Data Auditors***

| Configuration Parameter Name | Description |
|---|---|
| Generation Comments | Specify additional comments for the generated code. |
| Threshold Mode | Specify the mode that should be used to measure failure thresholds. The options are PERCENTAGE and SIX SIGMA. |
| Language | Language used to define the generated code. The options are PL/SQL (default) and UNDEFINED. Ensure that PL/SQL (default) is selected. |
| Deployable | Select this option to indicate that you want to deploy this data auditor. Warehouse Builder generates code only if the data auditor is marked as deployable. |

*Table 20–2 (Cont.) Data Auditor Configuration Parameters for Data Auditors*

| Configuration Parameter Name | Description |
|---|---|
| Referred Calendar | Specify the schedule to associate with the data auditor. The schedule defines when the data auditor will run. |

### Code Generation Options

Table 20–3 describes the code generation options that you can set for data auditors.

*Table 20–3 Code Generation Options for Data Auditors*

| Configuration Parameter Name | Description |
|---|---|
| ANSI SQL Syntax | Select this option to use ANSI SQL code in the generated code. If this option is not selected, then Oracle SQL syntax is generated. |
| Commit Control | Specifies how commit is performed. The options available for this parameter are: Automatic, Automatic Correlated, and Manual. Ensure that this parameter is set to Automatic. |
| Enable Parallel DML | Select this option to enable parallel DML at run time. |
| Analyze table statements | Set this option to True to generate the statement used to collect statistics for the data auditor. If the target table is not in the same schema as the mapping and you want to analyze the table, then you must grant `ANALYZE ANY` privilege to the schema owning the mapping. |
| Optimized Code | Select this option to indicate that optimized code should be generated. |
| Generation Mode | Select the mode in which optimized code should be generated. The options that you can select are: All Operating Modes, Row based, Row based (target only), Set based, Set based fail over to row based, and Set based fail over to row based (target only). |
| Use Target Load Ordering | Select this option to generate code for target load ordering. |
| Error Trigger | Specify the name of the error trigger procedure. |
| Bulk Processing Code | Select this option to generate bulk processing code. |

## Auditing Data Objects Using Data Auditors

After you create a data auditor, you can use it to monitor the data in your data objects. This ensures that the data rule violations for the objects are detected. When you run a data auditor, any records that violate the data rules defined on the data objects are written to the error tables.

There are two ways of using data auditors:

- Manually Running Data Auditors
- Scheduling a Data Auditor to Run

### Manually Running Data Auditors

To check if the data in the data object adheres to the data rules defined for the object, you must run the data auditor. You can run data auditors from the Design Center or the Control Center Manager. To run a data auditor from the Design Center, right-click the data auditor and select **Start**. In the Control Center Manager, select the data auditor, and from the File menu, select **Start**. The results are displayed in the Job Details window as described in "Data Auditor Execution Results" on page 20-7.

## Scheduling a Data Auditor to Run

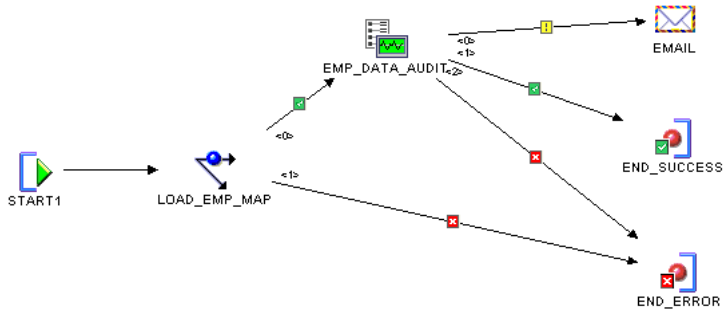
You can schedule the execution of a data auditor using the following steps:

1. Create a process flow that contains a Data Auditor Monitor activity that represents the data auditor.
2. Schedule this process flow to run at a predefined time.

   For more information about scheduling objects, see "Defining Schedules" on page 11-2.

Figure 20–1 displays a process flow that contains a Data Auditor Monitor activity. In this process flow, `LOAD_EMP_MAP` is a mapping that loads data into the `EMP` table. If the data load is successful, then the data auditor `EMP_DATA_AUDIT` is run. The data auditor monitors the data in the `EMP` table based on the data rules defined for the table.

***Figure 20–1 Data Auditor Monitor Activity in a Process Flow***



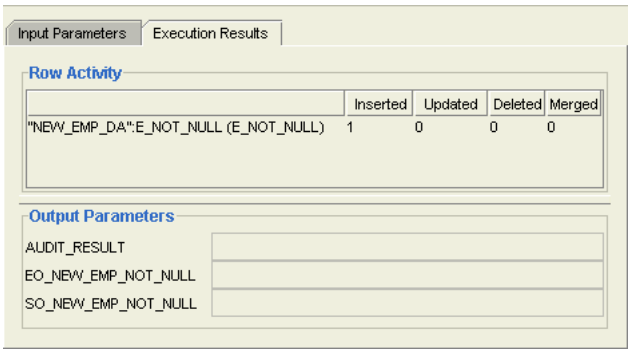

## Data Auditor Execution Results

After you run a data auditor, the Job Details window displays the details of the execution. The Job Details window contains two tabs: Input Parameters and Execution Results. Note that the Job Details window is displayed only when you set the deployment preference Show Monitor to true.

> **See Also:** *Oracle Warehouse Builder Concepts* for more information about deployment preferences

Figure 20–2 displays the Job Details window containing the Input Parameters Tab and the Execution Results Tab.

***Figure 20–2 Data Auditor Execution Results***



#### Input Parameters Tab

The Input Parameters tab contains the values of input parameters used to run the data auditor.

#### Execution Results Tab

The Execution Results tab displays the results of running the data auditor. This tab contains two sections: Row Activity and Output Parameters.

The Row Activity section contains details about the inserts into the error table for each step. Note that when more than one data rule is specified, multitable insert may be used in the data auditor. In this case, the count of the number of rows will not be accurate.

For example, in the data auditor execution result in Figure 20–2, the data rule called E_NOT_NULL inserted one record into the error table.

The Output Parameters section contains the following three parameters:

- **AUDIT_RESULT:** Indicates the result of running the data auditor. The possible values for this parameter are as follows:
  - **0:** No data rule violations occurred.
  - **1:** At least one data rule violation occurred, but no data rule failed to meet the minimum quality threshold as defined in the data auditor.
  - **2:** At least one data rule failed to meet the minimum quality threshold.

  For more information about setting the threshold, see the step on choosing actions in "Creating Data Auditors" on page 20-3.
- **EO_<data_rule_name>:** Represents the calculated error quality for the specified data rule. Zero (0) indicates all errors and 100 indicates no errors.
- **SO_<data_rule_name>:** Represents the Six Sigma quality calculated for the specified data rule.

## Viewing Data Auditor Error Tables

When you run a data auditor, either manually or as part of the process flow, Warehouse Builder writes records that do not comply with defined data rules for the objects contained in the data auditor to error tables. Each object contained in the data auditor has a corresponding error table that stores noncompliant records for that object.

You view all noncompliant records that are written to error tables by using the Repository Browser.

**To view error tables created as a result of data auditor execution:**

1. Grant privileges on the error tables as described in "Granting Privileges on Error Tables" on page 20-9.
2. Use the Repository Browser to view the error tables. Perform the following steps:
   a. Open the Repository Browser as described in "Opening the Repository Browser" on page 13-5.
   b. View error tables using the Repository Browser as described in "Viewing Error Tables Created as a Result of Data Auditor Execution" on page 13-23.

### Granting Privileges on Error Tables

Before you view data stored in error tables using the Repository Browser, you must grant privileges on the error tables to the OWBSYS user. This enables the Repository Browser to access error table data.

**To grant privileges on error tables:**

1. In SQL*Plus, log in to the schema containing the error tables.

   The error table for an object is stored in the same schema as the object.

2. Run the SQL script *OWB_ORACLE_HOME*`\owb\rtp\sql\grant_error_table_privileges.sql`.

3. When prompted, enter the name of the error table for which you want to grant privileges.

   If you did not specify a name for the error table of an object using the Error Table Name property, Warehouse Builder provides a default name. For objects that use error tables, the default error table name is the object name suffixed by "_ERR".

4. Repeat Steps 2 and 3 for each error table to which you want to grant privileges.

# 21

# Data Cleansing and Correction with Data Rules

This chapter describes the data cleansing features of Oracle Warehouse Builder and how to use them.

This chapter contains the following topics:


- Overview of Data Rules in ETL and Automatic Data Correction
- Generating Correction Mappings from Data Profiling Results
- Cleansing and Transforming Source Data Based on Data Profiling Results


## Overview of Data Rules in ETL and Automatic Data Correction

After you derive data rules from profiling results, you can automate the process of correcting source data based on profiling results. You can create the schema and mapping corrections. The schema correction creates scripts that you can use to create a corrected set of data objects with the same structure as the source objects, but with the derived data rules applied. The mapping correction creates new correction mappings to take your data from the source objects and load them into new objects.

For a given set of data objects (tables, views and so on) and a given set of data rules applied to those objects, Warehouse Builder can automatically generate the following data correction objects and logic:

- Definitions for corrected schema objects, that is, tables that have the same columns as the source tables and the same data rules bound to them, but which have constraints, stricter data types and other structures that enforce the data rules being corrected. Details of how individual data rules are enforced on corrected schema objects are described in Table 21–1, " Data Rules Implementation for Schema Correction" on page 21-4
- Cleansing ETL mappings for loading a clean version of the source data into the new corrected tables. Compliant rows can be passed through to the clean tables without change. Noncompliant data can be filtered out, reported on, or corrected to be made compliant. Many common data correction algorithms are built into Warehouse Builder, or you can implement your own cleansing logic. Details of the available correction strategies are described in Table 21–2, " Cleansing Strategies for Data Correction" on page 21-6

To actually create your corrected data, you must then deploy the corrected schema objects, mappings and relevant data rules to the target location and either run the mappings or schedule them to run as needed. You can then implement further ETL using the cleansed schema objects as a source instead of the original dirty data.

You can also apply data rules in ETL mappings manually. When adding a data object such as a table to a mapping, you can select one or more data rules to be applied to the object. Error tables are created for the object, and you can define separate flows within your mapping for compliant and noncompliant data.

# Generating Correction Mappings from Data Profiling Results

When automatically generating corrections for source tables or other objects based on data rules, the objects generated include the following.

- Definitions for corrected schema objects, that is, tables that have the same columns as the source tables, but which have constraints, types and other structures that correspond to the data rules being corrected.
- Cleansing ETL mappings that move compliant source data into the target tables and either filter out noncompliant data or generate corrected, compliant data from the noncompliant data based on algorithms you specify.

To actually create your corrected data, you must then deploy the corrected schema objects, mappings and relevant data rules to the target database and either run the mappings or schedule them to run as needed.

## Prerequisites for Creating Corrections

The prerequisites for creating corrections are:

- You must already have a data profile where you have profiled the source data objects (tables, views and so on) to be corrected.
- You must already have data rules to be used to identify noncompliant data for correction.

## Steps to Create Correction Objects

The Data Profile Editor enables you to create mappings that will perform schema correction and data cleansing based on your data profiling results.

**To create corrections:**

1. If the data profile is not already open, open it by right-clicking the data profile in the Projects Navigator and selecting **Open**.
2. From the Profile menu, select **Create Correction**.

   The Create Correction Wizard is displayed.
3. On the Welcome page, click **Next**.
4. On the Select Target Module page, specify the target module that will contain the corrections and click **Next**.

   You can either create a new module or use an existing module.

   - To store the corrections in an existing target module, choose **Select an existing module**. The Available list displays the existing modules in which corrections can be stored. Select the module from this list.
   - To store the corrections in a new target module, select **Create a new target module**. The Create Module Wizard guides you through the steps of creating a new target module.

To remove correction objects created as a result of previous corrections, select **Remove previous correction objects**.

5. On the Select Objects page, select the objects for which corrections should be generated by moving them to the Selected list. Click **Next**.

   The **Filter** list enables you to filter the objects that are available for selection. The default selection is All Objects. You can display only particular types of data objects such as tables or views.

6. On the Select Data Rules and Data Types page, select the corrections that must be generated to perform schema correction. Click **Next**.

   See "Selecting the Data Rules and Data Types for Corrected Schema Objects" on page 21-3 for information about specifying data corrections.

7. (Optional) On the Data Rules Validation page, note the validation errors, if any, and correct them before proceeding.

   If correction objects from a previous data correction action exist for the objects selected for correction, this page displays a message. Click **Next** to remove previously created correction objects.

8. On the Verify and Accept Corrected Tables page, select the objects that you want to correct and click **Next**.

   See "Selecting the Objects to Be Corrected" on page 21-4 for more information about how to specify how objects should be corrected.

9. On the Choose Data Correction Actions page, specify the correction actions to be performed to cleanse source data and click **Next**.

   See "Choosing Data Correction and Cleansing Actions" on page 21-5 for more details about specifying the actions that perform data correction and cleansing.

10. On the Summary page, click **Finish** to create the correction objects.

The correction schema is created and added to the Projects Navigator. The correction objects and mappings are displayed under the module that you specify as the target module on the Select Target Module page of the Create Correction Wizard. The correction object uses the same name as the source object. The name of the correction mapping is the object name prefixed with M_. The correction mapping is used to cleanse source data and load it into the corrected target object.

## Selecting the Data Rules and Data Types for Corrected Schema Objects

Use the Data Rules and Data Types page to select the schema corrections that should be generated for the corrected data objects. Based on the data profiling results, Warehouse Builder populates this page with data type corrections and data rules that you can apply to the data object.

Schema correction consists of correcting data type definitions and defining data rules that should be applied to the corrected objects. The objects selected for correction are displayed on the left side of the page and are organized into a tree by modules. The panel on the right contains two tabs: Data Rules and Data Types. Select an object by clicking the object name and then define how schema correction should be performed for this object using the Data Rules and Data Types tabs.

**Data Rules** The Data Rules tab displays the available data rules for the object selected in the object tree. Specify the data rules that should be generated for the corrected object by selecting the check box to the left of the data rule. Warehouse

Builder uses these data rules to create constraints on the tables during the schema generation.

The Bindings section contains details about the table column to which the rule is bound. Click a rule name to display the bindings for that rule.

Warehouse Builder uses different methods of enforcing data rules on corrected schema objects. The method used depends on the type of data rule that you are implementing.

Table 21–1 describes the methods used for object schema correction. It also lists the data rule types for which each correction is used.

***Table 21–1 Data Rules Implementation for Schema Correction***

| Schema Correction Method | Description | Data Rule Types for which Correction Method Can be Used |
|---|---|---|
| Create Constraints | Creates a constraint reflecting the data rule on the correction table. If a constraint cannot be created, then a validation message is displayed on the Data Rules Validation page of the Apply Data Rule Wizard. | Custom<br>Domain List<br>Domain Pattern List<br>Domain Range<br>Common Format<br>No Nulls<br>Unique Key |
| Change the data type | Changes the data type of the column to `NUMBER` or `DATE` according to the results of profiling. The data type is changed for data rules of type Is Number and Is Name. | Is Number<br>Is Date |
| Create a lookup table | Creates a lookup table and adds the appropriate foreign key or unique key constraints to the corrected table and the lookup table. | Functional Dependency |
| Name and Address Parse | Adds additional name and address attributes to the correction table. The name and address attributes correspond to a selection of the output values of the Name and Address operator. In the map that is created to cleanse data, a Name and Address operator is used to perform name and address cleansing. | Name and Address |

**Data Types** The Data Types tab displays the columns that are selected for correction. The change could be a modification of the data type, precision, or from fixed-length to variable-length. The Documented Data Type column on this tab displays the existing column definition and the New Data Type column displays the proposed correction to the column definition.

To correct a column definition, select the check box to the left of the column name.

## Selecting the Objects to Be Corrected

Use the Verify and Accept Corrected Tables page to confirm the objects that you want to correct and to provide additional details about how data correction should be performed. This page contains the objects you selected for schema correction on the Data Rules and Data Types page.

Use the following steps to specify how your data objects should be corrected.

1. In the Verify and Accept Corrected Tables that Will be Generated section, select **Create** to the left of a data object to create this data object in the corrected schema.

The Definition of the Corrected Table section displays the corrections details for the selected data object. The Columns tab displays the details of columns that will be created in the corrected data object. The Constraints tab displays details of constraints that will be created on the corrected data object. The Data Rules tab displays details of data rules that will be created on the corrected data object.

2. On the Columns tab of the Definition of the Corrected Table section:
   - Select **Create** to the left of a column name to create this column in the corrected data object.
   - Deselect **Create** to the left of a column name to remove this column from the corrected object.
   - Edit the Data Type, Length, Precision, Seconds Precision, and Scale for a column by clicking the value and entering the new value. However, you cannot modify a column name.
3. On the Constraints tab of the Definition of the Corrected Table section:
   - Click **Add Constraint** to create additional constraints.
   - Select the constraint and click **Delete** to remove a constraint from the corrected data object.
4. On the Data Rules tab of the Definition of the Corrected Table section:
   - Select the check box to the left of a data rule to apply this derived data rule to the corrected data object.

     Ensure that the Bindings column contains the column to which the data rule should be applied.
   - Click **Apply Rule** to apply a new data rule to the corrected object. The Apply Data Rule Wizard guides you through the process of applying a data rule.

## Choosing Data Correction and Cleansing Actions

When you decide to automatically generate corrected objects based on data profiling results, you must specify how inconsistent data from the source object should be cleansed before being stored in the corrected object. To do this, you specify a cleansing strategy for each data rule that is applied to the correction object.

The Choose Data Correction Actions page enables you to specify how to correct source data. This page contains two sections: Select a Corrected Table and Choose Data Correction Actions. The Select a Corrected Table section lists the objects that you selected for corrections. This section contains the following columns for each data object that you selected for correction:

- **Correct:** Select this option to enable generation of correction objects for the data object listed in the Table column.
- **Table:** Represents the name of the data object for which correction actions are being specified.
- **Load Option:** Indicates which records should be loaded by the correction mapping. Select **All Records** to indicate that the generated correction mapping should load all records. Select **Corrected Objects** to indicate that the generated correction mapping should load only the records being corrected.
- **Audit Option:** Select this option to create a data auditor for the table represented by the Table column.
- **Description:** Represents a description for the correction mapping that is created.

Select a data object in the Select a Corrected Table section to display the affiliated data rules in the Choose Data Correction Actions section.

### Choosing Data Correction Actions

For each data rule, you must choose a correction action that specifies how data values that violate data rules set for the data object should be handled. Use the list in the Action column to specify the correction action that you want to perform.

The correction actions that you can choose are:

- **Ignore:** The data rule is ignored and, therefore, no values are rejected based on this data rule.
- **Report:** The data rule is run only after the data has been loaded for reporting purposes. It is similar to the Ignore option, except that a report containing values that do not adhere to the data rules is created. This action can be used for some rule types only.
- **Cleanse:** The values rejected by this data rule are moved to an error table where cleansing strategies are applied. When you select this option, you must specify a cleansing strategy as described in "Specifying the Cleansing Strategy" on page 21-6.

### Specifying the Cleansing Strategy

For each data rule, use the **Cleansing Strategy** list to specify how data that violates a set data rule should be cleansed. This option is enabled only if you select **Cleanse** in the Action column. The cleansing strategy depends on the type of data rule and the rule configuration. Error tables are used to store the records that do not conform to the data rule.

Table 21–2 describes the cleansing strategies and lists the types of data rules for which each strategy is applicable.

***Table 21–2 Cleansing Strategies for Data Correction***

| Cleansing Strategy | Description | Applicable to Data Rule Types |
|---|---|---|
| Remove | Does not populate the target table with error records | All |
| Custom | Creates a function in the target table that contains a header, but no implementation details. You must add the implementation details to this function. | Domain List<br>Domain Pattern List<br>Domain Range<br>Common Format<br>No Nulls<br>Name and Address<br>Custom |
| Use Existing Function | Select from a list of existing functions to perform the correction | Domain List<br>Domain Pattern List<br>Domain Range<br>Common Format<br>No Nulls<br>Name and Address<br>Custom |

*Table 21–2 (Cont.) Cleansing Strategies for Data Correction*

| Cleansing Strategy | Description | Applicable to Data Rule Types |
|---|---|---|
| Set to Min | Sets the attribute value of the error record to the minimum value defined in the data rule | Domain Range rules that have a minimum value defined |
| Set to Max | Sets the attribute value of the error record to the maximum value defined in the data rule | Domain Range rules that have a maximum value defined |
| Similarity | Uses a similarity algorithm based on permitted domain values to find a value that is similar to the error record. If no similar value is found, then the original value is used. | Domain List rules with character data types |
| Soundex | Uses a soundex algorithm based on permitted domain values to find a value that is similar to the error record. If no soundex value is found, then the original value is used. | Domain List rules with character data types |
| Merge | Uses the match-merge algorithm to merge duplicate records into a single row | Unique Key |
| Set to Mode | Uses the mode value to correct the error records if a mode value exists for the functional dependency partition that fails | Functional Dependency |

**See Also:** "Types of Data Rules" on page 19-2

## Viewing the Correction Tables and Mappings

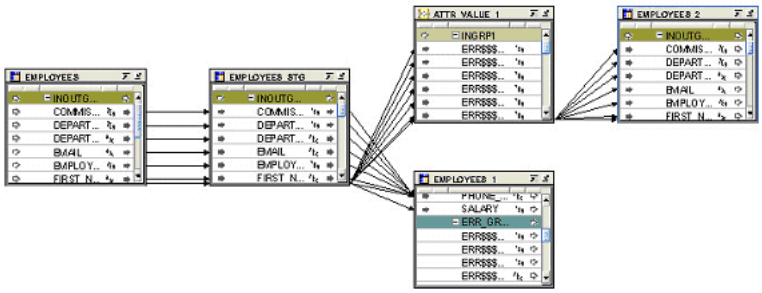
You can view the correction tables in the Table Editor to see the data rules and constraints created as part of the design of your table. You can also view the correction mappings as you can view any other ETL mapping.

**To view the correction mappings:**

1. Double-click the mapping to open the object in the Mapping Editor.
2. After the mapping is open, select **View** and then **Auto Layout** to view the entire mapping.

   Figure 21–1 displays a correction map generated by the Create Correction Wizard.

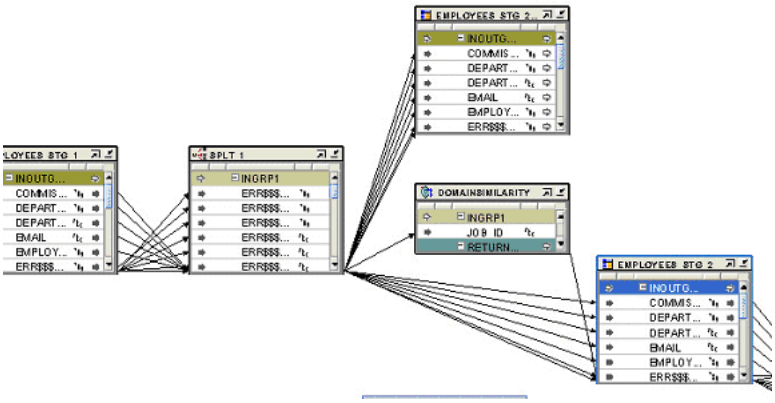
*Figure 21–1 Generated Correction Mapping*




3. Select the submapping ATTR_VALUE_1 and click **Visit Child Graph** on the toolbar to view the submapping.

Figure 21–2 displays the submapping that is displayed.

***Figure 21–2 Correction Submapping***




The submapping is the element in the mapping that performs the actual correction cleansing that you specified in the Create Correction Wizard. In the middle of this submap is the DOMAINSIMILARITY transformation that was generated as a function by the Create Correction Wizard.

# Cleansing and Transforming Source Data Based on Data Profiling Results

After you generate correction objects, you must deploy and execute the correction objects to perform schema correction and data cleansing. Your data is corrected after you run the correction mappings with the data rules. The relevant data rules also remain bound to the objects in the corrected schema for optional use in data auditors.

Correcting your schema and cleansing data requires the following steps:

1. Deploying Schema Corrections
2. Deploying Correction Mappings

## Deploying Schema Corrections

When you perform schema correction based on data profiling results, Warehouse Builder generates the schema correction actions that you specified and generates corrected data objects. The name of the corrected data object is the name of the original source object prefixed with TMP_.

When deploying schema corrections, deploy all corrected data objects, along with any data rules that were defined for the corrected objects as part of the data correction process.

## Deploying Correction Mappings

When you generate correction mappings to cleanse source data based on data profiling results, Warehouse Builder creates the correction mappings in the workspace. The name of the correction mapping for a particular data object is the name of the data object prefixed with M_. For example, the correction mapping generated to cleanse the `DEPT` table is called `M_DEPT`.

**To deploy the correction mappings created as part of the data correction process:**

1. Grant the `SELECT` privilege on the source tables to `PUBLIC`.

For example, your correction mapping contains the table `EMPLOYEES` from the `HR` schema. You can successfully deploy this correction mapping only if the `SELECT` privilege is granted to `PUBLIC` on the `HR.EMPLOYEES` table.

2. Deploy the correction tables created as a result of data profiling.

   You can right-click the table in the Projects Navigator and select **Deploy**. Or you can use the Control Center to deploy data objects.

3. Deploy the correction mappings generated to cleanse source data.

4. To cleanse source data and load it into the corrected tables, execute the correction mapping as you would any other ETL mapping.

   To execute the mapping, right-click the mapping in the Projects Navigator and select **Start**.

You can also schedule this mapping to run like any other mapping or include it in process flows.

# 22

# Name and Address Cleansing

This chapter discusses the name and address cleansing features of Oracle Warehouse Builder.

This chapter contains the following topics:


- About Name and Address Cleansing in Warehouse Builder on page 22-1
- Using the Name and Address Operator to Cleanse and Correct Name and Address Data on page 22-19
- Managing the Name and Address Server on page 22-23


## About Name and Address Cleansing in Warehouse Builder

Warehouse Builder includes name and address cleansing functionality and can integrate with third-party name and address cleansing tools from a number of vendors. Warehouse Builder parses the names and addresses, and uses methods specific to this type of data, such as matching common nicknames and abbreviations. You can compare the input data to the data libraries supplied by third-party name and address cleansing software vendors, identify and correct errors and inconsistencies in name and address source data. You can then further augment your records with information such as postal routes and geographic coordinates.

**Note:** Warehouse Builder exposes its name and address cleansing functionality through the Name and Address operator, used in a Warehouse Builder ETL mapping.

Users of third-party ETL products can still use Warehouse Builder for name and address cleansing, while retaining their existing ETL solution.

- Use the third-party ETL tool to load name and address cleansing input data in a staging table, or use an existing table as a source
- Use a Warehouse Builder ETL mapping to apply name and address cleansing, and load the corrected data into an output table
- Use the third-party ETL tool to pick up the cleansed results from the output table for further processing.

Because the deployed code for the mapping is just a PL/SQL package loaded in the database where the name and address cleansing takes place, this technique can be used from any ETL tool that can call logic from a PL/SQL package.

Also note that data libraries are not bundled with Warehouse Builder. Licenses must be purchased directly from third-party vendors.

**Note:** The Name and Address operator requires separate licensing and installation of third-party name and address cleansing software. See *Oracle Warehouse Builder Installation and Administration Guide for Windows and UNIX*.

## Types of Name and Address Cleansing Available in Warehouse Builder

The errors and inconsistencies corrected by the Name and Address operator include variations in address formats, use of abbreviations, misspellings, outdated information, inconsistent data, and transposed names. The operator fixes these errors and inconsistencies by:

- Parsing the name and address input data into individual elements.
- Standardizing name and address data, using standardized versions of nicknames and business names and standard abbreviations of address components, as approved by the postal service of the appropriate country. Standardized versions of names and addresses facilitate matching and householding, and ultimately help you obtain a single view of your customer.
- Correcting address information such as street names and city names. Filtering out incorrect or undeliverable addresses can lead to savings on marketing campaigns.
- Augmenting names and addresses with additional data such as gender, postal code, country code, apartment identification, or business and consumer identification. You can use this and other augmented address information, such as census geocoding, for marketing campaigns that are based on geographical location.

  Augmenting addresses with geographic information facilitates geography-specific marketing initiatives, such as marketing only to customers in large metropolitan areas (for example, within an *n*-mile radius of large cities); marketing only to customers served by a company's stores (within an *x*-mile radius of these stores).

Oracle Spatial, an option with Oracle Database, and Oracle Locator, packaged with Oracle Database, are two products that you can use with this feature.

The Name and Address operator also enables you to generate postal reports for countries that support address correction and postal matching. Postal reports often qualify you for mailing discounts. For more information, see "About Postal Reporting" on page 22-5.

## Example: Correcting Address Information

This example follows a record through a mapping using the Name and Address operator. This mapping also uses a Splitter operator to demonstrate a highly recommended data quality error handling technique.

### Example Input

In this example, the source data contains a `Customer` table with the row of data shown in Table 22–1.

***Table 22–1 Sample Input to Name and Address Operator***

| Address Column | Address Component |
|---|---|
| Name | Joe Smith |
| Street Address | 8500 Normandale Lake Suite 710 |
| City | Bloomington |
| ZIP Code | 55437 |

The data contains a nickname, a last name, and part of a mailing address, but it lacks the customer's full name, complete street address, and the state in which he lives. The data also lacks geographic information such as latitude and longitude, which can be used to calculate distances for truckload shipping.

### Example Steps

This example uses a mapping with a Name and Address operator to cleanse name and address records, followed by a Splitter operator to load the records into separate targets depending on whether they were successfully parsed. This section explains the general steps required to design such a mapping.

**To make the listed changes to the sample record:**

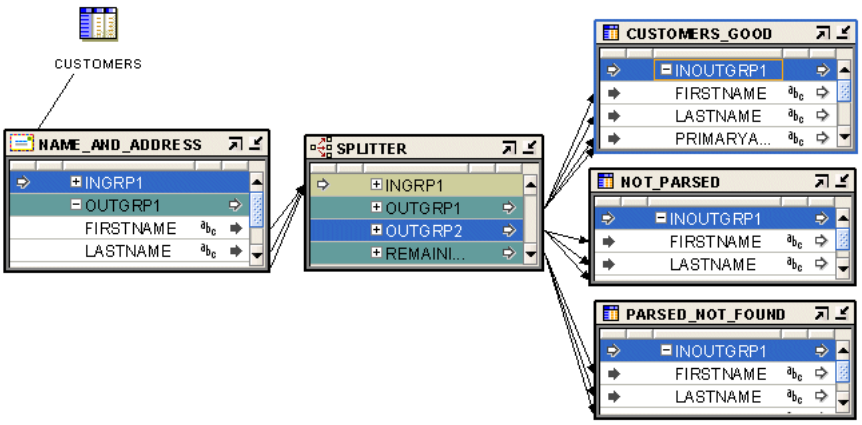
1. In the Mapping Editor, begin by adding the following operators to the canvas:
   - A `CUSTOMERS` table from which you extract the records. This is the data source. It contains the data in Table 22–1.
   - A Name and Address operator. This action starts the Name and Address Wizard. Follow the steps of the wizard.
   - A Splitter operator. For information about using this operator, see "Splitter Operator" on page 26-39.
   - Three target operators into which you load the successfully parsed records, the records with parsing errors, and the records whose addresses are parsed but not found in the postal matching software.
2. Map the attributes from the `CUSTOMERS` table to the Name and Address operator ingroup. Map the attributes from the Name and Address operator outgroup to the Splitter operator ingroup.

You are not required to use the Splitter operator, but it provides an important function in separating good records from problematic records.

3. Define the split conditions for each of the outgroups in the Splitter operator and map the outgroups to the targets.

Figure 22–1 shows a mapping designed for this example. The data is mapped from the `CUSTOMERS` source table to the Name and Address operator, and then to the Splitter operator. The Splitter operator separates the successfully parsed records from those that have errors. The output from OUTGRP1 is mapped to the `CUSTOMERS_GOOD` target. The split condition for OUTGRP2 is set such that records whose `Is Parsed` flag is `False` are loaded to the NOT_PARSED target. That is, the Split Condition for OUTGRP2 is set as INGRP1.ISPARSED='F'. The Records in the REMAINING_RECORDS group are successfully parsed, but their addresses are not found by the postal matching software. These records are loaded to the PARSED_NOT_FOUND target.

***Figure 22–1 Name and Address Operator Used with a Splitter Operator in a Mapping***




## Example Output

If you run the mapping designed in this example, the Name and Address operator standardizes, corrects, and completes the address data from the source table. In this example, the target table contains the address data as shown in Table 22–2. Compare it with the input record from Table 22–1 on page 22-3.

***Table 22–2 Sample Output from Name and Address Operator***

| Address Column | Address Component |
|---|---|
| First Name Standardized | JOSEPH |
| Last Name | SMITH |
| Primary Address | 8500 NORMANDALE LAKE BLVD |
| Secondary Address | STE 710 |
| City | BLOOMINGTON |
| State | MN |
| Postal Code | 55437-3813 |
| Latitude | 44.849194 |
| Longitude | -093.356352 |

*Table 22–2 (Cont.) Sample Output from Name and Address Operator*

| Address Column | Address Component |
|---|---|
| Is Parsed | True |
| Is Good Name | True |
| Is Good Address | True |
| Is Found | True |
| Name Warning | False |
| Street Warning | False |
| City Warning | False |

In this example, the following changes were made to the input data:

- Joe Smith was separated into separate columns for `First_Name_Standardized` and `Last_Name`.
- Joe was standardized into JOSEPH and Suite was standardized into STE.
- Normandale Lake was corrected to NORMANDALE LAKE BLVD.
- The first portion of the postal code, 55437, was augmented with the ZIP+4 code to read 55437-3813.
- Latitude and longitude locations were added.
- The records were tested in various ways, and the good records were directed to a different target from the ones that have problems.

## About Postal Reporting

All address lists used to produce mailings for discounted automation postal rates must be matched by postal report-certified software. Certifications depend on the third-party vendors of name and address software and data. The certifications may include the following:

- **United States Postal Service:** Coding Accuracy Support System (CASS)
- **Canada Post:** Software Evaluation and Recognition Program (SERP)
- **Australia Post:** Address Matching Approval System (AMAS)

### United States Postal Service CASS Certification

The Coding Accuracy Support System (CASS) was developed by the United States Postal Service (USPS) in cooperation with the mailing industry. The system provides mailers a common platform to measure the quality of address-matching software, focusing on the accuracy of five-digit ZIP Codes, ZIP+4 Codes, delivery point codes, and carrier route codes applied to all mail. All address lists used to produce mailings for automation rates must be matched by CASS-certified software.

To meet USPS requirements, the mailer must submit a CASS report in its original form to the USPS.

### Canada Post SERP Certification

Canada Post developed a testing program called Software Evaluation and Recognition Program (SERP), which evaluates software packages for their ability to validate, or

validate and correct, mailing lists to Canada Post requirements. Postal programs that meet SERP requirements are listed on the Canada Post Web site.

Canadian postal customers who use Incentive Lettermail, Addressed Admail, and Publications Mail must meet the Address Accuracy Program requirements. Customers can obtain a Statement of Accuracy by comparing their databases to Canada Post's address data.

### Australia Post AMAS Certification

The Address Matching Approval System (AMAS) was developed by Australia Post to improve the quality of addressing. It provides a standard by which to test and measure the ability of address-matching software to:

- Correct and match addresses against the Postal Address File (PAF).
- Append a unique Delivery Point Identifier (DPID) to each address record, which is a step toward barcoding mail.

AMAS enables companies to develop address matching software which:

- Prepares addresses for barcode creation
- Ensures quality addressing
- Enables qualification for discounts on PreSort letters lodgements

PreSort Letters Service prices are conditional upon customers using AMAS Approved Software with Delivery Point Identifiers (DPIDs) being current against the latest version of the PAF.

A declaration that the mail was prepared appropriately must be made when using the Presort Lodgement Document, available from post offices.

## Input Role Descriptions

For each attribute that you select for Name or Address cleansing, you must specify an input role to indicate the type of data that is stored in the source attribute. Warehouse Builder provides a set of predefined input roles from which you can select the most suitable one for your data.

For example, the Employees table contains the columns last_name and city. You can select the Last Name and City respectively for these columns.

Table 22–3 describes the input roles for the Name and Address Operator.

***Table 22–3 Name and Address Operator Input Roles***

| Input Role | Description |
|---|---|
| Pass Through | Any attribute that requires no processing |
| First Name | First name, nickname, or shortened version of the first name. |
| Middle Name | Middle name or initial. Use when there is only one middle name, or for the first of several middle names (for example, "May" in Ethel May Roberta Louise Mertz). |
| Middle Name 2 | Second middle name (for example, "Roberta" in Ethel May Roberta Louise Mertz) |
| Middle Name 3 | Third middle name (for example, "Louise" in Ethel May Roberta Louise Mertz) |
| Last Name | Last name or surname. |

*Table 22–3 (Cont.) Name and Address Operator Input Roles*

| Input Role | Description |
|---|---|
| First Part Name | First part of the Person name, including:<br>■ Pre name<br>■ First name<br>■ Middle name(s)<br>Use when these components are contained in one source column. |
| Last Part Name | Last part of Person Name, including:<br>■ Last name<br>■ Post Name<br>Use when these components are all contained in one source column. |
| Pre Name | Information that precedes and qualifies the name (for example, Ms., Mr., or Dr.) |
| Post Name | Generation or other information qualifying the name (for example, Jr. or Ph.D.) |
| Person | Full person name, including:<br>■ First Part Name (consisting of Pre Name, First Name, and Middle Names)<br>■ Last Part Name (consisting of Last Name and Post Name)<br>Use when these components are all contained in one source column. |
| Person 2 | Designates a second person if the input includes multiple personal contacts |
| Person 3 | Designates a third person if the input includes multiple personal contacts |
| Firm Name | Name of the company or organization |
| Primary Address | Box, route, or street address, including:<br>■ Street name<br>■ House number<br>■ City map grid direction (for example, SW or N)<br>■ Street type (for example, Avenue, Street, or Road)<br>This does not include the Unit Designator or the Unit Number. |
| Secondary Address | The second part of the street address, including:<br>■ Unit Designator<br>■ Unit Number<br>For example, in a secondary address of Suite 2100, the Unit Designator is STE (a standardization of "Suite") and the Unit Number is 2100. |
| Address | Full address line, including:<br>■ Primary Address<br>■ Secondary Address<br>Use when these components share one column. |
| Address 2 | Generic address line |
| Neighborhood | Neighborhood or barrio, common in South and Latin American addresses. |
| Locality Name | The city (shi) or island (shima) in Japan. |
| Locality 2 | The ward (ku) in Japan. |

*Table 22–3 (Cont.) Name and Address Operator Input Roles*

| Input Role | Description |
|---|---|
| Locality 3 | The district (machi) or village (mura) in Japan |
| Locality 4 | The subdistrict (aza, bu, chiwari, or sen) in Japan |
| City | Name of city |
| State | Name of state or province |
| Postal Code | Postal code, such as a ZIP code in the United States or a postal code in Canada |
| Country Name | Full country name |
| Country Code | The ISO 3166-1993 (E) 2-character or 3-character country code. For example, US or USA for United States; CA or CAN for Canada |
| Last Line | Last address line, including:<br>■ City<br>■ State or province<br>■ Postal code<br>Use when these components are all contained in one source column. |
| Last Line 2 | For Japanese adaptors, specifies additional line information that appears at the end of an address |
| Line1... Line10 | Use for free-form name, business, personal, and address text of any type. These roles do not provide the parser with any information about the data content. Whenever possible, use the discrete input roles provided instead. |

## Descriptions of Output Components

Use output components to define attributes that will store data cleansed by the Name and Address operator. Any attributes with an input role of Pass Through are automatically displayed as output components. You can define additional output components to store cleansed data.

### Categories of Output Components

Output components are grouped in the following categories:

- Pass Through
- Name
- Address
- Extra Vendor
- Error Status
- Country-Specific

### Pass Through

The Pass Through output component is for any attribute that requires no processing. When you create a `Pass Through` input role, the corresponding `Pass Through` output component is created automatically. You cannot edit a `Pass Through` output component, but you can edit the corresponding input role.

### Name

Table 22–4 describes the Name output components. Many components can be used multiple times to process a record, as noted in the table. For example, in records with two occurrences of Firm Name, you can extract both by adding two output attributes. Assign one as the First instance, and the other as the Second instance.

***Table 22–4 Name Output Components***

| Subfolder | Output Component | Description |
|---|---|---|
| None | Pre Name | Title or salutation appearing before a name (for example, Ms. or Dr.). Can be used multiple times. |
| None | First Name Standardized | Standard version of first name; for example, Theodore for Ted or James for Jim. Can be used multiple times. |
| None | Middle Name Standardized | Standardized version of the middle name; for example, Theodore for Ted or James for Jim. Use when there is only one middle name, or for the first of several middle names. Can be used multiple times. |
| None | Middle Name 2 Standardized | Standardized version of the second middle name; for example, Theodore for Ted or James for Jim. Can be used multiple times. |
| None | Middle Name 3 Standardized | Standardized version of the third middle name; for example, Theodore for Ted or James for Jim. Can be used multiple times. |
| None | Post Name | Name suffix indicating generation; for example, Sr., Jr., or III. Can be used multiple times. |
| None | Other Post Name | Name suffix indicating certification, academic degree, or affiliation; for example, Ph.D., M.D., or R.N. Can be used multiple times. |
| None | Title | Personal title, for example, Manager. Can be used multiple times. |
| None | Name Designator | Personal name designation; for example, ATTN (to the attention of) or C/O (care of). Can be used multiple times. |
| None | Relationship | Information related to another person; for example, Trustee For. Can be used multiple times. |
| None | SSN | Social security number |
| None | Email Address | E-mail address |
| None | Phone Number | Telephone number |
| None | Name/Firm Extra | Extra information associated with the firm or personal name |
| None | Person | First name, middle name, and last name. Can be used multiple times. |
| Person | First Name | The first name found in the input name. Can be used multiple times. |
| Person | Middle Name | Middle name or initial. Use this for a single middle name, or for the first of several middle names; for example, "May" in Ethel May Roberta Louise Mertz. Can be used multiple times. |
| Person | Middle Name 2 | Second middle name; for example, "Roberta" in Ethel May Roberta Louise Mertz. Can be used multiple times. |

*Table 22–4 (Cont.) Name Output Components*

| Subfolder | Output Component | Description |
|---|---|---|
| Person | Middle Name 3 | Third middle name; for example, "Louise" in Ethel May Roberta Louise Mertz. Can be used multiple times. |
| Person | Last Name | Last name or surname. Can be used multiple times. |
| Derived | Gender | Probable gender:<br>■ M = Male<br>■ F = Female<br>■ N = Neutral (either male or female)<br>■ Blank = Unknown<br>Can be used multiple times. |
| Derived | Person Count | Number of persons that the record references; for example, a record with a Person name of "John and Jane Doe" has a Person Count of 2. |
| Business | Firm Name | Name of the company or organization, including divisions. Can be used multiple times. |
| Business | Firm Count | Number of firms referenced in the record. Can be used multiple times. |
| Business | Firm Location | Location within a firm; for example, Accounts Payable |

## Address

Table 22–5 describes the Address output components. In records with dual addresses, you can specify which line is used as the Normal Address (and thus assigned to the Address component) and which is used as the Dual Address for many output components, as noted in the table.

*Table 22–5 Address Output Components*

| Subfolder | Output Component | Description |
|---|---|---|
| None | Address | Full address line, including:<br>■ Primary Address<br>■ Secondary Address<br>Can be used as the Normal Address or the Dual Address. |
| None | Primary Address | Box, route, or street address, including:<br>■ Street name<br>■ House number<br>■ City map grid direction; for example, SW or N<br>■ Street type; for example, Avenue, Street, or Road.<br>Does not include the output components Unit Designator or Unit Number. Can be used as the Normal Address or the Dual Address. |
| Primary Address | Street Number | Number that identifies the address, such as a house or building number, sometimes referred to as the primary range. For example, in 200 Oracle Parkway, the `Street Number` value is 200. Can be used as the Normal Address or the Dual Address. |

*Table 22–5 (Cont.) Address Output Components*

| Subfolder | Output Component | Description |
|---|---|---|
| Primary Address | Pre Directional | Street directional indicator appearing before the street name; for example, in 100 N University Drive, the `Pre Directional` value is "N". Can be used as the Normal Address or the Dual Address. |
| Primary Address | Street Name | Name of street. Can be used as the Normal Address or the Dual Address. |
| Primary Address | Primary Name 2 | Second street name, often used for addresses at a street intersection. |
| Primary Address | Street Type | Street identifier; for example, ST, AVE, RD, DR, or HWY. Can be used as the Normal Address or the Dual Address. |
| Primary Address | Post Directional | Street directional indicator appearing after the street name; for example, in 100 15th Ave. S., the `Post Directional` value is "S". Can be used as the Normal Address or the Dual Address. |
| None | Secondary Address | The second part of the street address, including:<br>■ Unit Designator<br>■ Unit Number<br>For example, in a secondary address of Suite 2100, `Unit Designator` is "STE" (a standardization of "Suite") and `Unit Number` is '2100'. Can be used as the Normal Address or the Dual Address. |
| Secondary Address | Unit Designator | Type of secondary address, such as APT or STE. For example, in a secondary address of Suite 2100, `Unit Designator` is "STE" (a standardization of "Suite"). Can be used as the Normal Address or the Dual Address. |
| Secondary Address | Unit Number | A number that identifies the secondary address, such as the apartment or suite number. For example, in a secondary address of Suite 2100, `Unit Number` is "2100". Can be used as the Normal Address or the Dual Address. |
| Secondary Address | Non-postal Secondary Address | A secondary address that is not in official postal format |
| Secondary Address | Non-postal Unit Designator | A unit designator that is not in official postal format |
| Secondary Address | Non-postal Unit Number | A unit number that is not in official postal format |
| Address | Last Line | Final address line, including:<br>■ City<br>■ State, province, or county<br>■ Formatted postal code if the address was fully assigned |
| Last Line | Neighborhood | Neighborhood or barrio, common in South and Latin American addresses |
| Last Line | City | Name of city. The U.S. city names may be converted to United States Postal Service preferred names. |

*Table 22–5 (Cont.) Address Output Components*

| Subfolder | Output Component | Description |
|---|---|---|
| Last Line | City Abbreviated | Abbreviated city name, composed of 13 characters for the United States |
| Last Line | City Abbreviated 2 | Alternative abbreviation for the city name |
| Last Line | Alternate City | An alternate name for a city that may be referenced by more than one name. In the United States, a city may be referenced by its actual name or the name of a larger urban area. For example, Brighton, Massachusetts may have Boston as an alternate city name. |
| Last Line | Locality Code | The last three digits of the International Mailsort Code, which represents a geographical region or locality within each country. Locality Codes are numeric in the range 000 to 999. |
| Last Line | Locality Name | In the United Kingdom, the following address is assigned Locality Name KNAPHILL:<br>`Chobham Rd`<br>`Knaphill`<br>`Woking GU21 2TZ` |
| Last Line | Locality 2 | The ward (ku) in Japan |
| Last Line | Locality 3 | The district (machi) or village (mura) in Japan |
| Last Line | Locality 4 | The subdistrict (aza, bu, chiwari, or sen) in Japan |
| Last Line | County Name | The name of a county in the United Kingdom, United States, or other country |
| Last Line | State | Name of state or province |
| Last Line | Postal Code | Full postal code with spaces and other nonalphanumeric characters removed |
| Last Line | Postal Code Formatted | Formatted version of postal code that includes spaces and other nonalphanumeric characters, such as dashes |
| Last Line | Delivery Point | A designation used in the United States and Australia.<br>■ For the United States, this is the 2-digit postal delivery point, which is combined with a full 9-digit postal code and check digit to form a delivery point bar code.<br>■ For Australia, this is a 9-digit delivery point. |
| Last Line | Country Code | The ISO 3166-1993 (E) 2-character country code, as defined by the International Organization for Standardization; for example, "US" for United States or 'CA' for Canada. |
| Last Line | Country Code 3 | The ISO 3166-1993 (E) 3-character country code, as defined by the International Organization for Standardization; for example, "USA" for United States, "FRA" for France, or "UKR" for Ukraine. |
| Last Line | Country Name | The full country name |
| Address | Address 2 | A second address line, typically used for Hong Kong addresses that have both a street address and a building or floor address |
| Address | Last Line 2 | Additional information that appears at the end of an address in Japan |

*Table 22–5 (Cont.) Address Output Components*

| Subfolder | Output Component | Description |
|---|---|---|
| Other Address Line | Box Name | The name for a post office box address; for example, for "PO Box 95", the Box Name is "PO BOX". Can be used as the Normal Address or the Dual Address. |
| Other Address Line | Box Number | The number for a post office box address; for example, for "PO Box 95", the Box Number is "95". Can be used as the Normal Address or the Dual Address. |
| Other Address Line | Route Name | Route name for a rural route address. For an address of "Route 5 Box 10", the Route Name is "RTE" (a standardization of "Route"). Can be used as the Normal Address or the Dual Address. |
| Other Address Line | Route Number | Route number for a rural route address. For an address of "Route 5 Box 10", the Route Number is "5". Can be used as the Normal Address or the Dual Address. |
| Other Address Line | Building Name | Building name, such as "Cannon Bridge House". Building names are common in the United Kingdom. |
| Other Address Line | Complex | Building, campus, or other complex. For example,<br>`USS John F. Kennedy`<br>`Shadow Green Apartments`<br>`Cedarvale Gardens`<br>`Concordia College`<br>You can use the Instance field in the Output Components dialog box to specify which complex should be returned if an address has more than one complex. |
| Other Address Line | Miscellaneous Address | Miscellaneous address information.<br>In records with multiple miscellaneous fields, you can extract them by specifying which instance to use in the Output Components page. |
| Geography | Latitude | Latitude in degrees north of the equator: Positive for north of the equator; negative for south (always positive for North America) |
| Geography | Longitude | Longitude in degrees east of the Greenwich Meridian: positive for east of GM; negative for west (always negative for North America) |
| Geography | Geo Match Precision | Indicates how closely the location identified by the latitude and longitude matches the address |

### Extra Vendor

Twenty components are open for vendor-specified usage.

### Error Status

Table 22–6 describes the Error Status output components. See "Handling Errors in Name and Address Data" on page 22-18 for usage notes about the Error Status components.

*Table 22–6 Error Status Output Components*

| Subfolders | Output Component | Description |
|---|---|---|
| Name and Address | Is Good Group | Indicates whether the name group, address group, or name and address group was processed successfully.<br>■ T =<br>For name groups, the name has been successfully parsed.<br>For address groups, the address has been found in a postal matching database if one is available, or has been successfully parsed if no postal database is installed.<br>For name and address groups, both the name and the address have been successfully processed.<br>■ F = The group was not parsed successfully.<br>Using this flag in conjunction with another flag, such as the `Is Parsed` flag, followed by the Splitter operator, enables you to isolate unsuccessfully parsed records in their own target, where you can address them separately. |
| Name and Address | Is Parsed | Indicates whether the name or address was parsed:<br>■ T = The name or address was parsed successfully, although some warning conditions may have been flagged.<br>■ F = The name or address cannot be parsed.<br>Check the status of warning flags such as `Name Warning` or `City Warning`. |
| Name and Address | Parse Status | Postal matching software parse status code |
| Name and Address | Parse Status Description | Text description of the postal matching software parse status |
| Name Only | Is Good Name | Indicates whether the name was parsed successfully:<br>■ T = The name was parsed successfully, although some warning conditions may have been flagged.<br>■ F = The name cannot be parsed. |
| Name Only | Name Warning | Indicates whether the parser found unusual or possibly erroneous data in a name:<br>■ T = The parser had difficulty parsing a name or found unusual data. Check the `Parse Status` component for the cause of the warning.<br>■ F = No difficulty parsing name |

*Table 22–6 (Cont.) Error Status Output Components*

| Subfolders | Output Component | Description |
|---|---|---|
| Address Only | Is Good Address | Indicates whether the address was processed successfully:<br>■ T = Successfully processed. Either the address was found in the postal matching database or, if no postal matching database is installed for the country indicated by the address, the address was successfully parsed.<br>■ F = Not successfully processed. If a postal matching database is installed for the country indicated by the address, then the address was not found in the database. If no postal matching database is available for the country, then the address cannot be parsed.<br>Use this component when you have a mix of records from both postal-matched and non-postal-matched countries. |
| Address Only | Is Found | Indicates whether the address is listed in the postal matching database for the country indicated by the address:<br>■ T = The address was found in a postal matching database.<br>■ F = The address was not found in a postal matching database. This status may indicate either that the address is not a legal address, or that postal matching is not available for the country.<br>This flag is true only if all of the other "Found" flags are true. If postal matching is available, then this flag is the best indicator of record quality. |
| Address Only: Is Found | City Found | T = The postal matcher found the city; otherwise, F. |
| Address Only: Is Found | Street Name Found | T = The postal matcher found the street name; otherwise, F. |
| Address Only: Is Found | Street Number Found | T = The postal matcher found the street number within a valid range of numbers for the named street, otherwise, F. |
| Address Only: Is Found | Street Components Found | T = The postal matcher found the street components, such as the Pre Directional or Post Directional; otherwise, F. |
| Address Only: Is Found | Non-ambiguous Match Found | Indicates whether the postal matcher found a matching address in the postal database:<br>■ T = The postal matcher found a match between the input record and a single entry in the postal database.<br>■ F = The address is ambiguous. The postal matcher found that the address matched several postal database entries and could not make a selection. For example, if the input address is "100 4th Avenue," but the postal database contains "100 4th Ave N" and "100 4th Ave S," the input's missing directional causes the match to fail. |
| Address Only | City Warning | T = The parser found unusual or possibly erroneous data in a city; otherwise, F. |

*Table 22–6 (Cont.) Error Status Output Components*

| Subfolders | Output Component | Description |
|---|---|---|
| Address Only | Street Warning | T = The parser found unusual or possibly erroneous data in a street address; otherwise, F. |
| Address Only | Is Address Verifiable | T = Postal matching is available for the country of the address; otherwise, F.<br>F does not indicate whether or not a postal matching database is installed for the country in the address. It only indicates that matching is not available for a particular address. |
| Address Only | Address Corrected | Indicates whether the address was corrected in any way during matching. Standardization is not considered correction in this case.<br>■ T = Some component of the address was changed, aside from standardization. One of the other Corrected flags must also be true.<br>■ F = No components of the address were changed, with the possible exception of standardization. |
| Address Only: Address Corrected | Postal Code Corrected | T = The postal code was corrected during matching, possibly by the addition of a postal extension; otherwise, F. |
| Address Only: Address Corrected | City Corrected | T = The city name was corrected during matching; otherwise, F.<br>Postal code input is used to determine the city name preferred by the postal service. |
| Address Only: Address Corrected | Street Corrected | T = The street name was corrected during matching; otherwise, F.<br>Some correct street names may be changed to an alternate name preferred by the postal service. |
| Address Only: Address Corrected | Street Components Corrected | T = One or more street components, such as `Pre Directional` or `Post Directional`, were corrected during matching. |
| Address Only | Address Type | Type of address. The following are common examples; actual values vary with vendors of postal matching software:<br>■ B= Box<br>■ F = Firm<br>■ G= General Delivery<br>■ H= High-rise apartment or office building<br>■ HD= High-rise default, where a single Zip+4 postal code applies to the entire building. The Name and Address operator can detect a finer level of postal code assignment if a floor or suite address is provided, in which case the record is treated as an H type, with a more specific Zip+4 code for that floor or suite.<br>■ M= Military<br>■ P= Post Office Box<br>■ R= Rural Code<br>■ S= Street |

*Table 22–6 (Cont.) Error Status Output Components*

| Subfolders | Output Component | Description |
|---|---|---|
| Address Only | Parsing Country | Country parser that was used for the final parse of the record |

## Country-Specific

Table 22–7 describes the output components that are specific to a particular country.

*Table 22–7 Country-Specific Output Components*

| Subfolder | Output Component | Description |
|---|---|---|
| United States | ZIP5 | The 5-digit United States postal code |
| United States | ZIP4 | The 4-digit suffix that is added to the 5-digit United States postal code to further specify location. |
| United States | Urbanization Name | Urban unit name used in Puerto Rico |
| United States | LACS Flag | T = Address requires a LACS conversion and should be submitted to a LACS vendor; otherwise, F.<br>The Locatable Address Conversion System (LACS) provides new addresses when a 911 emergency system has been implemented. The 911 address conversions typically involve changing rural-style addresses to city-style street addresses, but they may involve renaming or renumbering existing city-style addresses. |
| United States | CART | The 4-character USPS Carrier route |
| United States | DPBC Check Digit | Check digit for forming a delivery point bar code |
| United States | Automated Zone Indicator | T = The mail in this zip code is sorted by bar code sorting equipment; otherwise, F. |
| United States | Urban Indicator | T = An address is located within an urban area; otherwise, F. |
| United States | Line of Travel | United States Postal Service (USPS) line of travel |
| United States | Line of Travel Order | United States Postal Service (USPS) line of travel order |
| United States: Census/Geography | Metropolitan Statistical Area | Metropolitan Statistical Area (MSA) number. For example, "0000" indicates that the address does not lie within any MSA, and typically indicates a rural area. |
| United States: Census/Geography | Minor Census District | Minor Census District |
| United States: Census/Geography | CBSA Code | A 5-digit Core-Based Statistical Area (CBSA) code that identifies metropolitan and micropolitan areas. |
| United States: Census/Geography | CBSA Descriptor | Indicates whether the CBSA is metropolitan (population of 50,000 or more) or micropolitan (population of 10,000 to 49,999). |
| United States: Census/Geography | FIPS Code | The complete (state plus county) code assigned to the county by the Federal Information Processing Standard (FIPS). Because FIPS county codes are unique within a state, a complete FIPS Code includes the 2-digit state code followed by the 3-digit county code. |

*Table 22–7 (Cont.) Country-Specific Output Components*

| Subfolder | Output Component | Description |
|---|---|---|
| United States: Census/Geography | FIPS County | The 3-digit county code as defined by the Federal Information Processing Standard (FIPS). |
| United States: Census/Geography | FIPS Place Code | The 5-digit place code as defined by the Federal Information Processing Standard (FIPS). |
| United States: Geography | Census ID | United States Census tract and block-group number. The first six digits are the tract number; the final digit is the block-group number within the tract. These codes are used for matching to demographic-coding databases. |
| Canada | Installation Type | A type of Canadian postal installation:<br>■ STN= Station<br>■ RPO = Retail Postal Outlet<br>For example, for the address, "PO Box 7010, Scarborough ON M1S 3C6," the Installation Type is "STN". |
| Canada | Installation Name | Name of a Canadian postal installation. For example, for the address, "PO Box 7010, Scarborough ON M1S 3C6," the Installation Name is "AGINCOURT". |
| Hong Kong | Delivery Office Code | A mailing code used in Hong Kong. For example, the following address is assigned the Delivery Office Code "WCH":<br>`Oracle`<br>`39/F The Lee Gardens`<br>`33 Hysan Ave`<br>`Causeway Bay` |
| Hong Kong | Delivery Beat Code | A mailing code used in Hong Kong. For example, the following address is assigned the Delivery Beat Code "S06":<br>`Oracle`<br>`39/F The Lee Gardens`<br>`33 Hysan Ave`<br>`Causeway Bay` |

## Handling Errors in Name and Address Data

Name and Address parsing, like any other type of parsing, depends on identification of keywords and patterns containing those keywords. Free-form name and address data is sometimes difficult to parse because the keyword set is large and it is never 100% complete. Keyword sets are built by analyzing millions of records, but each new data set is likely to contain some undefined keywords.

Because most free-form name and address records contain common patterns of numbers, single letters, and alphanumeric strings, parsing can often be performed based on just the alphanumeric patterns. However, alphanumeric patterns may be ambiguous, or a particular pattern may not be found. Name and Address parsing errors set parsing status codes that you can use to control data mapping.

Because the criteria for quality vary among applications, numerous flags are available to help you determine the quality of a particular record. For countries with postal matching support, use the `Is Good Group` flag, because it verifies that an address is a valid entry in a postal database. Also use the `Is Good Group` flag for U.S. Coding

Accuracy Support System (CASS) and Canadian Software Evaluation and Recognition Program (SERP) certified mailings.

Unless you specify postal reporting, an address does not have to be found in a postal database to be acceptable. For example, street intersection addresses or building names may not be in a postal database, but they may still be deliverable. If the `Is Good Group` flag indicates failure, then additional error flags can help determine the parsing status.

The `Is Parsed` flag indicates success or failure of the parsing process. If `Is Parsed` indicates parsing success, then you may still want to check the parser warning flags, which indicate unusual data. You may want to check those records manually.

If `Is Parsed` indicates parsing failure, you must preserve the original data to prevent data loss.

Use the Splitter operator to map successful records to one target and failed records to another target.

## Using the Name and Address Operator to Cleanse and Correct Name and Address Data

The Name and Address operator accepts one PL/SQL input and generates one PL/SQL output.

If you experience timeout errors, you may need to increase the socket timeout setting of the Name and Address Server. The timeout setting is the number of seconds that the server will wait for a parsing request from a mapping before the server drops a connection. The default setting is 600 seconds (10 minutes). After the server drops a connection because of inactivity, subsequent parsing requests fail with a NAS-00021 error.

For most mappings, long time lapses between parsing requests are rare. However, maps operating in row-based mode with a Filter operator may have long time lapses between record parsing requests, because of the inefficiency of filtering records in row-based mode. For this type of mapping, you may need to increase the socket timeout value to prevent connections from being dropped.

To increase the socket timeout setting, see "Managing the Name and Address Server" on page 22-23.

### Creating a Mapping with a Name and Address Operator

The Name and Address operator has one input group and one output group.

**To create a mapping with a Name and Address operator:**

1. Drag and drop the operators representing the source data and the operator representing the cleansed data onto the Mapping Editor canvas:

   For example, if your source data is stored in a table, and the cleansed data will be stored in another table, drag and drop two Table operators that are bound to the tables onto the canvas.

2. Drag and drop a Name and Address operator onto the Mapping Editor canvas.

   The Name and Address Wizard is displayed.

3. On the Name page, specify a name and an optional description for the Name and Address operator.

Or, you can retain the default name displayed in the Name field.

4. On the Definitions page, select values that define the type of source data.

   See "Specifying Source Data Details and Setting Parsing Type" on page 22-21.

5. On the Groups page, optionally rename the input and output groups.

   The Name and Address operator has one input group, INGRP1, and one output group, OUTGRP1. You cannot edit, add, or delete groups. If the input data requires multiple groups, then create a separate Name and Address operator for each group.

6. On the Input Connections page, select attributes from any operator in your mapping that you want to copy and map to the Name and Address operator.

   **To complete the Input Connections page for an operator:**

   a. Select complete groups or individual attributes from the Available Attributes panel.

      To search for a specific attribute or group by name, type the text in **Search for** and click **Go.** To find the next match, click **Go** again.

      Hold the **Shift** key down to select multiple groups or attributes. If you want to select attributes from different groups, then you must first combine the groups with a Joiner or Set operator.

      **Note:** If you have not created any operators for the source data, the Available Attributes section is empty.

   b. Use the right-arrow button between the two panels to move your selections to the Mapped Attributes panel.

      The Mapped Attributes section lists the attributes that will be processed by the Name and Address operator.

7. On the Input Attributes page, assign input roles to each attribute that you selected on the Input Attributes page.

   Input roles indicate the type of name and address information that resides in a line of data. Whenever possible, choose discrete roles (such as City, State, and Postal Code) rather than nondiscrete ones (such as Last Line). Discrete roles improve parsing.

   **See Also:** "Input Role Descriptions" on page 22-6

   For attributes that have the input role set to Pass Through, specify the data type details using the Data Type, Length, Precision, Scale, and Seconds Precision fields.

8. On the Output Attributes page, define output attributes that determine how the Name and Address operator handles parsed data. The output attribute properties characterize the data extracted from the parser output.

   Any attributes that have the Pass Through input role assigned are automatically listed as output attributes. You can add additional output attributes.

   **Note:** The attributes for output components with the Pass Through role cannot be changed

**To add output attributes:**

a. Click an empty row on the Output tab and enter the attribute name.

   You can rename the output attribute by selecting the name and typing the new name.

b. Click the Ellipsis button on the Output Component field to select an output component for the attribute.

   **See Also:** "Descriptions of Output Components" on page 22-8 for the descriptions of output components

   Ensure that you add error handling flags such as Is Parsed, Is Good Name, and Is Good Address. You can use these flags with the Splitter operator to separate good records from the records with errors and load them into different targets.

c. Specify the data type details for the output attribute using the Data Type, Length, Precision, Scale, and Seconds Precision fields.

9. For countries that support address correction and postal matching, use the Postal Report page to specify the details for the postal report.

   See "Specifying Postal Report Details" on page 22-22.

### Specifying Source Data Details and Setting Parsing Type

Use the Definitions page or the Definitions tab to provide information about your source data and to specify the type of parsing to be performed on the source data. Set the following values: Parsing Type, Primary Country, and Dual Address Assignment.

**Parsing Type** Select one of the following parsing types:

- **Name Only:** Select this option when the input data contains only name data. Names can include both personal and business names. Selecting this option instead of the more generic Name and Address option may improve performance and accuracy, depending on the adapter.
- **Address Only:** Select this option when the input data contains only address data and no name data. Selecting this option instead of the more generic Name and Address option may improve performance and accuracy, depending on the adapter.
- **Name and Address:** Select this option when the input data contains both name and address data.

> **Note:** You can only specify the parsing type when you first add the Name and Address operator to your mapping. You cannot modify the parsing type in the editor.

**Primary Country** Select the country that best represents the country distribution of your data. The primary country is used by some providers of name and address cleansing software as a hint for the appropriate parser or parsing rules to use on the initial parse of the record. For other name and address service providers, external configuration of their installation controls this behavior.

**Dual Address Assignment** A dual address contains both a Post Office (PO) box and a street address for the same address record. For records that have dual addresses, your

selection determines which address becomes the normal address and which address becomes the dual address. A sample dual address is:

```
PO Box 2589
4439 Mormon Coulee Rd
La Crosse WI 54601-8231
```

Note that the choice for Dual Address Assignment affects which postal codes are assigned during postal code correction, because the street address and PO box address may correspond to different postal codes.

- **Street Assignment:** The street address is the *normal address* and the PO Box address is the *dual address*. This means that the `Address` component is assigned the street address. In the preceding example, the `Address` is 4439 MORMON COULEE RD. This choice corrects the postal code to 54601-8220.
- **PO Box Assignment:** The PO Box address is the *normal address* and the street address is the *dual address*. This means that the `Address` component is assigned the Post Office (PO) box address. In the preceding example, the `Address` is PO BOX 2589. This choice corrects the postal code to 54602-2589.
- **Closest to Last Line:** Whichever address occurs closest to the last line is the *normal address*; the other is the *dual address*. This means that the `Address` component is assigned the address line closest to the last line. In the preceding example, the `Address` is 4439 MORMON COULEE RD. This choice corrects the postal code to 54601-8220.

This option has no effect for records having a single street or PO box address.

---

**Note:** Dual Address Assignment may not be supported by all name and address cleansing software providers.

---

### Specifying Postal Report Details

Country certification varies with different vendors of name and address cleansing software. The most common country certifications are United States, Canada, and Australia. The process provides mailers with a common platform to measure the quality of address-matching software, focusing on the accuracy of postal codes (in the case of the United States, of 5-digit ZIP Codes and ZIP+4 Codes), delivery point codes, and carrier route codes applied to all mail. Some vendors of name and address cleansing software may ignore these parameters and require external setup for generating postal reports. For more information, see "About Postal Reporting" on page 22-5.

To specify postal reporting, select Yes in the Postal Report files and then provide values for the fields:

**Processor Name:** The use of this field varies with vendors of name and address cleansing software. Typically, this value appears on the United States Coding Accuracy Support System (CASS) report.

**List Name:** An optional reference field that appears on the United States and United Kingdom reports under the List Name section, but is not included in other reports. The list name provides a reference for tracking multiple postal reports (for example, "July 2005 Promotional Campaign").

**Processor Address Lines:** These address lines may appear on various postal reports. Various name and address cleansing software vendors use these fields differently. They often contain the full address of your company.

# Managing the Name and Address Server

An external Name and Address server provides an interface between Oracle Database and third-party name and address processing libraries. This section discusses details of configuring, starting, and stopping the Name and Address server.

## Configuring the Name and Address Server

The Name and Address operator generates PL/SQL code, which calls the `UTL_NAME_ADDR` package installed in the Runtime Schema. A private synonym, `NAME_ADDR`, is defined in the target schema to reference the `UTL_NAME_ADDR` package. The `UTL_NAME_ADDR` package calls Java packages, which send processing requests to an external Name and Address server, which then interfaces with third-party Name and Address processing libraries, such as Trillium.

You can use the server property file, `NameAddr.properties`, to configure server options. This file is located in `owb/bin/admin` under the Oracle home that you specified when installing the server components. The following code illustrates several important properties with their default settings.

```
TraceLevel=0
SocketTimeout=180
ClientThreads=4
Port=4040
```

The `TraceLevel` property is often changed to perform diagnostics on server communication and view output from the postal matching program parser. Other properties are rarely changed.

- **TraceLevel:** Enables output of file `NASvrTrace.log` in the `owb/bin/admin` folder. This file shows all incoming and outgoing data, verifies that your mapping is communicating with the Name and Address server, and that the Name and Address server is receiving output from the service provider. The trace log shows all server input and output and is most useful for determining whether any parsing requests are being made by an executing mapping. Set `TraceLevel=1` to enable logging. However, tracing degrades performance and creates a large log file. Set `TraceLevel=0` to disable logging for production.
- **SocketTimeOut:** Specifies the number of seconds that the Name and Address server will wait for a parsing request before closing the connection. You can increase this time to 1800 (30 minutes) when running concurrent mappings to prevent timing out.
- **ClientThreads:** Specifies the number of threads used to service client connections. One client connection is made for each database session or slave session if a map is parallelized. Most maps are parallelized, and the number of parallel processes is proportional to the number of processors. On a single-processor computer, two parallel processes are spawned for large maps. On a four processor computer, up to eight processes may be spawned. Parallelism may also be controlled by database initialization settings such as Sessions.

  For the best performance, set ClientThreads to the maximum number of clients that will be connected simultaneously. The actual number of connected clients is recorded in `NASvr.log` after a map run. You should increase the value of ClientThreads when the number of client connections shown in the log is greater.

  When the number of clients exceeds the number of threads, all clients are still serviced because the threads are shared among clients.

- **Port:** Specifies the port on which the server listens and was initially assigned by the installer. This value may be changed if the default port conflicts with another process. If the port is changed, then the port attribute must also be changed in the *runtime_schema*.nas_connection table to enable the utl_name_addr package to establish a connection.

## Starting and Stopping the Name and Address Server

Whenever you edit the properties file or perform table maintenance, you must stop and restart the Name and Address server for the changes to take effect.

**To manually stop the Name and Addresss server:**

- In Windows, run *OWB_ORACLE_HOME*/owb/bin/win32/NAStop.bat.
- In UNIX, run *OWB_ORACLE_HOME*/owb/bin/unix/NAStop.sh.

**To manually restart the Name and Address Server:**

- In Windows, run *OWB_ORACLE_HOME*/owb/bin/win32/NAStart.bat.
- In UNIX, run *OWB_ORACLE_HOME*/owb/bin/unix/NAStart.sh.

Alternatively, you can also automatically restart the Name and Address Server. However, before automatic startup, ensure that you grant the Execute privilege for the script *OWB_ORACLE_HOME*/owb/bin/unix/NAStart.sh to the OWBSYS schema.

For example, log in to SQL*Plus using the SYS user as SYSBDBA and execute the following:

```
SQL> EXEC DBMS_JAVA.GRANT_PERMISSION( 'OWBSYS', 'SYS:java.io.FilePermission',
     '/owb_11g/oracle/owb/bin/unix/NAStart.sh', 'execute' );
```

Here, /owb_11g is the path in which Oracle Warehouse Builder is installed.

# 23

# Matching, Merging, and Deduplication

This chapter discusses the matching, merging and data deduplication features of Oracle Warehouse Builder.

This chapter contains the following topics:


- About Matching and Merging in Warehouse Builder on page 23-1
- Using the Match Merge Operator to Eliminate Duplicate Source Records on page 23-22


## About Matching and Merging in Warehouse Builder

Warehouse Builder implements general-purpose data matching and merging capabilities that can be applied to any type of data.

You can write the list of rows matched by your algorithms to a target table. You can also implement complex deduplication logic to generated merged records, again using a variety of built-in merge rules or implementing your own merge rules.

Warehouse Builder matching and merging provides the following functionality:

- Determine matches using built-in algorithms, such as the Jaro-Winkler and Levenshtein edit distance algorithms, or using a custom algorithm you implement.
- Use weighting to determine matches between records.
- Generate a table containing candidate matches, as input to some other merge logic, such as an existing master data management application
- Generate a table with merged data records, with merge logic based on built-in merge rules, custom-implemented merge logic, or complex merge rules that can combine packaged and custom rules
- Cross reference data to track and audit matches.
- Built-in advanced matching rules for person, firm and address data

Warehouse Builder matching and merging can be combined with Warehouse Builder name and address cleansing functionality to support **householding**, which is the process of identifying unique households in name and address data.

See Chapter 22, "Name and Address Cleansing" on page 23-1 for details on name and address cleansing.

**Note:** Warehouse Builder exposes its matching and merging functionality through the Match Merge operator used in a Warehouse Builder ETL mapping. Users of third-party ETL products can still use Warehouse Builder for matching and merging, while retaining their existing ETL solution.

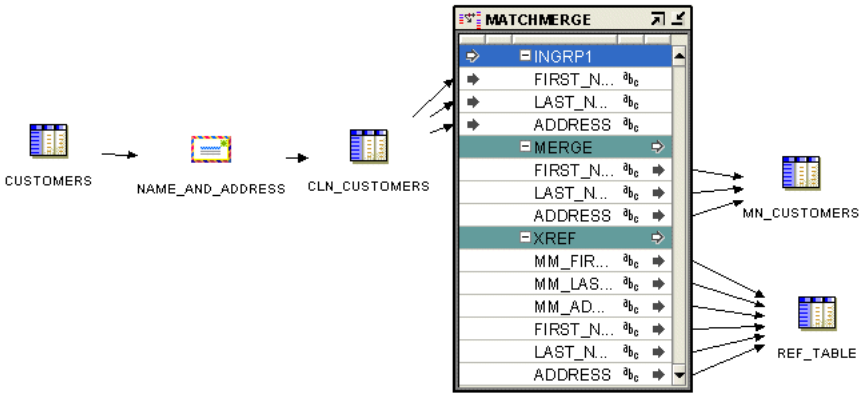
- Use the third-party ETL tool to load match-merge input data in a staging table
- Use a Warehouse Builder ETL mapping to apply match-merge and load the results into an output table
- Use the third-party ETL tool to pick up the merged results from the output table for further processing

Because the deployed code for the mapping is just a PL/SQL package loaded in the database where the matching and merging takes place, this technique can be used from any ETL tool that can call logic from a PL/SQL package.

## Example: A Basic Mapping with a Match Merge Operator

Figure 23–1 shows a mapping that uses a Match Merge operator. Notice that the Match Merge operator is preceded by a Name and Address operator, NAMEADDR, and a staging table, CLN_CUSTOMERS. In many scenarios, when cleansing and deduplicating name and address data, it makes sense to combine the Match Merge operator with the Name and Address operator in a mapping. Performing name and address cleansing on your source data provides clean and standardized input data for matching and merging. This improves the quality of your results, and can improve performance because cleansed rows are more easily identified as matches

***Figure 23–1 Match Merge Operator in a Mapping***




The simple mapping represents the flow of data for the matching and merging process:

- The Customers table provides input to the Name and Address operator, which stores its output in the CLN_CUSTOMERS table.
- The CLN_CUSTOMERS table provides FIRST, LAST, and ADDRESS columns as inputs to the Match Merge operator.
- The Match Merge operator provides FIRST, LAST, and ADDRESS input to the MM_CUSTOMERS table (the actual deduplicated rows), as well as FIRST, LAST,

ADDRESS, MM_FIRST, MM_LAST, and MM_ADDRESS input to the REF_TABLE table, which identifies the groups of matched rows from the input.

Details of how this matching process works are described in "Overview of the Matching and Merging Process" on page 23-3.

## Overview of the Matching and Merging Process

Matching determines which records refer to the same logical data. Warehouse Builder provides a variety of match rules to compare records. Match rules range from a simple exact match to sophisticated algorithms that can discover and correct common data entry errors.

Merging consolidates matched records into a single consolidated "golden" record based on survivorship rules called merge rules that you select or define for creating a merged value for each column.

If you have some other tool, such as a packaged MDM application, that already has logic for merging duplicate records, you can still use Warehouse Builder to generate the set of candidate matched rows and store those in an intermediate table.

**See Also:**

- "Match Rules" on page 23-5
- "Merge Rules" on page 23-19

### Elements of Matching and Merging Records

The following concepts and terms are important in understanding the matching and merging process.

- Match Bins
- Match Bin Attributes
- Match Record Sets
- Merged Records

#### Match Bins

Match bins are containers for similar records and are used to identify potential matches. The match bin attributes are used to determine how records are grouped into match bins. While performing matching, Warehouse Builder compares only records within the same match bin. Match bins limit the number of potential matches in a data set, thus improving performance of the match algorithm.

#### Match Bin Attributes

Before performing matching, Warehouse Builder divides the source records into smaller groups of similar records. *Match bin attributes* are the source attributes used to determine how records are grouped. Records having the same match bin attributes reside in the same match bin. Match bin attributes also limit match bins to manageable sets.

Select match bin attributes carefully to fulfill the following two conflicting needs:

- Ensure that any records that could match reside in the same match bin.
- Keep the size of the match bin as small as possible.

A small match bin is desirable for efficiency, because records that are binned together must be tested against each other to identify matches. The larger the bin, the slower the performance.

#### Match Record Sets

A match record set consists of one or more similar records within the match bin. After matching, each match bin will contain one or more match record sets. You can define match rules that determine if two records are similar.

#### Merged Records

A merged record contains data that is merged using multiple records in the match record set. Each match record set generates its own merged record.

### Process for Matching and Merging Records

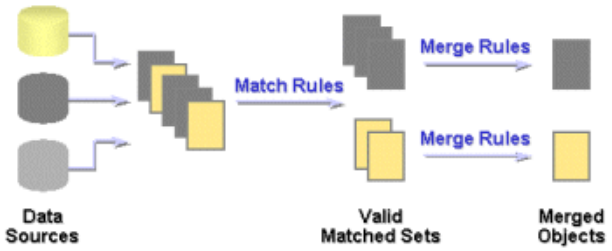
You use the Match Merge operator to match and merge records. This operator accepts records from an input source, determines the records that are logically the same, and constructs a new merged record from the matched records.

The high-level tasks involved in matching and merging process include the following:

- Constructing Match Bins
- Constructing Match Record Sets
- Constructing Merge Records

Figure 23–2 represents high-level tasks involved in the matching and merging process.

***Figure 23–2 Matching and Merging Process***




#### Constructing Match Bins

The match bin is constructed using the match bin attributes. Records with the same match bin attribute values will reside in the same match bin. A small match bin is desirable for efficiency.

#### Constructing Match Record Sets

Match rules are applied to all the records in each match bin to generate one or more match record sets. Match rules determine if two records match. The matching algorithm is an *n* X *n* algorithm where all records in the match bin are compared.

One important point of this algorithm is the transitive matching. Consider three records A, B, and C. If record A is equal to record B and record B is equal to record C, then this means that record A is equal to record C.

> **See Also:** "Match Rules" on page 23-5 for information about the types of match rules and how to create them

### Constructing Merge Records

A single merge record is constructed from each match record set. You can create specific rules to define merge attributes by using merge rules.

> **See Also:** "Merge Rules" on page 23-19 for more information about the types of merge rules

## Match Rules

Match rules are used to determine if two records are logically similar. Warehouse Builder enables you to use different types of rules to match source records. You can define match rules using the MatchMerge Wizard or the MatchMerge Editor. Use the editor to edit existing match rules or add new rules.

Match rules can be active or passive. Active rules are generated and executed in the order specified. Passive rules are generated but are not automatically executed. A passive rule may be executed by a custom rule.

Table 23–1 describes the types of match rules.

***Table 23–1 Types of Match Rules***

| Match Rule | Description |
|---|---|
| All Match | Matches all rows within a match bin |
| None Match | Turns off matching. No rows match within the match bin. |
| Conditional | Matches rows based on the algorithm you set. For more information about Conditional match rules and how to create one, see "Conditional Match Rules" on page 23-5. |
| Weight | Matches rows based on scores that you assign to the attributes. For more information about Weight match rules and how to create one, see "Weight Match Rules" on page 23-10. |
| Person | Matches records based on the names of people. For more information about Person match rules and how to create one, see "Person Match Rules" on page 23-12. |
| Firm | Matches records based on the name of the organization or firm. For more information about Firm match rules and how to create one, see "Firm Match Rules" on page 23-14. |
| Address | Matches records based on postal addresses. For more information about Address match rules and how to create one, see "Address Match Rules" on page 23-16. |
| Custom | Matches records based on a custom comparison algorithm that you define. For more information about Custom match rules and how to create one, see "Custom Match Rules" on page 23-18. |

## Conditional Match Rules

Conditional match rules specify the conditions under which records match.

A conditional match rule enables you to combine multiple attribute comparisons into one composite rule. When more than one attribute is involved in a rule, two records are considered to be a match only if all comparisons are true. Warehouse Builder displays an AND icon in the left-most column of subsequent conditions.

You can specify how attributes are compared using comparison algorithms.

### Attribute

Identifies the attribute that will be tested for a particular condition. You can select from any input attribute (INGRP1).

### Position

The order of execution. You can change the position of a rule by clicking on the row header and dragging the row to its new location. The row headers are the boxes to the left of the Attribute column.

### Algorithm

A list of methods that can be used to determine a match. Table 23–2 describes the algorithms.

### Similarity Score

The minimum similarity value required for two strings to match, as calculated by the Edit Distance, Standardized Edit Distance, Jaro-Winkler, or Standardized Jaro-Winkler algorithms. Enter a value between 0 and 100. A value of 100 indicates an exact match, and a value of 0 indicates no similarity.

### Blank Matching

Lists options for handling empty strings in a match.

## Comparison Algorithms

Each attribute in a conditional match rule is assigned a comparison algorithm, which specifies how the attribute values are compared. Multiple attributes may be compared in one rule with a separate comparison algorithm selected for each.

Table 23–2 describes the types of comparisons.

***Table 23–2 Types of Comparison Algorithms for Conditional Match Rules***

| Algorithm | Description |
|---|---|
| Exact | Attributes match if their values are exactly the same. For example, "Dog" and "dog!" would not match, because the second string is not capitalized and contains an extra character.<br>For data types other than `STRING`, this is the only type of comparison allowed. |
| Standardized Exact | Standardizes the values of the attributes before comparing them for an exact match. With standardization, the comparison ignores case, spaces, and nonalphanumeric characters. Using this algorithm, "Dog" and "dog!" would match. |
| Soundex | Converts the data to a Soundex representation and then compares the text strings. If the Soundex representations match, then the two attribute values are considered matched. |

*Table 23–2 (Cont.) Types of Comparison Algorithms for Conditional Match Rules*

| Algorithm | Description |
|---|---|
| Edit Distance | A "similarity score" in the range 0 to 100 is entered. If the similarity of the two attributes is equal to or greater than the specified value, then the attribute values are considered matched.<br><br>The similarity algorithm computes the edit distance between two strings. A value of 100 indicates that the two values are identical; a value of zero indicates no similarity whatsoever.<br><br>For example, if the string "tootle" is compared with the string "tootles", then the edit distance is 1. The length of the string "tootles" is 7. The similarity value is therefore (6/7)*100 or 85.<br><br>The algorithm used here is the Levenshtein edit distance algorithm. |
| Standardized Edit Distance | Standardizes the values of the attribute before using the Similarity algorithm to determine a match. With standardization, the comparison ignores case, spaces, and nonalphanumeric characters. |
| Partial Name | The values of a string attribute are considered a match if the value of one entire attribute is contained within the other, starting with the first word. For example, "Midtown Power" would match "Midtown Power and Light", but would not match "Northern Midtown Power". The comparison ignores case and nonalphanumeric characters. |
| Abbreviation | The values of a string attribute are considered a match if one string contains words that are abbreviations of corresponding words in the other. Before attempting to find an abbreviation, this algorithm performs a Std Exact comparison on the entire string. The comparison ignores case and nonalphanumeric character.<br><br>For each word, the match rule will look for abbreviations, as follows. If the larger of the words being compared contains all of the letters from the shorter word, and the letters appear in the same order as the shorter word, then the words are considered a match.<br><br>For example, "Intl. Business Products" would match "International Bus Prd". |
| Acronym | The values of a string attribute are considered a match if one string is an acronym for the other. Before attempting to identify an acronym, this algorithm performs a Std Exact comparison on the entire string. If no match is found, then each word of one string is compared to the corresponding word in the other string. If the entire word does not match, then each character of the word in one string is compared to the first character of each remaining word in the other string. If the characters are the same, then the names are considered a match.<br><br>For example, "Chase Manhattan Bank NA" matches "CMB North America". The comparison ignores case and nonalphanumeric characters. |

*Table 23–2 (Cont.) Types of Comparison Algorithms for Conditional Match Rules*

| Algorithm | Description |
|---|---|
| Jaro-Winkler | Matches strings based on their similarity value using an improved comparison system over the Edit Distance algorithm. The Jaro-Winkler algorithm accounts for the length of the strings and penalizes more for errors at the beginning. It also recognizes common typographical errors.<br><br>The strings match when their similarity value is equal to or greater than the Similarity Score that you specify. A similarity value of 100 indicates that the two strings are identical. A value of zero indicates no similarity whatsoever. Note that the value actually calculated by the algorithm (0.0 to 1.0) is multiplied by 100 to correspond to the Edit Distance scores. |
| Standardized Jaro-Winkler | Eliminates case, spaces, and nonalphanumeric characters before using the Jaro-Winkler algorithm to determine a match. |
| Double Metaphone | Matches phonetically similar strings using an improved coding system over the Soundex algorithm. It generates two codes for strings that could be pronounced in multiple ways. If the primary codes match for the two strings, or if the secondary codes match, then the strings match. The Double Metaphone algorithm accounts for alternate pronunciations in Italian, Spanish, French, and Germanic and Slavic languages. Unlike the Soundex algorithm, Double Metaphone encodes the first letter, so that "Kathy" and "Cathy" evaluate to the same phonetic code. |

### Creating Conditional Match Rules

To define a conditional match rule, complete the following steps:

1. On the top portion of the Match Rules tab or the Match Rules page, select **Conditional** in the Rule Type column.

   A Details section is displayed.

2. Click **Add** to add a new row.
3. Select an attribute in the Attribute column.
4. In the Algorithm column, select a comparison algorithm. See Table 23–2 for descriptions.
5. Specify a similarity score for the Edit Distance, Standardized Edit Distance, Jaro-Winkler, or Standardized Jaro-Winkler algorithms.
6. Select a method for handling blanks.

## Match Rules: Basic Example

The following discussions illustrate how some basic match rules apply to real data and how multiple match rules can interact with each other.

### Example: Matching and Merging Customer Data

Consider how you could use the Match Merge operator to manage a customer mailing list. Use matching to find records that refer to the same person in a table of customer data containing 10,000 rows.

For example, you can define a match rule that screens records that have similar first and last names. Through matching, you may discover that 5 rows could refer to the same person. You can then merge those records into one new record. For example, you can create a merge rule to retain the values from the one of the five matched records

with the longest address. The newly merged table now contains one record for each customer.

Table 23–3 shows records that refer to the same person prior to using the Match Merge operator.

***Table 23–3 Sample Records***

| Row | First Name | Last Name | SSN | Address | Unit | Zip |
|---|---|---|---|---|---|---|
| 1 | Jane | Doe | NULL | 123 Main Street | NULL | 22222 |
| 2 | Jane | Doe | 111111111 | NULL | NULL | 22222 |
| 3 | J. | Doe | NULL | 123 Main Street | Apt 4 | 22222 |
| 4 | NULL | Smith | 111111111 | 123 Main Street | Apt 4 | 22222 |
| 5 | Jane | Smith-Doe | 111111111 | NULL | NULL | 22222 |

Table 23–4 shows the single record for Jane Doe after using the Match Merge operator. Notice that the new record includes data from different rows in the sample.

***Table 23–4 Match-Merge Results***

| First Name | Last Name | SSN | Address | Unit | Zip |
|---|---|---|---|---|---|
| Jane | Doe | 111111111 | 123 Main Street | Apt 4 | 22222 |

### Example: How Multiple Match Rules Combine

If you create more than one match rule, Warehouse Builder determines two rows match if those rows satisfy any of the match rules. In other words, Warehouse Builder evaluates multiple match rules using OR logic.

The following example illustrates how Warehouse Builder evaluates multiple match rules.

In the top portion of the Match Rules tab, create two match rules as described in Table 23–5.

***Table 23–5 Two Match Rules***

| Name | Position | Rule Type | Usage | Description |
|---|---|---|---|---|
| Rule_1 | 1 | Conditional | Active | Match SSN |
| Rule_2 | 2 | Conditional | Active | Match Last Name and PHN |

In the lower portion of the tab, assign the details to Rule_1 as described in Table 23–6.

***Table 23–6 Details for Rule_1***

| Attribute | Position | Algorithm | Similarity Score | Blank Matching |
|---|---|---|---|---|
| SSN | 1 | Exact | 0 | Do not match if either is blank |

For Rule_2, assign the details as described in Table 23–7.

*Table 23–7 Details for Rule_2*

| Attribute | Position | Algorithm | Similarity Score | Blank Matching |
|---|---|---|---|---|
| LastName | 1 | Exact | 0 | Do not match if either is blank |
| PHN | 2 | Exact | 0 | Do not match if either is blank |

Assume that you have the data listed in Table 23–8.

*Table 23–8 Example Data*

| Row | First Name | Last Name | PHN | SSN |
|---|---|---|---|---|
| A | John | Doe | 650-123-1111 | NULL |
| B | Jonathan | Doe | 650-123-1111 | 555-55-5555 |
| C | John | Dough | 650-123-1111 | 555-55-5555 |

According to Rule_1, rows B and C match. According to Rule_2, rows A and B match. Therefore, because Warehouse Builder handles match rules using OR logic, all three records match.

### Example of Transitive Matching

The general rule is, if A matches B, and B matches C, then A matches C. Assign a conditional match rule based on similarity such as described in Table 23–9.

*Table 23–9 Conditional Match Rule*

| Attribute | Position | Algorithm | Similarity Score | Blank Matching |
|---|---|---|---|---|
| LastName | 1 | Similarity | 80 | Do not match if either is blank |

Assume that you have the data listed in Table 23–10.

*Table 23–10 Sample Data*

| Row | First Name | Last Name | PHN | SSN |
|---|---|---|---|---|
| A | John | Jones | 650-123-1111 | NULL |
| B | Jonathan | James | 650-123-1111 | 555-55-5555 |
| C | John | Jamos | 650-123-1111 | 555-55-5555 |

Jones matches James with a similarity of 80, and James matches Jamos with a similarity of 80. Jones does not match Jamos because the similarity is 60, which is less than the threshold of 80. However, because Jones matches James, and James matches Jamos, all three records match (Jones, James, and Jamos).

## Weight Match Rules

A weighted match rule enables you to assign an integer weight to each attribute included in the rule. You must also specify a threshold. For each attribute, the Match Merge operator multiplies the weight by the similarity score, and sums the scores. If the sum equals or exceeds the threshold, then the two records being compared are considered a match.

Weight match rules are most useful when you need to compare a large number of attributes, without having a single attribute that is different causing a non-match, as can happen with conditional rules.

Weight rules implicitly invoke the similarity algorithm to compare two attribute values. This algorithm returns an integer, a percentage value in the range 0 to 100, which represents the degree to which two values are alike. A value of 100 indicates that the two values are identical; a value of zero indicates no similarity whatsoever.

#### Similarity Algorithm

The method used to determine a match. Choose from these algorithms:

- **Edit Distance:** Calculates the number of deletions, insertions, or substitutions required to transform one string into another.
- **Jaro-Winkler:** Uses an improved comparison system over the Edit Distance algorithm. It accounts for the length of the strings and penalizes more for errors at the beginning. It also recognizes common typographical errors.

#### Attribute

Identifies the attribute that will be tested for a particular condition. You can select from any input attribute (INGRP1).

#### Maximum Score

The weight value for the attribute. This value should be greater than the value of Required Score to Match.

#### Score When Blank

The similarity value when one of the records is empty.

#### Required Score to Match

A value that represents the similarity required for a match. A value of 100 indicates that the two values are identical. A value of zero indicates there is no similarity.

### Example of Weight Match Rules

Table 23–11 displays the attribute values contained in two separate records that are read in the following order.

***Table 23–11 Example of Weight Match Rule***

| Record Number | First Name | Middle Name | Last Name |
|---|---|---|---|
| Record 1 | Robert | Steve | Paul |
| Record 2 | | Steven | Paul |

You define a match rule that uses the Edit Distance similarity algorithm. The Required Score to Match is 120. The attributes for first name and middle name are defined with a Maximum Score of 50 and Score When Blank of 20. The attribute for last name has a Maximum Score of 80 and a Score When Blank of 0.

Consider an example of the comparison of Record 1 and Record 2 using the weight match rule.

- Because first name is blank for Record 2, the Blank Score = 20.

- The similarity of middle name in the two records is 0.83. Since the weight assigned to this attribute is 50, the similarity score for this attribute is 41.5 (0.83 X 50).
- Because the last name attributes are the same, the similarity score for the last name is 1. The weighted score is 80 (1 X 80).

The total score for this comparison is 143 (20+43+80). Since this is more than the value defined for Required Score to Match, the records are considered a match.

### Creating Weight Match Rules

To use the Weight match rule, complete the following steps:

1. On the Match Rules tab or the Match Rules page, select **Weight** as the Rule Type.

   The Details tab is displayed at the bottom of the page.
2. Select **Add** at the bottom of the page to add a new row.
3. For each row, select an attribute to add to the rule using the Attribute column.
4. In **Maximum Score,** assign a weight to each attribute. Warehouse Builder compares each attribute using a similarity algorithm that returns a score between 0 and 100 to represent the similarity between the rows.
5. In **Score When Blank**, assign a value to be used when the attribute is blank in one of the records.
6. In **Required score to match,** assign an overall score for the match.

   For two rows to be considered a match, the total counts must be greater than the value specified in the Required score to match parameter.

## Person Match Rules

Built-in Person rules provide an easy and convenient way for matching names of individuals. Person match rules are most effective when the data has first been corrected using the Name and Address operator.

When you use Person match rules, you must specify which data within the record represents the name of the person. The data can come from multiple columns. Each column must be assigned an input role that specifies what the data represents.

To define a Person match rule, you must define the Person Attributes that are part of the rule. For example, you can create a Person match rule that uses the Person Attributes first name and last name for comparison. For each Person Attribute, you must define the Person Role that the attribute uses. Next you define the rule options used for the comparison. For example, while comparing last names, you can specify that hyphenated last names should be considered a match.

### Person Roles

Table 23–12 describes the roles for different parts of a name that are used for matching. On the Match Rules page or Match Rules tab, use the Roles column on the Person Attributes tab to define person details.

*Table 23–12 Name Roles for Person Match Rules*

| Role | Description |
|---|---|
| Prename | Prenames are compared only if the following are true:<br>▪ The Last_name and, if present, the middle name (Middle_name_std, Middle_name_2_std, and Middle_name_3_std roles) in both records match.<br>▪ The "Mrs. Match" option is selected.<br>▪ Either record has a missing First_name_std. |
| First Name Standardized | Compares the first names. By default, the first names must match exactly, but you can specify other comparison options as well.<br>First names match if both are blank. A blank first name will not match a nonblank first name unless the Prename role has been assigned and the "Mrs. Match" option is set. If a Last_name role has not been assigned, then a role of First_name_std must be assigned. |
| Middle Name Standardized, Middle Name 2 Standardized, Middle Name 3 Standardized | Compares the middle names. By default, the middle names must match exactly, but other comparison options can be specified. If more than one middle name role is assigned, then attributes assigned to the different roles are cross-compared.<br>For example, values for Middle_name_std will be compared not only against other Middle_name_std values, but also against Middle_name_2_std, if that role is also assigned. Middle names match if either or both are blank. If any of the middle name roles are assigned, then the First_name_std role must also be assigned. |
| Last Name | Compares the last names. By default, the last names must match exactly, but you can specify other comparison options. The last names match if both are blank, but not if only one is blank. |
| Maturity Post Name | Compares the post name, such as "Jr.", "III," and so on. The post names match if the values are exactly the same, or if either value is blank. |

## Person Details

Table 23–13 describes the options that determine a match for Person match rules. Use the Details tab of the Match Rules tab or the Match Rules page to define person details.

*Table 23–13 Options for Person Match Rule*

| Option | Description |
|---|---|
| Detect switched name order | Detects switched name orders such as matching "Elmer Fudd" to "Fudd Elmer". You can select this option if you selected First Name and Last Name roles for attributes on the Person Attributes tab. |
| Match on initials | Matches initials to names such as "R'" and "Robert". You can select this option for first name and middle name roles. |
| Match on substrings | Matches substrings to names such as "Rob" to "Robert". You can select this option for first name and middle name roles. |
| Similarity Score | Records are considered a match if the similarity is greater than or equal to the score. For example, "Susan" will match "Susen" if the score is less than or equal to 80.<br>Uses a similarity score to determine a match, as calculated by the Edit Distance or Jaro-Winkler algorithm. A value of 100 requires an exact match, and a value of 0 requires no similarity whatsoever. |

*Table 23–13 (Cont.) Options for Person Match Rule*

| Option | Description |
|---|---|
| Match on Phonetic Codes | Determines a match using either the Soundex or the Double Metaphone algorithm. |
| Detect compound name | Matches compound names to names such as "De Anne" to "Deanne". You can select this option for the first name role. |
| "Mrs" Match | Matches prenames to first and last names such as "Mrs. Washington" to "George Washington". You can select this option for the prename role. |
| Match hyphenated names | Matches hyphenated names to unhyphenated names such as "Reese-Jones" to "Reese". You can select this option for the last name role. |
| Detect missing hyphen | The operator detects missing hyphens, such as matching "Hillary Rodham Clinton" to "Hillary Rodham-Clinton". You can select this option for the last name role. |

### Creating Person Match Rules

To define a Person match rule, complete the following steps:

1. On the Match Rules tab, select **Person** as the Rule Type.

   The Person Attributes tab and Details tab are displayed at the bottom of the page.

2. In the left panel of the Person Attributes tab, select the attributes that describe a full name and use the right arrow to move them to the Name Roles section.

3. For each attribute, select the role that it plays in a name.

   You must define either the Last Name or First Name Standardized for the match rule to be effective. See Table 23–12 for the types of roles that you can assign.

4. Select the Details tab and select the applicable options as listed in Table 23–13.

## Firm Match Rules

Built-in Firm match rules provide an easy and convenient way for matching business names. Firm match rules are most effective when the data has first been corrected using the Name and Address operator. Similar to the Person rule, this rule requires users to set what data within the record represents the name of the firm. The data can come from multiple columns and each column specified must be assigned an input role that indicates what the data represents.

Note that you need not assign a firm role to every attribute, and not every role needs to be assigned to an attribute. The attributes assigned to firm roles are used in the match rule to compare the records. The attributes are compared based on the role that they have been assigned and other comparison options that you have set. For a complete list of firm roles and how each role is treated in a firm match rule, see "Firm Roles" on page 23-14.

### Firm Roles

Firm roles define the parts of a firm name that are used for matching. The options that you can select for firm role are Firm1 or Firm2. If you select one attribute for firm name, then select Firm1 as the role. If you select two attributes, then designate one of them as Firm1 and the other as Firm2.

- **Firm1:** If this role is assigned, then the business names represented by Firm1 are compared. Firm1 names will not be compared against Firm2 names unless the

Cross-match firm1 and firm2 box is checked. By default, the firm names must match exactly, but other comparison options can also be specified. Firm1 names do not match if either or both names are blank.

- **Firm2:** If this role is assigned, then the values of the attribute assigned to Firm2 will be compared. Firm2 names will not be compared against Firm1 names unless the Cross-match firm1 and firm2 box is checked. By default, the firm names must match exactly, but other comparison options can also be specified. Firm2 names do not match if either or both names are blank. If a Firm1 role is not assigned, then a Firm2 role must be assigned.

### Firm Details

Table 23–14 describes the rule options that you can set for each component of the firm name to determine a match.

***Table 23–14 Options for Firm Rules***

| Option | Description |
|---|---|
| Strip noise words | Removes the following words from Firm1 and Firm2 before matching: THE, AND, CORP, CORPORATION, CO, COMPANY, INC, INCORPORATED, LTD, TO, OF, and BY. |
| Cross-match firm1 and firm2 | When comparing two records for matching, in addition to matching firm1 to firm1 and firm2 to firm2 of the respective records, match firm1 against firm2 for the records. |
| Match on partial firm name | Uses the Partial Name algorithm to determine a match. For example, match "Midtown Power" to "Midtown Power and Light". |
| Match on abbreviations | Uses the Abbreviation algorithm to determine a match. For example, match "International Business Machines" to "IBM". |
| Match on acronyms | Uses the Acronym algorithm to determine a match. For example, match "CMB, North America" to "Chase Manhattan Bank, NA". |
| Similarity score | Uses a similarity score to determine a match, as calculated by the Edit Distance or Jaro-Winkler algorithm. Enter a value between 0 and 100 as the minimum similarity value required for a match. A value of 100 requires an exact match, and a value of 0 requires no similarity whatsoever.<br>Two records are considered as a match if the similarity is greater than or equal to the value of similarity score. |

### Creating Firm Match Rules

To define a Firm match rule, complete the following steps:

1. On the Match Rules tab or the Match Rules page, select **Firm** as the Rule Type.

   The Firm Attributes tab and Details tab are displayed at the bottom of the page.

2. In the left panel of the Firm Attributes tab, select one or two attributes that represent the firm name and click the right shuttle button.

   The attributes are moved to the Firm Roles box.

3. For each attribute, click **Roles**. From the list, select Firm 1 for the first attribute, and Firm 2 for the second attribute, if it exists.

4. On the Details tab, select the applicable options. For more details, see "Firm Details" on page 23-15.

## Address Match Rules

Address match rules provide a method of matching records based on postal addresses. Address match rules are most effective when the data has first been corrected using a Name and Address operator.

Address match rules work differently depending on whether or not the address being processed has been corrected using the Name and Address operator. Generally, corrected addresses have already been identified in a postal matching database, and are therefore syntactically correct, legal, and existing addresses according to the Postal Service of the country containing the address. Corrected addresses can be processed more quickly, because the match rule can make certain assumptions about their format.

Uncorrected addresses may be syntactically correct, but have not been found in a postal matching database. Addresses may have not been found because they are not in the database, or because there is no postal matching database installed for the country containing the address. Address match rules determine whether an address has been corrected based on the Is_found role. If the Is_found role is not assigned, then the match rule performs the comparisons for both the corrected and uncorrected addresses.

To create an Address match rule, assign address roles to the various attributes. The attributes assigned to address roles are used in the match rule to compare the records. Attributes are compared depending on which role they have been assigned, and what other comparison options have been set.

### Address Roles

Table 23–15 describes the address roles that you can select for each part of an address.

***Table 23–15 Address Roles***

| Role | Description |
|---|---|
| Primary Address | Compares the primary addresses. Primary addresses can be, for example, street addresses ("100 Main Street") or PO boxes ("PO Box 100"). By default, the primary addresses must match exactly, but a similarity option can also be specified.<br>The Primary_address role must be assigned. |
| Unit Number | Unit numbers (such as suite numbers, floor numbers, or apartment numbers) are compared if the primary addresses match. The unit numbers match if both are blank, but not if one is blank, unless the Match on blank secondary address option is set. If the Allow differing secondary address option is set, then the unit numbers are ignored. |
| PO Box | Compares the Post Office Boxes. The PO Box is just the number portion of the PO Box ("100"), and is a subset of the primary address, when the primary address represents a PO Box ("PO Box 100"). If the primary address represents a street address, then the PO Box will be blank. |
| Dual Primary Address | The Dual_primary_address is compared against the other record's Dual_primary_address and Primary_address to determine a match. |
| Dual Unit Number | Compares the Dual_unit_number address with the Dual_unit_number and Unit_number of the other record. The unit numbers will match if one or both are blank. To assign the Dual_unit_number role, the Dual_primary_address role must also be assigned. |
| Dual PO Box | Dual_PO_Box address of a record is compared with the Dual_PO_Box and the PO_Box of the other record. To assign the Dual_PO_Box role, the Dual_primary_address role must also be assigned. |

*Table 23–15 (Cont.) Address Roles*

| Role | Description |
|---|---|
| City | Compares the cities for uncorrected addresses. For corrected addresses, the cities are only compared if the postal codes do not match. If both City and State roles match, then the address line roles, such as Primary_address, can be compared.<br>By default, the cities must match exactly. But you may specify a last line similarity option. The cities match if both are blank, but not if only one is blank. If the City role is assigned, then the State role must also be assigned. |
| State | Assign this role only when also assigning the City role.<br>The states are compared for uncorrected addresses. For corrected addresses, the states are only compared if the postal codes do not match. If both State and City roles match, then the address line roles, such as Primary_address, can be compared. By default, the states must match exactly, but a last line similarity option may be specified. The states match if both are blank, but not if only one is blank. If the State role is assigned, then the City role must also be assigned. |
| Postal Code | For uncorrected address data, the operator does not use Postal Code.<br>The postal codes are compared for corrected addresses. For uncorrected addresses, the Postal_code role is not used. To match, the postal codes must be exactly the same. The postal codes are not considered a match if one or both are blank. If the postal codes match, then the address line roles, such as Primary_address, can be compared. If the postal codes do not match, then City and State roles are compared to determine whether the address line roles should be compared. |
| Is Found | The Is_found_flag attributes are not compared, but instead are used to determine whether an address has been found in a postal matching database, and therefore represents a legal address according to the postal service of the country containing the address. This determination is important because the type of comparison done during matching depends on whether or not the address has been found in the postal database. |

## Address Details

Table 23–16 describes the options for determining a match for an Address rule.

*Table 23–16 Options for Address Roles*

| Option | Description |
|---|---|
| Allow differing secondary address | Allow addresses to match even if the unit numbers are not null and are different. |
| Match on blank secondary address | Allow addresses to match even if exactly one unit number is null. |
| Match on either street or post office box | Match records if either the street address or the post office box match. |
| Address line similarity | Match if address line similarity >= the score. All spaces and non-alpanumeric characters are removed before the similarity is calculated. |
| Last line similarity | Match if the last line similarity >= score. The last line consists of city and state. All spaces and nonalphanumeric characters are removed before the similarity is calculated. |

## Creating Address Match Rules

To define an Address match rule, complete the following steps:

1. On the Match Rules tab or the Match Rules page, select **Address** as the Rule Type.

   The Address Attributes tab and Details tab are displayed at the bottom of the page.

2. In the left panel of the Address Attributes tab, select the attribute that represents the primary address. Use the right shuttle key to move it to the Address Roles Attributes column.

3. Click **Role Required** and designate that attribute as the Primary Address.

   You must designate one attribute as the primary address. If you do not assign the Primary Address role, then the match rule is invalid.

4. Add other attributes and designate their roles as necessary. See Table 23–15 for the types of roles that you can assign.

5. Select the Details tab and select the applicable options as listed in Table 23–16.

## Custom Match Rules

Custom match rules enable you to write your own comparison algorithms to match records. You can use any input attributes or match functions within this comparison. You can use an active custom rule to control the execution of passive rules.

Consider the following three passive built-in rules:

- `NAME_MATCH`: built-in name rule
- `ADDRESS_MATCH`: built-in address rule
- `TN_MATCH`: built-in conditional rule

You can create a custom rule to specify that two records can be considered a match if any two of these rules are satisfied. Example 23–1 describes the PL/SQL code used to create the custom match rule that implements this example.

***Example 23–1 Creating a Custom Rule Using Existing Passive Rules***

```
BEGIN
   RETURN(
       (NAME_MATCH(THIS_,THAT_) AND ADDRESS_MATCH(THIS_,THAT_))
     OR
       (NAME_MATCH(THIS_,THAT_) AND TN_MATCH(THIS_,THAT_))
     OR
       (ADDRESS_MATCH(THIS_,THAT_) AND TN_MATCH(THIS_,THAT_))
         );
END;
```

### Creating Custom Match Rules

To define a Custom match rule, complete the following steps:

1. On the Match Rules tab or the Match Rules page, select **Custom** as the Rule Type.

   A Details field is displayed at the bottom of the page with the skeleton of a PL/SQL program.

2. Click **Edit** to open the Custom Match Rules Editor.

   For more information about using the editor, select **Help Topic** from the Help menu.

3. To enter PL/SQL code, use any combination of the following:

- To read in a file, select **Open File** from the Code menu.
- To enter text, first position the cursor using the mouse or arrow keys, then begin typing. You can also use the commands on the Edit and Search menus.
- To reference any function, parameter, or transformation in the navigation tree, first position the cursor, then double-click or drag-and-drop the object onto the Implementation field.

4. To validate your code, select **Validate** from the Test menu.

   The validation results appear on the Messages tab.

5. To save your code, select **Save** from the Code menu.
6. To close the Custom Match Rules Editor, select **Close** from the Code menu.

## Merge Rules

Matching produces a set of records that are logically the same. Merging is the process of creating one record from the set of matched records. A Merge rule is applied to attributes in the matched record set to obtain a single value for the attribute in the merged record.

You can define one Merge rule for all the attributes in the Merge record or define a rule for each attribute. For instance, if the merged record is a customer record, it may have attributes such as ADDRESS1, ADDRESS2, CITY, STATE, and ZIP. You can write five rules that select the value of each attribute from up to five different records, or one Record rule that selects the values of all five attributes from one record. Use Record rules when multiple attributes compose a logical unit, such as an address. For example, City, State, and Zip Code might be three different attributes, but the data for these attributes should all come from the same record.

Table 23–17 describes the types of merge rules.

***Table 23–17 Merge Rule Types***

| Merge Rule | Description |
|---|---|
| Any | Uses the first nonblank value |
| Match ID | Merges records that have already been output from another Match Merge operator |
| Rank | Ranks the records from the match set. The associated attribute from the highest ranked record will be used to populate the merge attribute value |
| Sequence | Specify a database sequence for this rule. The next value of the sequence will be used for the value. |
| Min Max | Specify an attribute and a relation to choose the record to be used as a source for the merge attribute. |
| Copy | Choose a value from a different previously merged value. |
| Custom | Create a PL/SQL package function to select the merge value. The operator will provide the signature of this function. The user is responsible for the implementation of the rule from "BEGIN" to "END;" The matched records and merge record are parameters for this function. |
| Any Record | Identical to the Any rule, except that an Any Record rule applies to multiple attributes |
| Rank Record | Identical to the Rank rule, except that a Rank Record rule applies to multiple attributes |

*Table 23–17 (Cont.) Merge Rule Types*

| Merge Rule | Description |
|---|---|
| Min Max Record | Identical to the Min Max rule, except that a Min Max Record rule applies to multiple attributes |
| Custom Record | Identical to the Custom rule, except that a Custom Record rule applies to multiple attributes |

## Match ID Merge Rule

Use the Match ID merge rule to merge records that have already been output in the XREF group from another Match Merge operator. No other operator is valid for this type of input. For more information, see "Example: Using Two Match Merge Operators for Householding" on page 23-24.

### Next Value of the Sequence

Identifies the sequence that will be used by the rule.

### Sequences list

Lists all sequences defined in the current project.

### Select Sequence

Sets the sequence for the rule to the sequence currently selected in the list. Move a sequence from the sequences list to Select Sequence.

## Rank and Rank Record Merge Rules

Use the Rank and Rank Record rules when merging data from multiple sources. These rules enable you to identify your preference for certain sources. Your data must have a second input attribute on which the rule is based.

For example, the second attribute might identify the data source, and these data sources are ranked in order of reliability. The most reliable value would be used in the merged record. The merge rule might look like this:

```
INGRP1.SOURCE = 'Order Entry'
```

### Name

An arbitrary name for the rule. Warehouse Builder creates a default name such as RULE_0 for each Rank merge rule. You can replace these names with meaningful ones.

### Position

The order of execution. You can change the position of a rule by clicking on the row header and dragging the row to its new location. The row headers are the boxes to the left of the Name column.

### Expression Record Selection

The custom SQL expression used in the ranking. Click the Ellipsis button to display the Rank Rule Editor (also called the Expression Builder User Interface). Use this editor to develop the ranking expression.

## Sequence Merge Rule

The Sequence rule uses the next value in a sequence.

#### Next Value of the Sequence

Identifies the sequence that will be used by the rule.

#### Sequences list

Lists all sequences defined in the current project.

#### Select Sequence

Sets the sequence for the rule to the sequence currently selected in the list.

## Min Max and Min Max Record Merge Rules

The Min Max and Min Max Record rules select an attribute value based on the size of another attribute value in the record.

For example, you might select the First Name value from the record in each bin that contains the longest Last Name value.

#### Selecting Attribute

Lists all input attributes. Select the attribute whose values provide the order.

#### Attribute Relation

Select the characteristic for choosing a value in the selected attribute.

- **Minimum**. Selects the smallest numeric value or the oldest date value.
- **Maximum**. Selects the largest numeric value or the most recent date value.
- **Shortest**. Selects the shortest character value.
- **Longest**. Selects the longest character value.

## Copy Merge Rule

The Copy rule uses the values from another merged attribute.

#### Merged Attribute

Lists the other merged attributes, which you selected on the Merge Attributes page.

## Custom and Custom Record Merge Rules

The Custom and Custom Record rules use PL/SQL code that you provide to merge the records. The following is an example of a Custom merge rule, which returns the value of the TAXID attribute for record 1.

```
BEGIN
RETURN M_MATCHES(1)."TAXID";
END;
```

The following is an example of a Custom Record merge rule, which returns a record for record 1:

```
BEGIN
RETURN M_MATCHES(1);
END;
```

#### Merge Rules Detail

Displays the PL/SQL code composing your custom algorithm. You can edit code directly in this field or use the Custom Merge Rule Editor.

#### Edit

Displays the Custom Merge Rule Editor.

## Using the Match Merge Operator to Eliminate Duplicate Source Records

Use the Match Merge operator to identify matching records in a data source and to merge them into a single record.

The Match Merge operator has one input group and two output groups, Merge and Xref. The source data is mapped to the input group. The Merge group contains records that have been merged after the matching process is complete. The Xref group provides a record of the merge process. Every record in the input group will have a corresponding record in the Xref group. This record may contain the original attribute values and the merged attributes.

The Match Merge operator uses an ordered record stream as input. From this stream, it constructs the match bins. From each match bin, matched sets are constructed. From each matched set, a merged record is created. The initial query will contain an `ORDER BY` clause consisting of the match bin attributes.

### Steps to Use a Match Merge Operator

**To match and merge source data using the Match Merge operator:**

1. Drag and drop the operators representing the source data and the operator representing the merged data onto the Mapping Editor canvas.

   For example, if your source data is stored in a table, and the merged data will be stored in another table, drag and drop two Table operators that are bound to the tables onto the canvas.

2. Drag and drop a Match Merge operator onto the Mapping Editor canvas.

   The MatchMerge Wizard is displayed.

3. On the Name page, the Name field contains a default name for the operator. You can change this name or accept the default name.

   You can enter an optional description for the operator.

4. On the Groups page, you can rename groups or provide descriptions for them.

   This page contains the following three groups:

   - **INGRP1:** Contains input attributes.
   - **MERGE:** Contains the merged records (usually this means fewer records than INGRP1).
   - **XREF:** Contains the link between the original and merged data sets. This is the tracking mechanism used when a merge is performed.

5. On the Input Connections page, move the attributes that you want to match and merge from the Available Attributes section to the Mapped Attributes section. Click **Next**.

   The Available Attributes section of this page displays nodes for each operator on the canvas. Expand a node to display the attributes contained in the operator,

select the attributes, and use the shuttle arrows to move selected attributes to the Mapped Attributes section.

---

**Note:** The Match Merge operator requires an ordered input data set. If you have source data from more than one operator, then use a Set Operation operator to combine the data and obtain an ordered data set.

---

6. On the Input Attributes page, review the attribute data types and lengths.

   In general, if you go through the wizard, you need not change any of these values. Warehouse Builder populates them based on the output attributes.

7. On the Merge Output page, select the attributes to be merged from the input attributes.

   These attributes appear in the Merge output group (the cleansed group). The attributes in this group retain the name and properties of the input attributes.

8. On the Cross Reference Output page, select attributes for the XREF output group.

   The Source Attributes section contains all the input attributes and the Merge attributes that you selected on the Merge Output page. The attributes from the Merge group are prefixed with MM_. The other attributes define the unmodified input attribute values. Select at least one attribute from the Merge group that will provide a link between the input and Merge groups.

9. On the Match Bins page, specify the match bin attributes. These attributes are used to group source data into match bins.

   After the first deployment, you can choose whether to match and merge all records or only new records. To match and merge only the new records, select **Match New Records Only**.

   You must designate a condition that identifies new records. The Match Merge operator treats the new records in the following way:

   - No matching is performed for any records in a match bin unless the match bin contains a new record.
   - Old records are not compared with each other.
   - A matched record set is not presented to the merge processing unless the matched record set contains a new record.
   - An old record is not presented to the Xref output unless the record is matched to a new record.

   For more information about match bin attributes and match bins, see "Overview of the Matching and Merging Process" on page 23-3.

10. On the Define Match Rules page, define the match rules that will be used to match the source data.

    Match rules can be active or passive. A passive match rule is generated but not automatically invoked. You must define at least one active match rule.

    For more information about the match rules, the types of match rules that you can define, and the steps used to define them, see "Match Rules" on page 23-5.

11. On the Merge Rules page, define the rules that will be used to merge the sets of matched records created from the source data.

You can define Merge rules for each attribute in a record or for the entire record. Warehouse Builder provides different types of Merge rules.

For more information about the types of Merge rules and the steps to create Merge rules, see "Merge Rules" on page 23-19.

12. On the Summary page, review your selections. Click **Back** to modify any selection that you made. Click **Next** to complete creating the Match Merge operator.
13. Map the Merge group of the Match Merge operator to the input group of the operator that stores the merged data.

## Considerations When Designing Mappings Containing Match Merge Operators

Be aware of the following considerations as you design your mapping:

- **Operating modes:** A mapping that contains a Match Merge operator can only run in set-based mode. Operators may accept either set-based or row-based input and generate either set-based or row-based output. SQL is set-based, so a set of records is processed at one time. PL/SQL is row-based, so each row is processed separately. When the Match Merge operator matches records, it compares each row with the subsequent row in the source, and generates row-based code only.
- **SQL-based operators before Match Merge:** The Match Merge operator accepts set-based SQL input, but generates only row-based PL/SQL output. Any operators that generate only SQL code must precede the Match Merge operator. For example, the Joiner, Lookup, and Set operators generate set-based SQL output, so they must precede the Match Merge operator. If set-based operators appear after Match Merge operator, then the mapping is invalid. If you need to process the output of a match-merge mapping using a set-based SQL operator, then stage the output in an intermediate table.
- **PL/SQL input:** The Match Merge operator requires SQL input except from another Match Merge operator, as described in "Example: Using Two Match Merge Operators for Householding" on page 23-24. If you want to precede a Match Merge operator with an operator that generates only PL/SQL output, then you must first load the data into a staging table.
- **Refining data from Match Merge operators:** To achieve greater data refinement, map the XREF output from one Match Merge operator into another Match Merge operator. This scenario is the one exception to the SQL input rule for Match Merge operators. With additional design elements, the second Match Merge operator accepts PL/SQL. For more information, see "Example: Using Two Match Merge Operators for Householding" on page 23-24.

### Restrictions on Using the Match Merge Operator

- Because the match-merge process generates only PL/SQL, you cannot map the Merge or XREF output groups of the Match Merge operator to a SQL-only operator such as a Sorter operator or another Match Merge operator.
- Because the Match Merge operator only accepts SQL input, you cannot map the output of the Name and Address operator directly to the Match Merge operator. You must use a staging table.

## Example: Using Two Match Merge Operators for Householding

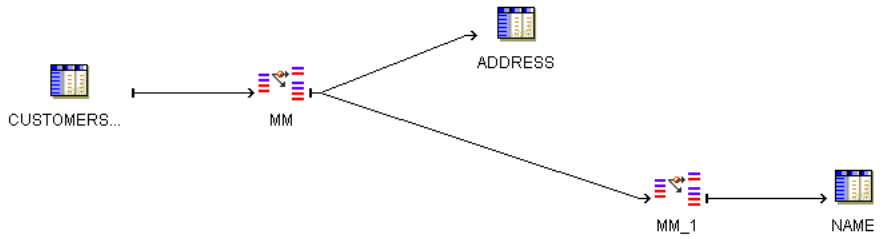
Most match-merge operations can be performed by a single Match Merge operator. However, if you are directing the output to two different targets, then you may need to use two Match Merge operators in succession.

For example, when householding name and address data, you may need to merge the data first for addresses and then again for names. Assuming that you map the MERGE output to a target table, you can map the XREF group to another Match Merge operator.

> **Note:** Although you could map the XREF group to a staging table, this intermediate step adds significant overhead. The match-merge functionality is designed to support maximum performance using two Match Merge operators together as described in this section.

Figure 23–3 shows a mapping that uses two Match Merge operators. The XREF group from MM is mapped directly to MM_1. For this mapping to be valid, you must assign the Match ID generated for the first XREF group as the Match Bin rule on the second Match Merge operator.

***Figure 23–3 Householding Data: XREF Group Merged to Second Match Merge Operator***




> **Note:** A more complete solution for the householding problem might apply name and address cleansing on individual records before performing the matching and merging to group customers into households.

# Part IV

## Reference

This part contains the following chapters:


- Chapter 24, "Mappings and Process Flows Reference"
- Chapter 25, "Source and Target Operators"
- Chapter 26, "Data Flow Operators"
- Chapter 27, "Activities in Process Flows"
- Chapter 28, "Warehouse Builder Transformations Reference"
- Chapter 29, "Warehouse Builder Code Template Tools and Substitution Methods Reference"

# 24

# Mappings and Process Flows Reference

This chapter contains the following topics:


- Configuring ETL Objects
- Configuring Mappings Reference
- Configuring Process Flows Reference


## Configuring ETL Objects

Earlier in the design phase, you defined a logical model for your target system using Oracle Warehouse Builder design objects. This chapter includes reference information for assigning physical properties to mappings and process flows. This chapter presents configuration parameters in the order in which they appear in the user interface.

This section contains the following topics:


- Configuring Mappings Reference on page 24-1
- Configuring Process Flows Reference on page 24-15


## Configuring Mappings Reference

When you configure mappings properly, you can improve the Extract, Transform, and Load (ETL) performance. Use this section as a reference for setting configuration parameters that govern how data is loaded and to optimize code for better performance.

Configuration parameters for mappings are classified into the following categories:


- Runtime Parameters on page 24-1
- Code Generation Options on page 24-4


In addition to these parameters, you can configure the source and target operators in the mapping as described in "Sources and Targets Reference" on page 24-8. There are some additional parameters that you can configure for CT mappings as described in "Configuring Code Template (CT) Mappings" on page 24-12.

### Runtime Parameters

When you configure Runtime parameters for a mapping, you set the default behaviors for the mapping. You can override these parameters when you execute the mapping either in the Control Center, the Process Flow Editor, or Oracle Enterprise Manager.

The Runtime parameters include the following parameters:

- Analyze Table Sample Percentage
- Bulk Size
- Chunk Execute Resume Task
- Chunk Force Resume
- Chunk Number of Times to Retry
- Chunk Parallel Level
- Commit Frequency
- Default Audit Level
- Default Operating Mode
- Default Purge Group
- Maximum Number of Errors
- Number of Threads to Process Chunks

### Analyze Table Sample Percentage

When you select the Analyze Table Statements option, Warehouse Builder estimates when gathering statistics on the target tables. After data is loaded into the target tables, statistics used for cost-based optimization are gathered on each target table. You can set this parameter to the percentage of rows in each target table used for this analysis.

### Bulk Size

Use **Bulk Size** to specify the number of rows in each bulk for PL/SQL Bulk Processing. Warehouse Builder uses the Bulk Size parameter only when the Bulk Processing Code option is selected and the operating mode is set to row-based. For more information, see *Oracle PL/SQL Reference Guide.*

### Chunk Execute Resume Task

This parameter specifies whether the parallel chunking task must be resumed. Select False to run a new parallel chunking task. Select True to resume the parallel chunking task. At runtime, you can override the value set for this parameter by using the Input Parameters dialog box.

### Chunk Force Resume

This parameter specifies whether the parallel chunking task must be resumed if it has been stopped due to errors. Set this parameter to True to force Warehouse Builder to resume the parallel chunking task. At runtime, you can override the value set for this parameter by using the Input Parameters dialog box.

### Chunk Number of Times to Retry

This parameter specifies the number of times that Warehouse Builder attempts to resume a failed parallel chunking task. Enter a numeric value that specifies the number of attempts. At runtime, you can override the value set for this parameter by using the Input Parameters dialog box.

### Chunk Parallel Level

This parameter specifies the number of threads that must be used to process the chunks in parallel. Enter a numeric value for this parameter. Setting this parameter to NULL or zero causes the mapping to be executed serially.

At runtime, you can override the value set for this parameter by using the Input Parameters dialog box.

### Commit Frequency

Commit frequency applies only to non-bulk mode mappings. Bulk mode mappings commit according to the bulk size.

If you set the **Default Operating Mode** to row-based and deselect Bulk Processing Code**,** then use the **Commit Frequency** parameter to determine the number of rows to be processed before a commit operation. Warehouse Builder commits data to the database after processing the number of rows specified in this parameter.

If you select the Bulk Processing Code option, set the Commit Frequency equal to the Bulk Size. If the two values differ, then Bulk Size overrides Commit Frequency and a commit operation is implicitly performed for every bulk size.

### Default Audit Level

Use **Default Audit Level** to indicate the audit level used when executing the package. Audit levels dictate the amount of audit information captured in the runtime schema when the package is run. The audit level settings are:

- **None:** No auditing information is recorded in run time.
- **Statistics:** Statistical auditing information is recorded in run time.
- **Error Details:** Error information and statistical auditing information is recorded in run time.
- **Complete:** All auditing information is recorded in run time. Running a mapping with the audit level set to **Complete** generates a large amount of diagnostic data, which may quickly fill the allocated tablespace.

### Default Operating Mode

For mappings with a PL/SQL implementation, select a default operating mode. The operating mode, you select can greatly affect mapping performance. For details on how operating modes affect performance, see "Set-Based Versus Row-Based Operating Modes" on page 10-4. You can select one of the following operating modes:

- **Set based:** A single SQL statement that inserts all data and performs all operations on the data is generated. This increases the speed of Data Manipulation Language (DML) operations. Set based mode offers optimal performance but minimal auditing details.
- **Row based:** Statements that process data row by row are generated. The select statement is a SQL cursor. All subsequent statements are PL/SQL. Because data is processed row by row, the row-based operating mode has the slowest performance but offers exhaustive auditing details.
- **Row based (Target Only):** A cursor select statement is generated and attempts are made to include as many operations as possible in the cursor. For each target, Warehouse Builder generates a PL/SQL insert statement and inserts each row into the target separately.

- **Set based fail over row based:** The mapping is executed in set based mode. If an error occurs, then the execution fails and the mapping is started over again in the row-based mode. This mode is recommended for use only in test environments and is not recommended for use in production environments.
- **Set based fail over row based (Target Only):** The mapping is first executed in set based mode. If an error occurs, then the execution fails over to **Row based (Target Only)** mode. This mode is recommended for use only in test environments and is not recommended for use in production environments.

### Default Purge Group

Default Purge Group is used when executing the package. Each audit record in the runtime schema is assigned to the purge group specified.

### Maximum Number of Errors

Use **Maximum Number of Errors** to indicate the maximum number of errors allowed while executing the package. Execution of the package terminates when the number of errors exceeds the maximum number of errors value.

The Maximum Number of Errors parameter applies to the count of errors for the entire mapping run, whether run in set-based, row-based, or failover modes. Consider the following cases:

- Maximum number of errors is set to 50 and the mapping is run in set-based mode. The data did not load successfully. One error resulted from failure of the set-based load DML statement. The mapping return status is WARNING.
- Maximum number of errors is set to 50, the mapping is run in set-based mode, and Enable Constraint parameter is set to false. The data is loaded successfully, but 60 constraint violation errors occurred during reenabling of the constraint. The mapping return status is ERROR.
- Max number of errors is set to 50 and the mapping is run in row-based mode. Some of the data loaded successfully, but with many errors. The mapping will terminate after reaching the 50th error. The mapping return status is ERROR.
- Max number of errors is set to 50 and the mapping is run in set-based failover to row-based mode. The data did not load successfully in the set-based mode. One error resulted from the failure of the set-based load DML statement. Some of the data loaded successfully in the row-based mode, but with many errors. The mapping will terminate after reaching the 49th error in the row-based mode because there was one error counted in set-based mode. The mapping return status is ERROR.

### Number of Threads to Process Chunks

The Number of Threads to Process Chunks parameter represents the number of threads used to process the chunks of source data. Setting this parameter to zero means that the mapping will not be run in parallel.

## Code Generation Options

The Code Generation Options include the following:

- ANSI SQL Syntax
- Commit Control
- Analyze Table Statements

- Enable Parallel DML
- Optimized Code
- Authid
- Use Target Load Ordering
- ERROR TRIGGER
- Bulk Processing Code
- Generation Mode

### ANSI SQL Syntax

If you select this option, ANSI SQL syntax is generated. Otherwise, Oracle SQL syntax is generated.

### Commit Control

**Automatic**: This is the default setting. Warehouse Builder loads and then automatically commits data based on the mapping design. This setting is valid for all mapping types. For multiple targets in a single mapping, data is committed based on target by target processing (insert, update, delete).

**Automatic Correlated**: Automatic correlated commit is a specialized type of automatic commit that applies only to PL/SQL mappings with multiple targets.Warehouse Builder considers all targets collectively and commits or rolls back data uniformly across all targets.

The mapping behavior varies according to the operating mode that you select. For more information about automatic correlated commit, see "Committing Data from a Single Source to Multiple Targets" on page 10-7.

**Manual**: Select manual commit control for PL/SQL mappings when you want to interject complex business logic or perform validations before committing data.

You have the following options for specifying manual commits:

- You can define the commit logic within the mapping as described in "Embedding Commit Logic into the Mapping" on page 10-9.
- You can commit data in a process flow or from a SQL*Plus session as described in "Committing Data Independently of Mapping Design" on page 10-10.

**No Commit**: If you set this option, then Warehouse Builder mapping does not issue a commit while the mapping executes.

### Analyze Table Statements

If you select this option, code is generated for analyzing the target table after the target is loaded, if the resultant target table is double or half its original size.

If the target table is not in the same schema as the mapping and you want to analyze the table, then you must grant ANALYZE ANY to the schema owning the mapping.

### Enable Parallel DML

If you select this option, parallel DML is enabled at run time. Executing DML statements in parallel improves the response time of data-intensive operations in large databases that are present in a data warehouse.

### Optimized Code

Select this option to improve performance for mappings that include the Splitter operator and inserts into multiple target tables. When this option is selected and the mapping is executed by Oracle*9i* or later, a single SQL statement is generated (`multi_table_insert`) that inserts data into multiple tables based on the same set of source data.

Note that the multiple table insert is performed only if this option is selected and the Oracle target module database is Oracle*9i* or later. The multiple table insert is performed only for mappings in set-based mode that include a Splitter operator, and does not include active operators, such as an Aggregator or Joiner operator, between the Splitter and the target. In addition, the multiple insert is available only for tables. It is not available for views, materialized views, dimensions, or cubes. Each target table must have fewer than 999 columns. For detailed instructions on how to create a mapping with multiple targets, see "Example: Creating Mappings with Multiple Targets" on page 26-40.

Do not select this option for mappings run in row-based mode or for mappings executed by Oracle*8i* server. Also, do not select this option when auditing information for individual targets is required.

When this option is selected, one total SELECT and INSERT count is returned for all targets.

### Authid

Specifies the AUTHID option to be used while generating the code. The options that you can select are Current_User, Definer, or None.

### Use Target Load Ordering

For PL/SQL mappings with multiple targets, you can generate code that defines an order for loading the targets. This is important when a parent-child relationship exists between two or more targets in a mapping. The option is selected by default.

### ERROR TRIGGER

Specify the name of the error trigger procedure in this field.

### Bulk Processing Code

If this configuration parameter is selected and the operating mode is set to row-based, Warehouse Builder generates PL/SQL bulk processing code. PL/SQL bulk processing improves row-based ETL performance by collecting, processing, and writing rows in bulk, instead of doing it row by row. The size of each bulk is determined by the configuration parameter Bulk Size. Set-based mode offers optimal performance, followed by bulk processing, and finally by row-based mode. For more information, see *Oracle PL/SQL Reference Guide.*

### Generation Mode

By default, when code is generated for a mapping, the code for all possible operating modes is generated. That is, if you set the Default Operating Mode to **Set based**, Warehouse Builder still generates code for all possible operating modes when **Generation Mode** is set to All Operating Modes. This enables you to switch the operating modes for testing purposes at run time.

## Chunking Options

The parameters in the Chunking Options section enable you to configure parallel chunking for a PL/SQL mapping.

Expand the Details node under Chunking Options to specify the following parallel chunking options:

- Chunking Method
- Chunk by Number Column

  Expand the Chunk by Number Column node to set the following parameters: Chunk Table, Chunk Column, and Chunk Size.
- Chunk by ROWID

  Expand the Chunk by ROWID node to set the following parameters: Chunk Table, Chunk Type, and Chunk Size.
- Chunk by SQL Statement

  Expand the Chunk by SQL Statement node to set the following parameters: SQL Statement and SQL Statement Chunk Type.

### Chunking Method

Use this parameter to select the method to be used for performing parallel chunking. Following are the options you can select for this parameter:

- **None:** Does not perform parallel chunking.
- **Chunk By Number Column:** Uses a numeric column in the source data as a basis for dividing the source data into chunks. When you select this option, you must set the parameters listed under the Chunk By Number Column node.
- **Chunk by ROWID:** Divides the source rows into chunks depending on the `ROWID` of the column in the source data. When you select this option, you must set the parameters listed under the Chunk By ROWID node.
- **Chunk by SQL Statement:** Uses a SQL statement to divide the source data into chunks. When you select this option, you must set the parameters listed under the Chunk By SQL Statement node.

### Chunk Table

Use Chunk Table to select the source table on which parallel data chunking must be performed.

### Chunk Column

The Chunking Column parameter is applicable when you perform parallel chunking using a numeric column in the source table. Use this parameter to select the source column based on which parallel chunking is performed.

### Chunk Size

Use Chunk Size to specify the range of each chunk.

### Chunk Type

Use chunk Type to specify how source data must be divided into chunks based on the `ROWID` of the source columns. The options you can select are By data blocks and By rows.

### SQL Statement

Specify the SQL statement using which the source data is divided into smaller chunks.

### SQL Statement Chunk Type

Use this parameter to specify how the SQL statement, specified using SQL Statement, chunks the source data. Select one of the following options:

- **NUMBER_COLUMN:** Data is chunked using a numeric column in the source data.
- **ROWID:** Data is chunked using the `ROWID` of columns

## SCD Updates

The parameters in the SCD Updates section enable you to configure serial chunking for a PL/SQL mapping. The SCD Updates options contains one parameter called Chunking Strategy.

### Chunking Strategy

The Chunking Strategy parameter enables to perform serial chunking. Select Serial to perform serial chunking on the source data. Select None to disable serial chunking for the PL/SQL mapping.

## Sources and Targets Reference

For relational and dimensional sources and targets such as tables, views, and cubes, Warehouse Builder displays the following set of properties for each operator:

- Use LCR APIs
- Database Link
- Location
- Conflict Resolution
- Schema
- Partition Exchange Loading
- Hints
- Constraint Management
- SQL*Loader Parameters

### Use LCR APIs

By default, this setting is enabled and DML is performed using LCR APIs if available. If no LCR APIs are available, then the standard DML is used.

### Database Link

This parameter is maintained for backward compatibility only.

In previous releases, you could select a database link by name from the list. Source operators can be configured for schemas and database links, but targets can be configured for schemas only. Sources and targets can reside in different schemas, but they must reside in the same database instance.

### Location

This setting specifies the location that is used to access the source or target operator.

### Conflict Resolution

Enable this setting to detect and resolve any conflicts that may arise during DML operations using the LCR APIs.

### Schema

This parameter is maintained for backward compatibility only.

In previous releases, you could link the mapping to a particular schema by clicking on the Schema field and typing a name.

### Partition Exchange Loading

Use the settings in this section to enable Partition Exchange Loading (PEL) into a target table. For specific information about each of these settings and additional information about how to design mappings for PEL, see "Improved Performance through Partition Exchange Loading" on page 10-21.

### Hints

Define loading or extraction hints. Application developers often develop insights into their data. For example, they know that a query runs much faster if a set of tables is joined in one order rather than another. Warehouse Builder can incorporate these insights into the generated SQL code packages as SQL Optimizer Hints.

When you select a hint from the Hints dialog box, the hint appears in the **Existing Hints** field. Enter additional text as appropriate in the Extra Text column. The editor includes the hint in the mapping definition as is. There is no validation or checking on this text.

You can define loading hints for mappings that load data in INSERT or UPDATE mode. By default, commonly used hints such as APPEND and PARALLEL are added. For all loading modes other than INSERT, the APPEND hint causes no effect and you can choose to remove it.

Hint is available during mapping configuration. To configure a hint:

1. In the Projects Navigator, expand the **Databases** folder, and then the required module.
2. In the module, expand the Mappings node.
3. Right-click the required mapping and select **Configure**.

   The Configuration tab displays the configuration parameters of the mapping.
4. In the Configuration tab, expand the required operator type and then expand the required operator.
5. Expand the Hints node and click the Ellipsis button to the right of a hint type to enter a hint.

For information about optimizer hints and how to use them, see *Oracle Database Performance Tuning Guide*.

### Constraint Management

Configure the following Constraint Management parameters:

- **Exceptions Table Name:** All rows that violate their foreign key constraints during reenabling are logged into the specified exceptions table. No automatic truncation of this table is performed either before or after the load. Constraint violations are also loaded into the runtime audit error tables.

  For SQL and PL/SQL loading, if you do not specify an exceptions table, invalid rows are loaded into a temporary table located in the default tablespace and then loaded into the Runtime Audit error table. The table is dropped at the end of the load.

  If you are using SQL*Loader direct path loading, you must specify an exception table. Consult the SQL*Loader documentation for more information.

- **Enable Constraints:** If you set this option to False, Warehouse Builder disables constraints on the target tables, loads data, and then reenables the constraints. Constraint violations found during reenable are identified in the runtime audit error table and, if specified, loaded into an exceptions table. If you set this option to True, then Warehouse Builder does not manage constraints and the data from the source is loaded into the target table.

  When you disable constraints, loading is quicker because a constraint check is not performed. However, if exceptions occur for any rows during reenabling, the constraints for those rows will remain in a nonvalidated state. These rows are logged in the runtime audit error table by their `ROWID`. You must manually inspect the error rows to take any necessary corrective action.

  The disabling and enabling of constraints happens on the target table. When the Enable Constraints parameter is set to True, the constraints on the target table will be disabled prior to the loading of data, and will be reenabled after the loading of data. When the constraints are reenabled, the entire table is scanned and rows that violate the constraints are logged in the exceptions table. These rows are reported as constraint violation errors in the audit browser.

  Consider a scenario where the target table is empty and the Enable Constraints parameter is set to True. Initially suppose that the source table has 10 rows, of which 2 rows violate the constraint on the target table. When the mapping is executed, the constraints on the target table are first disabled, then data is loaded (all 10 rows), and then constraints on the target table are reenabled. When the constraints are reenabled, the 2 rows that violate the constraints are logged into the exceptions table. The audit browser reports that there are 2 constraint violation errors.

  Later, the mapping is again executed with a new source table containing 20 rows, of which 5 rows violate the constraint on the target table. After the data is loaded into the target table (all 20 rows), the target table has 30 rows. When the constraints on the target table are reenabled, 7 rows will be logged in to the exceptions table and reported as constraint violation errors in the audit browser. These include the 5 rows reported newly as well as the 2 rows reported initially. This is because Warehouse Builder scans the entire target table, which means that all 30 rows will be checked and therefore the 2 rows with violations from the first data load will still be included. Warehouse Builder cannot identify only the new rows added when the mapping was executed the second time. Therefore, unless you truncate the target table before each data load, you will always see the constraint violations from the previous data loads reported each time.

  Setting the Enable Constraints option to True is subject to the following restrictions:

  – For set-based operating mode, the foreign key constraints on the targets are disabled before loading, and then reenabled after loading. This parameter has

no effect on foreign key constraints on other tables referencing the target table. If the load is done using SQL*Loader instead of a SQL or PL/SQL package, then a reenable clause is added to the .ctl file.

– For set-based fail over to row-based and set-based fail over to row-based (target only) operating modes, the deselect setting disables the foreign key constraints on the targets before loading and then reenables them if the load succeeds in set-based mode. This setting has no effect on foreign keys referencing other tables. If the load fails over to row-based, then loading will repeat in row-based mode and all constraints remain enabled.

---

**Note:** Constraint violations created during reenabling will not cause the load to fail from set-based mode over to row-based mode.

---

– For row-based or row-based (target only) operating modes, all foreign key constraints remain enabled even if the option is not selected.

– For the TRUNCATE/INSERT DML type, the deselect setting disables foreign key constraints on other tables referencing the target table before loading, and then reenables the constraints after loading, regardless of the default operating mode.

### SQL*Loader Parameters

When you have a Table operator that contains inputs from a flat file, you must configure the following SQL*Loader Parameters properties:

- **Partition Name:** Indicates that the load is a partition-level load. Partition-level loading enables you to load one or more specified partitions or subpartitions in a table. Full database, user, and transportable tablespace mode loading does not support partition-level loading. Because incremental loading (incremental, cumulative, and complete) can be done only in full database mode, partition-level loading cannot be specified for incremental loads. In all modes, partitioned data is loaded in a format such that partitions or subpartitions can be selectively loaded.
- **Sorted Indexes Clause:** Identifies the indexes on which the data is presorted. This clause is allowed only for direct path loads. Because the data sorted for one index is not usually in the right order for another index, you specify only one index in the SORTED INDEXES clause. When the data is in the same order for multiple indexes, all indexes can be specified at once. All indexes listed in the SORTED INDEXES clause must be created before you start the direct path load.
- **Singlerow:** Intended for use during a direct path load with APPEND on systems with limited memory, or when loading a small number of records into a large table. This option inserts each index entry directly into the index, one record at a time. By default, SQL*Loader does not use SINGLEROW to append records to a table. Index entries are stored in a temporary area and merged with the original index at the end of the load. Although this method achieves better performance and produces an optimal index, it requires extra storage space. During the merge, the original index, the new index, and the space for new entries all simultaneously occupy storage space. With the SINGLEROW option, storage space is not required for new index entries or for a new index. Although the resulting index may not be as optimal as a freshly sorted one, it takes less space to produce. It also takes more time, because additional UNDO information is generated for each index insert. This option is recommended when the available storage is limited. It is also recommended when the number of records to be loaded is small compared to the size of the table. A ratio of 1:20 or less is considered small.

- **Trailing Nullcols:** Sets SQL*Loader to treat any relatively positioned columns that are not present in the record as null columns.
- **Records To Skip:** Invokes the SKIP command in SQL*Loader. SKIP specifies the number of logical records from the beginning of the file that should not be loaded. By default, no records are skipped. This parameter continues loads that have been interrupted for some reason. It is used for all conventional loads, for single-table direct loads, and for multiple-table direct loads when the same number of records is loaded into each table. It is not used for multiple-table direct loads when a different number of records is loaded into each table.
- **Database File Name:** Specifies the names of the export files to import. The default extension is .dmp. Because you can export multiple export files, you may must specify multiple file names to be imported. You must have read access to the imported files. You must also have the IMP_FULL_DATABASE role.

## Configuring Code Template (CT) Mappings

The configuration parameters for Code Template (CT) mappings are similar to the parameters for PL/SQL mappings. However, for CT mappings, the nodes SCD Updates and Chunking Options are not displayed because these options only apply to PL/SQL mappings.

You can set an additional parameters called Use Enclosure Character and Source File Operator for CT mappings. These parameters are displayed in the Configuration tab of the mapping.

### Use Enclosure Character

Select this parameter to enclose SQL identifiers in the generated code of CT mappings. Warehouse Builder supports delimited identifiers in CT mapping across heterogeneous database systems.

For example, if the Use Enclosure Character parameter is selected, and the enclosure character is set to square brackets, the SQL statement generated is as follows:

```
SELECT [My Empno], [My Emp Name] FROM [My Employees]
```

By default, the enclosure character used for Oracle Database, IBM DB2 UDB, and SQL Server is the double quotation marks. You can set the enclosure character that should be used for each platform by using OMB*Plus. For example, to change the enclosure character for the SQL Server platform to square brackets, use the following OMB*Plus command.

```
OMBALTER PLATFORM 'SQLSERVER' SET PROPERTIES (ENCLOSURE_CHAR)
VALUES ('/[/]');
```

For a CT mapping that has the Use Enclosure Character parameter selected to work correctly, you must set up the correct enclosure character for any new platform that you create. For example, you create a new platform called `MYSQL` that references a MySQL database. Your CT mapping uses a table from the MySQL database and you want to generate code using the double quotation mark as the enclosure character. Before you generate code for the CT mapping, you must set the enclosure character for the MYSQL platform using the following statement:

```
OMBALTER PLATFORM 'MYSQL' SET PROPERTIES (ENCLOSURE_CHAR) VALUES
('"')
```

**Note:** The Use Enclosure Parameter is not supported for CT mappings that contain Flat File operators.

### SQL Loader Data Files

Use the SQL Loader Data Files section to set the Source File Operator parameter. For CT mappings that contain one or more Flat File operators, you can set parameters for the associated data files.

#### Source File Operator

Use this property to specify the Flat File operator that must be associated with the current data file.

If your CT mapping contains only one Flat File operator, setting the Source File Operator parameter is optional. However, if your CT mapping contains more than one Flat File operator, you must set this parameter. This parameter enables Warehouse Builder to understand the relationship between the data file and the Flat File operator.

This property is available starting with Oracle Warehouse Builder 11g Release 2 (11.2.0.2).

## Configuring Flat File Operators

The Configuration tab of the Flat File operator contains additional settings for Flat File operators, depending on how the operators are used in the mapping.

- **Flat File Operators as a Target:** A PL/SQL deployment code package is generated. For information about configuring the parameters associated with a Flat File operator used as a target, see "Flat File Operators as a Target" on page 24-13.
- **Flat File Operator as a Source:** SQL*Loader scripts are generated. For information about the parameters associated with a Flat File operator used as a source, see "Flat File Operator as a Source" on page 24-14.

## Flat File Operators as a Target

**To configure properties unique to mappings with flat file targets:**

1. Select a mapping from the Projects Navigator, select **Design** from the menu bar, and select **Configure.**

   Or, right-click the mapping you want to configure and select **Configure**.

   Warehouse Builder displays the Configuration tab for the mapping.

2. Choose the parameters that you want to configure and click the space to the right of the parameter name to edit its value.

   For each parameter, you can either select an option from a list, enter a value, or click the Ellipsis button to display another properties dialog box.

3. Select the **Deployable** option to generate a set of scripts for mapping objects marked as deployable. If this option is not selected for a mapping, then scripts are not generated for that mapping.

4. Set **Language** to the type of code that you want to generate for the selected mapping. The options you can choose from depend upon the design and use of the operators in the mapping. Depending on the mapping, you can select from PL/SQL, ABAP (for an SAP source mapping), or SQL*Loader.

5. Specify the location to deploy the mapping.
6. Under **Runtime Parameters**, set the **Default Operating Mode** to **Row based (target only)**. This type of mapping will not generate code in any other default operating mode. For a description of each runtime parameter, see "Runtime Parameters" on page 24-1.
7. Set the **Code Generation Options** as described in "Code Generation Options" on page 24-4.
8. Set the Sources and Targets Reference as described in "Sources and Targets Reference" on page 24-6.
9. For **Access Specification,** specify the name of the flat file target in **Target Data File Name.** For the **Target Data File Location,** specify a target file located on the computer where you installed the Runtime Platform. Select **Output as XML file** if you want the output to be in an xml file.

## Flat File Operator as a Source

**To configure a mapping with a Flat File operator as a source:**

1. Select a mapping from the Projects Navigator, select **Design** from the menu bar, and select **Configure.** Or, right-click the mapping that you want to configure and select **Configure**.
2. Select the parameters that you want to configure and click the space to the right of the parameter name to edit its value.

   For each parameter, you can specify whether you want the parameter to be selected, select an option from a list, enter a value, or click the Ellipsis button to display another properties dialog box.
3. Select the **Deployable** option to generate SQL*Loader script.
4. Specify the **Log File Location** and **Log File Name.**
5. Select **Continue Load.**

   If SQL*Loader runs out of space for data rows or index entries, the load is discontinued. If the **Continue Load** option is selected, then an attempt is made to continue discontinued loads.
6. In **Nls Characterset,** specify the character set to place in the CHARACTERSET clause.
7. Select **Direct Mode** to indicate that a direct path load will be done. If this option is not selected, then a conventional load will be done. In general, direct mode is faster.
8. Select **Operation Recoverable** to indicate that the load is recoverable. If this option is not selected, then the load is not recoverable and records are not recorded in the redo log.
9. Configure the following parameters that affect the OPTIONS clause in the SQL*Loader script that is generated for mappings with flat file sources.

   **Perform Parallel Load**: If this option is selected, direct loads can operate in multiple concurrent sessions.

   **Errors Allowed:** If the value specified is greater than 0, then the ERRORS = n option is generated. SQL*Loader terminates the load at the first consistent point after it reaches this error limit.

**Records To Skip:** If the value specified is greater than 0, then the SKIP = n option is generated. This value indicates the number of records from the beginning of the file that should not be loaded. If the value is not specified, then no records are skipped.

**Records To Load:** If the value specified is greater than 0, then the LOAD = n option is generated. This value specifies the maximum number of records to load. If a value is not specified, then all of the records are loaded.

**Rows Per Commit:** If the value specified is greater than 0, then the ROWS = n option is generated. For direct path loads, the value identifies the number of rows to read from the source before a data is saved. For conventional path loads, the value specifies the number of rows in the bind array.

**Read Size:** If the value specified is greater than 0, then the READSIZE = n option is generated. The value is used to specify the size of the read buffer.

**Bind Size:** If the value specified is greater than 0, then the BINDSIZE = n option is generated. The value indicates the maximum size, in bytes, of the bind array.

**Read Buffers:** If the value specified is greater than 0, then the READBUFFERS = n clause is generated. READBUFFERS specifies the number of buffers to use during a direct path load. Do not specify a value for READBUFFERS unless it is necessary.

**Preserve Blanks:** If this option is selected, then the PRESERVE BLANKS clause is generated. PRESERVE BLANKS retains leading white space when optional enclosure delimiters are not present. It also leaves the trailing white space intact when fields are specified with a predetermined size.

**Database File Name:** This parameter enables you to specify the characteristics of the physical files to be loaded. The initial values of these parameter is set from the properties of the flat file used in the mapping.

If this parameter is set to a nonblank value, then the FILE= option is generated. The value specified is enclosed in single quotation marks in the generated code.

**Control File Location** and **Control File Name:** The control file name necessary for audit details.

For more information about each SQL*Loader option and clause, see *Oracle Database Utilities*.

10. Expand the **Runtime Parameters** to configure your mapping for deployment.

    **Audit:** Select this option to perform an audit when the package is executed.

    **Default Purge Group:** The Default Purge Group is used when executing the package. Each audit record in the runtime schema is assigned to the purge group specified.

11. Expand **Sources and Targets Reference** to set the physical properties of the operators in the mapping as described in "Sources and Targets Reference" on page 24-8.

## Configuring Process Flows Reference

**To configure a process flow module:**

1. Right-click the process flow module and select **Configure.**

   Warehouse Builder displays the Configuration tab for the process flow module.

2. Set the parameters for **Evaluation Location** and **Identification Location.**

   **Evaluation Location** is the location from which this process flow is evaluated.

   **Identification Location** provides the location where the generated code will be deployed to.

**To configure a process flow package**:

1. Right-click the process flow package and select **Configure**.

   Warehouse Builder displays the Configuration tab for the process flow package.

2. Set the parameters for **Referred Calendar** and **Generation Comments.**

   **Referred Calendar** provides the schedule to associate with this package.

   **Generation Comments** provides additional comments for the generated code.

Click any of the activities of a package to view its properties.

Under **Path Settings,** set the following properties for each activity in the process flow:

**Execution Location:** The location from which this activity is executed. If you configured Oracle Enterprise Manager, then you can select an OEM agent to execute the process flow.

**Remote Location:** The remote location for FTP activities only.

**Working Location:** The working location for FTP, FILE EXISTS, and External Process activities only.

**Deployed Location:** The deployment location. This setting applies to transformation activities only. For activities referring to predefined transformations, you must change the setting from **Use Default Location** and specify a valid location.

Under **General Properties,** you can view the bound name, which is the name of the object that the activity represents in the process flow. Only mapping, transformation, and subprocess activities have bound names.

Under **Execution Settings,** select the option **Use Return as Status**.

This setting governs the behavior for activities that return `NUMBER` in their output. These activities include the FTP, User Defined, and Transform activities. When you select **Use Return as Status**, the Process Flow Editor assigns the outgoing transition conditions based on the following numeric return values for the activity:

1 = Success Transition

2 = Warning Transition

3 = Error Transition

# 25

# Source and Target Operators

This chapter provides details on how to use operators as sources and targets in an Oracle Warehouse Builder mapping.

This chapter contains the following topics:


- List of Source and Target Operators
- Using Oracle Source and Target Operators
- Using Remote and non-Oracle Source and Target Operators
- Using Flat File Source and Target Operators


## List of Source and Target Operators

The source and target operators are:


- Constant Operator on page 25-9
- Construct Object Operator on page 25-9
- Cube Operator on page 25-10
- Data Generator Operator on page 25-12
- Dimension Operator on page 25-14
- Expand Object Operator on page 25-18
- External Table Operator on page 25-19
- Flat File Operator on page 25-32
- Mapping Input Parameter Operator on page 25-20
- Mapping Output Parameter Operator on page 25-21
- Materialized View Operator on page 25-22
- Queue Operator on page 25-23
- Sequence Operator on page 25-25
- Table Operator on page 25-26
- Varray Iterator Operator on page 25-29
- View Operator on page 25-30

# Using Oracle Source and Target Operators

Oracle source and target operators refer to operators that are bound to Oracle data objects in the workspace. Use these operators in a mapping to load data into or source data from Oracle data objects.

## Setting Properties for Oracle Source and Target Operators

The Property Inspector displays the properties of the selected operator. It contains the following categories of parameters for source and target operators:

- **Change Data Capture:** This category is displayed only for tables and views. It contains the following properties: Capture Consistency, Change Data Capture Filter, Enabled, and Trigger Based Capture.
- **Conditional Loading:** You can set the following properties: Target Filter for Update, Target Filter for Delete, and Match By Constraint.
- **Data Chunking:** This category is displayed for tables, views, and materialized views. It contains the following properties: Chunk Filter Condition, Chunking Enabled, and Parallel Chunk Filter Condition. For information about these properties, see "Chunking for Table Operators" on page 25-27.
- **Error Table:** You can set the Error Table Name, Roll up Errors, and Select Only Errors from this Operator properties. This section of properties is displayed only for the following mapping operators: Table, View, Materialized View, External Table, and Dimension.
- **General:** Under the General node, you can set Primary Source, Target Load Order, and the Loading Type. Depending upon the type of target, you can set different values for the Loading Type as described in Loading Types for Oracle Target Operators and Loading Types for Flat Files.
- **Keys (read-only):** You can view the Key Name, Key Type, and Referenced Keys. If the operator functions as a source, then the key settings are used in conjunction with the Join operator. If the operator functions as a target, then the key settings are used in conjunction with the Match By Constraint parameter.
- **File Properties:** Under the file properties, you can view the Bound Name.
- **Temp Stage Table:** This category is displayed for tables, views, materialized views, and external tables.It contains the following properties: Extra DDL Clauses, Is Temp Staging Table, and Temp Stage Table ID. These properties are described in "Creating Temporary Tables While Performing ETL" on page 25-28.

### Capture Consistency

The Capture Consistency determines the type of Change Data Capture performed. Select one of the following options:

- **Consistent Set:** Performs consistent set Change Data Capture.
- **Non Consistent Set:** Performs non consistent set Change Data Capture.
- **None:** Does not perform Change Data Capture.

### Change Data Capture Filter

The Change Data Capture Filter property represents the filter used to capture changes for a particular subscriber.

### Enabled

Select the Enabled property to enable the functionality that performs Change Data Capture.

### Trigger Based Capture

Select the Trigger Based Capture property to indicate changes are captured and propagated using triggers on the source tables.

### Primary Source

Oracle Application Embedded Data Warehouse (EDW) users, refer to EDW documentation. All other users can disregard this parameter.

### Loading Types for Oracle Target Operators

Select a loading type for each target operator using the Loading Type property.

For all Oracle target operators, except for dimensions and cubes, select one of the following options.

- **CHECK/INSERT:** Checks the target for existing rows. If there are no existing rows, then the incoming rows are inserted into the target.
- **DELETE:** The incoming row sets are used to determine which of the rows on the target are to be deleted.
- **DELETE/INSERT:** Deletes all rows in the target and then inserts the new rows.
- **INSERT:** Inserts the incoming row sets into the target. The insert operation fails if a row already exists with the same primary or unique key.
- **INSERT/UPDATE:** For each incoming row, the insert operation is performed first. If the insert fails, then an update operation occurs. If there are no matching records for update, then the insert is performed. If you select `INSERT/UPDATE` and the Default Operating Mode is set to **Row based**, then you must set unique constraints on the target. If the operating mode is set to **Set based**, then Warehouse Builder generates a `MERGE` statement.
- **NONE:** No operation is performed on the target. This setting is useful for testing. Extraction and transformations run but have no effect on the target.
- **TRUNCATE/INSERT:** Truncates the target and then inserts the incoming row set. If you select this option, then the operation cannot be rolled back even if the execution of the mapping fails. Truncate permanently removes the data from the target.
- **UPDATE:** Uses the incoming row sets to update existing rows in the target. If no rows exist for the specified match conditions, then no changes are made.

  If you set the configuration parameter **PL/SQL Generation Mode** of the target module to Oracle 10g, Oracle 10*g*R2, Oracle 11gR1, or Oracle 11gR2, the target is updated in set-based mode. The generated code includes a MERGE statement without an insert clause. For modules configured to generate Oracle 9*i* and earlier versions of PL/SQL code, the target is updated in row-based mode.
- **UPDATE/INSERT:** If the Default Operating Mode is set to **Row based**, for each incoming row, the update is performed first followed by an insert if no rows are updated. If the Default Operating Mode is set to **Set based**, then a `MERGE` statement is generated. Set-based mode can only be generated if the PL/SQL Generation Mode parameter of the target module is Oracle 10g or higher.

For dimensions and cubes, the Loading Type property has the following options: Load and Remove. Use Load to load data into the dimension or cube. Use Remove to remove data from the dimension or cube.

### Loading Types for Flat File Targets

Configure **SQL*Loader parameters** to define SQL*Loader options for your mapping. The values chosen during configuration directly affect the content of the generated SQL*Loader and the run time control files. SQL*Loader provides two methods for loading data:

- **Conventional Path Load:** Executes a SQL INSERT statement to populate tables in Oracle Database.
- **Direct Path Load:** Eliminates much of the Oracle Database overhead by formatting Oracle data blocks and writing the data blocks directly to the database files. Because a direct load does not compete with other users for database resources, it can usually load data at or near disk speed.

  Certain considerations such as restrictions, security, and backup implications are inherent to each method of access to database files. For more information, see *Oracle Database Utilities*.

  When designing and implementing a mapping that extracts data from a flat file using SQL*Loader, you can configure different properties affecting the generated SQL*Loader script. Each load operator in a mapping has an operator property called Loading Type. The value contained by this property affects how the SQL*Loader INTO TABLE clause for that load operator is generated. Although SQL*Loader can append, insert, replace, or truncate data, it cannot update any data during its processing.

  Table 25–1 lists the INTO TABLE clauses associated with each load type and their affect on data in the existing targets.

***Table 25–1 Loading Types and INTO TABLE Relationship***

| Loading Types | INTO TABLE Clause | Affect on Target with Existing Data |
|---|---|---|
| INSERT/UPDATE | APPEND | Adds additional data to target |
| DELETE/INSERT | REPLACE | Removes existing data and replaces with new (DELETE trigger fires) |
| TRUNCATE/INSERT | TRUNCATE | Removes existing data and replaces with new (DELETE trigger fires) |
| CHECK/INSERT | INSERT | Assumes target table is empty |
| NONE | INSERT | Assumes target table is empty |

### Target Load Order

This property enables you to specify the order in which multiple targets within the same mapping are loaded. Warehouse Builder determines a default load order based on the foreign key relationships. Use this property to overrule the default order.

### Target Filter for Update

If the condition evaluates to true, the row is included in the update loading operation.

### Target Filter for Delete

If the condition evaluates to true, the row is included in the delete loading operation.

### Match By Constraint

When loading target operators with the UPDATE or the DELETE conditions, you can specify matching criteria. You can set matching and loading criteria manually or choose from several built-in options. Use Match By Constraint to indicate whether unique or primary key information on a target overrides the manual matching and loading criteria set on its attributes. When you click the property Match By Constraint, Warehouse Builder displays a list containing the constraints defined on that operator and the built-in loading options.

If you select **All Constraints**, all manual attribute load settings are overruled and the data is loaded as if the load and match properties of the target attributes were set as displayed in Table 25–2.

When you select **All Constraints**, the load setting Load Column when Updating Row is not automatically assumed to be No for key attributes. However, when performing MERGE generation, this will be validated and a validation warning is displayed when certain attributes that are used for UPDATE matching are also used for UPDATE loading.

***Table 25–2 All Constraints Target Load Settings***

| Load Setting | Key Attribute | All Other Attributes |
|---|---|---|
| Match Column when Updating Row | YES | NO |
| Match Column when Deleting Row | YES | NO |

If you select **No Constraints,** all manual load settings are honored, and the data is loaded accordingly.

If you select a constraint previously defined for the operator, all manual attribute load settings are overruled, and the data is loaded as if the load and match properties of the target were set as displayed in Table 25–3.

When you select a previously defined constraint, the load setting Load Column when Updating Row is not automatically assumed to be No for key attributes. However, when performing MERGE generation, a validation warning is displayed when certain attributes that are used for UPDATE matching are also used for UPDATE loading

***Table 25–3 Target Load Settings for a Selected Constraint***

| Load Setting | Selected Key Attributes | All Other Attributes |
|---|---|---|
| Match Column when Updating Row | YES | NO |
| Match Column when Deleting Row | YES | NO |

### Reverting Constraints to Default Values

If you made changes at the attribute level and you want to default all settings, click **Advanced.** A list containing the loading options is displayed. Warehouse Builder defaults the settings based on the constraint type that you select.

For example, if you want to reset the match properties for all key attributes, click **Advanced,** select **No Constraints,** and click **OK.** The manual load settings are overwritten and the data is loaded based on the settings displayed in Table 25–4.

*Table 25–4 Default Load Settings for Advanced No Constraints*

| Load Setting | All Key Attributes | All Other Attributes |
|---|---|---|
| Load Column when Inserting Row | YES | NO |
| Load Column when Updating Row | YES | YES |
| Match Column when Updating Row | NO | NO |
| Match Column when Deleting Row | NO | NO |

Alternatively, if you click **Advanced** and select **All Constraints,** the manual load settings are overwritten and data is loaded based on the settings displayed in Table 25–5.

*Table 25–5 Default Load Settings for Advanced All Constraints*

| Load Setting | All Key Attributes | All Other Attributes |
|---|---|---|
| Load Column when Inserting Row | YES | YES |
| Load Column when Updating Row | NO | YES |
| Match Column when Updating Row | YES | NO |
| Match Column when Deleting Row | YES | NO |

### Bound Name

The name used by the code generator. If an operator is currently bound and synchronized, then this property is read-only. If an operator is not yet bound, then you can edit the bound name within the Mapping Editor before you synchronize it to a workspace object.

### Key Name

Name of the primary, foreign, or unique key.

### Key Columns

Local columns that define this key. Each key column is comma-delimited if the operator contains more than one key column.

### Key Type

Type of key, either primary, foreign, or unique.

### Referenced Keys

If the operator contains a foreign key, **Referenced Keys** displays the primary key or unique key for the referenced object.

### Error Table Name

The name of the error table that stores the invalid records during a load operation.

### Roll up Errors

Select **Yes** to roll up records selected from the error table by the error name. Thus all errors generated by a particular input record will be rolled up into a single record with the error names concatenated in the error name attribute.

### Select Only Errors from this Operator

Rows selected from the error table will contain only errors created by this operator in this map execution

## Setting Attribute Properties

For each attribute in a source and target operator, parameters are categorized into the following types:

- **General:** Under the General properties, you can view the Bound Name property, the Business name, and Physical name.
- **Chunking_column:** This category is displayed only for attributes in Table, View, and Materialized View operators. It contains one property Chunking Number Column.
- **Code Template Metadata Tags:** This category contains the following properties: SCD, UD1, UD2, UD3, UD4, UD5, and UPD.
- **Data Type Information:** The data type properties are applicable to all operators. They include Data Type, Precision, Scale, Length, and Fractional Seconds Precision.
- **Loading Properties:** The operators for tables, dimensions, cubes, views, and materialized views have a Loading Properties category. This category contains the following settings: Load Column When Inserting Row, Load Column When Updating Row, Match Column When Updating Row, Update: Operation, and Match Column When Deleting Row.

Certain operators contain properties that are specific to that particular operator. These properties are listed under the Operator Specific Properties node and are described in the sections that discuss that operator.

### Bound Name

Name used by the code generator to identify this item. By default, it is the same name as the item. This is a read-only setting when the operator is bound.

### Data Type

Data type of the attribute.

### Precision

The maximum number of digits this attribute will have if the data type of this attribute is a number or a float. This is a read-only setting.

### Scale

The number of digits to the right of the decimal point. This only applies to number attributes.

### Length

The maximum length for a `CHAR`, `VARCHAR`, or `VARCHAR2` attribute.

### Fractional Seconds Precision

The number of digits in the fractional part of the datetime field. It can be a number between 0 and 9. This property is used only for `TIMESTAMP`, `TIMESTAMP WITH TIME ZONE`, and `TIMESTAMP WITH LOCAL TIME ZONE` data types.

### Load Column When Inserting Row

This setting prevents data from moving to a target even though it is mapped to do so. If you select **Yes** (default), then the data will reach the mapped target.

### Load Column When Updating Row

This setting prevents the selected attribute data from moving to a target even though it is mapped to do so. If you select **Yes** (default), then the data reaches the mapped target attribute. If all columns of a unique key are not mapped, then the unique key is not used to construct the match condition. If no columns of a unique key are mapped, then an error is displayed. If a column (not a key column) is not mapped, then it is not used in loading.

### Match Column When Updating Row

This setting updates a data target row only if there is a match between the source attribute and mapped target attribute. If a match is found, then an update occurs on the row. If you set this property to **Yes** (default), then the attribute is used as a matching attribute. If you use this setting, then all the key columns must be mapped. If there is only one unique key defined on the target entity, then use constraints to override this setting.

### Update: Operation

You can specify an update operation to be performed when a matching row is located. An update operation is performed on the target attribute using the data of the source attribute. Table 25–6 lists the update operations that you can specify and describes the update operation logic.

***Table 25–6 Update Operations***

| Operation | Example | Result If Source Value = 5 and Target Value = 10 |
|---|---|---|
| **=** | TARGET = SOURCE | TARGET = 5 |
| **+=** | TARGET = SOURCE + TARGET | TARGET = 15 (5 + 10) |
| **-=** | TARGET = TARGET - SOURCE | TARGET = 5 (10 - 5) |
| **=-** | TARGET = SOURCE - TARGET | TARGET = negative 5 (5 - 10) |
| ***=** | TARGET = SOURCE * TARGET | TARGET = 50 (5 * 10) |
| **/=** | TARGET = TARGET / SOURCE | TARGET = 2 (10 / 5) |
| **=/** | TARGET = SOURCE / TARGET | TARGET = 0.5 (5 /10) |
| **\|\|=** | TARGET = TARGET \|\| SOURCE | TARGET = 105 (10 concatenated with 5) |
| **=\|\|** | TARGET = SOURCE \|\| TARGET | TARGET = 510 (5 concatenated with 10) |

### Match Column When Deleting Row

Deletes a data target row only if there is a match between the source attribute and mapped target attribute. If a match is found, then a delete operation occurs on the row. If you set this property to **Yes** (default), then the attribute is used as a matching attribute. Constraints can override this setting.

### Chunking Number Column

Select Chunking Number column for an attribute to use that attribute as the chunking attribute. This property is applicable only for parallel chunking.

## Constant Operator

The Constant operator enables you to define constant values. You can place constants anywhere in any PL/SQL or ABAP mapping.

The Constant operator produces a single output group that contains one or more constant attributes. Warehouse Builder initializes constants at the beginning of the execution of the mapping.

For example, use a Constant operator to load the value of the current system date into a Table operator. In the Expression Builder, select the public transformation `SYSDATE` from the list of predefined transformations.

For more information about public transformations, see Chapter 4, "Overview of Transforming Data".

**To define a Constant operator in a PL/SQL or ABAP mapping:**

1. Drop a Constant operator onto the Mapping Editor canvas.
2. Right-click the Constant operator and select **Open**.

   The Constant Editor dialog box is displayed.
3. On the Output tab, create an output attribute by clicking the blank cell in the Attribute column and entering the name of the output attribute.

   The default data type assigned is NUMERIC. You can modify the data type and any other parameters associated with it such as length, precision, and so on.
4. Enter the expression associated with the output attribute.

   Use the Expression field for an output attribute to enter the expression. Or, click the Ellipsis button to the right of the Expression field to use the Expression Builder dialog box to define an expression.

   The length, precision, and scale properties assigned to the output attribute must match the values returned by the expressions defined in the mapping. For `VARCHAR`, `CHAR`, or `VARCHAR2` data types, enclose constant string literals within single quotation marks, such as, `'my_string'`.
5. Click **OK** to close the Constant Editor dialog box.

## Construct Object Operator

The Construct Object operator enables you to create SQL object data types (object types and collection types), PL/SQL object types, and cursors in a mapping by using the individual attributes that they comprise.

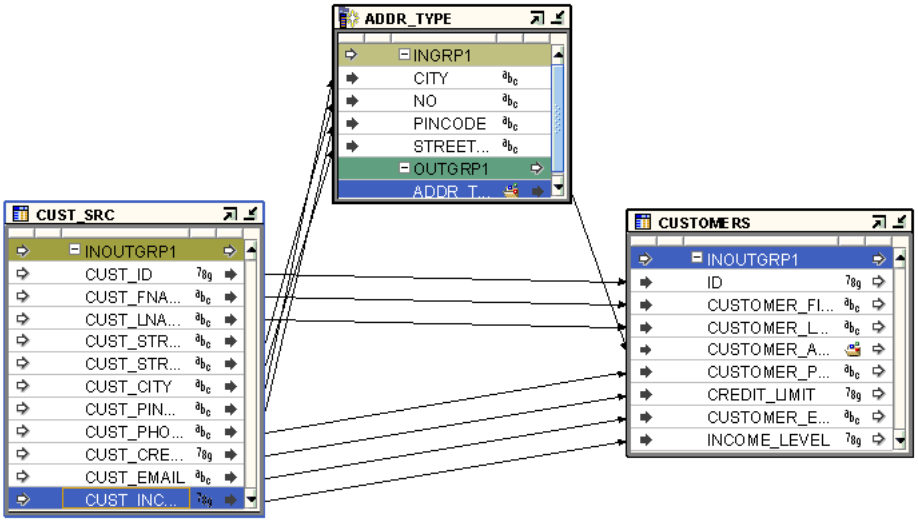
For example, you can use a Construct Object operator to create a SQL object type that is used to load data into a table that contains a column whose data type is an object type. You can also use this operator to create the payload that loads data into an advanced queue. This operator also enables you to construct a SYS.REFCURSOR object.

The Construct Object operator has one input group and one output group. The input group represents the individual attributes that comprise the object type. The output of the Construct Object operator is an object type that is created using the individual

attributes. In a mapping, the data type of the output attribute of the Construct Object operator should match the target attribute to which it is being mapped.

Figure 25–1 displays a mapping that uses a Construct Object operator. The source table CUST_SRC uses separate attributes to store each component of the customer address. But the target table CUSTOMERS uses an object type to store the customer address. To load data from the CUST_SRC table into the CUSTOMERS table, the customer address should be an object type whose signature matches that of the customer address in CUSTOMERS. The Construct Object operator takes the individual attributes from CUST_SRC, that store the customer address as input, and constructs an object type. The Construct Object operator is bound to the user-defined data type CUST_ADDR stored in the workspace.

***Figure 25–1 Construct Object Operator in a Mapping***




**To define a Construct Object operator in a mapping:**

1. Drag and drop a Construct Object operator onto the Mapping Editor canvas.
2. Use the Add Construct Object dialog box to create or select an object. For more information about these options, see "Using the Add Operator Dialog Box to Add Operators" on page 5-15.
3. Map the individual source attributes that are used to construct the object to the input group of the Construct Object operator.
4. Map the output attribute of the Construct Object operator to the target attribute. The data type of the target attribute should be an object type.

   Note that the signatures of the output attribute of the Construct Object operator and the target attribute should be the same.

## Cube Operator

Use the Cube operator to source data from or load data into cubes.

The Cube operator contains a group with the same name as the cube. This group contains an attribute for each of the cube measures. It also contains the attributes for the surrogate identifier and business identifier of each dimension level that the cube references. Additionally, the Cube operator displays one group for each dimension that the cube references.

If you specify an Orphan Management Policy and create an error table for a cube, when you add this cube to a mapping, the Cube operator contains a group called ERROR_<*cube_name*>. This is an output group that contains attributes that are displayed in the Cube operator details, but not in the Cube operator on the mapping canvas. To create data flows using these attributes, display these attributes on the canvas by selecting the ERROR_<*cube_name*> group on the canvas and from the Graph menu, selecting **Select Display Set**, and then **All**.

You can bind a Cube operator to a cube defined in any Oracle module in the current project. You can also synchronize the Cube operator and update it with changes made to the cube to which it is bound. To synchronize a Cube operator, right-click the Cube operator on the Mapping Editor canvas and select **Synchronize**.

### Cube Operator Properties

The Cube operator has the following properties that you can use to load a cube.

**Loading Type** Use the Loading Type property to specify if you are loading data into the cube or removing data from the cube. Set one of the following values for this property.

- INSERT_LOAD

  All records from the source data set are inserted into the cube. Oracle recommends that you set this option with orphan management.

- LOAD

  The records from the source data set are merged into the cube. Thus, if a record that is being loaded from the source already exists in the cube, this record is updated. Any records in the source data set that do not exist are inserted.

- REMOVE

  The records in the cube that match the incoming source records are deleted from the cube.

**Target Load Order** This property determines the order in which multiple targets within the same mapping are loaded. Warehouse Builder determines a default order based on the foreign key relationships. You can use this property to overrule the default order.

**Enable Source Aggregation** Set this property to True to aggregate source data before loading the cube. The source data is grouped by all dimension keys.

The default Aggregation function on cube measure attributes is SUM. You can changed the setting Source Aggregation Function property of the cube measure.

If you set the Orphan Management Policy for the cube to Default Dimension Record and you set the Enable Source Aggregation property of the Cube operator to False, execution errors may occur when the cube table is updated. Thus, in this scenario, Warehouse Builder displays a warning during cube validation.

**Solve the Cube** Select YES for this property to aggregate the cube data while loading the cube. This increases the load time, but decreases the query time. The data is first loaded and then aggregated.

**Incremental Aggregation** Select this option to perform incremental loads. This means that if the cube has been solved earlier, subsequent loads will only aggregate the new data.

**AW Staged Load** If you set AW Staged Load to true, the set-based AW load data is staged into a temporary table before loading into the AW.

**AW Truncate Before Load** Indicates whether all existing cube values should be truncated before loading the cube. Setting this property to YES truncates existing cube data.

### Cube Attribute Properties

You can set the following properties for attributes in a Cube operator.

**Update:Operation** This property is only applicable to cubes with a ROLAP implementation and to attributes that represent cube measures.

Specifies the type of update operation for cube measures while loading the cube. The options that you can select are +=, -=, /=,=,=-, =||, and ||=. The default values is = and using this value inserts the source fact records into the cube.

For example, if you set this property to +=, the source attribute value that is mapped to the cube measure is added to the existing measure value. If there are multiple source fact records with the same dimensionality, then ensure that you use an Aggregator operator to aggregate these records before loading them into the cube.

**Null Data Value** Specifies the value that is interpreted as null by the orphan management policy of the cube. While loading cubes, you can use the Orphan Tab of the Cube editor to specify how records with null dimension key values and records with invalid dimension key values are treated.

The default value for this property is NULL.

## Data Generator Operator

Use a Data Generator operator to introduce a sequence, record number, or system date into a mapping. You can use a single Data Generator operator to map more than one of these functions.

---

**Recommendation:** For PL/SQL mappings, use a Constant Operator or Sequence Operator instead of a Data Generator operator.

---

For mappings with flat file sources and targets, the Data Generator operator connects the mapping to SQL*Loader to generate the data stored in the database record.

The following functions are available:

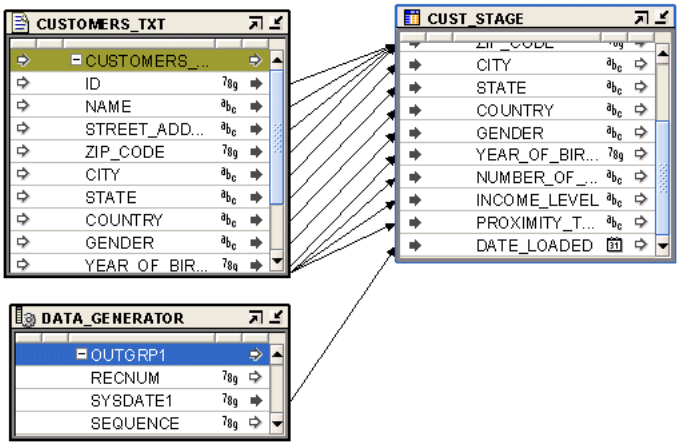
- RECNUM
- SYSDATE1
- SEQUENCE

Warehouse Builder can generate data by specifying only sequences, record numbers, system dates, and constants as field specifications. SQL*Loader inserts as many records as are specified by the LOAD keyword.

The Data Generator operator has one output group with predefined attributes corresponding to Record Number, System Date, and a typical Sequence. Use the Data Generator operator to obtain record number, system date, or a sequence. For all other functions, use a Constant operator or Expression operator.

Figure 25–2 shows a mapping that uses the Data Generator operator to obtain the current system date. The data from a flat file CUSTOMERS_TXT is loaded in to a staging table CUST_STAGE. The staging table contains an additional attribute for the date on which the data was loaded. The SYSDATE1 attribute of the Data Generator operator is mapped to the DATE_LOADED attribute of the staging table CUST_STAGE.

***Figure 25–2 Data Generator in a Mapping***




**To define a Data Generator in a SQL*Loader mapping:**

1. Drop a Data Generator operator onto the Mapping Editor canvas.
2. Select the SEQUENCE attribute from the Data Generator operator and map it to the target column.

   Warehouse Builder displays the properties of this attribute in the Property Inspector.
3. In the Expression field, click the Ellipsis button to open the Expression Builder and define an expression.
4. (Optional) Repeat steps 2 and 3 for the RECNUM attribute.

### Setting a Column to the Data File Record Number

To set an attribute to the number of the records that the record was loaded from, map from the RECNUM attribute. Records are counted sequentially from the beginning of the first data file, starting with record 1. RECNUM increments as each logical record is assembled. It increments for records that are discarded, skipped, rejected, or loaded. For example, if you use the option SKIP=10, the first record loaded has a RECNUM of 11.

### Setting a Column to the Current Date

A column mapped from SYSDATE1 gets the current system date, as defined by the SQL SYSDATE function.

The target column must be of type CHAR or DATE. If the column is of type CHAR, then the date is loaded in the format dd-mon-yy. If the system date is loaded into a DATE column, then you can access it in the time format and the date format. A new system date/time is used for each array of records inserted in a conventional path load and for each block of records loaded during a direct path load.

### Setting a Column to a Unique Sequence Number

Map the SEQUENCE attribute to the target column to generate sequence numbers for the column. The SEQUENCE keyword ensures a unique value for a column. SEQUENCE increments for each record that is loaded or rejected. It does not increment for records that are discarded or skipped.

The default sequence generated is SEQUENCE (COUNT). You can edit the sequence expression in the property expression, but you must provide the syntax.

The combination of column name and the SEQUENCE function is a complete column specification. Table 25–7 lists the options available for sequence values.

***Table 25–7 Sequence Value Options***

| Value | Description |
|---|---|
| COUNT | The sequence starts with the number of records already in the table plus the increment |
| integer | Specifies the beginning sequence number |
| MAX | The sequence starts with the current maximum value for the column plus the increment. |
| incr | The value that the sequence number is to increment after a record is loaded or rejected |

If records are rejected during loading, the sequence of inserts is preserved despite data errors. For example, if four rows are assigned sequence numbers 10, 12, 14, and 16 in a column, and the row with 12 is rejected, the valid rows with assigned numbers 10, 14, and 16, not 10, 12, 14 are inserted. When you correct the rejected data and reinsert it, you can manually set the columns to match the sequence.

## Dimension Operator

Use the Dimension operator to source data from or load data into dimensions and Slowly Changing Dimensions.

The Dimension operator contains one group for each level in the dimension. The groups use the same name as the dimension levels. The level attributes of each level are listed under the group that represents the level.

You cannot map a data flow to the surrogate identifier attribute or the parent surrogate identifier reference attribute of any dimension level. Warehouse Builder automatically populates these columns when it loads a dimension.

You can bind and synchronize a Dimension operator with a dimension stored in the workspace. To avoid errors in the generated code, ensure that the workspace dimension is deployed successfully before you deploy the mapping that contains the Dimension operator. To synchronize a Dimension operator with the workspace dimension, right-click the dimension on the Mapping Editor canvas and select **Synchronize**.

If you specify an Orphan Management Policy and create an error table for a dimension, when you add this dimension to a mapping, the Dimension operator contains a group called ERROR_<*dimension_name*>. This is an output group that contains attributes that are displayed in the dimension operator details, but not in the Dimension operator on the mapping canvas. To create data flows using these attributes, display these attributes on the canvas by selecting the ERROR_<*dimension_name*> group on the canvas and from the Graph menu, selecting **Select Display Set**, and then **All**.

**To use a Dimension operator in a mapping:**

1. Drag and drop a Dimension operator onto the Mapping Editor canvas.

   Warehouse Builder displays the Add Dimension dialog box.

2. Use the Add Dimension dialog box to select a dimension.

   Alternatively, you can combine Steps 1 and 2 into one single step. In the Mapping Editor, navigate to the Projects Navigator. Select the dimension and drag and drop it onto the Mapping Editor canvas.

3. Map the attributes from the Dimension operator to the target, or map attributes from the source to the Dimension operator.

## Dimension Operator Properties

Use the Property Inspector to set options that define additional details about loading or removing data from a dimension or Slowly Changing Dimension.

You can set properties at the following three levels: operator, group that represents each level in the dimension, and level attribute. The following sections describe the Dimension operator properties. The properties are categorized as follows: AW Properties, Dimension Properties, Error Table, History Logging Properties, and Orphan Management Policies.

**Target Load Order** Specifies the order in which multiple targets within the same mapping are loaded. Warehouse Builder determines a default order based on the foreign key relationships. Use this property to overrule the default order.

### AW Properties

**AW Name** Represents the name of the analytic workspace in which the dimension data is stored.

**Aw Staged Load** This property is applicable to MOLAP dimensions only. Select this option to stage the set-based load data into a temporary table before loading into the analytic workspace.

Each group in the Dimension operator represents a dimension level. You can set the following properties for each dimension level:

- **Extracting Type:** Represents the extraction operation to be performed when the dimension is used as a source. Select **Extract Current Only (Type 2 Only)** to extract current records only from a Type 2 SCD. This property is valid only for Type 2 SCDs. Select **Extract All** to extract all records from the dimension or SCD.
- **Default Expiration Time of Open Record:** This property is applicable to Type 2 SCDs only. It represents a date value that is used as the expiration time of a newly created open record. The default value is NULL.

> **Note:** If you set the Commit Control property to Manual, ensure that you set the Automatic Hints Enable property to false. Otherwise, your mapping may not execute correctly.

**Aw Truncate Before Load** This property is applicable to MOLAP dimensions only. It indicates whether all existing dimension data should be truncated before loading fresh

data. Set this property to YES to truncate any existing dimension data before you load fresh data.

#### Dimension Properties

**Loading Type** Represents the type of operation to be performed on the dimension. The options that you can select are as follows:

- **LOAD:** Select this value to load data into the dimension or Slowly Changing Dimension.
- **REMOVE:** Select this value to delete data from the dimension or Slowly Changing Dimension.

While you are loading or removing data, a lookup is performed to determine if the source record exists in the dimension. The matching is performed by the natural key identifier. If the record exists, then a REMOVE operation removes existing data. A LOAD operation updates existing data and then loads new data.

Note that when you remove a parent record, the child records will have references to a nonexistent parent.

**Type 2 Extract/Remove Current Only** This property is applicable only to Type 2 SCDs. Use this property to specify which records are to be extracted or removed. You can set the following values for this property:

- **YES:** When you are extracting data from the Type 2 SCD, only the current record that matches the business identifier in the source data is extracted. When you are removing data from a Type 2 SCD, only the current record that matches the business identifier in the source data is closed (expiration date is set either to SYSDATE or to the date defined in the Default Expiration Time of Open Record property).

  Note that in a Type 2 SCD that uses a snowflake implementation, you cannot remove a record if it has child records.
- **NO:** When you are extracting data from a Type 2 SCD, all the records, including historical records, that match the business identifier from the source data are extracted from the dimension.

  When you are removing data from the Type 2 SCD, all records, including historical records, that match the business identifier in the source data set are deleted.

#### Error Table

**DML Error Table Name** Represents the name of the table that stores DML errors associated with the dimension. To log DML errors, you must enable DML error logging for the dimension. For more information about DML error logging, see "Using DML Error Logging" on page 15-4.

**Error Table Name** Represents the name of the error table that stores logical errors caused by enforcing data profiling and orphan management. If you specify a value for the Error Table Name property of a dimension, then the Error Table Name property of the Dimension operator associated with this dimension displays the same name and you cannot edit the name. Else, specify the name if the error table.

**Truncate Error Table(s)** This property is applicable to error tables only and not to DML error tables. Set this property to **Yes** to truncate error tables very time they are used.

#### History Logging Properties

**Default Effective Time of Initial Record** This property is applicable to Type 2 SCDs only. It represents the default value assigned as the effective time for the initial load of a particular dimension record. The default value set for this property is `SYSDATE`.

**Default Effective Time of Open Record** This property is applicable to Type 2 SCDs only. It represents the default value set for the effective time of the open records, after the initial record. The default value of this property is `SYSDATE`. This value should not be modified.

**Default Expiration Time of Open Record** This property is applicable to Type 2 SCDs only. It represents a date value that is used as the expiration time of a newly created open record for all the levels in the dimension. The default value is NULL.

**Support Multiple History Loading** This property is applicable only to Type 2 SCDs. Select this option to load multiple rows for a particular business identifier during a single load operation. Then you select this option, the mapping is run in row-based non-bulk mode.

To load multiple records for a particular business identifier, ensure that the effective date of the Type 2 levels are loaded from a source or transformation operator.

Typically, this situation would arise when your dimension records change multiple times within during the period between two dimension updates. For example, you update your dimension only once per day, but there are multiple changes to a dimension record within that day.

**Support Out of Order History Loading** This property is applicable only to Type 2 SCDs. Setting this property to true enables you to load out-of-order changes to historical records in consecutive data loads.

You can also use this property in conjunction with Support Multiple History Loading. However, using this property has a performance overhead.

**Type 2 Gap** This property is applicable to Type 2 SCDs only. It represents the time interval between the expiration time of an old record and the effective time of the current record when a record is versioned.

When the value of a triggering attribute is updated, the current record is closed and a new record is created with the updated values. Because the closing of the old record and opening of the current record occur simultaneously, it is useful to have a time interval between the expiration time of the old record and the effective time of the open record, instead of using the same value for both.

**Type 2 Gap Units** This property is applicable for Type 2 SCDs only. It represents the unit of time used to measure the gap interval represented in the Type2 Gap property. The options are: Seconds, Minutes, Hours, Days, and Weeks. The default value is Seconds.

#### Orphan Management Policies

**Create Default Level Records** Indicates if default level records should be created for the dimension to which the Dimension operator is bound. Set this property to Yes to create default rows for the business identifier and surrogate identifier of the dimension.

The values used by the default record depend on the orphan management policy that you selected for the dimension to which the Dimension operator is bound. If you specified No Maintenance as the orphan management policy of the dimension, then use the Default Value property of the attributes in each level of Dimension operator to specify the values that the default record should use. If you set the orphan management policy of the dimension to Default Parent and specified the attribute values to be used for the default record, then these values are automatically displayed in the Default Value property of the attributes and these values are used for the default records. If you do not specify default values for the attributes in the dimension levels, then the default records are created using NULL values.

You can use this property to generate default records for time dimensions too. Time dimensions do not have an Orphan tab where you can use to set the orphan management policy. But, if you need to use a time dimension with a cube that has its orphan management policy set to a value other than No Maintenance, you can generate default records for the time dimension by setting the Default Value property of the level attributes and then setting the Create Default Level Records property of the time dimension to Yes.

**LOAD Policy for Invalid Keys** Represents the orphan management policy to be used to load records that contain an invalid parent record. The options are No Maintenance, Default Parent, and Reject Orphan.

**LOAD Policy for NULL Keys** Represents the orphan management policy to be used to load records that contain a NULL parent key reference. The options are No Maintenance, Default Parent, and Reject Orphan.

**Record Error Rows** Select Yes to store orphan records contained in the source data set that is used to load the dimension in the error table. The error table is represented by the Error Table Name property.

## Expand Object Operator

The Expand Object operator enables you to expand an object type and obtain the individual attributes that comprise the object type.

You can bind and synchronize an Expand Object operator with a workspace object type. To avoid generation errors in the mapping, ensure that you deploy the workspace object type before you deploy the mapping.

The Expand Object operator has one input group and one output group. The input group represents the object type that you want to expand in order to obtain its individual attributes. When you bind an Expand Object operator to a workspace object, the output group of the operator contains the individual attributes that comprise the object type.

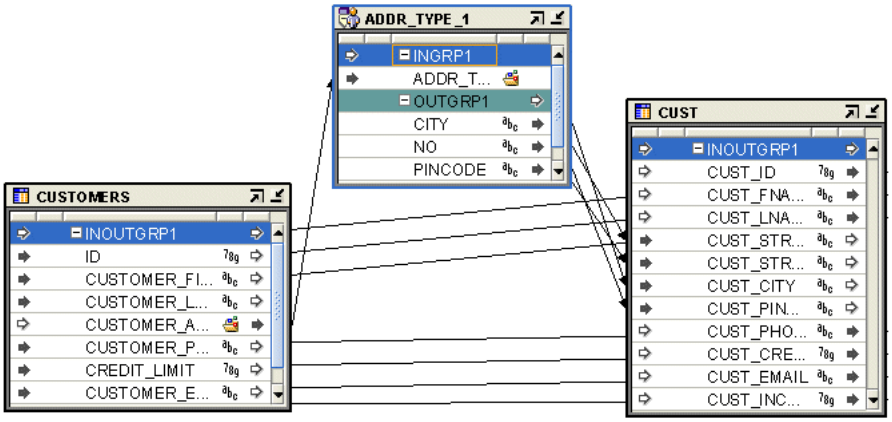
To successfully deploy a mapping that contains an Expand Object operator, ensure that the following conditions are satisfied.

- The schema that contains the source tables must be on the same instance as the warehouse schema.
- The warehouse schema is granted the SELECT privilege on the source tables.

- The warehouse schema is granted the EXECUTE privilege on all the object types and nested tables used in the Expand Object operator.

Figure 25–3 displays a mapping that uses an Expand Object operator. The source table CUSTOMERS contains a column CUSTOMER_ADDRESS of data type ADDR_TYPE, a SQL object type. But the target table CUST contains four different columns, of Oracle built-in data types, that store each component of the customer address. To obtain the individual attributes of the column CUSTOMER_ADDRESS, create an Expand Object operator that is bound to the object type ADDR_TYPE. You then map the CUSTOMER_ADDRESS column to the input group of an Expand Object operator. The output group of the Expand Object operator contains the individual attributes of the column CUSTOMER_ADDRESS. Map these output attributes to the target operator.

***Figure 25–3 Expand Operator in a Mapping***




**To define an Expand Object operator in a mapping:**

1. Drag and drop an Expand Object operator onto the Mapping Editor canvas.
2. Use the Add Expand Object dialog box to create or select an object. For more information about these options, see "Using the Add Operator Dialog Box to Add Operators" on page 5-15.
3. Map the source attribute that needs to be expanded to the input group of the Expand Object operator.

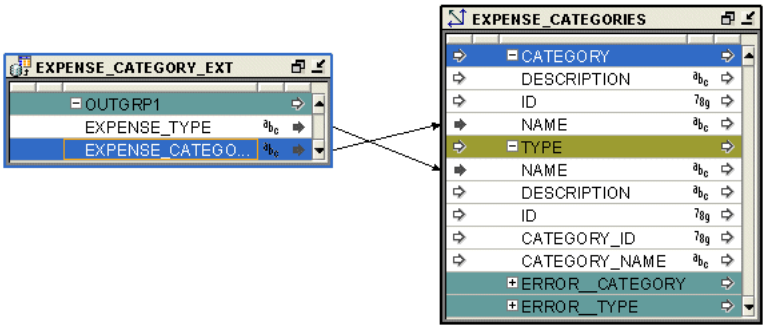
   Note that the signature of the input object type should be same as that of the Expand Object operator.
4. Map the output attributes of the Expand Object operator to the target attributes.

## External Table Operator

The External Table operator enables you to source data stored in external tables in the workspace. You can then load the external table data into another workspace object or perform transformations on the data. For example, you can source data stored in an external table, transform the data using mapping operators, and then load the data into a dimension or a cube.

Figure 25–4 displays a mapping that uses the External Table operator. The External Table operator EXPENSE_CATEGORY_EXT is bound to the external table of the same name in the workspace. The data stored in this external table is used to load the dimension EXPENSE_CATEGORIES.

*Figure 25–4 External Table Operator in a Mapping*




To create a mapping that contains an External Table operator:

1. Drag and drop an External Table operator onto the Mapping Editor canvas.

   Warehouse Builder displays the Add External Table dialog box.

2. Use the Add External Table dialog box to create or select an external table. For more information about these options, see "Using the Add Operator Dialog Box to Add Operators" on page 5-15.

3. Map the attributes from the output group of the External Table operator to the target operator or the intermediate transformation operator.

## Mapping Input Parameter Operator

You can introduce information external to Warehouse Builder as input into a mapping using a Mapping Input Parameter.

For example, you can use a Mapping Input Parameter operator to pass SYSDATE to a mapping that loads data to a staging area. Use the same Mapping Input Parameter operator to pass the timestamp to another mapping that loads the data to a target.

When you generate a mapping, a PL/SQL package is created. Mapping input parameters become part of the signature of the main procedure in the package.

The Mapping Input Parameter operator has a cardinality of one. It creates a single row set that can be combined with another row set as input to the next operator.

Each Mapping Input Parameter operator becomes an output attribute in the Mapping Input Parameter operator. These output attributes can then be used by connecting them to other operators within the Mapping Editor.

When you define the Mapping Input Parameter operator, you specify a data type and an optional default value.

**To define a Mapping Input Parameter operator in a mapping:**

1. Drag and drop a Mapping Input Parameter operator onto the Mapping Editor canvas.

2. Right-click the Mapping Input Parameter operator and select **Open Details**.

   The Mapping Input Parameter Editor is displayed.

3. Select the Output Attributes link on the left, to display the Output Attributes tab.

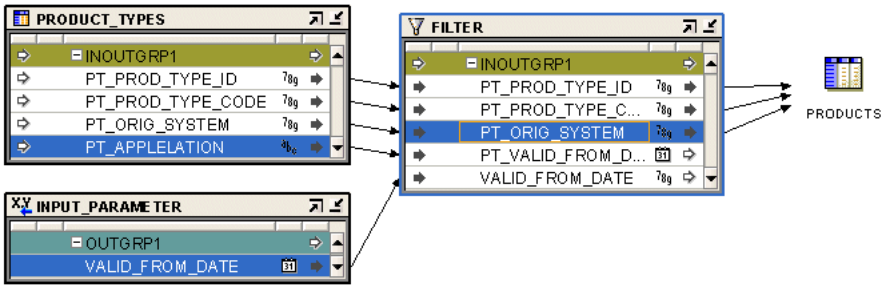
4. To add an output attribute, click a blank field in the Attribute column and provide a name for the output attribute. Also specify details such as data type, length, precision, scale, and seconds description for the attribute, as applicable.

You can rename the attributes and define the data type and other attribute properties.

5. Click **OK** to close the Mapping Input Parameter Editor.
6. Connect the output attribute of the Mapping Input Parameter operator to an attribute in the target operator.

   Figure 25–5 displays the mapping that uses a Mapping Input Parameter operator.

***Figure 25–5 Mapping Editor Showing A Mapping Input Parameter***




## Mapping Output Parameter Operator

Use a single Mapping Output Parameter operator to send values out of a PL/SQL mapping to applications external to Warehouse Builder.

A Mapping Output Parameter operator is not valid for a SQL*Loader mapping. When you generate a mapping, a PL/SQL package is created. Mapping Output Parameters become part of the signature of the main procedure in the package.

The Mapping Output Parameter operator has only one input group and no output groups. You can have only one Mapping Output Parameter operator in a mapping. Only attributes that are not associated with a row set can be mapped into a Mapping Output Parameter operator. For example, constant, input parameter, output from a premapping process, or output from a post process can all contain attributes that are not associated with a row set.

**To define a Mapping Output Parameter operator in a mapping:**

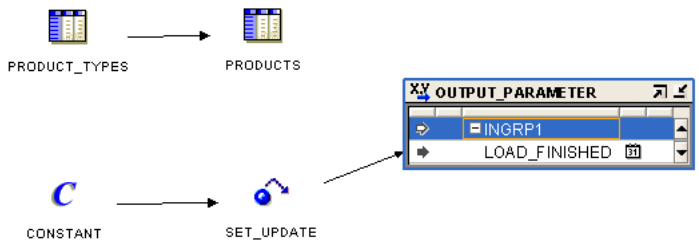
1. Drag and drop a Mapping Output Parameter operator onto the Mapping Editor canvas.
2. Right-click the Mapping Output Parameter operator and select **Open Details.**

   The Mapping Output Parameter Editor is displayed.
3. Click the Input Attributes tab link on the left to display the Input Attributes page.
4. To add an input attribute, click a blank field in the Attribute column and provide a name for the input attribute. Also specify details such as data type, length, precision, scale, and seconds description for the attribute, as applicable.

   You can rename the attributes and define the data type and other attribute properties.
5. Click **OK** to close the Mapping Output Parameter editor.
6. Connect the input attribute of the Mapping Output Parameter operator to an attribute in the target operator.

Figure 25–6 displays an example of a Mapping Output Parameter operator used in a mapping.

***Figure 25–6 Mapping Editor Showing An Output Parameter Operator***




## Materialized View Operator

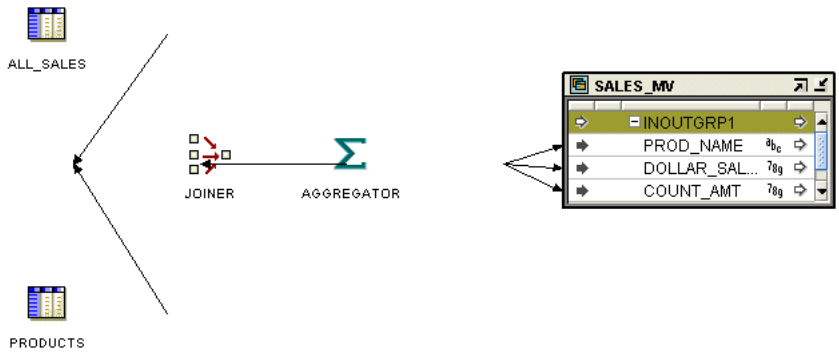
The Materialized View operator enables you to source data from or load data into a materialized view stored in the workspace.

For example, you can use the data stored in a materialized view to load a cube. The Materialized View operator has one Input/Output group called INOUTGRP1. You cannot add additional groups to this operator, but you can add attributes to the existing Input/Output group.

You can bind and synchronize a Materialized View operator to a workspace materialized view. The workspace materialized view must be deployed before the mapping that contains the Materialized View operator is generated to avoid errors in the generated code package.

Figure 25–7 displays a mapping that uses a Materialized View operator. The data from the two source tables `PRODUCTS` and `ALL_SALES` is joined using a Joiner operator. This data is then aggregated using an Aggregator operator. The aggregated data is used to load the materialized view `SALES_MV`.

***Figure 25–7 Mapping that Contains a Materialized View Operator***




**To create a mapping that contains a Materialized View operator:**

1. Drag and drop a Materialized View operator onto the Mapping Editor canvas.

   Warehouse Builder displays the Add Materialized View dialog box.

2. Use the Add Materialized View dialog box to create or select a materialized view. For more information about these options, see "Using the Add Operator Dialog Box to Add Operators" on page 5-15.
3. Map the attributes of the Materialized View operator.

   If you are using the Materialized View operator as a target, connect the source attributes to the Materialized View operator attributes. If you are using the materialized view as a source, then connect the Materialized View operator attributes to the target.

## Queue Operator

A Queue operator enables you to use advanced queues as sources or targets in mappings.

Some of the most critical tasks in creating and maintaining a data warehouse include refreshing existing data, and adding new data from the operational databases. Use the Queue operator to capture changes made to source objects and send those changes to a staging database or directly to a data warehouse or operational data store.

> **Note:** You cannot use a Queue operator as a source and target in the same mapping.

### Using a Queue Operator

You have the following options for using a Queue operator:

- **Define a new Queue operator:** Drag a Queue operator from the Palette onto the mapping. The Mapping Editor displays a wizard.
- **Edit an existing Queue operator:** Right-click the Queue operator and select **Open Details**.

For an example of using a Queue operator, see "LCR Cast Operator" on page 26-22.

Whether you are using the operator wizard or the Operator Editor, complete the following tasks:

- Selecting the Queue
- Selecting the Source Type for a Queue Operator
- Selecting the User-Defined or Primary Type for a Queue Operator
- Selecting the Source Object
- Specifying the Source Changes to Process

The tasks that you must perform depend on the payload type of the advanced queue to which the Queue operator is bound. For all payload types, except `SYS.ANYDATA`, you need to only select the queue to which the Queue operator should be bound. For a payload type of `SYS.ANYDATA`, you must complete all the tasks listed.

### Selecting the Queue

Use the Select Queue page of the Queue Operator Wizard or the Select page of the Queue Operator Editor to select the advanced queue to which the operator is bound.

The node tree on this page lists the advanced queues in the current project. Select the advanced queue to which your Queue operator should be bound.

### Selecting the Source Type for a Queue Operator

Use the Select Source Type page to specify if the queue will be used as a real-time queue or a batch queue. Also specify the type of messages that the queue will receive.

Select one of the following options to indicate the type of queue:

- **Real-Time Source**

  Select this option to indicate that the Queue operator represents a real-time source. Real-time queues enable you to populate source changes to the target objects instantly. All DML changes to the source objects associated with the Queue operator, as specified in "Selecting the Source Object" on page 25-25, are instantly added to the AQ.

  Mappings that contain real-time sources are called real-time mappings. You need to just deploy real-time mappings once. Subsequently, whenever source changes are added to the queue, Warehouse Builder automatically runs the mapping and publishes the changes to the target objects.

- **Batch Source**

  Select this option to indicate that the Queue operator represents a batch source. Batch mappings populate the target objects with changes from the source only when you explicitly execute the mapping.

  When you define a batch source, you do not need to provide any more details in the wizard or editor for a batch source. All the wizard or editor pages related to the other tasks for defining the Queue operator are disabled.

For real-time queues that use a `SYS.ANYDATA` payload, you must specify the format of the messages in the queue. Select one of the following options to specify the message format:

- **Oracle Capture Process Message Format**

  Specifies that the messages received are in the form of LCRs. Warehouse Builder takes care of capturing DML changes, formatting the changes into LCRs and adding the LCRs to the queue.

  For more information about LCRs, see *Oracle Streams Concepts and Administration*.

  When you choose this option, you must specify the source table and the DML operations that need to be captured as described in "Selecting the Source Object" on page 25-25 and "Specifying the Source Changes to Process" on page 25-25.

- **User-Defined Message Format**

  Specifies that the messages received by the queue are in a user-defined format. When you choose this option, specify the user-defined type that represents the message format using the Select User-Defined or Primary Type page as described in "Selecting the User-Defined or Primary Type for a Queue Operator" on page 25-24.

### Selecting the User-Defined or Primary Type for a Queue Operator

When your queue uses a user-defined message format, you must specify the user-defined type that represents the message format. Use the User-Defined or Primary Type page to select the user-defined type.

This page contains a node tree that you can to select the user-defined type. The Primary Data Types node lists the primary data types you can select. If your message format uses primary data types, then expand this node and select a primary data type. A separate node is displayed for each Oracle module that contains user-defined types.

Expand the required module node and select the user-defined type they represents the queue message format.

### Selecting the Source Object

Use the Select Source page to specify the source tables for which you want to capture data changes. This page is enabled only if your queue is a real-time source that uses the Oracle capture message format.

The Available Tables section lists the tables for which can capture data changes. Select the tables and use the arrows to move them to the Selected Tables section. You can choose multiple tables by holding down the Ctrl key and selecting the tables.

### Specifying the Source Changes to Process

Use the Source Changes to Process page to specify the DML changes that should be captured for the tables selected on the Select Source page. This page is enabled only if your queue is a real-time source that uses an Oracle Capture Message Format.

The Identify Changes to Process section lists the tables selected on the Select Objects page. For each tables, use the check boxes to the right of the table name to select the DML operations that should be captured. Select **Insert** for a table to capture any rows inserted in the table. To capture any modifications made to a table, select **Update** to the right of the table. To capture rows deleted from a table, select **Delete** to the right of the table name.

## Sequence Operator

A Sequence operator generates sequential numbers that increment for each row.

For example, you can use the Sequence operator to create surrogate keys while loading data into a dimension table. You can connect a Sequence to a target operator input or to the inputs of other types of operators. You can combine the sequence outputs with outputs from other operators.

Because sequence numbers are generated independently of tables, the same sequence can be used for multiple tables. Sequence numbers may not be consecutive, because the same sequence can be used by multiple sessions.

This operator contains an output group containing the following output attributes:

- **CURRVAL:** Generates from the current value
- **NEXTVAL:** Generates a row set of consecutively incremented numbers beginning with the next value

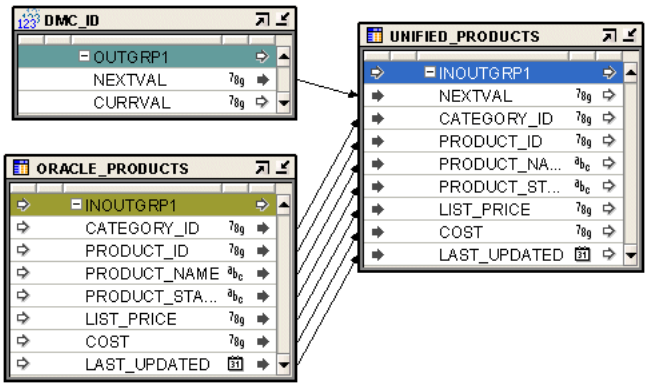
You can bind and synchronize Sequences to a workspace sequence in one of the modules. The workspace sequence must be generated and deployed before the mapping containing the Sequence is deployed to avoid errors in the generated code package. See "Using the Add Operator Dialog Box to Add Operators" on page 5-15 for more information.

Generate mappings with sequences using row-based mode. Sequences may be incremented even if rows are not selected. If you want a sequence to start from the last number, then do not run your SQL package in set-based or in set-based with failover operating modes. See "Runtime Parameters" on page 24-1 for more information about configuring mode settings.

Figure 25–8 shows a mapping that uses a Sequence operator to automatically generate the primary key of a table. The NEXTVAL attribute of the Sequence operator is mapped to an input attribute of the target table `UNIFIED_PRODUCTS`. The other input

attributes from the source table, ORACLE_PRODUCTS, are mapped directly to the target.

***Figure 25–8 Sequence Operator in a Mapping***




**To define a Sequence operator in a mapping:**

1. Drag and drop the Sequence operator onto the Mapping Editor canvas.

   Warehouse Builder displays the Add Sequence dialog box.

2. Use the Add Sequence dialog box to create or select a sequence. For more information about these options, see "Using the Add Operator Dialog Box to Add Operators" on page 5-15.

3. Connect the required output attribute from the Sequence operator to a target attribute.

## Table Operator

The Table operator enables you to source data from and load data into tables stored in the workspace.

You can bind and synchronize a Table operator to a workspace table. To avoid errors in the generated code package, the workspace table must be deployed before the mapping that contains the Table operator is generated.

Figure 25–8 displays a mapping that uses Table operators as both source and target. Figure 25–2 displays a mapping that uses the Table operator as a target.

**To define a Table operator in a mapping:**

1. Drag and drop a Table operator onto the Mapping Editor canvas.

   Warehouse Builder displays the Add Table dialog box.

2. Use the Add Table dialog box to create or select a table. For more information about these options, see "Using the Add Operator Dialog Box to Add Operators" on page 5-15.

3. Map the attributes of the Table operator.

   If you are using the table as a target, connect the source attributes to the Table operator attributes. If you are using the table as a source, then connect the Table operator attributes to the target.

### Merge Optimization for Table Operators

Beginning with the Oracle Warehouse Builder 10.2.0.3 release, you can enable the Merge Optimization property for Table operators. When set to True, this property optimizes the invocation or execution of expressions and transformations in the `MERGE` statement.

For example, consider a mapping in which the target table contains a column that is part of the update operation only and is mapped to a transformation. In previous releases, Warehouse Builder would execute the transformation for all rows, including rows that did not require transformation. Beginning in this release, if Merge Optimization is enabled, then Warehouse Builder calls the transformation only in the update part of the `MERGE` statement.

### Chunking for Table Operators

Chunking enables you to divide the source data in a mapping into chunks. The chunks are defined by a data partitioning algorithm and each chunk is then processed and loaded into the targets separately.

For more information about chunking, see "About Chunking for PL/SQL Mappings" on page 5-4.

To use source data chunking for a Table operator, select the Table operator on the mapping canvas. The Property Inspector displays the properties of the Table operator.

You can perform both serial or parallel chunking. To perform serial chunking, set the Chunking Enabled and Chunk Filter Condition options under the SCD Updates node. To perform parallel chunking, set the Chunking Expression option under the Chunking node.

#### Chunking Enabled

Select this option to enable serial data chunking for the source table represented by the Table operator.

You can enable data chunking for only certain source tables in a mapping. For example, if your mapping contains three Table operators and you enable data chunking for only one Table operator, the entire mapping functionality is executed multiple times, once for each data chunk. However, since the other two tables do not have chunking enabled, these sources will provide data rows only during the first iteration of the mapping.

#### Chunk Filter Condition

The Chunk Filter Condition enables you to specify the condition used to divide source data into multiple chunks while performing serial chunking. For each iteration, the Chunk Filter Condition filters the source data from the source.

Warehouse Builder provides a predefined mapping constant, get_chunk_iterator, that must be used in all chunk filter conditions. This is an iteration count that starts at 1 and is incremented for each map execution in the chunk processing. You can set the condition to use a value from the data source.

For example, your filter condition can be `get_chunk_iterator = INOUTGRP1.CHUNK_GRP_NUM`, where `CHUNK_GRP_NUM` is an attribute in the source table.

#### Chunking Expression

Use Chunking Expression to set the condition used to divide source data into multiple chunks for parallel chunking.

## Creating Temporary Tables While Performing ETL

Warehouse Builder enables you to use temporary tables while extracting and loading data. Temporary tables are useful when you must extract data from remote sources into multiple targets.

Temporary staging tables are typically used in the dimension loading logic that is automatically generated by the Dimension operator submapping expansion. This prevents problems that would be caused by doing lookups on the target table.

The following properties enable you to create temporary tables while performing ETL.

### Is Temp Stage Table

When you set the Is Temp Stage Table property to True, any existing bindings for the Table operator are ignored. A temporary staging table is deployed with the mapping and all loading and extracting operations from this Table operator are performed from the staging table.

The name of the deployed table in the database is based on the operator name, with a unique identifier appended to the name to prevent name conflicts. The table is automatically dropped when the map is dropped or redeployed. Before each execution of the mapping, the table is automatically truncated.

When you set this property to its default value of False, it has no effect.

### Extra DDL Clauses

Use this property to add additional clauses to the DDL statement that is used to create the table. For example, use the following `TABLESPACE` clause to allocate storage for the temporary table in the `MY_TBLSPC` tablespace, instead of in the default tablespace: `TABLESPACE MY_TBLSPC`.

If you do not provide a value for the Extra DDL Clauses property, this property has no effect on the table creation.

### Temp Stage Table ID

Use the Temp Stage Table ID property to specify the internal identifier used for the temporary staging table by the code generator. If any other temporary staging table in the mapping has the same value for the Temp Stage Table ID property, then the same deployed temporary staging table will be bound to both operators. This enables multiple usages of the same temporary staging table in the same mapping.

## DML Error Logging

You can perform DML error logging on target tables. Use the property **DML Error Table Name** of the Table operator to specify the name of the error table that stores errors encountered while performing DML operations on that table. The error table is created when you execute the mapping containing the Table operator.

For more information about DML error logging, see "Using DML Error Logging" on page 15-4.

## Data Rules and Loading Tables

In addition to logging DML errors, you can also store logical errors such as data profiling and orphan management errors. Use the **Error Table Name** property for the Table operator to specify the name of the table that stores logical errors for the repository table associated with the Table operator. If you have specified a name for

the Error Table Name property of the table, then the Error Table Name property of the Table operator associated with this table automatically uses the same name.

## Varray Iterator Operator

When you have an object of type nested table or Varray, you can use the Varray Iterator operator to iterate through the values in the table type.

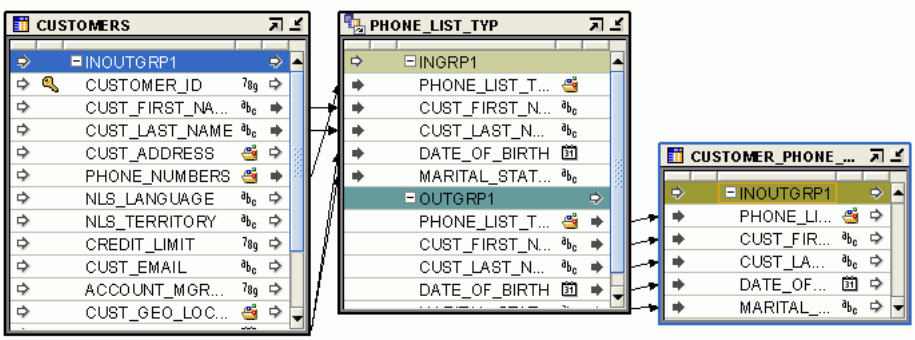
This operator accepts a table type attribute as the source, and generates a value that is of the base element type defined in the nested table or Varray type. If the operator is bound, then reconciliation operations are performed on the operator. Reconciliation operations are not supported for unbound Varray Iterator operators.

You can create the Varray Iterator operator as either a bound operator or an unbound operator. You cannot synchronize or validate an unbound Varray Iterator operator. It has only one input group and one output group. You can have an input group with attributes of other data types. However, there must be at least one table type attribute.

The attributes of the output group are a copy of the attributes of the input group. The only difference is that instead of the table type to which the operator is bound (which is one of the input group attributes), the output group will have an attribute that is the same as the base element type of the input group. The input group is editable. The output group is not editable.

Figure 25–9 displays a mapping that contains a Varray Iterator operator.

***Figure 25–9 Varray Iterator Operator in a Mapping***




**To define a Varray Iterator operator in a mapping:**

1. Drag and drop a Varray Iterator operator onto the Mapping Editor canvas.

   Warehouse Builder displays the Add Varray Iterator dialog box.

2. From the **Add Varray Iterator** dialog box, select either an unbound operator or a bound operator.

   - If you select the unbound operator, then a Varray Iterator operator with no attributes is created. You must create these attributes manually.
   - If you select the bound operator, then you must select one of the available nested table or Varray types shown in the tree. The output attribute is the same as the base element.

3. Click **OK**.

4. Map the attributes of the Varray Iterator operator.

For an unbound operator, right-click the unbound Varray Iterator operator on the Mapping Editor canvas and then select **Open Details**. This opens the Varray Iterator

editor dialog box. You can add attributes to the input group by using the **Add** button. You can only change the data type of the attributes in the output group.

## View Operator

The View operator enables you to source data from or load data into a view stored in the workspace.

You can bind and synchronize a View operator to a workspace view. The workspace view must be deployed before the mapping that contains the View operator is generated to avoid errors in the generated code package.

**To define a View operator in a mapping:**

1. Drag and drop a View operator onto the Mapping Editor canvas.

   Warehouse Builder displays the Add View dialog box.

2. Use the Add View dialog box to create or select a view. For more information about these options, see "Using the Add Operator Dialog Box to Add Operators" on page 5-15.

3. Map the attributes of the View operator.

   If you are using the view as a target, connect the source attributes to the View operator attributes. If you are using the view as a source, then connect the View operator attributes to the target.

### Using the View Operator for Inline Views

You can use the View operator to create inline views in a mapping. For inline views, you must set the following operator properties.

**Inlined:** Select this property to indicate that the operator represents an inline view.

**View Query:** Use this property to specify the query text for the inline view. The query text must have column aliases that correspond to the operator attribute names.

# Using Remote and non-Oracle Source and Target Operators

You can bind a target operator in a mapping to an object in a remote Oracle Database location or a non-Oracle Database location such as SQL Server or DB2 through a Gateway location. Such operators are referred to as Gateway targets. Use database links to access these targets. The database links are created using the locations. SAP targets are not supported, because it is not possible to generate a database link to access SAP tables remotely from an Oracle database.

There are certain restrictions on using remote or Gateway targets in a mapping, as described in the following sections:

- Limitations of Using Non-Oracle or Remote Targets
- Warehouse Builder Workarounds for Non-Oracle and Remote Targets

## Limitations of Using Non-Oracle or Remote Targets

The following limitations apply when you use a remote or Gateway target in a mapping:

- You cannot set the Loading Type property of the target operator to TRUNCATE/INSERT.

This results in a validation error when you validate the mapping.

- For Gateway targets, setting the Loading Type property of the target operator to INSERT/UPDATE produces the same result as setting the loading type to INSERT.
- The RETURNING clause is not supported in a DML statement.

  The RETURNING clause enables you to obtain the ROWIDs of the rows that are loaded into the target using row-based mode. These ROWIDs are recorded by the runtime auditing system. But in a remote or Gateway target, the RETURNING clause is not generated, and nulls are passed to the runtime auditing system for the ROWID field.
- In set-based mode, you cannot load data from an Oracle database into a remote or Gateway target. All other modes, including set-based failover, are supported.

  When you set the Operating Mode property of the target operator to set-based, a runtime error occurs.
- Row-based bulk processing is not supported.

### Warehouse Builder Workarounds for Non-Oracle and Remote Targets

When you use a remote or Gateway target in a mapping, default workarounds are used for certain restricted activities. These workarounds are listed for your information only. You need not explicitly do anything to enable these workarounds.

The default workarounds used for a remote or a Gateway target are as follows:

- When you set the loading type of a target to INSERT/UPDATE or UPDATE/INSERT in Oracle 9*i* database and to UPDATE in Oracle Database 11*g*, a MERGE statement is generated to implement this mapping in set-based mode. But a MERGE statement cannot be run against remote or Gateway targets. Thus, when you use a remote or Gateway target in a mapping, code is generated without a MERGE statement. The generated code is the same as that generated when the PL/SQL generation mode is set to Oracle8*i*.
- For set-based DML statements that reference a database sequence that loads into a remote or Gateway target, the GLOBAL_NAMES parameter must be set to TRUE. When code is generated for a mapping, this parameter is set to TRUE if the mapping contains a remote or Gateway target.
- For a multitable insert to a remote or Gateway target, an INSERT statement is generated per table instead of a multitable insert statement.
- While performing bulk inserts on a remote or non-Oracle Database, bulk processing code is not generated. Instead, code that processes one row at a time is generated. This means that the Generate Bulk property of the operator is ignored.

---

**Note:** The loading types used for remote or Gateway targets are the same as the ones used for other Oracle target operators. For more information about the loading type property, see "Loading Types for Oracle Target Operators" on page 25-3.

---

## Using Flat File Source and Target Operators

The Flat File operator enables you to use flat files as sources or targets in a mapping. The following section describes the usage of Flat File operators.

## Flat File Operator

Use a Flat File operator to extract data from or load data into flat file.

A Flat File operator can be used either as a source or target in a mapping. However, the two are mutually exclusive within the same mapping. There are differences in code generation languages for flat file sources and targets. Subsequently, mappings can contain a mix of flat files, relational objects, and transformations, but with the restrictions discussed later in this section.

You have the following options for Flat File operators:

- Using previously imported flat files
- Importing and binding new flat files into your mapping
- Defining new flat file sources or targets in mappings

**To use a Flat File operator in a mapping:**

1. Drag and drop a Flat File operator onto the Mapping Editor canvas.
2. Use the Add Flat File dialog box to create or select an object. For more information about these options, see "Using the Add Operator Dialog Box to Add Operators" on page 5-15.
3. Map the attributes from the Flat File operator to the target, or map attributes from the source to the Flat File operator.

For examples of using flat files as sources and targets in a mapping, see Chapter 7, "Using SQL*Loader, SAP, and Code Template Mappings".

### Flat File Source Operators

You can introduce data from a flat file into a mapping using either a Flat File operator or an External Table operator. If you are loading large volumes of data, then loading from a flat file enables you to use the DIRECT PATH SQL*Loader option, which results in better performance.

If you are not loading large volumes of data, you can benefit from many of the relational transformations available in the external table feature.

> **See Also:** *Oracle Warehouse Builder Sources and Targets Guide* for a comparison of external tables and flat files

As a source, the Flat File operator acts as the row set generator that reads from a flat file using the SQL*Loader utility. Do not use a Flat File source operator to map to a flat file target or to an external table. When you design a mapping with a Flat File source operator, you can use the following operators:

- Filter Operator
- Constant Operator
- Data Generator Operator
- Sequence Operator
- Expression Operator
- Transformation Operator
- Other relational target objects, excluding the External Table operator.

> **Note:** If you use the Sequence, Expression, or Transformation operators, you cannot use the SQL*Loader Direct Load setting as a configuration parameter.

When you use a flat file as a source in a mapping, remember to create a directory from the target location to the flat file location for the mapping to deploy successfully.

### Flat File Target Operators

A mapping with a flat file target generates a PL/SQL package that loads data into a flat file instead of loading data into rows in a table.

> **Note:** A mapping can contain a maximum of 50 Flat File target operators at one time.

You can use an existing flat file with either a single record type or multiple record types. If you use a multiple-record-type flat file as a target, then you can only map to one of the record types. If you want to load all of the record types in the flat file from the same source, then you can drop the same flat file into the mapping as a target again and map to a different record type. For an example of this, see "Using Direct Path Loading to Ensure Referential Integrity in SQL*Loader Mappings" on page 10-18. Alternatively, create a separate mapping for each record type that you want to load.

For more information about creating a new flat file target, see *Oracle Warehouse Builder Sources and Targets Guide*.

### Setting Properties for Flat File Source and Target Operators

You can set properties for a Flat File operator as either a source or target. You can set Loading Types for Flat Files and the Field Names in the First Row setting. All other settings are read-only and depend upon how you imported the flat file.

### Loading Types for Flat Files

Select a loading type from the list:

- **Insert:** Creates a new target file. If there is an existing target file, then the newly created file replaces the previous file.
- **Update:** Creates a new target file if one does not already exist. If there is an existing target file, then that file is appended.
- **None:** No operation is performed on the data in the target file. This setting is useful for testing purposes. All transformations and extractions are run without affecting the target.

### Field Names in the First Row

Set this property to **True** if you want to write the field names in the first row of the operator or **False** if you do not.

# 26

# Data Flow Operators

The Mapping Editor provides a set of pre-built mapping operators. These operators enable you to define common transformations that specify how data moves from the source to the target.

This chapter provides details on how to use operators in a mapping to transform data. Some operators have wizards that assist you in designing the mapping. And some operators allow you to start the Expression Builder as an aide to writing SQL expressions.

This chapter contains the following topics:


- List of Data Flow Operators on page 26-1
- About Operator Wizards on page 26-2
- About the Expression Builder on page 26-3


## List of Data Flow Operators

The list of data flow operators is as follows:


- Aggregator Operator on page 26-5
- Anydata Cast Operator on page 26-9
- Deduplicator Operator on page 26-10
- Expression Operator on page 26-11
- Filter Operator on page 26-12
- Joiner Operator on page 26-13
- LCR Cast Operator on page 26-22
- LCR Splitter Operator on page 26-23
- Lookup Operator on page 26-23
- Pivot Operator on page 26-28
- Post-Mapping Process Operator on page 26-34
- Pre-Mapping Process Operator on page 26-35
- Set Operation Operator on page 26-36
- Sorter Operator on page 26-37
- Splitter Operator on page 26-39
- Subquery Filter Operator on page 26-41

- Table Function Operator on page 26-43
- Transformation Operator on page 26-46
- Unpivot Operator on page 26-47


## About Operator Wizards

For operators that require you to make numerous design decisions, wizards guide you in defining the operator. Each wizard begins with a welcome page that provides an overview of the steps you must perform. And each wizard concludes with a summary page listing your selections. Use **Next** and **Back** to navigate through the wizard. To close an operator wizard, click **Finish** on any of the wizard pages.

The following operators have wizards to assist you:

- Lookup Operator
- Match Merge operator, see "Using the Match Merge Operator to Eliminate Duplicate Source Records"
- Name and Address operator, see "Using the Name and Address Operator to Cleanse and Correct Name and Address Data"
- Queue Operator
- Pivot Operator
- Unpivot Operator

Once you become proficient with defining an operator, you may prefer to disable the wizard and use the operator editor instead. To start the operator editor, right-click the operator on the Mapping Editor and select **Open Details**. The operator editor displays the same content as the wizard except in a tab format rather than wizard pages.

Whether you are using an operator wizard or the operator editor, you must complete the following pages for each operator:

- Operator Wizard General Page
- Operator Wizard Groups Page
- Operator Wizard Input and Output Pages
- Operator Wizard Input Connections

### Operator Wizard General Page

Use the General page to specify a name and optional description for the operator. By default, the wizard assigns the operator type as the name. For example, the default name for a new pivot operator is "Pivot".

### Operator Wizard Groups Page

Edit group information on the Groups tab.

Each group has a name, direction, and optional description. You can rename groups for most operators but cannot change group direction for any of the operators. A group can have one of these directions: Input, Output, Input/Output.

Depending on the operator, you can add and remove groups from the Groups tab. For example, you add input groups to Joiners and output groups to Splitters.

### Operator Wizard Input and Output Pages

The operator editor displays a tab for each type of group displayed on the Groups tab. Each of these tabs displays the attribute name, data type, length, precision, scale and optional description.

Depending on the operator, you may be able to add, remove, and edit attributes. The Mapping Editor grays out properties that you cannot edit. For example, if the data type is `NUMBER`, you can edit the precision and scale but not the length.

### Operator Wizard Input Connections

Use the Input Connections page to copy and map attributes into the operator. The attributes you select become mapped members in the input group. The Available Attributes panel displays a list of all the operators in the mapping.

**To complete the Input Connections page for an operator:**

1. Select complete groups or individual attributes from the Available Attributes panel.

   To search for a specific attribute or group by name, type the text in **Search for** and select **Go.** To find the next match, select **Go** again.

   Hold the **Shift** key down to select multiple groups or attributes. If you want to select attributes from different groups, then you must first combine the groups with a Joiner or Set operator.

2. Use the right arrow button between the two panels to move your selections to the Mapped Attributes panel.

   You can use the left arrow to remove groups or attributes from the input connections list. Warehouse Builder removes the selection from the input group and removes the data flow connection between the source operator and the current operator.

## About the Expression Builder

Some of the data flow operators require that you create expressions. An expression is a statement or clause that transforms data or specifies a restriction. These expressions are portions of SQL that are used inline as part of a SQL statement. Each expression belongs to a type that is determined by the role of the data flow operator. You can create expressions using Expression Builder, or by typing them into the expression field located in the Property Inspector of the operator or operator attributes.

### Opening the Expression Builder

You can open the Expression Builder from the Property Inspector of the operator for operators such as filters, joiners, and aggregators. For operators such as expressions, data generators, splitters, and constants, you can open the Expression Builder from the Property Inspector of the operator attribute.

**To open the Expression Builder:**

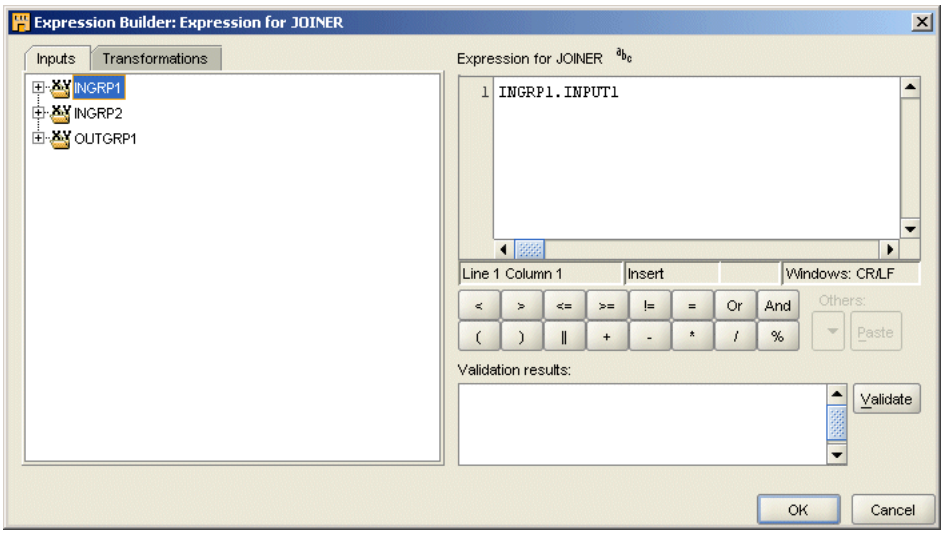
1. On the Mapping Editor, select the operator or the attribute for which you want to open the Expression Builder.

   The Property Inspector displays the properties of the selected operator or operator attribute.

2. In the Property Inspector, click the Ellipsis button in the property that you want to set using the Expression Builder.

   The Expression Builder displays as shown in Figure 26–1.

***Figure 26–1 Expression Builder Interface***




3. Create an expression by:
   - Typing text into the Expression field on the right of the Expression Builder.
   - Dragging items from the Inputs and Transformations tabs on the left panel and dropping them into the **Expression** field on the right.
   - Double clicking on items from the Inputs and Transformations tabs on the left panel.
   - Clicking arithmetic operator buttons available under the **Expression** field.
4. Click **Validate.**

   This verifies the accuracy of the Expression syntax.
5. Click **OK** to save the expression and close the Expression Builder.

## The Expression Builder User Interface

The Expression Builder contains the following parts:

- In the left panel, the navigation tree displays two tabs:
  - **Inputs Tab:** A list of input parameters.
  - **Transformations Tab:** A list of predefined functions and procedures located in the public Oracle Predefined library, the public Oracle Custom library, and a private Oracle library.
- **Expression Field:** At the top of the right panel is the **Expression** field. Use this field to type and edit expressions.
- **Arithmetic Operator Buttons:** Below the **Expression** field are buttons for arithmetic operators. Use these buttons to build an expression without typing. The arithmetic operators available vary by the type of data flow operator that is active.

- **Others:** A list of available SQL clauses that are appropriate for the active expression type.

  Beginning in Oracle 9*i*, the `CASE` function is recommended over the `DECODE` function because the `CASE` function generates both SQL and PL/SQL while `DECODE` is limited to SQL. If you use the `DECODE` function in an expression, then it is promoted to `CASE` where appropriate during code generation. This enables you to deploy the `DECODE` functionality in all operating modes (such as setbased or rowbased) and transparently across Oracle Database releases (8.1, 9.0 and higher).

  For example, the function

  ```
  DECODE (T1.A, 1, 'ABC', 2, 'DEF', 3, 'GHI', 'JKL')
  ```

  is converted to the following:

  ```
  CASE T1.A WHEN 1 THEN 'ABC'
  WHEN 2 THEN 'DEF'
  WHEN 3 THEN 'GHI'
  ELSE 'JKL'
  ```

- **Validate Button:** Use this button to validate the current expression in the Expression Builder. Validation ensures that all mapping objects referred to by the expression have associated workspace objects. The expressions you create with the Expression Builder are limited to the operator inputs and to any transformations available in a project. This limitation protects the expression from becoming invalid because of changes external to the operator. If the deployment database is different from the design workspace, then it may not accept the expression. If this happens, then the expression may be valid but incorrect against the database. In this case, expression errors can only be found at deployment time.
- **Validation Results Field:** At the bottom of the right panel is the **Validation Results** field. After you select the **Validate** button to the right of this field, this field displays the validation results.

## Aggregator Operator

The Aggregator operator calculates data aggregations, such as summations and averages, on the input data. It provides an output row set that contains the aggregated data.

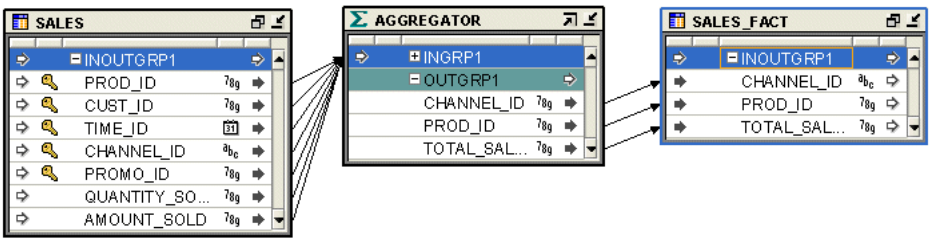
The Aggregator operator has one input group and one output group. For the output group, define a GROUP BY clause that specifies the attributes over which the aggregation is performed. You can optionally specify a HAVING clause that restricts the aggregated data. Each attribute in the output group has the same cardinality. The number of rows in the output row set is less than or equal to the number of input rows.

You can use a single Aggregator operator to perform multiple aggregations. Although you can specify a different aggregation function for each attribute in the output group of an Aggregator, each Aggregator supports only one GROUP BY and one HAVING clause.

Figure 26–2 shows a mapping that uses the Aggregator operator to aggregate the total sales over channels and products. Use the Expression property of the output attribute to specify that the aggregate function to be applied to the attribute `TOTAL_SALES` is SUM. Use the Group By property of the Aggregator operator to specify that the sales

are aggregated over channel ID and product ID. The output of the Aggregator operator is mapped to the target table `SALES_FACT`.

***Figure 26–2 Aggregator Operator in a Mapping***




**To define an Aggregator operator in a mapping:**

1. Drag and drop an **Aggregator** operator onto the Mapping Editor canvas.
2. On the canvas, connect source attributes to the input group of the Aggregator operator.
3. Right-click the Aggregator operator and select **Open Details**.

   Warehouse Builder displays the Aggregator Editor.
4. On the Output Attributes tab, create the output attributes that store the aggregated data.

   To create an output attribute, click the empty cell under the Attribute column and enter the attribute name. The default data type assigned to the attribute is `NUMBER`. You can change the data type and other parameters related to the data type such as length, precision, and so on.

   If the output attribute refers to an input attribute (from the Input group), the Group By Clause is automatically set.

   In the example displayed in Figure 26–2, you add an output attribute and rename it to TOTAL_SALES.
5. Define an expression for each output attribute. You can directly enter the expression in the Expression column associated with the attribute. Or, click the Ellipsis button to the right of the Expression field to display the Expression Builder. For detailed instructions on using the Expression Builder, see "Aggregate Function Expression" on page 26-8.

   In the example displayed in Figure 26–2, you define the expression as SUM(amount_sold).
6. Click **OK** to close the Aggregator Editor.
7. Define a Group By clause and an optional Having clause for the operator. For detailed instructions, see "Group By Clause" on page 26-6 and "Having Clause" on page 26-7.
8. Map the attributes in the output group of the Aggregator operator to the input group of the target.

## Group By Clause

The Group By clause defines how to group the incoming row set to return a single summary row for each group. An ordered list of attributes in the input group specifies how this grouping is performed. The default GROUP BY clause is NONE.

**To define the Group By Clause:**

1. Select the Aggregator operator on the Mapping Editor canvas.

   The Property Inspector displays the properties of the Aggregator operator.

2. Click the Ellipsis button to the right of the Group By Clause property.

   The Expression Builder is displayed.

3. Move the attributes that you want to use to group source data from the Inputs tab to the Group By Clause for Aggregator section. When you select more than one attribute, separate attributes using a comma.

4. Click **OK.**

## Having Clause

The Having clause is a boolean condition that restricts the groups of rows returned in the output group to those groups for which this condition is true. If this clause is not specified, then all summary rows for all groups are returned in the output group.

**To define the Having Clause:**

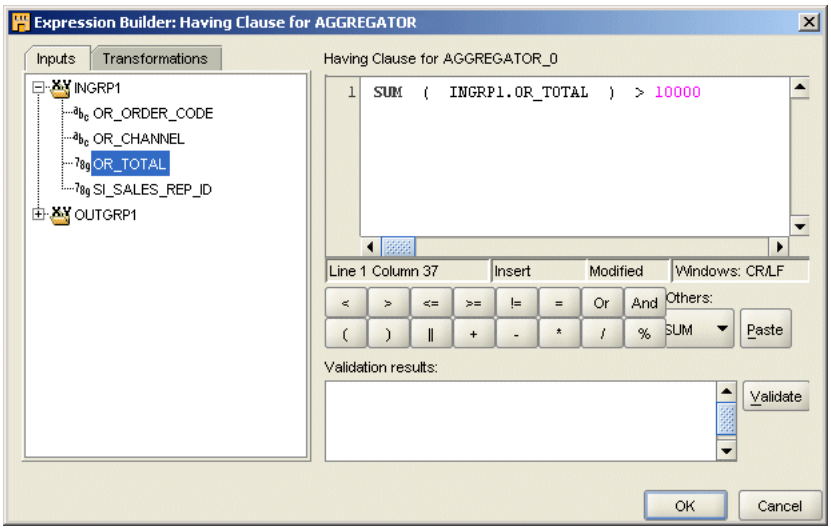
1. Select the Aggregator operator on the mapping canvas.

   The Property Inspector displays the properties of the Aggregator operator.

2. Click the Ellipsis button to the right of the Having Clause property.

   The Expression Builder dialog box for the Having Clause displays as shown in Figure 26–3.

***Figure 26–3 Having Clause Dialog Box***




3. Create an expression for the Having Clause of the Aggregator operator.

   For example, Figure 26–3 shows a sample Having Clause expression.

4. Click **OK** to close the Expression Builder.

5. Map the attributes you edited from the output group of the Aggregator operator to the attributes in the target.

## Aggregate Function Expression

The Expression property of an attribute defines the aggregation functions to be performed on the attribute. For each ungrouped output attribute, select whether the aggregation expression should be a DISTINCT or ALL result. ALL is the default setting. For example,

- ALL: `Select AVG(ALL sal) from emp;`
- DISTINCT: `Select AVG(DISTINCT sal) from emp;`

A DISTINCT result removes all duplicate rows before the average is calculated.

An ALL result returns an average value on all rows.

If no aggregation function is necessary, specify NONE for the function. Specifying NONE on the attribute aggregation automatically adds the attribute to the resulting GROUP BY function.

**To define expressions for output attributes:**

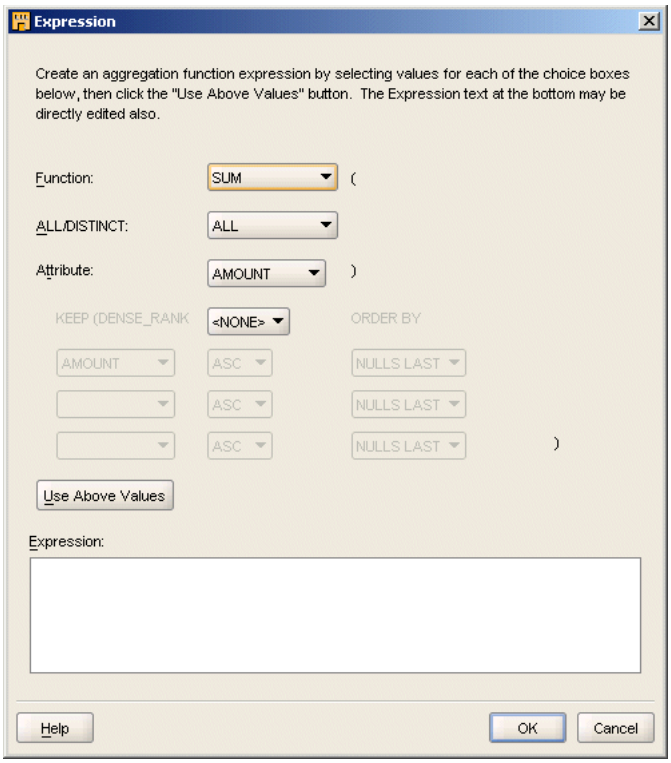
1. In the Aggregator operator on the mapping canvas, select the output attribute for which you want to define an aggregate function.

   The Property Inspector displays the properties of the selected output attribute.

2. Click the Ellipsis button to the right of the **Expression** property.

   The Expression dialog box displays as shown in Figure 26–4.

***Figure 26–4 Expression Dialog Box***




3. Select an aggregate function from the **Function** list.

   The aggregate functions you can select are as follows: AVG, COUNT, GROUP_ID, GROUPING, GROUPING_ID, MAX, MEDIAN, MIN, None, STDDEV, STDDEV_

POP, STDDEV_SAMP, SUM, VAR_POP, VAR_SAMP, VARIANCE, and WB_RT_CONCAT.

In the example displayed in Figure 26–2, you select SUM as the aggregate function.

4. Select either ALL or DISTINCT as the aggregation expression.
5. Select the attribute that should be aggregated from the **Attribute** list.

   In the example displayed in Figure 26–2, you select the attribute `amount_sold` from the list.
6. Click **Use Above Values** to display the aggregate expression in the Expression field.
7. Click **OK.**

## Anydata Cast Operator

Anydata Cast operator enables you to convert an object of type Sys.AnyData to either a primary type or to a user-defined type. The Anydata Cast operator accepts an Anydata attribute as a source and transforms the object to the desired type.

The Anydata Cast operator is used with user-defined data types and primitive data types. This operator acts as a filter. The number of attributes in the output group is $n$+1 where $n$ is the number of attributes in the input group. This operator has one input group and one output group. The input group is editable. The output group is not editable. In an output group, you can only rename the attributes and change the data type of only the cast target. You cannot change the data type of any other output group attribute.

You can connect attributes to the input group. Each output group gets a copy of the input group attributes, including the Anydata attributes. You must choose an Anydata attribute of the input group as the source of the Cast operation.

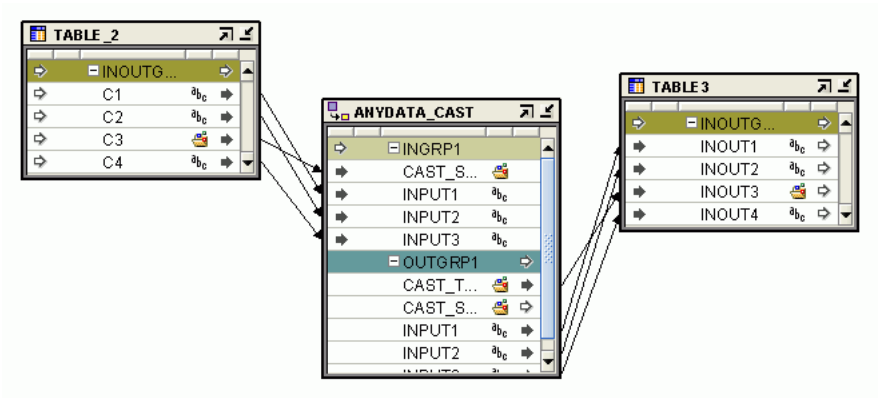
If you change the data type to which you are going to cast the Anydata attribute, then you must:

1. Edit the output group attribute that is the target of the Cast operation
2. Change the data type of the attribute.

Because the Anydata Cast operator is unbound, it will not support any synchronization operations.

Figure 26–5 displays a mapping that uses an Anydata Cast operator.

*Figure 26–5 Anydata Cast in a Mapping*




**To define a Anydata Cast operator in a mapping:**

1. Drop an Anydata Cast operator onto the Mapping Editor canvas.

   The AnyData Cast dialog box is displayed. The tree inside the dialog box has one parent node that will open to display the primary data types (other than Anydata). Each of the other parent nodes will correspond to the modules.

2. Select the target type for casting and click **Finish**.

3. Right-click the ANYDATA CAST operator and select **Open Details**.

   Warehouse Builder displays the ANYDATA_CAST Editor.

4. On the Input Attributes tab, click **Add** and specify the attribute name, data type, and other properties.

5. Click **OK** to close the operator editor.

6. Map the attributes of the output group of the Anydata Cast operator to the target.

# Deduplicator Operator

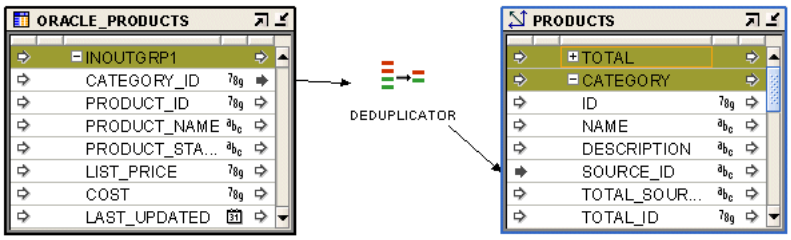
The Deduplicator operator enables you to remove duplicate data in a source by placing a DISTINCT clause in the select code represented by the mapping.

For example, when you load data from a source table into a dimension, the higher levels within a dimension may be duplicated in the source.

All attributes from a source rowset must pass through the Deduplicator operator. You cannot map part of the output from a source rowset and part of the output from the Deduplicator operator to the same target table.

Figure 26–6 displays a mapping that uses the Deduplicator operator to remove duplicate values in the source while loading data into the `PRODUCTS` dimension. The source table contains duplicate values for category ID because more than one products may belong to the same category. The Deduplicator operator removes these duplicates and loads distinct values of category ID into the `PRODUCTS` dimension.

*Figure 26–6 Deduplicator in a Mapping*




**To remove duplicates:**

1. Drop the Deduplicator operator onto the Mapping Editor canvas.
2. Connect the attributes from the source operator to the input/output group of the Deduplicator operator.
3. Connect the attributes from the Deduplicator operator group to the attributes of the target operator.

# Expression Operator

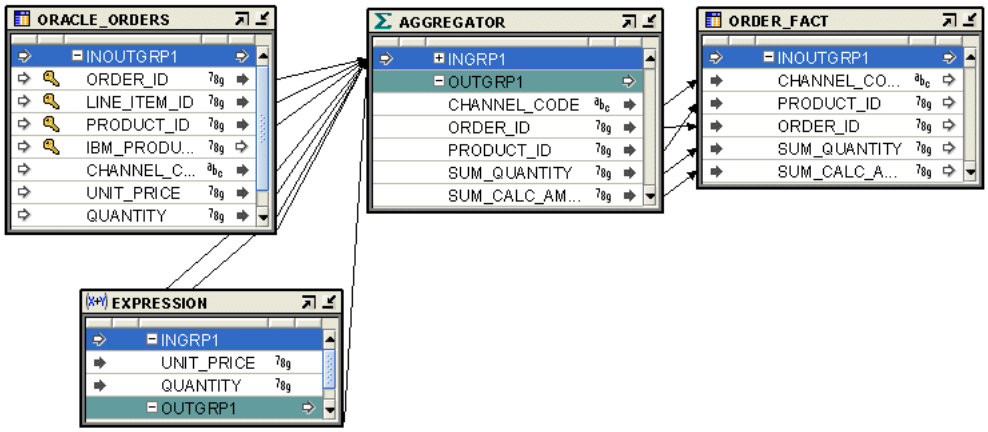
Use the Expression operator to write SQL expressions that define non-procedural algorithms for one output parameter of the operator.

The expression text can contain combinations of input parameter names, variable names, and library functions. Use the Expression operator to transform the column value data of rows within a row set using SQL-type expressions, while preserving the cardinality of the input row set. To create these expressions, open the Attribute properties window for the output attribute and then open the Expression Builder.

By default, the Expression operator contains one input group and one output group.

Figure 26–7 shows a mapping that uses the Expression operator. The transaction table `ORACLE_ORDERS` contains order details such as product ID, unit price, and quantity sold. The `ORDERS_FACT` table contains an aggregation of the total sales amount across channels, products, and orders. The Expression operator is used to compute the amount of sale for each product by multiplying the unit price by the quantity sold. The Aggregator operator aggregates the sale amounts over channel code, product ID, and order ID before loading the target table.

*Figure 26–7 Expression Operator in a Mapping*



Do not use the Expression operator to write aggregation functions. Use the Aggregator operator. See "Aggregator Operator" on page 26-5 for more information about the Aggregator operator.

Also, do not use a line-end or double-dash comment (--) after an expression in an expression operator if the expression is connected to an aggregation operator, as this causes a deployment compile error. This is because an aggregation operator in this scenario adds parenthesis, and the commenting removes one of them, creating unbalanced parenthesis.

**To define an Expression operator in a mapping:**

1. Drag and drop an Expression operator onto the Mapping Editor canvas.
2. Right-click the Expression operator and select **Open Details**.

   Warehouse Builder displays the Expression Editor.
3. On the Output Attributes tab, create an output attribute by clicking on a blank cell under the Attribute column. The default data type assigned is NUMERIC. You can modify the data type and other parameters associated with the data type.
4. Define the expression used for the output attribute.

   Enter the expression directly in the Expression field of the output attribute. Or click the Ellipsis button to the right of the Expression field to display the Expression builder dialog box that enables you to specify the expression.
5. Click **OK** to close the Expression Editor.
6. Connect the Expression output attribute to the appropriate target attribute.

## Filter Operator

You can conditionally filter out rows using the Filter operator.

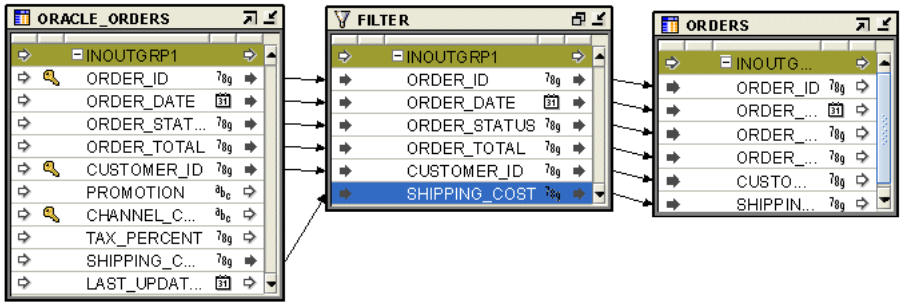
You connect a source operator to the Filter operator, apply a filter condition, and send a subset of rows to the next operator. The Filter operator filters data from a source to a target by placing a WHERE clause in SQL statement or a IF statement of the generated PL/SQL code. You specify the filter condition using the Expression Builder. The filter condition can be based on all supported data types and can contain constants.

A Filter operator has only one input/output group that can be connected to both a source and target row set. The resulting row set is a filtered subset of the source row set-based on a boolean filter condition expression. All rows that are required at the target must pass through the Filter operator. No row set can bypass the filter and be directly inserted in the target.

For a mapping that contains a Filter operator, code that displays the filter condition expression as a WHERE clause for set-based view mode is generated. The filter input names in the original filter condition are replaced by actual column names from the source table, qualified by the source table alias.

Figure 26–8 shows the mapping that uses the Filter operator to move selected data to the target table. The `ORACLE_ORDERS` table contains orders data. Use the Filter Condition property of the Filter operator to move only the booked orders which were last updated on the current system date into the `ORDERS` table.

*Figure 26–8 Filter in a Mapping*




**To define a Filter operator in a mapping:**

1. Drag and drop the **Filter** operator onto the Mapping Editor canvas.
2. Connect source attributes to the input/output group of the Filter operator.
3. Select the Filter operator header.

   The Property Inspector displays the properties of the Filter operator.
4. Click the Ellipsis button to the right of the Filter Condition property.

   Warehouse Builder displays the Expression Builder dialog box for the filter condition.
5. Define a filter condition expression using the Expression Builder.
6. Click **OK** to close the Expression Builder.
7. Connect the Filter operator outputs to the input/output group in the target.

## Adding Self Joins in a Mapping

The Mapping Editor enables you to recursively join a table, view, or other source data operators onto itself.

Also known as *tree walking*, recursively joining a table back onto itself enables you to retrieve records in a hierarchy. For example, consider a table that contains employee data including the manager for each employee. Using tree walking, you could determine the hierarchy of employees reporting up to a given manager.

To perform tree walking:

1. Create a mapping and add the desired source data operator such as a Table, View, or a Materialized View operator, which contains the hierarchal definition.
2. Connect that source data operator to a Filter operator.
3. In the Filter operator, define the filter condition with CONNECT BY as the first two words. Make sure that you include only the connect by logic in the Filter operator. That is, do not include any AND or OR logic in the filter.

# Joiner Operator

The Joiner operator joins multiple row sets from different sources with different cardinalities, and produces a single output row set.

In general, a Joiner operator corresponds to a SQL `JOIN` clause. For more information about SQL joins, see *Oracle Database SQL Language Reference*.

The Joiner operator uses a boolean condition called the join condition that relates column values in each source row set to at least one other row set. The Joiner operator results in a `WHERE` clause in the generated SQL query. When executed on Oracle 9*i* or higher, ANSI full outer joins are supported.

You can use the Joiner operator to create inner joins, outer joins, equijoins, and non-equijoins. You can also create self joins by using a Filter operator as described in "Adding Self Joins in a Mapping" on page 26-13.

For all Joiners, you must set a join condition, as described in "Join Conditions" on page 26-14.

For Joiners that define outer joins, you must use one of two methods to specify the outer join type:

- Insert Oracle SQL outer join operators (+) in the join condition, as described in "Join Conditions" on page 26-14.
- Set joiner input roles on the input groups, as described in "Join Input Roles and Inner and Outer Joins" on page 26-19.

---

**Note:**

- You cannot use both joiner input role settings and join conditions to specify outer joins.
- Operators placed between data sources and a Joiner can generate complex SQL or PL/SQL.
- Nested Joiners with no intervening operators that generate a subquery (for example, an aggregator) are generally consolidated into a single join in the generated code. You may notice this in reviewing generated code for a mapping that uses multiple nested join operators.

---

## Join Conditions

The Join Condition property of a Joiner determines the join condition in the generated code for the Joiner. By inserting the outer join operator (+), you can also use the join condition to create inner, left outer, right outer, or full outer joins.

The relational operator in a join condition is not restricted to equality. You can also use other operators such as, >, <, !=, >=, <= . Aggregation functions are not supported in join conditions.

### Join Conditions and Cartesian Products

If two tables in a join query do not have a join condition specified, the Cartesian product of the two tables is returned.

### Default Join Conditions and Foreign Keys

If the input row sets are related through foreign keys, that relationship can be used to form a default join condition.

> **Tip:** OWB can only create a default join condition for you if you have mapped your sources to the input groups of the Joiner operator before editing the join condition.

You can use this default condition or you can modify it further. If the sources are not related through foreign keys, then you must define a join condition manually.

If the default foreign keys result in duplicate WHERE clauses, the Joiner operator will remove the duplicate clauses when creating the default join condition. This can happen if the join condition references several foreign keys. For example, if table T1 has a foreign key FK1 pointing to unique key UK1 in table T2 and table T2 has a foreign key FK2 pointing to unique key UK2 in T1, the resulting join condition

```
T1.A = T2.A AND T1.B = T2.B
        /*All instances of FK1 -> UK1 are reduced to one WHERE clause*/
      AND T2.B = T1.B AND T2.C = T1.C
        /*All instances of FK2 -> UK2 are reduced to one WHERE clause */
```

is generated by the Joiner operator as

```
T2.A = T2.A AND T1.B = T2.B AND T2.C = T1.C
```

### Using Join Conditions for Outer Joins

In a join condition using Oracle SQL, you can use the outer join operator (+) to create an outer join in the generated SQL.

> **Note:** Join input roles provide an alternative to use of outer join operators for controlling outer join behavior in a Joiner, and may be more maintainable than using outer join operators in a complex join expression with multiple input groups.
>
> See "Join Input Roles and Inner and Outer Joins" on page 26-19 for details.

For example:

```
SELECT ...
FROM A, B
WHERE A.key = B.key (+);
```

This example creates a left outer join.

### Using Join Conditions for Full Outer Joins

If your target is based on Oracle 9*i* or later, the Joiner operator also supports full outer joins using the join condition. To specify a full outer join condition, you must place the outer join operator on both sides of a relational operator. For example:

```
T1.A (+) = T2.B (+)
```

The code generated for the above join expression is:

```
SELECT ...
FROM T1 FULL OUTER JOIN T2 ON (T1.A = T2.B);
```

> **Note:** Although OWB allows a full outer join to be specified by using the "(+)" notation on both sides of the condition predicate, this construct is actually invalid Oracle SQL syntax. Therefore, during code generation, OWB translates any such join condition into ANSI SQL syntax.

When creating a full outer join from the join condition, note the following:

- The ANSI join syntax is generated only if you specify a full outer join condition in the joiner. Otherwise, the following Oracle SQL join syntax is generated:

  ```
  SELECT ...
  FROM T1, T2
  WHERE T1.A = T2.B;
  ```

- You can specify both full outer join and join conditions in the same joiner. However, if both conditions are specified for the same sources, the more restrictive join type is used for generating code. For example, if you specify:

  ```
  T1.A(+) = T2.A(+) and T1.B = T2.B
  ```

  Warehouse Builder will generate an inner join statement instead of a full outer join because `T1.B = T2.B` is more restrictive than the full outer join condition between T1 and T2.

- You cannot specify a full outer join and partial outer join condition in the same joiner. If you specify a full outer join, then you cannot specify a partial outer join anywhere in the join condition. For example, `T1.A (+) = T2.A (+) and T2.B = T3.B (+)` is not valid.

- To create a full outer join in Oracle8*i*, you must use multiple SQL statements.

  The Expression Builder allows the following syntax for a full outer join:

  ```
  TABLE1.COL1 (+) = TABLE2.COL2 (+)
  ```

  The code generator translates the preceding expression into an ANSI SQL 1999 full outer join statement, similar to:

  ```
  SELECT ...
  FROM table1 FULL OUTER JOIN table2 ON (table1.col1 = table2.col2)
  ```

  This syntax is not supported by Oracle8*i*. Specifying a full outer join to an Oracle8*i* database results in a validation error.

### Grouping Join Conditions

When you create a join between more than two tables containing multiple conditions, you must clearly indicate which conditions should be grouped together.

Follow these guidelines while defining joins that contain multiple conditions:

- Use parentheses in the join condition to force clauses to be combined into one single condition.

  For example, if the join condition is specified as:

  ```
  A.ID = B.ID AND (B.ID (+) = C.ID (+) AND B.ID > 10)
  ```

  Warehouse Builder generates the following code (in this example assume ANSI code generation is specified):

  ```
  SELECT ...
  FROM
      "A"  "A"
   JOIN   "B"  "B" ON ( ( "A"."ID" = "B"."ID" ) )
   FULL OUTER JOIN   "C"  "C" ON ( "B"."ID" = "C"."ID"  )
    WHERE
    ( "B"."ID" > 10 )
  ```

The last clause, `B.ID (+) > 10`, is not included in any join condition but is treated as a WHERE clause.

If you use parentheses to group the clauses as follows:

```
A.ID = B.ID AND (B.ID (+) = C.ID (+) AND B.ID > 10)
```

Warehouse Builder generates the following code:

```
SELECT ...
FROM
    "A"  "A"
 JOIN   "B"  "B" ON ( ( "A"."ID" = "B"."ID" ) )
 FULL OUTER JOIN   "C"  "C" ON ( ( "B"."ID" = "C"."ID" and "B"."ID"  > 10 )/*
OPERATOR JOINER JOIN CONDITION */ )
```

The clause `B.ID (+) > 10` is now included in the `FULL OUTER JOIN` condition.

- You can use the outer join condition sign (+) to force Warehouse Builder to generate a condition within the join.

  For example, in the join condition

  ```
  A.ID = B.ID AND B.ID (+) = C.ID (+) AND B.ID (+) > 10
  ```

  the presence of the outer join operator in the clause `B.ID (+) > 10` forces OWB to include the condition `B.ID > 10` in the join, instead of generating a separate WHERE clause.

  > **Note:** The use of the (+) operator to force grouping in this manner is a behavior specific to OWB code generation and may not yield the same results as the same expression in standard Oracle SQL in some cases.

  The generated code is:

  ```
  SELECT ...
  FROM
      "A"  "A"
   JOIN  ( SELECT
  /* B.INOUTGRP1 */
    "B"."ID" "ID",
    "B"."NAME" "NAME",
    "B"."ATTR" "ATTR"
  FROM
    "B"  "B" ) "B" ON ( ( ( "A"."ID" = "B"."ID" ) ) AND ( ( "B"."ID" > 10 ) ) )
   FULL OUTER JOIN   "C"  "C" ON ( "B"."ID" = "C"."ID"  )
  ```

  In this case, since parenthesis is not used, the single condition B.ID (+) > 10 is moved into the first ON clause.

**Note:** When you configure ANSI SQL code generation for a mapping containing a Joiner operator, it is recommended that you inspect the generated code to verify that the conditions are grouped as intended. Sometimes, because of ambiguous conditions, the generated code may group conditions differently from what you expected.

You can force grouping of join conditions using one of the following techniques:

- You can use multiple nested Joiner operators in a mapping, with each operator having two input groups. This makes explicit the handling of join conditions and grouping.
- Alternatively, you can set the ANSI SQL Syntax parameter to false. This generates the `WHERE` clause exactly as specified by the Join Condition property.

**Code Template Mappings and Grouping Join Conditions** For code template mappings, if you do not follow either of the guidelines listed above, a validation warning is displayed and one of the following actions is performed:

- If the condition listed last cannot be combined with the condition adjacent to it, the last condition is moved to the WHERE clause.

  Consider the condition C.ENAME = A.ENAME AND A.DEPTNO = B.DEPTNO AND C.SAL (+) > 1000. The condition C.SAL (+) > 1000 can be paired with C.ENAME = A.ENAME. However, its placement is not consistent with such a pairing and so a validation warning is displayed.

- If the condition listed last can be combined with the condition adjacent to it, a combined group condition is formed.

  Consider the condition A.DEPTNO = B.DEPTNO AND C.ENAME = A.ENAME AND C.SAL (+) > 1000. The condition C.SAL (+) > 1000 can be paired with C.ENAME = A.ENAME. Thus it is included as part of the combined condition C.ENAME = A.ENAME AND C.SAL (+) > 1000.

### Join Conditions and Join Order

By default, the order of input groups in a joiner is used as the join order.

When generating ANSI SQL syntax, you can force a different join order by using a specific order in the join condition. Consider the following example:

```
SELECT ...
      FROM T1 FULL OUTER JOIN T2 ON (T1.A=T2.A)
                         JOIN T3 ON (T2.A=T3.A);
```

If you create input groups in another order, such as T1, T3, T2. Warehouse Builder will generate the following:

```
      SELECT ...
      FROM T1 JOIN T3 ON (1=1)
              JOIN T2 ON (T1.A=T2.A and T2.A=T3.A);
```

When T1 and T3 are joined, there is no join condition specified. Warehouse Builder fills in a condition 1=1 (essentially a boolean true) and the two conditions you specified are used to join (T1 JOIN T3) with T2.

Now consider an example where the input groups are created in the order T1, T3, T2, but you want the generated FROM clause order to be T1, T2, T3 as in the first example. Then create the join condition so that the sources appear in the order that they should appear in the generated code, like this:

```
T1.A(+) = T2.A(+) AND T2.A = T3.A
```

### Filters in Join Conditions

The filter condition is applied after join. For example, consider the following join:

```
Input1.c --- +
Input2.c --- +---> Joiner
Input3.c --- +
```

with the following conditions:

- **Condition 1:** Input1.c (+) = Input2.c (+)
- **Condition 2:** Input2.c = Input3.c
- **Condition 3:** Input1.c is not null

The first two conditions are true joins while the third is a filter condition. If ANSI code is to be generated, then the join condition is interpreted as

```
SELECT ...
FROM Input1 FULL OUTER JOIN Input2 ON (Input1.c = Input2.c)
JOIN Input3 ON (Input2.c = Input3.c)
WHERE Input1.c IS NOT NULL;
```

### Join Conditions and SAP Sources

The join condition is defined in a SQL context. For SAP sources, ABAP code can be generated by interpreting the SQL join condition in the ABAP context. ABAP can only join over defined foreign key relationships.

## Join Input Roles and Inner and Outer Joins

Join input roles on joiner input groups provide flexible control of the use of inner and outer joins on the input data flows.

Possible join input roles for an input group are: Standard (the default), Outer Join and Full Outer Join. Warehouse Builder can generate ANSI SQL or Oracle SQL for inner joins, left and right outer joins, and full outer joins based on these settings. Use the order of input groups and the input role settings on each input group to specify the type of join to create.

> **Note:** If using outer join operators in join expression to control outer joins, set the join input roles for all groups to Standard.

### Specifying Inner and Outer Join Types Using Join Input Roles

When considering two adjacent input groups, the join relationship between them in the generated code is determined by the join role property setting of the two groups. Assume that input group INGRP1 is connected to source table T1 and input group INGRP2 is connected to source table T2. The join relationship is determined by the following rules (in order of precedence):

*Table 26–1 Join Input Roles and Resulting Join Types*

| Input Roles | Resulting Join Operation |
|---|---|
| Both roles set to Standard (default) | Inner join<br>`T1 JOIN T2` |
| Either role set to Full Outer | Full outer join<br>`T1 FULL OUTER JOIN T2` |
| INGRP1 role set to Standard<br>INGRP2 role set to Outer | Left outer join<br>`T1 LEFT OUTER JOIN T2` |
| INGRP1 role set to Outer<br>INGRP2 role set to Standard | Right outer join<br>`T1 RIGHT OUTER JOIN T2` |
| Both roles set to Outer | See "Outer Join Input Roles with More than Two Input Groups" on page 26-20 |

## Outer Join Input Roles with More than Two Input Groups

When there are more than two input groups, Warehouse Builder iterates over the input groups, and generates a series of joins linking the input tables.

If one or the other input group has the input role Standard or Full Outer Join, then the join generated is determined using the rules described in Table 26–1, " Join Input Roles and Resulting Join Types" on page 26-20.

However, if two consecutive input groups both have the input role Outer Join, then the join generated may be a left outer join or a right outer join, depending upon the previous join generated. The rules for determining the join type to generated are in Table 26–2, " Join Generated for Consecutive Outer Join Input Roles" on page 26-20. In this table, assume that INGRP1, INGRP2 and INGRP3 are associated with tables T1, T2 and T3 respectively.

*Table 26–2 Join Generated for Consecutive Outer Join Input Roles*

| Input Roles | Resulting Join Operation |
|---|---|
| The two input groups are the first two input groups<br>INGRP1 = Outer<br>INGRP2 = Outer | Right outer join<br>`T1 RIGHT OUTER JOIN T2` |
| The preceding join generated was a Left Outer Join<br>INGRP1 = Standard<br>INGRP2 = Outer<br>INGRP3 = Outer | Left outer join<br>`T1 LEFT OUTER JOIN T2`<br>`LEFT OUTER JOIN T3` |
| The preceding join generated was not a Left Outer Join<br>INGRP1 = Outer<br>INGRP2 = Outer<br>INGRP3 = Outer | Right outer join<br>`T1 RIGHT OUTER JOIN T2`<br>`RIGHT OUTER JOIN T3` |

## Steps to Use a Joiner Operator in a Mapping

**To add a Joiner operator to a mapping:**

1. Drag and drop the Joiner operator onto the Mapping Editor canvas.
2. If you need to join more than two sources, right-click the Joiner operator and select **Open Details**.

   Warehouse Builder displays the Joiner Editor.

   On the Groups tab, create more input groups as needed.

   To create an input group, click the empty cell under the Group column and enter the new group name. The default direction assigned to the new group is `Input`.

   Click **OK** after adding your input groups.
3. For each input group, connect an output group from a source to the desired Joiner input group.

   Output attributes are created with data types matching the corresponding input data types.
4. If necessary, change the Input Role for each source.

   See "Join Input Roles and Inner and Outer Joins" on page 26-19.
5. Select the Joiner operator header.

   The Property Inspector displays the properties of the Joiner operator.
6. Click the Ellipsis button to the right of the Join Condition property.

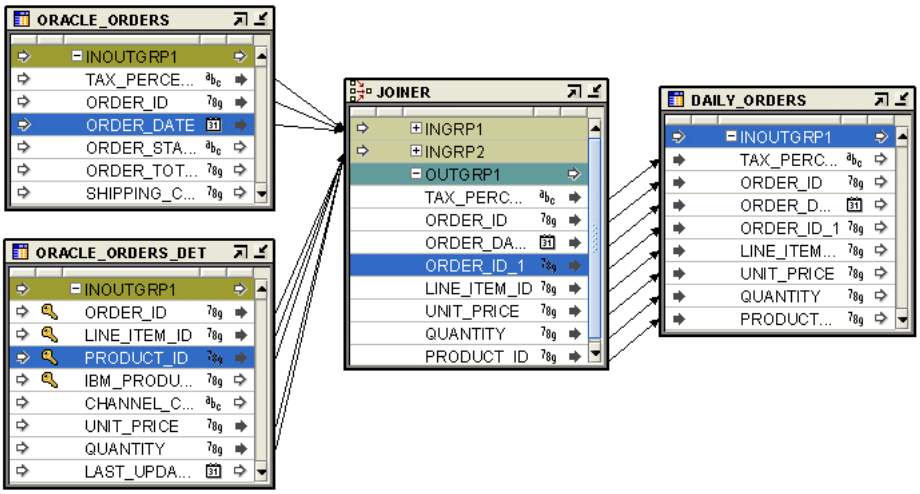
   The Expression Builder dialog box is displayed.
7. Define the join condition.

   See "Join Conditions" on page 26-14.
8. Click **OK** to close the Expression Builder.
9. Map the attributes of the output group of the Joiner operator to the target.

### Example: Using the Joiner Operator

Figure 26–9 shows a mapping that contains a Joiner operator. The two source tables `ORACLE_ORDERS` and `ORACLE_ORDER_LINES` are joined to combine the data from these tables into one table. The output of the Joiner operator is passed to the target table `DAILY_ORDERS`.

*Figure 26–9 Joiner in a Mapping*




# LCR Cast Operator

Use the LCR Cast operator to expand an LCR (Logical Change Record) object into its constituent columns. This enables you to update the target object with insert, update, or delete operations contained in the LCR. Typically, the LCR Cast operator is used just after a Queue operator in a real-time mapping that publishes source changes to target objects.

The LCR Cast operator must be bound to the table for which it stores change records. It contains one input group and one output group, both of which are non-editable. You cannot add groups to this operator, but you can rename the existing input or output group. The input group contains an attribute Event, of type SYS.LCR$_ROW_RECORD, that stores the LCRs. You must connect a `SYS.ANYDATA` attribute to the input group. The output group contains the columns of the tables to which the LCR Cast operator is bound.

**To add an LCR Cast operator to a mapping:**

1. Drag and drop a LCR Cast operator onto the Mapping Editor canvas.

   Warehouse Builder displays the Add LCR Cast dialog box.

2. Select the table to which the LCR Cast operator must be bound.

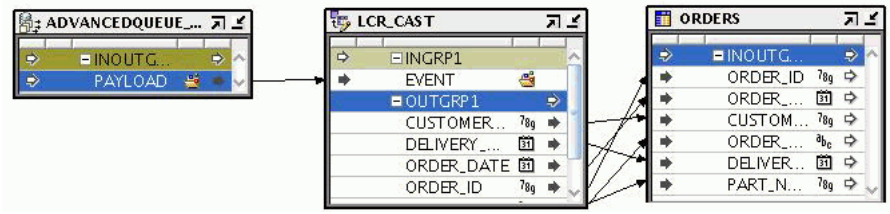
3. Connect the output attribute of a source operator to the input group of the LCR Cast operator.

4. Connect the attributes of the output group in the LCR Cast operator to the target table to which you want to publish the changes contained in the LCR.

Figure 26–10 describes an example of using the LCR Cast operator in a mapping. The Queue operator ADVANCED_QUEUE represents the advanced queue that stores the change records for the source table. The LCR Cast operator is bound to the target table into which the changed records from the source should be transferred.

The AQ contains a payload that is represented by the PAYLOAD attribute. The payload stores change data in the form of LCRs. Map the PAYLOAD attribute the LCR Cast operator to expand the LCR into the columns contained in the target table that is bound to the LCR Cast operator. You then map the output attributes of the LCR Cast operator to the target table `ORDERS`.

*Figure 26–10 Mapping that Uses an LCR Cast Operator*




## LCR Splitter Operator

Use the LCR Splitter operator to direct changes to different tables along data flow paths.

The LCR Splitter contains one input group and one output group. Both groups contain one attribute called Event of type SYS.LCR$_ROW_RECORD. The input group represents the represents the LCR object. You cannot add input or output groups.

## Lookup Operator

Use the Lookup operator to lookup data from a table, view, cube, or dimension. For example, use the Lookup operator when you define a mapping that loads a cube or when you define surrogate keys on the dimension.

You can use the same Lookup operator to lookup data from multiple objects.

> **See Also:** *Oracle Warehouse Builder Concepts* for information about surrogate identifiers

The key that you look up can be any unique value. It need not be a primary or unique key, as defined in an RDBMS. The Lookup operator reads data from a lookup table using the key input you supply and returns exactly one matching row. This operator returns a row for each input key. You can have multiple Lookup operators in the same mapping.

The output of the Lookup operator corresponds to the columns in the lookup object. In case multiple records are returned by the lookup operation, you can specify which of these records is selected.

The Lookup Wizard contains one input group and one output group. You can create additional input and output groups. The attributes in each input group must be connected from the same data source. Each output group is bound to one lookup object. Each lookup uses attributes from only one input group as search values. That is, each output group is associated with only one input group. The tooltip for each output group displays the input group associated with it and the lookup condition used.

Since an output group is bound to an object, its attributes are the columns in the object. You can create additional output attributes, that are derived from the object columns, by using the Expression property of the output attribute.

Each output attribute for the lookup has a property called DEFAULT VALUE. The DEFAULT VALUE property is used instead of NULL in the outgoing row set if no value is found in the lookup table for an input value. The generated code uses the NVL function. The Lookup always results in an outer-join statement.

The table, view, or dimension from which the data is being looked up is bound to the Lookup operator. You can synchronize a Lookup operator with the workspace object to which it is bound. But you cannot synchronize the workspace object with the Lookup operator. For more information about synchronizing operators, see "Synchronizing Operators and Workspace Objects" on page 5-28.

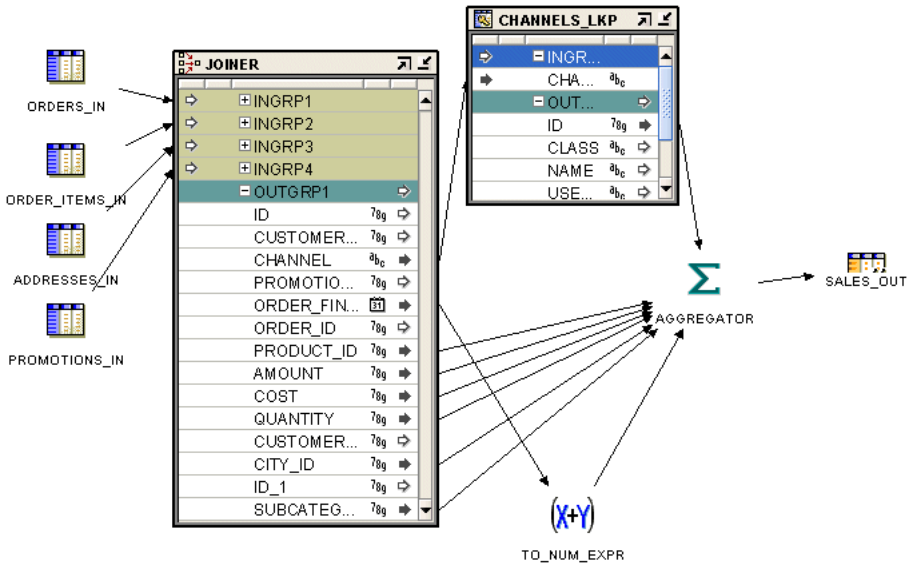
### Points to Keep In Mind while Using the Lookup Operator

The Lookup operator returns only one row. When the result of the lookup returns multiple rows, you must specify which of the rows must be used as the return value. You can select either the first row or the last row from the returned rows.

Since you are selecting the first or last row, the order in which the lookup results are returned is important. Since SQL queries do not guarantee repeatable order, you must specify the appropriate ORDER BY clause to sort lookup results. To eliminate non-determinism, ensure that you specify a detailed ordering clause that ensures that the desired record is picked as the lookup result.

Figure 26–11 shows a mapping that is used to load a cube. Data from four source tables is joined using a Joiner operator. But the data in the source tables only contains a channel name. To load the cube, we need the value of the surrogate identifier. A Lookup operator is used to lookup the surrogate identifier of the `CHANNELS` dimension and then load this value into the cube.

***Figure 26–11 Lookup Operator in a Mapping***




## Using the Lookup Operator

You have the following options for using a Lookup operator:

- **Define a new Lookup operator:** Drag a Lookup operator from the Palette onto the mapping. The Mapping Editor displays a wizard.
- **Edit an existing Lookup operator:** Right-click the Lookup operator and select **Open Details**.

Whether you are using the operator wizard or the operator editor, complete the following pages:

- Name

- Groups
- Lookup Tables
- Input Attributes
- Output Attributes
- Lookup Conditions
- Multiple Match Rows
- Type 2 History Lookup
- No-match Rows

### Name

Use the Name page or Name tab to specify a name and optional description for the Lookup operator.

### Groups

Use the Groups page to specify one input and one output group.

In a Lookup operator, the input group represents the data from the source that is contained across multiple attributes. The output group represents that data transformed into rows.

You can rename and add descriptions to the default input and output groups. You can also create additional input and output groups that you require. To create an input or output group, specify a name for the group, select the direction (Input or Output), and provide an optional description.

### Lookup Tables

Every output group is associated with a lookup table. Use the Lookup Tables page or the Lookup Tables tab to select the lookup table that must be associated with each output group.

The Group field displays all the output groups defined for the Lookup operator. Select an output group and specify the lookup table to which the output group is bound. To select the lookup table, click the list in the section below the Group field. The objects from which you can perform a lookup are listed in the tree displayed. Select the lookup object.

Repeat this step for all the output groups in the Lookup operator.

### Input Attributes

Use the Input Attributes page to define the input attributes of each input group.

The Group field lists all the input groups defined for the Lookup operator. Select an input group, and create the attributes for that group. Each input attribute contains a field called Default Value. Use this field to specify a default value for the input attribute.

### Output Attributes

Use the Output Attributes field to create output attributes in each output group. Since each output group is bound to a lookup object, the columns from the bound lookup object are automatically listed as output attributes for the group. Create any additional

output attributes that are required. These could be values that are derived from the existing attributes using expressions.

The Groups list lists the output groups defined for the Lookup operator. Select an output group to display the attributes in this group. To create an output attribute, click a blank cell in the Name field and enter the attribute name. Then specify the additional parameters of the attribute such as data type, default values, description, and so on.

Output attributes have an additional parameter called Expression. Use this column to specify the expression used to determine the value of the output attribute. You can enter the expression directly in the Expression column. Or, click the Ellipsis button to the right of the Expression column to display the Expression Builder interface. Use this interface to define the expression.

### Lookup Conditions

Use the Lookup page to provide details about the object on which the lookup is being performed. This object is referred to as the lookup result. You can perform a lookup on any table, view, or dimension that belongs to the current project.

On the Lookup page, also associate each output group with its corresponding input group. Each output group must be bound to one input group. This input group provides the values that are searched for in the lookup table represented by the output group.

The Output field lists the output groups and the Input field lists the input groups of the Lookup operator. Select and output group and from the Input list, select the input group to which it is bound. Repeat this process for all output groups.

Once the output groups are bound to the corresponding input groups, you must map the output attributes and the input attributes that must be compared to perform the lookup. Select the output group in the Output field and use the area at the bottom of this page to specify the lookup condition. The contents displayed in this area depend on whether you choose Simple Editing or Freestyle Editing.

If you selected Simple Editing, a table with two columns is displayed. Use the Lookup Table Column to select the column from the lookup table with which the attribute selected in the Input Attribute column is compared.

If you select Freestyle Editing, an interface similar to the Expression Builder is displayed. Use this to define the condition that will be used as the lookup condition. You can use an equality or a non-equality condition as the lookup condition.

If you select a dimension level for the lookup, the options displayed are the surrogate and business identifier of the dimension level and the primary key of the database table that stores the dimension data.

### Multiple Match Rows

Use the Multiple Match Rows page to define which row from the lookup result should be selected as the lookup result if the lookup returns multiple rows. Multiple rows are returned if the lookup condition specified matched more than one record.

For each output group, you must define the action to be taken if multiple rows are returned by the lookup operation.

Select an output group in the Output Group field and then specify values for this group as described in the following sections.

#### Selecting the Action to Perform When Multiple Rows are Returned

Select one of the following options:

- **Error: multiple rows cause mapping to fail**

  Select this option to indicate that when the mapping that contains this Lookup operator is run, if the lookup operation for the selected output group returns more than one row, the mapping execution fails.

- **All Rows (number of result rows may differ from the number of input rows)**

  Select this option to indicate that when the Lookup operator returns multiple rows for the selected output group, all the rows should be returned as the lookup result.

- **Select single row**

  Select this option to specify that when the Lookup operator returns multiple rows for the selected output group, only one row from the returned rows must be selected as the lookup result. When you select this option, the fields contained in the section below this option are enabled. Use these fields to specify which row from the lookup result set should be selected as the lookup result.

### Specifying the Row to Select as the Lookup Result

You must select the one row that should be selected from the multiple rows produced by the lookup operation only if you selected the Select single row option in the previous section.

For each output group, specify the following information about the row that must be selected from the multiple rows returned from the lookup operation.

**Row Position:** Select one of the following options:

- Any row

  Any one row among the result set returned by the Lookup operator is selected as the lookup result.

- First row

  The first row from the result set returned by the Lookup operator is selected as the lookup result.

- Last row

  The first row from the result set returned by the Lookup operator is selected as the lookup result.

- Nth row

  The *n*th row from the result set returned by the Lookup operator is selected as the lookup result. Click the list on the **Nth Row** field to specify the values of *n*.

**Order Result Set By**

Use this section to specify how the rows in the result set (containing multiple rows) should be ordered. Ordering columns is important when you select the first, last, or *n*th row from the result set as the lookup result. Ensure that you specify ordering conditions such that the row you want returned is selected.

The Available section lists the lookup table columns for the output group selected in the Output Group field. Select the columns that you want to use to order rows in the lookup result set and use the arrow to move them to the Selected section. In the Selected section, ensure that the columns are listed in the same order (from top to bottom) in which you want the result set to be ordered. For example, if you want to implement an ordering such as ORDER BY attr2, attr3, and then attr1, the attributes should be listed in the same order in the Selected section. You can use the arrows to the right of the Selected section to change the position of selected columns.

### No-match Rows

Use the No-match Rows page to indicate the action to be taken when there are no rows that satisfy the lookup condition specified on the Lookup page. You must specify an action for all the output groups.

Select an output group in the Output Group field and then choose one of the following options:

- **Return no row**

  This option does not return any row when no row in the lookup result satisfies the matching condition.

- **Return a row with the following default values**

  This option returns a row that contains default values when the lookup condition is not satisfied by the lookup result. Use the table below this option to specify the default values for each lookup column.

### Type 2 History Lookup

Use this page only if you selected a Type 2 SCD as the lookup result on the Lookup page. When the lookup result is a Type 2 SCD, you must specify which version of a particular record is to be used as a lookup.

For each output group that you bound to a Type 2 SCD, select the group in the Output Group field and then choose one of the following options:

- **Use the most current record**

  This option returns the current record that corresponds to the attribute being looked up using the lookup condition. The current record is the one with the latest timestamp.

- **Specify the historic date as a constant value**

  This option returns the record that contains the constant value that is specified using the **Date** and **Time** lists.

- **Choose an input attribute that holds the historic value**

  This option enables you return records that pertain to a date and time that is contained in one of the input attributes. Use the **Input Attribute** list to select the attribute that contains the historic value.

# Pivot Operator

The Pivot operator enables you to transform a single row of attributes into multiple rows.

Use this operator in a mapping when you want to transform data that is contained across attributes instead of rows. This situation can arise when you extract data from non-relational data sources such as data in a crosstab format.

## Example: Pivoting Sales Data

The external table `SALES_DAT` contains data from a flat file. There is a row for each sales representative and separate columns for each month.

> **See Also:** *Oracle Warehouse Builder Sources and Targets Guide* for more information about external tables

Figure 26–12 displays the flat file SALES_DAT.

***Figure 26–12 SALES_DAT***

| ID | Reg | M1 | M2 | M3 | M4 | M5 | M6 | M7 | M8 | M9 | M10 | M11 | M12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0675 | 4 | 10.5 | 11.4 | 9.5 | 8.7 | 7.4 | 7.5 | 7.8 | 9.7 | | | | |
| 0676 | 3 | 9.5 | 10.5 | 10.3 | 7.6 | 8.0 | 7.8 | 8.7 | 8.9 | | | | |
| 0679 | 3 | 8.7 | 7.4 | 7.5 | 7.8 | 9.7 | 10.3 | 7.6 | 8.0 | | | | |
| 0683 | 2 | 9.5 | 10.5 | 10.3 | 9.5 | 8.7 | 7.4 | 7.8 | 8.7 | | | | |
| 0684 | 1 | 11.4 | 9.5 | 8.7 | 7.4 | 7.5 | 10.3 | 9.5 | 8.7 | | | | |
| 0687 | 1 | 9.5 | 8.7 | 7.4 | 7.8 | 8.7 | 7.4 | 7.5 | 7.8 | | | | |
| 0690 | 1 | 8.7 | 7.4 | 7.8 | 8.7 | 11.4 | 9.5 | 8.7 | 7.4 | | | | |

Table 26–3 shows a sample of the data after a pivot operation is performed. The data that was formerly contained across multiple columns (M1, M2, M3...) is now contained in a single attribute (Monthly_Sales). A single ID row in SALES_DAT corresponds to 12 rows in pivoted data.

***Table 26–3 Pivoted Data***

| REP | MONTH | MONTHLY_SALES | REGION |
|---|---|---|---|
| 0675 | Jan | 10.5 | 4 |
| 0675 | Feb | 11.4 | 4 |
| 0675 | Mar | 9.5 | 4 |
| 0675 | Apr | 8.7 | 4 |
| 0675 | May | 7.4 | 4 |
| 0675 | Jun | 7.5 | 4 |
| 0675 | Jul | 7.8 | 4 |
| 0675 | Aug | 9.7 | 4 |
| 0675 | Sep | NULL | 4 |
| 0675 | Oct | NULL | 4 |
| 0675 | Nov | NULL | 4 |
| 0675 | Dec | NULL | 4 |

To perform the pivot transformation in this example, create a mapping like the one shown in Figure 26–13.

***Figure 26–13 Pivot Operator in a Mapping***

In this mapping that performs the pivot transformation, the data is read from the external table once, pivoted, aggregated, and written it to a target in set-based mode. It is not necessary to load the data to a target directly after pivoting it. You can use the Pivot operator in a series of operators before and after directing data into the target operator. You can place operators such as filter, joiner, and set operation before the

Pivot operator. Since pivoted data is not a row-by-row operation, you can also execute the mapping in set-based mode.

## The Row Locator

In the Pivot operator, the row locator is an output attribute that you create to correspond to the repeated set of data from the source. When you use the Pivot operator, a single input attribute is transformed into multiple rows and generates values for a row locator. In this example, since the source contains attributes for each month, you can create an output attribute named 'MONTH' and designate it as the row locator. Each row from SALES_DAT then yields 12 rows of pivoted data in the output.

Table 26–4 shows the data from the first row from SALES_DAT after the data is pivoted with 'MONTH' as the row indicator.

***Table 26–4 Data Pivoted By Row Indicator***

| REP | MONTH | MONTHLY_SALES | REGION |
|---|---|---|---|
| 0675 | Jan | 10.5 | 4 |
| 0675 | Feb | 11.4 | 4 |
| 0675 | Mar | 9.5 | 4 |
| 0675 | Apr | 8.7 | 4 |
| 0675 | May | 7.4 | 4 |
| 0675 | Jun | 7.5 | 4 |
| 0675 | Jul | 7.8 | 4 |
| 0675 | Aug | 9.7 | 4 |
| 0675 | Sep | NULL | 4 |
| 0675 | Oct | NULL | 4 |
| 0675 | Nov | NULL | 4 |
| 0675 | Dec | NULL | 4 |

## Using the Pivot Operator

You have the following options for using a Pivot operator:

- **Define a new Pivot operator:** Use the Pivot Wizard to add a new Pivot operator to a mapping. Drag a Pivot operator from the Palette onto the mapping. The Mapping Editor displays the Pivot Wizard.
- **Edit an existing Pivot operator:** Use the Pivot Editor to edit a Pivot operator you previously created. Right-click the Pivot operator and select **Open Details**. The Mapping Editor opens the Pivot Editor.

Whether you are using the Pivot Wizard or the Pivot Editor, complete the following pages:

- General
- Groups
- Input Connections
- Input Attributes
- Output Attributes

- Pivot Transform

### General

Use the General page to specify a name and optional description for the Pivot operator. By default, the wizard names the operator "Pivot".

### Groups

Use the Groups page to specify one input and one output group.

In a Pivot operator, the input group represents the data from the source that is contained across multiple attributes. The output group represents that data transformed into rows.

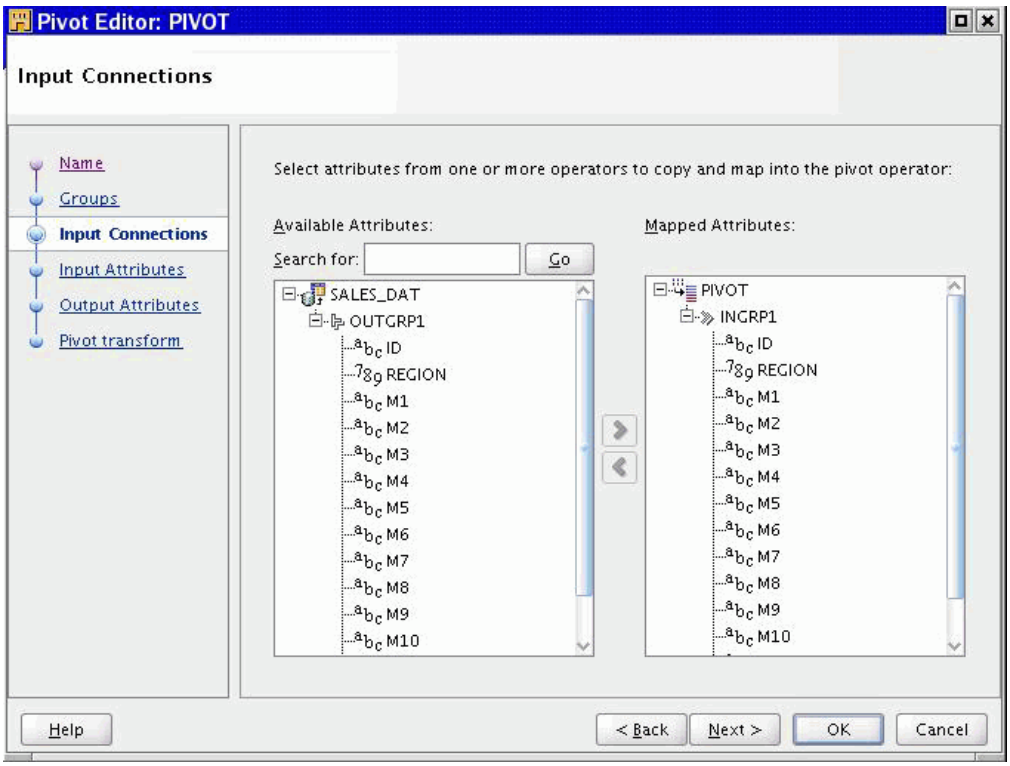
You can rename and add descriptions to the input and output groups. Since each Pivot operator must have exactly one input and one output group, the wizard prevents you from adding or removing groups or changing group direction.

### Input Connections

Use the Input Connections page to copy and map attributes into the Pivot operator. The attributes you select become mapped to the pivot input group. The left side of the page displays a list of all the operators in the mapping.

Figure 26–14 shows a group from the external table `SALES_DAT` selected as input for the Pivot operator.

***Figure 26–14 Pivot Operator Input Connections Tab***




**To complete the Input Connections page for a Pivot operator:**

1. Select complete groups or individual attributes from the left panel.

To search for a specific attribute or group by name, type the text in **Search for** and select **Go.** To find the next match, select **Go** again.

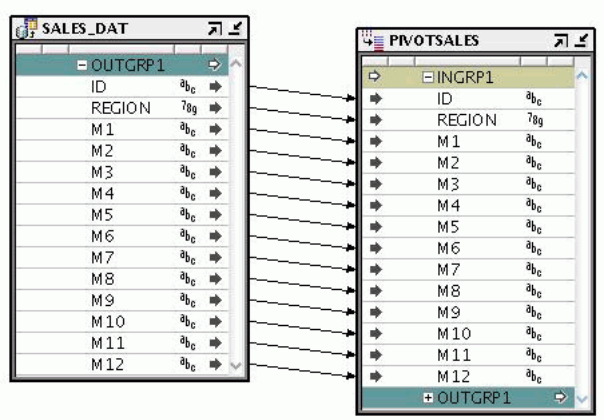
Press the Shift key to select multiple attributes. If you want to select attributes from different groups, then you must first combine the groups with a Joiner or Set operator.

2. Use the right arrow button in the middle of the page to move your selections to the right side of the wizard page.

   Use the left arrow to remove groups or attributes from the input connections list. Warehouse Builder removes the selection from the input group and removes the data flow connection between the source operator and the Pivot operator.

   Figure 26–15 shows a group from `SALES_DAT` copied and mapped into the `PIVOTSALES` operator.

***Figure 26–15 Attributes Copied and Mapped into Pivot In Group***




## Input Attributes

Use the Input Attributes page to modify the attributes you selected in the Input Connections tab or wizard page.

You can perform the following tasks from the Input Attributes page:

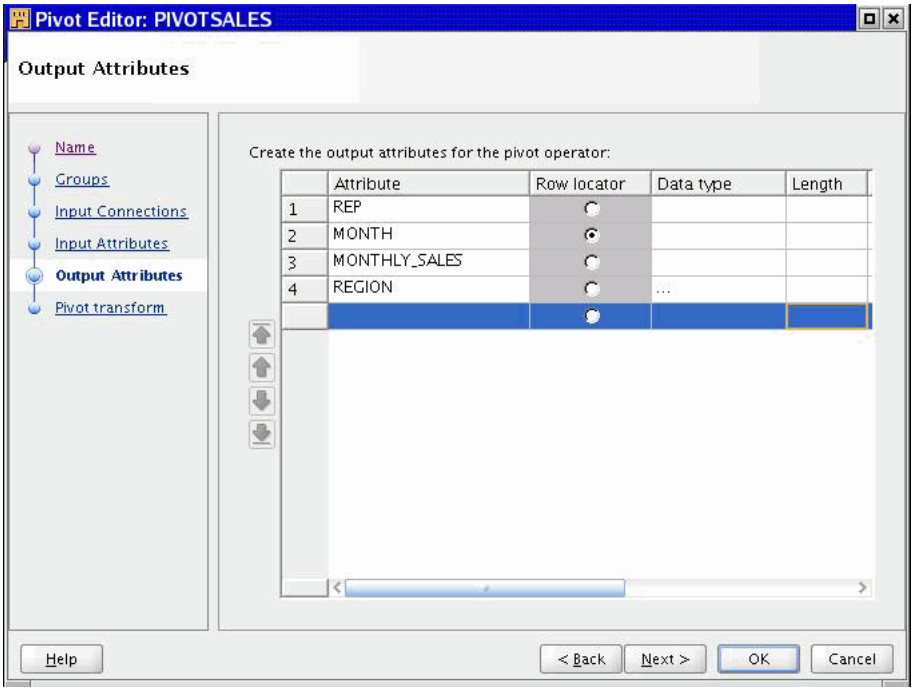
- **Add attributes:** Enter the attribute name and other attribute details in an empty row on this page.
- **Change attribute properties:** You can change the attribute name, data type, length, precision, and scale.
- **Add an optional description:** Type a description for the input attributes.
- **Designate attribute keys:** As an option, use the **Key** check box to indicate an attribute that uniquely identifies the input group.

## Output Attributes

Use the Output Attributes page to create the output attributes for the Pivot operator. If you designated any input attributes as keys on the Input Attributes tab or wizard page, then those input attributes are displayed as output attributes that you cannot edit or delete.

Figure 26–16 displays the output attributes with `MONTH` selected as the row locator.

*Figure 26–16 Pivot Output Attributes Tab*




You can perform the following tasks from the pivot Output Attributes Page:

- **Change attribute properties:** Except for attributes you designated as keys on the previous tab or wizard page, you can change the attribute name, data type, length, precision, and scale.
- **Add an optional description:** Type a description for the output attributes.
- **Designate a row locator:** Although you are not required to designate a row locator for the Pivot operator, it is recommended. When you identify the row locator on the Output Attributes page or tab, it is easier for you to match your output data to the input data.

  In the Pivot operator, the row locator is an output attribute that corresponds to the repeated set of data from the source. For example, if the source data contains separate attributes for each month, create an output attribute '`MONTH`' and designate it as the row locator.

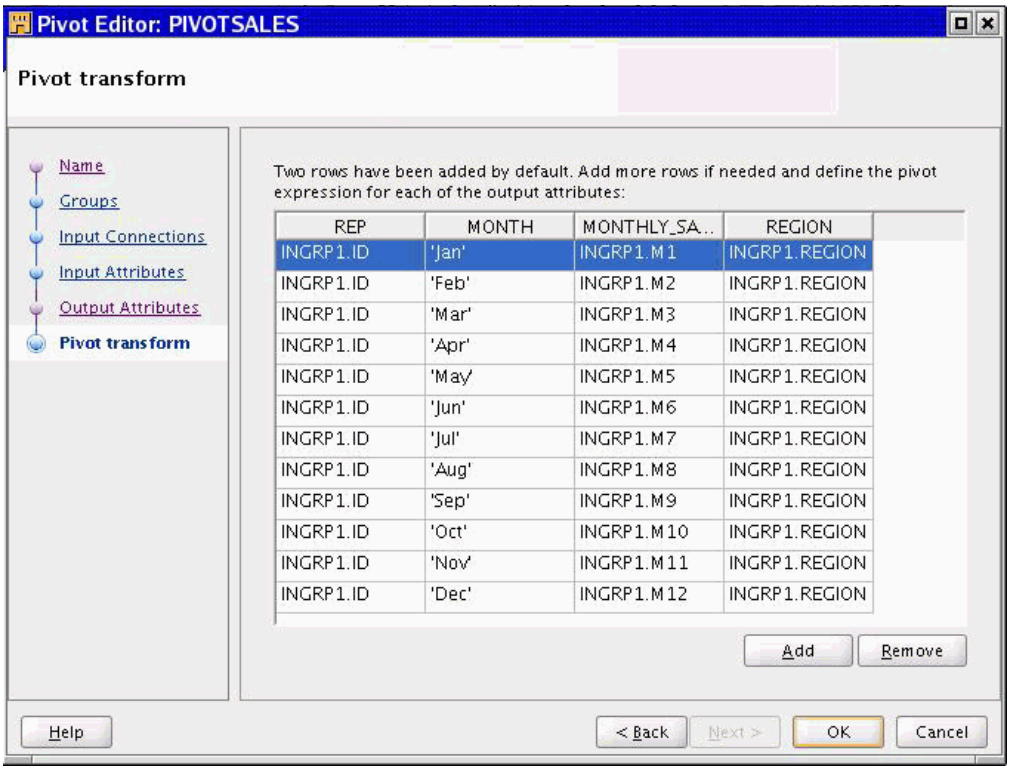
## Pivot Transform

Use the Pivot Transform page to write expressions for each output attribute.

By default, two rows are displayed. Use **Add** to specify how many rows of output you want from a single row in the source. For example, if your source contains an attribute for each quarter in a year, you can specify 4 rows of output for each row in the source. If the source data contains an attribute for each month in the year, then you can specify 12 rows of output for each row in the source.

Figure 26–17 shows the Pivot Transform tab with the pivot expressions defined for a source with an attribute for each month.

*Figure 26–17 Pivot Transform Tab*




Write pivot expressions based on the following types of output:

- **Row locator:** Specify a name for each row where the name is a value you want to load into the table. For example, if the row locator is 'MONTH', type 'Jan' for the first row.
- **Pivoted output data:** Select the appropriate expression from the list box. For example, for the row you define as 'Jan', select the expression that returns the set of values for January.
- **Attributes previously specified as keys:** Defines the expression for you.
- **Unnecessary data:** If the Pivot Transform page contains data that you do not want as output, use the expression 'NULL'. Warehouse Builder outputs a repeated set of rows with no data for attributes you define as 'NULL'.

When using the wizard to create a new Pivot operator, click **Finish** when you want to close the wizard. The Mapping Editor displays the operator you defined.

When using the Pivot Editor to edit an existing Pivot operator, click **OK** when you have finished editing the operator. The Mapping Editor updates the operator with the changes you made.

## Post-Mapping Process Operator

Use a Post-Mapping Process operator to define a procedure to be executed after running a PL/SQL mapping. For example, you can use a Post-Mapping Process operator to reenable and build indexes after a mapping completes successfully and loads data into the target.

The Post-Mapping Process operator calls a function or procedure after the mapping is executed. The output parameter group provides the connection point for the returned

value (if implemented through a function) and the output parameters of the function or procedure. There are no restrictions on the connections of these output attributes

The Post-Mapping Process operator contains groups corresponding to the number and direction of the parameters associated with the selected PL/SQL procedure or function. This list of groups and attributes can only be modified through synchronization with workspace objects.

You can map constants, data generators, mapping input parameters, and output from a Pre-Mapping Process into a Post-Mapping Process operator. The Post-Mapping Process operator is not valid for an SQL*Loader mapping.

After you add a Post-Mapping Process operator to the Mapping Editor, use the operator properties dialog box to specify run conditions in which to execute the process.

**To use a Post-Mapping Process operator in a mapping:**

1. Drag and drop a Post-Mapping Process operator onto the Mapping Editor canvas.

   Warehouse Builder displays the Add Post-Mapping Process dialog box.

2. Use the Add Post-Mapping Process dialog box to select or create a transformation. For more information about how to use the Add Post-Mapping Process dialog box, see "Using the Add Operator Dialog Box to Add Operators" on page 5-15.

3. Connect the output attribute of a source operator to the input/output group of the Post-Mapping Process operator.

4. Set the run conditions for the operator.

**To set run conditions for a Post-Mapping Process operator:**

1. From the mapping canvas, select a Post-Mapping Process operator.

   The Property Inspector displays the properties of the Post-Mapping Process operator.

2. Click **Post-Mapping Process Run Condition** and select one of the following run conditions:

   **Always:** The process runs regardless of errors from the mapping.

   **On Success:** The process runs only if the mapping completes without errors.

   **On Error:** The process runs only if the mapping completes with errors exceeding the number of allowed errors set for the mapping.

   **On Warning:** The process runs only if the mapping completes with errors that are less than the number of allowed errors set for the mapping.

   If you select **On Error** or **On Warning** and the mapping runs in row-based mode, you must verify the **Maximum Number of Errors** set for the mapping. To view the number of allowed errors, right-click the mapping in the Projects Navigator, select **Configure,** and expand **Runtime Parameters.**

## Pre-Mapping Process Operator

Use a Pre-Mapping Process operator to define a procedure to be executed before running a mapping.

For example, you can use a Pre-Mapping Process operator to truncate tables in a staging area before running a mapping that loads tables to that staging area. You can also use a Pre-Mapping Process operator to disable indexes before running a mapping

that loads data to a target. You can then use a Post-Mapping Process operator to reenable and build the indexes after running the mapping that loads data to the target.

The Pre-Mapping Process operator calls a function or procedure whose metadata is defined prior to executing a mapping. The output attribute group provides the return value (if implemented as a function) and the output parameters of the function or procedure. You can connect these attributes to any other operators downstream, and they do not have the connection restrictions that apply to the Post-Mapping Process operator.

When you drop a Pre-Mapping Process operator onto the Mapping Editor canvas, a dialog box opens displaying the available libraries, categories, functions, and procedures. After you select a function or procedure from the tree, the operator displays the attributes that correspond to the selected function or procedure.

The Pre-Mapping Process operator contains groups corresponding to the number and direction of the parameters associated with the selected PL/SQL procedure or function.

After you add a Pre-Mapping Process operator to the Mapping Editor, use the Property Inspector to specify the Run condition of the mapping.

**To use a Pre-Mapping Process operator in a mapping:**

1. Drag and drop a **Pre-Mapping Process** operator onto the Mapping Editor canvas.

   The Add Pre-Mapping Process dialog box is displayed.

2. Use the Add Pre-Mapping Process dialog box to select or create a transformation. For more information about how to use this dialog box, see "Using the Add Operator Dialog Box to Add Operators" on page 5-15.

3. Connect the output attribute of the Pre-Mapping Process operator to the input group of a target operator.

4. Set the run conditions for the operator.

**To set run conditions for a mapping with a Pre-Mapping Process operator:**

1. In the mapping canvas, select the Pre-Mapping Process operator.

   The Property Inspector displays the properties of the Pre-Mapping Process operator.

2. Click **Mapping Run Condition** and select one of the following run conditions:

   **Always:** Runs the mapping after the process completes, regardless of the errors.

   **On Success:** Runs the mapping only if the process completes without errors.

   **On Error:** Runs the mapping only if the process completes with errors.

## Set Operation Operator

Set operations combine the results of two component queries into a single result.

While a Joiner operator combines separate rows into one row, Set Operation operators combine all data rows into one output rowset using one of the various set operation conditions. In Set Operation operators, although the data is added to one output, the column lists are not mixed together to form one combined column list.

The Set Operation operator enables you to use following set operations in a mapping:

- Union (default)

- Union All
- Intersect
- Minus

By default, the Set Operation operator contains two input groups and one output group. You can add input groups by using the operator editor. The number of attributes in the output group matches the number of attributes in the input group containing the most number of attributes.

To use the Set Operation operator, all sets must have the same number of attributes and the data types of corresponding attributes must match. Corresponding attributes are determined by the order of the attributes within an input group. For example, attribute 1 in input group 1 corresponds to attribute 1 in input group 2.

You must apply the set operation in top-down order. The order of the input groups determines the execution order of the set operation. This order only affects the minus operation. For example, A minus B is not the same as B minus A. The order of the attributes within the first input group determines the structure of a set. For example, {empno, ename} is not the same as {ename, empno}.

**To use the Set Operation operator in a mapping:**

1. Drag and drop a **Set Operation** operator onto the Mapping Editor canvas.
2. Connect source attributes to the Set Operation operator groups.
3. Select the Set Operation operator header.

   The Property Inspector displays the properties of the Set Operation operator.
4. Click the list on the **Set Operation** property and select an operation from the list.
5. Connect the Set Operation output group to a target input group.

### Synchronizing the Attributes in a Set Operation Operator

The Set Operation operator in the Mapping Editor assists you in matching attributes between two data streams. To match attributes from two data streams in a mapping, define the data streams as input groups into the Set Operation operator. On the Input Attributes tab, click **Synchronize from <Input Group Name>**. The synchronize operation rearranges and adds attributes to the target group such that the target group most closely matches the source group. The synchronize operation uses the following rules to find or create a match in the target:

1. Looks for an existing attribute in the target that matches name and data type.
2. Looks for an existing attribute in the target whose description matches the source name, and the data type matches source data type.
3. If (1) and (2) fail, then a new attribute is created with the source name and data type, and is inserted in the correct matching position. Any unmatched target group attributes are indicated by `UNMATCHED` in the attribute description.

   To force a target attribute to match a specified source attribute, type the source group attribute as the target attribute description.

## Sorter Operator

You can produce a sorted row set using the Sorter operator.

The Sorter operator enables you to specify which input attributes are sorted and whether the sorting is performed in ascending or descending order. Warehouse Builder sorts data by placing an ORDER BY clause in the code generated by the mapping.

The Sorter operator has one input/output group. You can use the Sorter operator to sort data from any relational database source. You can place any operator after the Sorter operator.

## Order By Clause

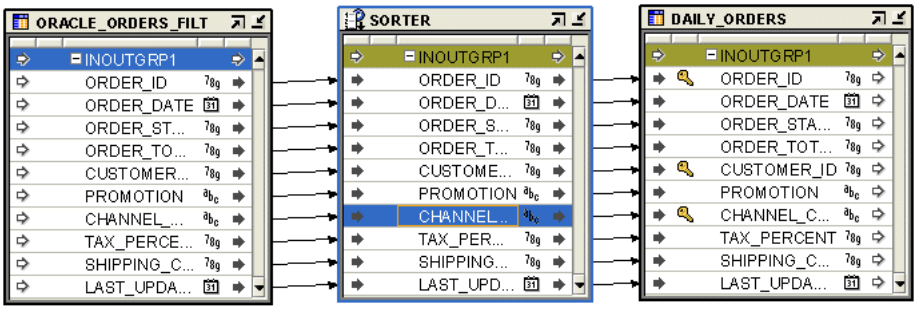
The Sorter operator contains the Order By clause. This clause is an ordered list of attributes in the input/output group to specify that sorting is performed in the same order as the ordered attribute list. You can set ascending or descending sorting for each attribute.

Most data in warehouses is loaded in batches. There can be some problems with the loading routines. For example, a batch of orders might contain a single order number multiple times with each order line representing a different state of the order. The order might have gone from status 'CREATED' to 'UPDATED' to 'BOOKED' during the day.

Because a SQL statement does not guarantee any ordering by default, the inserts and updates on the target table can take place in the wrong order. If the 'UPDATED' row is processed last, then it becomes the final value for the day although the result should be status 'BOOKED'. Warehouse Builder enables you to solve this problem by creating an ordered extraction query using the Sorter operator. The ORDER BY clause can use the last updated attribute. This will ensure that the records appear in the order in which they were created.

Figure 26–18 shows a mapping that uses the Sorter operator to sort the records from the `ORACLE_ORDERS` table. Use the Order By Clause property of the Sorter operator to sort the input records on the `ORDER_ID` and the `LAST_UPDATED` attributes.

***Figure 26–18 Sorter Operator in a Mapping***




**To use the Sorter operator in a mapping:**

1. Drag and drop the **Sorter** operator onto the Mapping Editor canvas.
2. Connect a source operator group to the Sorter input/output group.
3. Select the Sorter operator header.

   The Property Inspector displays the properties of the operator.
4. Click the Ellipsis button in the Order By Clause field.

   The Order By Clause dialog box is displayed.

5. Select the attributes you want to sort.

   Select an attribute from the Available Attributes list and click the right arrow button. Or, click the double right arrow button to select all of the Available Attributes.

6. Apply an ORDER BY clause to the attribute.

   Select the attribute in the ORDER BY Attributes list and select **ASC** (ascending) or **DESC** (descending) from the **ASC/DESC** list.

7. Click **OK.**

8. Connect the output of the Sorter operator to the target.

## Splitter Operator

You can use the Splitter operator to split data from one source to several targets.

The Splitter operator splits a single input row set into several output row sets using a boolean split condition. Each output row set has a cardinality less than or equal to the input cardinality. This is useful when you want to move data to different targets based on a data driven condition. Instead of moving the data through multiple filters, you can use a splitter.

As an option, you can optimize mappings that split data from one source to multiple targets for improved performance. For more information, see "Example: Creating Mappings with Multiple Targets" on page 26-40.

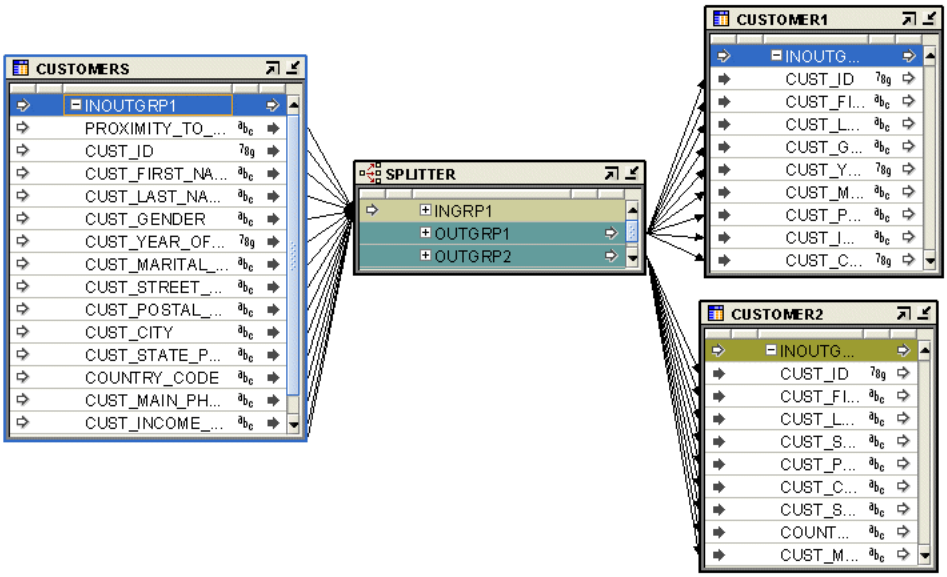
The Splitter operator contains one input group and three output groups. The output groups are OUTGRP1, OUTGRP2, and REMAINING_ROWS. You can create additional output groups, if required. You can delete the REMAINING_ROWS output group, but you cannot edit it.

In most cases, the output group REMAINING_ROWS contains all input rows that are not included in any output group. However, when the split condition contains an attribute whose value is null, the corresponding rows are not moved to the REMAINING_ROWS output group.

The Splitter operator contains the split condition. For code generation, the source columns are substituted by the input attribute names in the expression template. The expression is a valid SQL expression that can be used in a WHERE clause.

Figure 26–19 shows the mapping that uses the Splitter operator to split customer data from the source table `CUSTOMERS` into two separate tables. One table contains only the customer addresses and the other table contains the remaining customer details. Use the Split Condition property of each output group in the Splitter operator to specify which data should be moved to a particular target table.

*Figure 26–19 Splitter Operator in a Mapping*




**To use the Splitter operator in a mapping:**

1. Drag and drop the **Splitter** operator onto the Mapping Editor canvas.
2. Connect a group from a source operator to the input group of the Splitter operator.

   The output attributes are created with data types matching the corresponding input data types.
3. Select the output group of the Splitter operator.

   The Property Inspector displays the properties of the output group.
4. Click the Ellipsis button to the right of the Split Condition field.

   The Expression Builder dialog box is displayed.
5. Define the split condition.

   For example, the split condition can be UPPER(INGRP1.OR_CHANNEL) = 'DIRECT'.
6. Define expressions for the split condition of each output group except the REMAINING ROWS group.
7. Connect the output groups to the targets.

## Example: Creating Mappings with Multiple Targets

When you design a mapping with multiple targets, you have the option to optimize for improved performance. You may decide to not optimize if you require accurate auditing details for the mapping. If you decide to not optimize, then separate insert statements for each target are generated.

To optimize a multiple target mapping, you must take additional steps to generate a single insert statement for all targets combined. In this case, a multitable INSERT SQL statement is generated that takes advantage of parallel query and parallel DML services available in versions 9*i* and higher of the Oracle Database server.

**To optimize a mapping with multiple targets:**

1. Define a mapping in an Oracle target module configured to generate Oracle 9*i* or higher SQL.

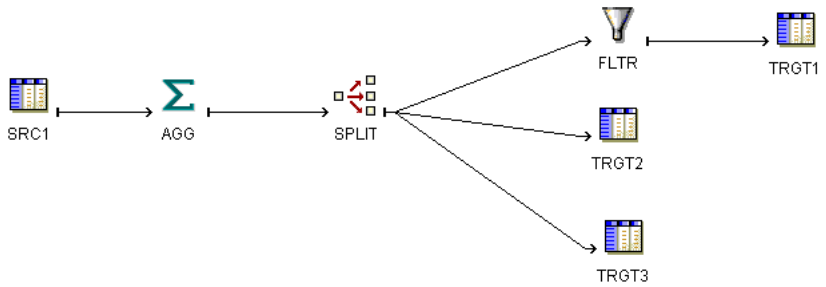
   Right-click the target module on the Projects Navigator and select **Configure.** Under **Deployment System Type** and **PL/SQL Generation Mode,** select Oracle 9*i* or higher.

2. In the Mapping Editor, design a mapping with a single source, a Splitter operator, and multiple targets.

   For the mapping to be optimized, the targets must be tables, not views or materialized views. Each target table must have less than 999 columns. Between the Splitter operator and the targets, do not include any operators that change the cardinality.

   For example, you can place a Filter between the Splitter and the targets as shown in Figure 26–20, but not a Joiner or Aggregator operator. These restrictions only apply if you choose to optimize the mapping.

***Figure 26–20 Example Mapping with Multiple Targets***




3. From the Projects Navigator, select the mapping and select **Design** from the menu bar, and select **Configure.** You can also right-click the mapping you want to configure and select **Configure.**

   Warehouse Builder displays the configuration properties dialog box for a mapping.

4. Expand **Runtime Parameters** and set **Default Operating Mode** to **Set based**.

5. Expand **Code Generation Options** and set **Optimized Code** to **True**.

When you run this mapping and view the generation results, one total SELECT and INSERT count for all targets is returned.

## Subquery Filter Operator

The Subquery Filter operator enables you to filter rows based on the results of a subquery. The conditions that you can use to filter rows are EXISTS, NOT EXISTS, IN, and NOT IN.

For example, the EMP table contains employee data. You can use a subquery to fetch a set of records from another table and then filter rows from the EMP table by using one of the conditions EXISTS, NOT EXISTS, IN, or NOT IN.

The Subquery Filter operator contains one input group INGRP1 and one Input/Output group INOUTGRP1. INGRP1 is mapped from the object that represents the subquery used to filter source data. The default condition used for filtering data is

EXISTS, which is indicated by an "E" displayed to the left of INGRP1. You can change this condition based on your requirement. The group INOUTGRP1 is mapped from the source data set that needs to be filtered. The filtered data is available as an output of this group.

**To create a mapping with a Subquery Filter operator:**

1. Drag and drop a Subquery Filter operator onto the mapping canvas.
2. Connect the source attributes that you want to filter to the Input/Output group INOUTGRP1 of the Subquery Filter operator.
3. Connect the required attributes from the object that you want to use as a subquery to the group INGRP1 of the Subquery Filter operator.
4. Select the Subquery Filter operator header.

   The Property Inspector displays the properties of the Subquery Filter operator.

   If the Property Inspector is not visible in the Design Center, select **Property Inspector** from the View menu.
5. In Subquery Filter Input Role field, select the condition that you want to use to filter input rows. The available options are: Exists, In, Not Exists, or Not In.
6. If a filter condition is required, click the Ellipsis button on the Subquery Filter Condition field to display the Expression Builder. Use this interface to specify the filter condition used to compare the input rowset and the rowset returned by the subquery.

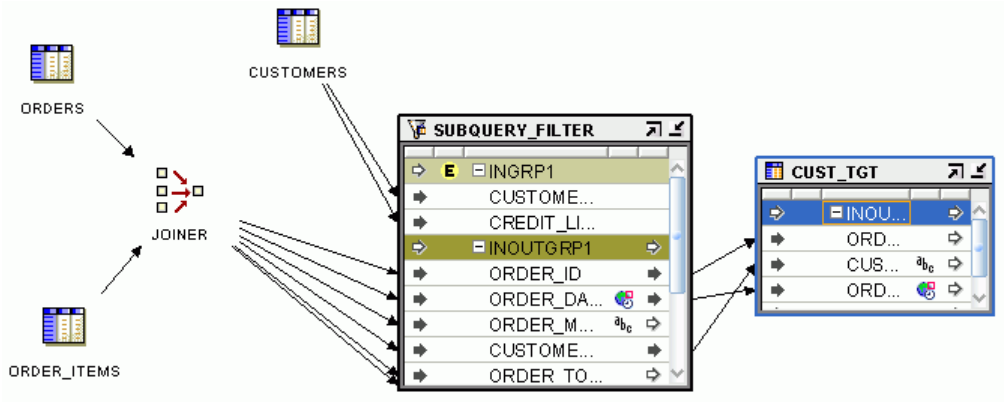
   A filter condition is not required if the input role is IN or NOT IN. For filter roles EXISTS and NOT EXISTS, you must specify a filter condition that relates the source query to EXISTS filtering subquery.
7. If the input role is IN or NOT IN, edit the properties of each input attribute and select the matching attribute from the Input/Output group of the subquery. This relates an input attribute from the source input query to each subquery group attribute, thus relating the source query to the filtering subquery.
8. Connect the Input/Output group of the Subquery Filter operator to the target.

Figure 26–21 displays a simple example of a Subquery Filter operator. In this mapping, orders data relating to customers whose credit limit is above a certain value is loaded into a target table called CUST_TGT. Order data is stored in the ORDERS and ORDERS_ITEMS tables. A Joiner operator is used to combine orders data from these tables. Use a Subquery Filter operator to filter order data based on the results of a subquery on the CUSTOMERS table.

The attributes, in the CUSTOMERS table, required for comparison are mapped to the input group of the Subquery Filter operator. The orders data, represented by the result of the Joiner operator, are mapped to the Input/Output group of the Subquery Filter operator. The Subquery Filter Condition property of the Subquery Filter operator is set to represent the condition used compare rows. In this example, the following condition was specified for the Subquery Filter Condition:

```
INGRP1.CUSTOMER_ID  =  INOUTGRP1.CUSTOMER_ID  AND  INGRP1.CREDIT_LIMIT  >= 75000
```

*Figure 26–21 Subquery Filter Operator in a Mapping*




Following is the code generated by Warehouse Builder for the Subquery Filter operator used in the mapping displayed in Figure 26–21.

```
SELECT
 "ORDERS"."ORDER_ID" "ORDER_ID",
  "ORDERS"."CUSTOMER_ID" "CUSTOMER_ID",
  "ORDERS"."ORDER_DATE" "ORDER_DATE"
FROM
     "OE"."ORDERS"@"ORA11@OE_SRC_LOCATION"  "ORDERS"
   JOIN
     "OE"."ORDER_ITEMS"@"ORA11@OE_SRC_LOCATION"  "ORDER_ITEMS" ON
          ( ( "ORDERS"."ORDER_ID" = "ORDER_ITEMS"."ORDER_ID" ) )
WHERE
  (EXISTS
    (SELECT  1
     FROM "OE"."CUSTOMERS"@"ORA11@OE_SRC_LOCATION"  "CUSTOMERS"
     WHERE
       ( "CUSTOMERS"."CUSTOMER_ID"  =  "ORDERS"."CUSTOMER_ID" )  AND
          ( "CUSTOMERS"."CREDIT_LIMIT"  >= 75000)
    )
  );
```

## Table Function Operator

Use Table Function operators to represent a table function in a mapping. Table function operators enable you to manipulate a set of input rows and return another set of rows of the same or different cardinality.

While a regular function only works on one row at a time, a table function enables you to apply the same complex PL/SQL logic on a set of rows and increase your performance. Unlike conventional functions, table functions can return a set of output rows that can be queried like a physical table.

The execution of the table function can also be parallelized where the returned rows are streamed directly to the next process without intermediate staging. Rows from a collection returned by a table function can also be pipelined or output one by one, as they are produced, instead of being output in a batch after processing of the entire table function input is completed.

Using table functions can greatly improve performance when loading your data warehouse.

A Table Function operator contains one input group and one output group.

**To define a Table Function operator in a mapping:**

1. Drag and drop a Table Function operator onto the canvas.

   The Add Table Function Operator dialog box is displayed.

2. Use an existing table function to transform data by choosing **Select from existing repository object and bind** and then selecting the table function from the tree.

   The Table Function operator is added to the canvas. The input group INGRP1 contains the input parameters defined for the table function.

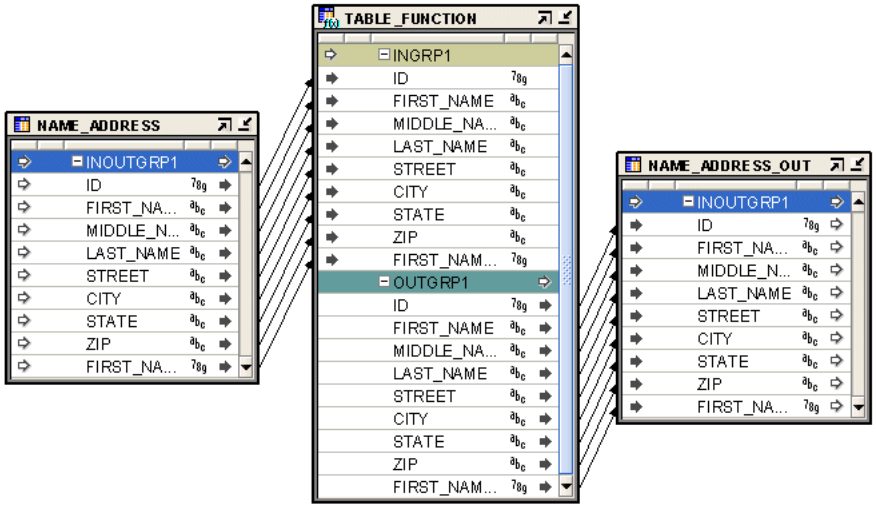
3. Map the operator that contains the input to the used by the table function to the parameters in the input group.

   Typically one or more of the input parameters are collection types. In such cases, if the source object does not contain collection types, use a Construct Object operator to create a collection type using the individual source attributes.

4. Map the Return group of the Table Function operator to the operator representing the transformation target.

Figure 26–22 shows a mapping that uses a Table Function operator to load data into a table.

***Figure 26–22 Table Function Operator in a Mapping***




### Characteristics of Table Functions

- They do not support the passing of parameters by name.
- If the return type is TABLE of PLS Record, the name you select must match the name of PLS Record field. It is possible to select only one subset of the fields of the PLS Record in the select list.
- If the return type is TABLE of T1%ROWTYPE, the name you select must match the name of the columns of the table T1.
- If the return type is TABLE of Object Type, the name you select list must match the name of Object Type attribute.
- If the return type is TABLE of Scalar (like TABLE of `NUMBER`), only Select COLUMN_VALUE can be used to retrieve the scalar values returned by the table function.

## Prerequisites for Using the Table Function Operator

Before you can use the Table Function operator in a mapping, create the table function in your target schema, external to Warehouse Builder. The table functions in the database that are supported by the unbound Table Function operator must meet the following requirements:

### Input

- Ref Cursor returning PLS Record (the fields of the PLS Record) must be supported scalar data types (0..n).
- There must be at least one input parameter.

### Output

- PLS Record (the fields of the PLS Record should be scalar data types supported by Warehouse Builder.
- Object Type (the attributes of the Object Type should be supported scalar data types).
- Supported scalar data types.
- ROWTYPE

For a Table Function operator in a mapping:

- You must add one parameter group for each ref cursor type parameter.
- Multiple scalar parameters can be part of a single scalar type parameter group.
- The parameter groups and the parameters in a group can be entered in any order.
- The positioning of the parameters in the Table Function operator must be the same as the positioning of the parameters in the table function created in your target warehouse.

## Table Function Operator Properties

You access the Table Function operator properties using the Property Inspector. The Property Inspector displays the properties of the object selected on the canvas. For example, when you select the input group of the Table Function operator, the Property Inspector displays the properties of the input parameter group.

### Table Function Operator Properties

The Table Function operator has the following properties.

**Table Function Name:** Represents the name of the table function. The name specified here must match the actual name of the table function.

**Table Function is Target:** Select this option to indicate that the table function is a target. By default, this property is selected.

**Bound Name:** Name of the table function in the repository to which the Table Function operator is bound.

### Input Parameter Properties

- **Parameter Position:** The position of the parameter in the table function signature. This property is only applicable to scalar parameters.

### Output Parameter Group Properties

- **Return Table of Scalar:** This property specifies whether the return of the table function is a TABLE of SCALAR or not. This information is required because the select list item for TABLE of SCALAR must be Select COLUMN_VALUE while in the other cases it should be an appropriate name.

### Output Parameter

- **Type Attribute Name:** The name of the field of the PLS Record, attribute of the Object Type, or column of the ROWTYPE. This property is not applicable if the return type is TABLE of SCALAR. This name is used to call the table function.

## Transformation Operator

Use the Transformation operator to transform the column value data of rows within a row set using a PL/SQL function, while preserving the cardinality of the input row set.

The Transformation operator must be bound to a function or procedure contained by one of the modules in the workspace. The inputs and outputs of the Transformation operator correspond to the input and output parameters of the bound workspace function or procedure. If the Transformation operator is bound to a function, then a result output is added to the operator that corresponds to the result of the function. The bound function or procedure must be generated and deployed before the mapping can be deployed, unless the function or procedure already exists in the target system.

Warehouse Builder provides pre-defined PL/SQL library functions in the runtime schema that can be selected as a bound function when adding a Transformation operator onto a mapping. In addition, you can choose a function or procedure from the public Oracle Custom library.

The Transformation operator contains the following properties:

- **Function Call:** The text template for the function call that is generated by the code generator with the attribute names listed as the calling parameters. For the actual call, the attribute names are replaced with the actual source or target columns that are connected to the attributes.
- **Function Name:** The name of the function or procedure, to which this operator is bound.
- **Procedure:** A boolean value indicating, if true, that the bound transformation is a procedure rather than a function with no returned value.
- **Data Type:** Indicates the data type of the input, output, or result parameter of the bound function that corresponds to the given attribute. If the output of a mapping transformation is of `CHAR` data type, then an RTRIM is applied on the result before moving the data to a target. This ensures that no extra spaces are contained in the output result.
- **Default Value:** The default value (blank if none) for the given attribute.
- **Optional Input:** A boolean value indicating, if true, that the given attribute is optional. If the attribute is optional, then it need not be connected in the mapping.
- **Function Return:** A boolean value indicating, if true, that the given output attribute is the result attribute for the function. The result attribute is a named result. Use this property if another output is a named result, or if you change the name of the result output.

**To use a Transformation operator in a mapping:**

1. Drag and drop a Transformation operator onto the Mapping Editor canvas.

   The Add Mapping Transformation dialog box is displayed.

2. Use the Add Mapping Transformation dialog box to create a new transformation or select one or more transformations. For more information about these options, see "Using the Add Operator Dialog Box to Add Operators" on page 5-15.

3. Connect the source attributes to the inputs of the Transformation operator.

4. Select an input attribute. If the Procedure property is set to True, then do not connect the input parameter.

5. Connect the Transformation operator output attributes to the target attributes.

---

**Note:** If the signature of a transformation is very long (exceeds the length of the Design Center window), you may not be able to drag and drop this transformation from the Projects Navigator onto the Mapping Editor canvas. In such cases, you can add the transformation to your mapping by using the Component Palette.

---

# Unpivot Operator

The Unpivot operator converts multiple input rows into one output row.

The Unpivot operator enables you to extract from a source once and produce one row from a set of source rows that are grouped by attributes in the source data. Like the Pivot operator, the Unpivot operator can be placed anywhere in a mapping.

## Example: Unpivoting Sales Data

Table 26–5 shows a sample of data from the `SALES` relational table. In the crosstab format, the `MONTH` column has 12 possible character values, one for each month of the year. All sales figures are contained in one column, `MONTHLY_SALES`.

***Table 26–5 Data in a Crosstab Format***

| REP | MONTH | MONTHLY_ SALES | REGION |
|---|---|---|---|
| 0675 | Jan | 10.5 | 4 |
| 0676 | Jan | 9.5 | 3 |
| 0679 | Jan | 8.7 | 3 |
| 0675 | Feb | 11.4 | 4 |
| 0676 | Feb | 10.5 | 3 |
| 0679 | Feb | 7.4 | 3 |
| 0675 | Mar | 9.5 | 4 |
| 0676 | Mar | 10.3 | 3 |
| 0679 | Mar | 7.5 | 3 |
| 0675 | Apr | 8.7 | 4 |
| 0676 | Apr | 7.6 | 3 |
| 0679 | Apr | 7.8 | 3 |

Figure 26–23 depicts data from the relational table SALES after unpivoting the table. The data formerly contained in the MONTH column (Jan, Feb, Mar...) corresponds to12 separate attributes (M1, M2, M3...). The sales figures formerly contained in the MONTHLY_SALES are now distributed across the 12 attributes for each month.

***Figure 26–23 Data Unpivoted from Crosstab Format***

| ID | Reg | M1 | M2 | M3 | M4 | M5 | M6 | M7 | M8 | M9 | M10 | M11 | M12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0675 | 4 | 10.5 | 11.4 | 9.5 | 8.7 | 7.4 | 7.5 | 7.8 | 9.7 | | | | |
| 0676 | 3 | 9.5 | 10.5 | 10.3 | 7.6 | 8.0 | 7.8 | 8.7 | 8.9 | | | | |
| 0679 | 3 | 8.7 | 7.4 | 7.5 | 7.8 | 9.7 | 10.3 | 7.6 | 8.0 | | | | |
| 0683 | 2 | 9.5 | 10.5 | 10.3 | 9.5 | 8.7 | 7.4 | 7.8 | 8.7 | | | | |
| 0684 | 1 | 11.4 | 9.5 | 8.7 | 7.4 | 7.5 | 10.3 | 9.5 | 8.7 | | | | |
| 0687 | 1 | 9.5 | 8.7 | 7.4 | 7.8 | 8.7 | 7.4 | 7.5 | 7.8 | | | | |
| 0690 | 1 | 8.7 | 7.4 | 7.8 | 8.7 | 11.4 | 9.5 | 8.7 | 7.4 | | | | |

## The Row Locator

When you use the Unpivot operator, multiple input rows are transformed into a single row based on the row locator. In the Unpivot operator, the row locator is an attribute that you must select from the source to correspond with a set of output attributes that you define. A row locator is required in an Unpivot operator. In this example, the row locator is the MONTH column from the SALES table and it corresponds to attributes M1, M2, M3... M12 in the unpivoted output.

## Using the Unpivot Operator

You have the following options for using an Unpivot operator:

- **Define a new Unpivot operator:** Drag an Unpivot operator from the Palette onto the mapping. The Mapping Editor displays a wizard.
- **Edit an existing Unpivot operator:** Right-click the Unpivot operator and select **Open Details.** The Mapping Editor opens the Unpivot Editor.

Whether you are using the Unpivot Wizard or the Unpivot Editor, complete the following pages:

- General
- Groups
- Input Connections
- Input Attributes
- Row Locator
- Output Attributes
- Unpivot Transform

### General

Use the General page to specify a name and optional description for the Unpivot operator. By default, the wizard names the operator "Unpivot".

### Groups

Use the Groups page to specify one input and one output group.

In an Unpivot operator, the input group represents the source data in crosstab format. The output group represents the target data distributed across multiple attributes.

You can rename and add descriptions to the input and output groups. Since each Unpivot operator must have exactly one input and one output group, the wizard prevents you from adding or removing groups or changing group direction.

### Input Connections

Use the Input Connections page to select attributes to copy and map into the Unpivot operator.

**To complete the Input connections page for an Unpivot operator:**

1. Select complete groups or individual attributes from the left panel.

   To search for a specific attribute or group by name, type the text in **Search for** and click **Go.** To find the next match, click **Go** again.

   Hold the Shift key down to select multiple groups or attributes. If you want to select attributes from different groups, then you must first combine the groups with a Joiner or Set operator.

2. Use the left to right arrow button in the middle of the page to move your selections to the right side of the wizard page.

   You can use the right to left arrow to move groups or attributes from the input connections list. Warehouse Builder removes the selection from the input group and removes the data flow connection between the source operator and the Unpivot operator.

### Input Attributes

Use the Input Attributes page to modify the attributes you selected in the Input Connections tab or wizard page.

You can perform the following tasks from the Unpivot Input Attributes page:

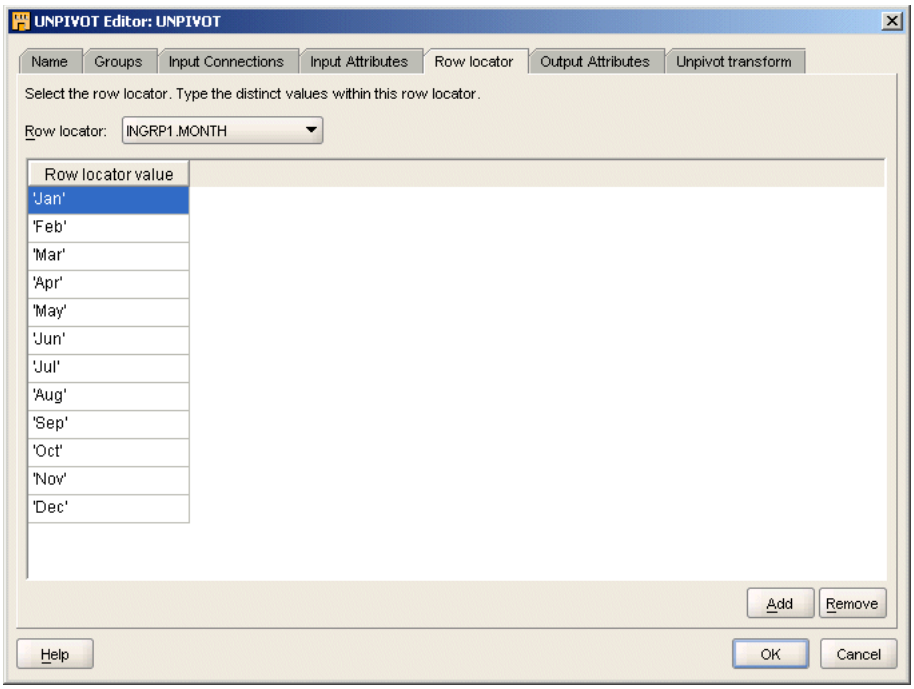
- **Add attributes:** Enter the attribute name and other attribute details in an empty row on the page.
- **Change attribute properties:** You can change the attribute name, data type, length, precision and scale.
- **Add an optional description:** Type a description for the input attributes.
- **Designate key attribute(s):** You must designate one or more key attributes for Unpivot operators. Use the **Key** check box to indicate the attribute(s) that uniquely identifies the input group. Input rows with the same value in their key attribute(s) produce one unpivoted output row.

### Row Locator

Use the Row locator page to select a row locator and assign values to the distinct values contained in the row locator.

Figure 26–24 shows the attribute `MONTH` selected as the row locator with values such as 'Jan', 'Feb', or 'Mar'.

*Figure 26–24 Unpivot Row Locator Page*




**To complete the Unpivot Row Locator page:**

1. Select an attribute from the Row locator list box.

   In the Unpivot operator, the row locator is the attribute from the source data that corresponds to a set of output attributes.

2. Use **Add** to specify the number of distinct values that exist in the row locator.

3. For each row locator value, type in the value as it appears in your source dataset.

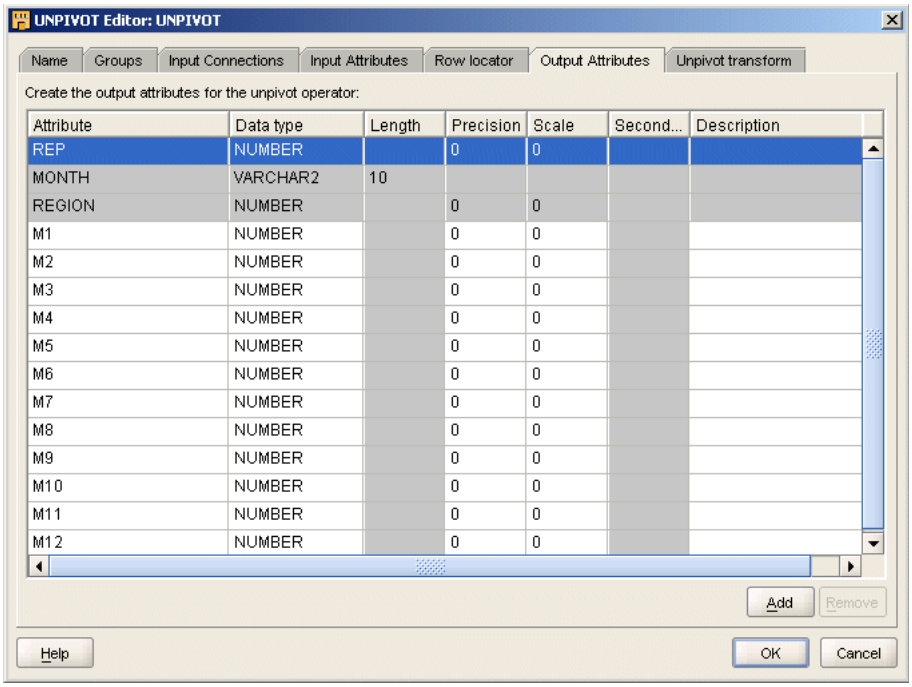
   For string values, enclose the text in single quotes. For example, if the row locator is `MONTH`, there would be a total of 12 distinct values for that attribute. Click **Add** to add a row for each distinct value. For row locator values, type values exactly as they appear in the source dataset. For instance, the row locator values as shown in Table 26–5 are 'Jan', 'Feb', and 'Mar.'

## Output Attributes

Use the Output Attributes tab to create the output attributes for the Unpivot operator.

Figure 26–25 displays the Output Attributes tab.

*Figure 26–25 Unpivot Output Attributes Page*




If you designated any input attributes as keys on the Input Attributes tab or wizard page, those input attributes are displayed as output attributes that you cannot edit or remove.

You can perform the following tasks from the Unpivot Output Attributes page:

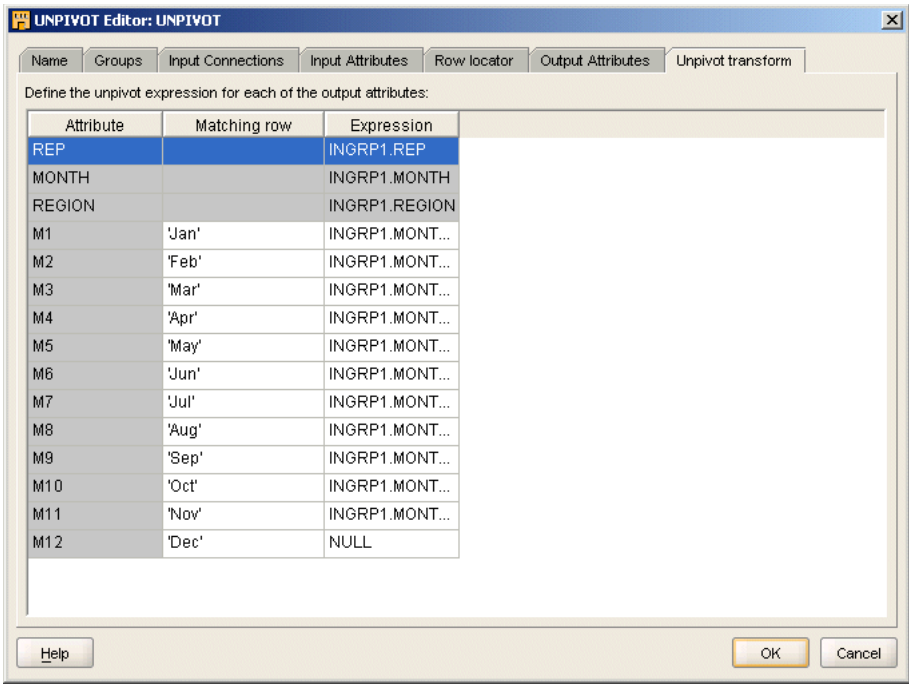
- **Add attributes:** To increase the number of output attributes to accommodate the rows you specified on the Row locator tab or wizard page, enter the attribute details in an empty cell of the page. If you specified 12 rows, then specify 12 output attributes plus attributes for any other input attributes that you did not designate as a key.
- **Change attribute properties:** Except for attributes you designated as keys on the Input Attributes tab or wizard page, you can change the attribute name, data type, length, precision, and scale.
- **Add an optional description:** Type a description for the output attributes.

## Unpivot Transform

Use the Unpivot Transform tab to write expressions for each output attribute.

Figure 26–26 displays the Unpivot Transform tab.

***Figure 26–26 Unpivot Transform Page***




For attributes you designated as keys, the matching row and expression is defined for you. Warehouse Builder displays the first row as the match for a key attribute. For all other output attributes, specify the matching row and the expression.

- **Matching row:** Select the appropriate option from the list box. For example, for the attribute you define as the first month of the year, 'M1', select 'Jan' from the list box.
- **Expression:** Select the appropriate expression from the list box. For all the new attributes you created to unpivot the data, select the same input attribute that contains the corresponding data. For example, the unpivot attributes M1, M2, M3... M12 would all share the same expression, INGRP1.MONTHLY_SALES. For all other output attributes, select the corresponding attribute from the list of input attributes.

# 27

# Activities in Process Flows

Process flows enable you to interrelate Oracle Warehouse Builder objects and external activities, such as e-mail, FTP, or operating system commands, and define flow of control between these different activities. Within a process flow, use Warehouse Builder activities to represent data objects, external objects, and control constructs. This enables you to accomplish a certain data warehouse task by create a data flow between various activities.

## Using Activities in Process Flows

Use this section as a reference for all the process flow activities. This section categorizes activities into the following types:

- Activities That Represent Objects
- Utility Activities
- Control Activities
- OS Activities

For detailed descriptions of each activity, see the alphabetical listing in the remainder of this section.

### Activities That Represent Objects

Table 27–1 lists the activities that represent objects that you previously created in Oracle Warehouse Builder. You can specify one or more incoming transitions. For outgoing transitions, you can use the success, warning, error, and unconditional transitions once each, and then also define an unlimited number of complex condition transitions.

***Table 27–1 Activities that Represent Objects***

| Icon | Activity | Brief Description |
|---|---|---|
| | Data Auditor Monitor | Adds to the process flow an existing data auditor monitor used in data profiling |
| | Mapping | Adds an existing mapping to the process flow |
| | Subprocess | Embeds an existing process flow within the process flow |

*Table 27–1 (Cont.) Activities that Represent Objects*

| Icon | Activity | Brief Description |
|---|---|---|
| | Transform | Adds an existing transformation to the process flow |
| | Web Service | Adds an existing Web service to the process flow |

### Committing Data

When you add activities that represent design objects, the process flow evaluates each of these activities as a separate transaction. For example, when you add mapping activities, the process flow commits and rolls back each mapping independently. In this design, it is not possible to control all the mappings by one commit or roll back statement.

To collectively commit or rollback multiple mappings, consider designing the process flow with a SQL*PLUS activity associated with a script that calls each mapping. For instructions, see "Committing Mappings through the Process Flow Editor" on page 10-12.

## Utility Activities

Table 27–2 lists each utility activity and shows the associated icon.

*Table 27–2 Utility Activities*

| Icon | Activity | Brief Description |
|---|---|---|
| | Assign | Assigns a value to a variable |
| | Enterprise Java Bean | Executes an Enterprise JavaBean from within a process flow |
| | Email | Sends an e-mail. For example, send an e-mail message about the status of activities in the process flow |
| | File Exists | Use the File Exists activity to check if a file is located on a specified drive or directory |
| | Java Class | Executes a Java class from within a process flow |
| | Manual | Halts a process flow and requires manual intervention to resume the process flow |
| | Notification | Sends an e-mail to a user and allows the user to select from a list of responses that dictates how the process flow proceeds |
| | OMBPlus | Represents an OMB*Plus script in a process flow |
| | Set Status | Interjects a success, warning, or error status |
| | Wait | Delays the progress of the process flow by a specified amount of time |

## Control Activities

Table 27–3 lists the activities that you use to control the process flow. The table shows the associated icon. It also lists the number of incoming and outgoing transitions allowed for each activity.

***Table 27–3 Control Activities***

| Icon | Activity | Brief Description | Incoming Transitions | Outgoing Transitions |
|---|---|---|---|---|
| | AND | Specifies the completion of all incoming activities before starting another activity | Two or more allowed. The number of incoming transitions must be less than or equal to the number of outgoing transitions from the upstream FORK. | Unconditional and complex transitions are not allowed. |
| | End (successfully) | Designates a path as being successful | One or more allowed | Not allowed |
| | End (with errors) | Designates a path as ending in errors | One or more allowed | Not allowed |
| | End (with warnings) | Designates a path as ending with warnings | One or more allowed | Not allowed |
| | End Loop | Defines the end of a For Loop or While Loop | One or more allowed | One to For Loop or While Loop only |
| | For Loop | Use this activity with an End Loop to define constructs that repeat | One from End Loop required plus more from other activities | One Loop condition and one Exit required |
| | FORK | Starts two or more activities after completing an activity | One or more allowed | Two or more unconditional transitions only |
| | OR | Starts an activity after the completion of any of two or more specified activities | Two or more allowed | One unconditional transition only |
| | Route | Defines exclusive OR and if-then-else scenarios | | |
| | While Loop | Run other activities while a condition is true | One from End Loop required plus more from other activities | One Loop condition and one Exit required |

## OS Activities

Table 27–4 lists the Operating System (OS) activities that can be initiated by a process flow.

*Table 27–4 OS Activities*

| Icon | Activity | Brief Description |
|---|---|---|
| | FTP | Starts a file transfer protocol command during a process flow. For example, use the FTP activity to move data files to the computer where a mapping runs. |
| | SQL*PLUS | Runs a SQL*Plus script in a process flow |
| | User Defined | Represents an activity that is not predefined and enables you to incorporate it into a process flow |

Because it is not desirable to allow a user have complete control over OS activities, Warehouse Builder enables you to determine which OS activities can be initiated by a process flow. This is primarily achieved by constraining the user's ability to execute operating system commands either by granting or revoking direct execution or by mandating that execution be performed through a third party, as described in "Setting a Security Constraint" on page 27-4. Further access control can be achieved by using a proxy command and parameters, which can be used to secure all executions.

This security feature is controlled by setting properties in the `Runtime.properties` file in the *$owb_home*`/owb/bin/admin` directory. This file contains Control Center property values that run the Control Center service. This file is set to read-only at Control Center service startup. If you make changes to the file, then you must restart the Control Center service for the changes to take effect.

### Setting a Security Constraint

By default, `security_constraint` for each of the OS activity commands is set to DISABLED:

```
property.RuntimePlatform.0.NativeExecution.FTP.security_constraint = DISABLED
property.RuntimePlatform.0.NativeExecution.Shell.security_constraint = DISABLED
property.RuntimePlatform.0.NativeExecution.SQLPlus.security_constraint = DISABLED
```

To enable an OS activity, you must set `security_constraint` to `NATIVE_JAVA` or `Scheduler`.

```
property.RuntimePlatform.0.NativeExecution.FTP.security_constraint = NATIVE_JAVA
property.RuntimePlatform.0.NativeExecution.Shell.security_constraint = NATIVE_JAVA
property.RuntimePlatform.0.NativeExecution.SQLPlus.security_constraint =
NATIVE_JAVA
```

`NATIVE_JAVA` allows direct execution by the Control Center service and `SCHEDULER` forces execution through `DBMS_SCHEDULER`.

### Setting a Proxy Command and Parameters

For each activity type, `USER DEFINED (Shell)`, `FTP`, and `SQLPlus`, there are two properties: the `proxy_command` property and the `proxy_parameter_list` property (optional).

If a proxy command is specified, then that command is run instead of the user's specified command and parameters. The user-specified command and parameters are passed as parameters to the proxy command following the proxy parameters. The proxy command then becomes the context in which the user's command is run.

The `proxy_command` property allows the proxy command to be specified.

To set a proxy command for the activities, set the proxy command as well as the proxy parameter list (optional) using the following command:

```
property.RuntimePlatform.0.NativeExecution.FTP.proxy_command
property.RuntimePlatform.0.NativeExecution.FTP.proxy_parameter_list
property.RuntimePlatform.0.NativeExecution.Shell.proxy_command
property.RuntimePlatform.0.NativeExecution.Shell.proxy_parameter_list
property.RuntimePlatform.0.NativeExecution.SQLPlus.proxy_command
property.RuntimePlatform.0.NativeExecution.SQLPlus.proxy_parameter_list
```

For example, to set a proxy command for Shell:

```
property.RuntimePlatform.0.NativeExecution.Shell.proxy_command = /bin/proxy_sh
property.RuntimePlatform.0.NativeExecution.Shell.proxy_parameter_list = ?-v?-n?
```

**Note:** Ideally, only the Warehouse Builder administrator must have the rights to modify the `Runtime.properties` file. The users should be granted read-only permission.

# AND

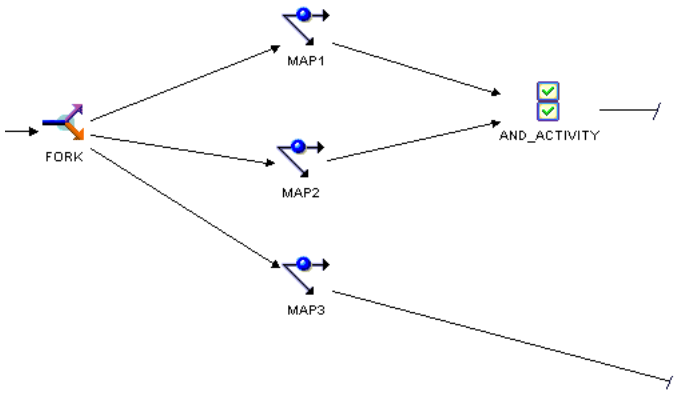
Use the AND activity to specify the completion of two or more activities before resuming the process flow.

The AND activity can have two or more incoming transitions. To correctly design process flows with an AND activity, you must place a FORK activity upstream of the AND. Also, the number of transitions going into the AND activity must be less than or equal to the number of outgoing transitions from the upstream FORK. The FORK is the only activity that enables you to assign multiple unconditional transitions and therefore ensure the completion of multiple activities as required by the AND activity.

The AND activity enables you to aggregate the outcome of the upstream activities. If all the upstream activities return SUCCESS, then the AND activity returns SUCESSES. If any upstream activity returns an ERROR, then the AND activity returns ERROR; otherwise a WARNING is returned. Any activity that does not have an outcome is considered to have returned SUCCESS. Use the SET_STATUS activity to force an outcome. The feature is particularly useful to test if a set of mappings that are running in parallel have all successfully completed.

Figure 27–1 shows the AND and FORK activities in a process flow. In this example, AND_ACTIVITY triggers downstream activities based on the completion of MAP1 and MAP2. The process flow is valid because the FORK activity has three outgoing transitions while AND_ACTIVITY has two incoming transitions. The process flow would also be valid if the transition and activities associated with MAP3 were deleted.

*Figure 27–1 AND Activity in a Process Flow*




For outgoing conditions, the AND activity can have one, two, or three conditional transitions. This results in three possible paths terminating in success, warning, and error activities.

## Assign

Use the Assign activity to assign a value to a variable. For example, use this activity to initialize a variable back to zero.

Table 27–5 describes the parameters of the Assign activity.

*Table 27–5 Assign Activity Parameters*

| Parameter | Description |
|---|---|
| Value | Enter the value to assign to the variable. |
| Variable | Select a variable that you previously defined in the editor. |

## Data Auditor Monitor

You can design process flows that proceed based on the results of profiling data. For example, you create logic that runs a mapping only if the quality of data meets a standard as determined by the threshold parameter.

Table 27–6 describes the parameters of the Data Auditor Monitor activity.

*Table 27–6 Data Auditor Monitor Activity Parameters*

| Parameter | Description |
|---|---|
| AUDIT_LEVEL | NONE<br>STATISTICS<br>ERROR_DETAILS<br>COMPLETE |
| BULK_SIZE | 1+ |
| COMMIT_FREQUENCY | 1+ |
| MAX_NO_OF_ERRORS | Maximum number of errors allowed after which the mapping terminates |

*Table 27–6 (Cont.) Data Auditor Monitor Activity Parameters*

| Parameter | Description |
|---|---|
| OPERATING_MODE | SET_BASED |
| | ROW_BASED |
| | ROW_BASED_TARGET_ONLY |
| | SET_BASED_FAIL_OVER_TO_ROW_BASED |
| | SET_BASED_FAIL_OVER_TO_ROW_BASED_TARGET_ONLY |

# Enterprise Java Bean

Use the Enterprise Java Bean activity type to call Enterprise JavaBeans (EJB) from within a process flow. EJBs are server-side components (managed by the J2EE container) that contain business logic. They enable you to create applications that are scalable, available to multiple clients, and support transactional processing.

Use the Enterprise Java Bean activity type to leverage functionality defined as an EJB within a process flow. For example, you have a suite of EJBs that implements complex business logic. You can directly integrate this business logic into a process flow by using the Enterprise Java Bean activity. Using Enterprise Java Bean activity provides better scalability, performance, and secure transactions.

> **Note:** To deploy process flows containing an Enterprise Java Bean activity, you must create a URI location that represents the J2EE platform containing the Enterprise JavaBeans.

Table 27–7 describes the parameters of the Enterprise Java Bean activity.

*Table 27–7 Enterprise Java Bean Activity Parameters*

| Parameter Name | Description |
|---|---|
| CLASS_NAME | Name of the class that implements the EJB |
| METHOD_NAME | Name of the method, within the class, that needs to be executed |
| RETURN_VALUE | String representation of the value returned by the method |
| PARAMETER_LIST | List of parameters that you want to pass to the Enterprise Java Bean |

## Example: Using an Enterprise Java Bean Activity to Leverage Existing Business Logic from EJBs

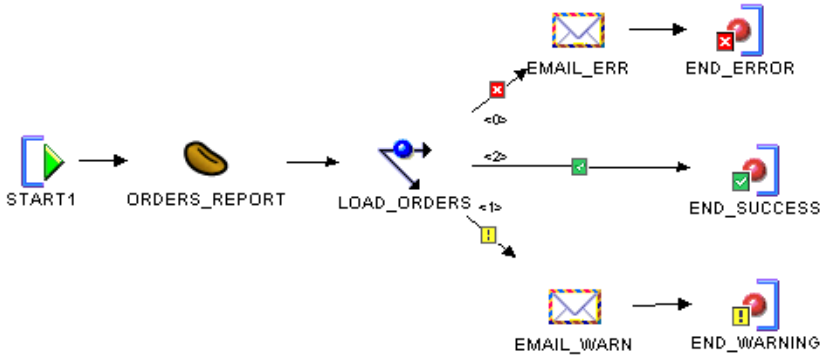
You have a suite of Enterprise JavaBeans that implement logic for an Order Processing application. You can leverage this existing functionality in a process flow. The Enterprise Java Bean activity enables you to directly integrate this functionality in your process flow.

The Order Processing application contains an EJB that produces a report of all orders made during any specified day. You want to create a process flow that produces a report of all orders made during the previous day prior to invoking ETL logic that loads this orders data into your data warehouse.

Figure 27–2 displays the process flow that provides the required functionality. The Enterprise Java Bean activity ORDERS_REPORT leverages the functionality provided

as an EJB Order Processing application. LOAD_ORDERS is a mapping that loads orders data into your warehouse after running the report.

***Figure 27–2 EJB Activity in a Process Flow***




For the ORDERS_REPORT activity, set the following parameter values:

- CLASS_NAME: ordersystem.reports
- METHOD_NAME: printReport
- PARAMETER_LIST: ReportName, PrintDevice

Create two custom parameters, ReportName and PrintDevice, in the ORDERS_REPORT activity by selecting **New Process Activity Parameter** at the top of the Structure Panel and set their values as specified in Table 27–8.

***Table 27–8 Values for Custom Parameters***

| | ReportName Parameter | PrintDevice Parameter |
|---|---|---|
| Direction | IN | IN |
| Literal | True | True |
| Value | DailyOrders | lpt1 |

Ensure that the deployed location for the Enterprise Java Bean activity is set to a URI location that points to the J2EE instance containing the application that supports the specified Enterprise Java Bean. For example, the URI location has its URI field set to:

```
ormi://myhost.example.com:23791/ReportsApp
```

## Example: Using an Enterprise Java Bean Activity to Load Data From one DB2 Table to Another

Your DB2 database contains two tables: `Orders` and `Orders_tgt`. You want to use Warehouse Builder to load data from `Orders` to `Orders_tgt`. You have a J2EE application that consists of various table utilities, one of which can be used to copy data from one table to another.

To leverage the table utility that copies data in your process flow, create an Enterprise Java Bean activity in your process flow. The parameters of the Enterprise Java Bean activity are set as follows:

- CLASS_NAME: mydb2.TableHandler
- METHOD_NAME: copyTable

- PARAMETER_LIST: SrcDataSource, TgtDataSource, SrcTable, TgtTable

Create the following custom parameters:

- `SrcDatasource`: Represents the name of the source data source
- `TgtDataSource`: Represents the name of the target data source
- `SrcTable`: Represents the source table
- `TgtTable`: Represents the target table

Table 27–9 lists the values to set for the custom parameters SrcDataSource, TgtDataSource, SrcTable, and TgtTable.

***Table 27–9 Custom Parameter Values for an Enterprise Java Bean Activity***

| | SrcDataSource | TgtDataTarget | SrcTable | TgtTable |
|---|---|---|---|---|
| Direction | IN | IN | IN | IN |
| Literal | True | True | True | True |
| Value | `OLTP_DataSource` | `Stage_DataSource` | `Orders` | `Orders_tgt` |

To deploy this process flow, create a URI location that represents the J2EE container of your table-utilities application and set the Deployed Location of the Enterprise Java Bean as described in "Example: Using an Enterprise Java Bean Activity to Leverage Existing Business Logic from EJBs" on page 27-7.

> **Note:** Because the J2EE platform provides better scalability, performance and security, Oracle recommends that you use EJBs to integrate functionality provided by Java into your process flows.

## Restrictions on Using an Enterprise Java Bean Activity

- The parameter types supported are as follows:
  - String
  - Integer
  - Float
  - Date
  - Boolean

  However, arrays of supported types (String, Integer, and so on) are not supported.
- Custom parameters with the Direction set to OUT are not supported.
- Any exceptions thrown during the execution of the Enterprise JavaBean are available only in the Repository Browser.
- You cannot perform the following actions within an Enterprise Java Bean activity:
  - Redirect input, output and error streams
  - Create and manage threads
  - Stop the Java Virtual Machine (JVM)
  - Load a native library
  - Listen on, accept connections on, or multicast from a network socket

- Directly read or write a file descriptor
- Create, modify, or delete files in the file system

# Email

You can send e-mail notifications after the completion of an activity in a process flow. You may find this useful, for example, for notifying administrators when activities such as mappings end in errors or warnings.

Table 27–10 lists the parameters that you set for the email activity.

***Table 27–10 Email Activity Parameters***

| Parameter | Description |
|---|---|
| SMTP Server | The name of that outgoing mail server. The default value is `localhost`. |
| Port | The port number for the outgoing mail server. The default value is `25`. |
| From_Address | The e-mail address from which process flow notifications are sent |
| Reply_To_Address | The e-mail address or mailing list to which recipients should respond |
| To_Address | The e-mail addresses or mailing lists that receive the process flow notification. Use a comma or a semicolon to separate multiple e-mail addresses. |
| CC_Address | The e-mail addresses or mailing lists that receive a copy of the process flow notification. Use a comma or a semicolon to separate multiple e-mail addresses. |
| BCC_Address | The e-mail addresses or mailing lists that receive a blind copy of the process flow notification. Use a comma or a semicolon to separate multiple e-mail addresses. |
| Importance | The level of importance for the notification. Select one of the following options for importance: Normal, High, or Low. |
| Subject | The text that appears in the e-mail subject line |
| Message_Body | The text that appears in the body of the email. To type in or paste text, select **Value** at the bottom of the Activity panel. The Process Flow Editor does not limit you on the amount of text that you can enter. |

For e-mail addresses, you can enter an e-mail address with or without the display name. For example, the following entries are correct:

```
jack.emp@example.com

Jack Emp<jack.emp@example.com>

Jack Emp[jack.emp@example.com]

Jack Emp[jack.emp@example.com],Jill Emp[jill.emp@example.com]

Jack Emp[jack.emp@example.com];Jill Emp[jill.emp@example.com]
```

To execute a process flow with an Email activity, you may need to access different host systems and ports. New security measures implemented in Oracle Database 11*g* Release 1 restrict access to hosts and ports. You must explicitly grant access to hosts and ports that the Email activity accesses by using the `DBMS_NETWORK_ACL_ADMIN` package.

For example, the user OWBSYS needs to send an e-mail through the mail server mail.example.com using port 25. The database administrator must perform the following steps:

1. Create an Access Control List (ACL) for the user OWBSYS by using the following command:

```
EXECUTE DBMS_NETWORK_ACL_ADMIN.CREATE_ACL
        ('acl_for_owb_cc.xml','ACL for Control Center','OWBSYS','CONNECT');
```

   The ACL has no access control effect unless it is assigned to a network target.

2. Assign the Access Control List (ACL) to a network host, and optionally specify a TCP port range. Use the following command:

```
EXECUTE DBMS_NETWORK_ACL_ADMIN.ASSIGN_ACL
        ('acl_for_owb_cc.xml','mail.example.com',25)
```

3. Commit the changes made by using the COMMIT command.

## End

Every path in the process flow must terminate in an End activity.

When you first create a process flow, a success type End activity is included by default. Use end types to indicate the type of logic contained in a path. Because a given activity such as a mapping has three possible outcomes, the editor includes three ending types, as shown in Table 27–11. You can use these ending types to design error handling logic for the process flow.

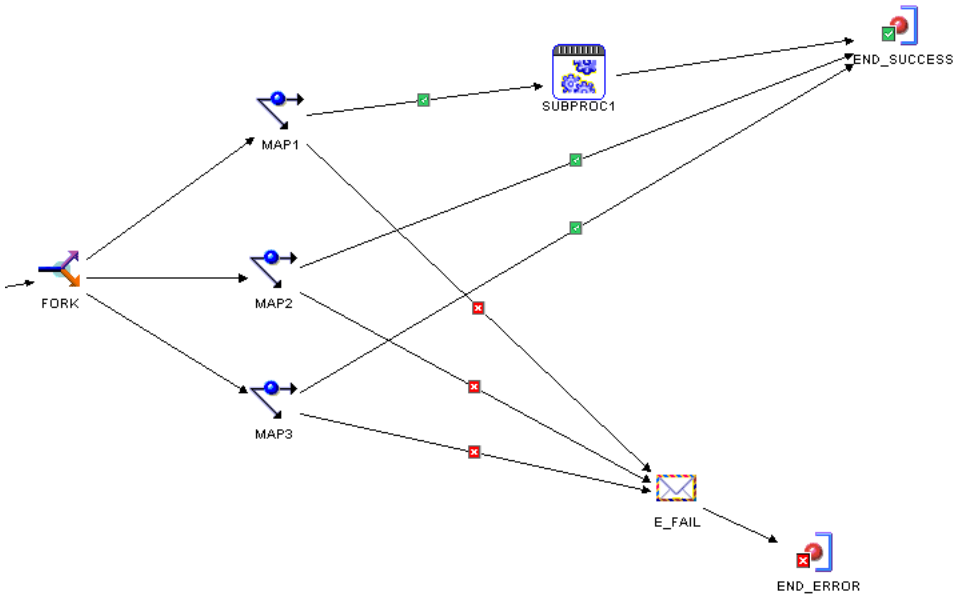
***Table 27–11 Types of End Activities***

| Icon | End Type | Description |
|---|---|---|
| | Success | Indicates that the path or paths contain logic dependent on the successful completion of an upstream activity |
| | Warning | Indicates that the path or paths contain logic dependent on an upstream activity completing with warnings |
| | Error | Indicates that the path or paths contain logic dependent on an upstream activity completing with errors |

You can design a process flow to include one, two, or all three types of endings. You can use each ending type only once, Duplicate ending types are not allowed. Each End activity can have a single or multiple incoming transitions.

In Figure 27–3, END_SUCCESS has three incoming transitions, each dependent on the successful completion of upstream activities. END_ERROR has one incoming transition from an Email activity that runs when any of the upstream mapping activities completes with errors.

*Figure 27–3 End Activities in a Process Flow*




By default, every process flow includes an END_SUCCESS. Although you cannot change an End activity to another type, you can add different types of End activity.

**To add end activities to a process flow:**

1. From the palette on the Process Flow Editor, drag and drop the desired End icon onto the canvas.

   Warehouse Builder does not allow you to select ending types already present in the process flow.

2. Click **OK.**

   Warehouse Builder adds the End activity or activities to the canvas.

## End Loop

The editor adds an End Loop for each For Loop and While Loop that you add to the canvas.

The End Loop activity must have a single unconditional outgoing transition to its For Loop or While Loop activity. All the flows that are part of the loop must converge on the End Loop activity to ensure that no parallel flows remain for either the next loop interaction or the exit of the loop.

## File Exists

Use the File Exists activity to verify the existence of a file before running the next activity. In the Activities panel, enter the name of the file.

The File Exists activity checks only once. If the file exists, then the process flow proceeds with the success transition. If the file does not exist, then the process flow proceeds with the warning transition. The File Exists activity triggers the error transition only in a catastrophic failure such as a Tcl error when using OMB*Plus.

The File Exists activity has one parameter called PATH. Specify a fully qualified file name, a directory name, or a semicolon-separated list for this parameter. The paths are normally tested in the same host that is running the Control Center service.

The security constraints of the underlying operating system may disallow access to one or more files, giving the impression that they do not exist. If all the paths exist, then the activity returns EXISTS. If none of the paths exist, then the activity returns MISSING. If some paths exist, then the activity returns SOME_EXIST.

## FORK

Use the FORK activity to start multiple, concurrent activities after the completion of an activity.

You can assign multiple incoming transitions to a FORK activity. The FORK activity is the only activity that enables you to assign multiple unconditional outgoing transitions for parallel process.

For example, in Figure 27–4, the process flow carry out the activities named FTP, FDS, and EMAIL in parallel after completing MAP1.

***Figure 27–4 FORK Activity Ensures Parallel Process***

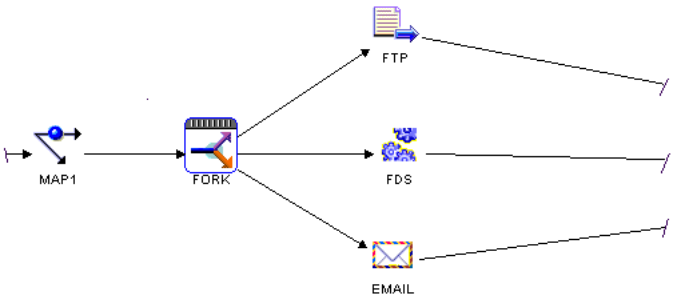
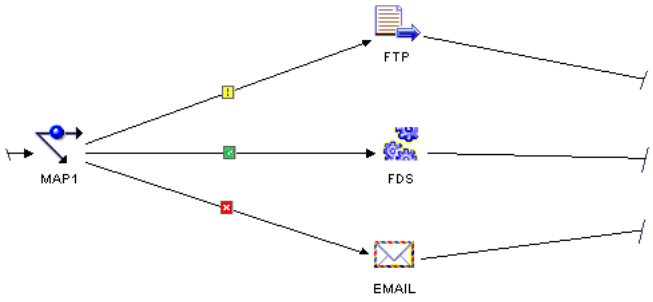



Figure 27–5 shows the same activities without the FORK activity. In this case, only one of the activities runs based on the completion state of MAP1.

***Figure 27–5 Absence of FORK Activity Results in Conditional Process***




The Process Flow Editor does not limit the number of outgoing transitions or concurrent activities that you can assign from a FORK. When you are designing for concurrent execution, design the FORK based on limitations imposed by the workflow engine or server that you use to run the process flow.

The outgoing FORK activity transition cannot have complex expressions.

# For Loop

Use the For Loop to repeatedly run activities that you include in the loop and then exit and resume the process flow.

When you add a For Loop activity, the editor also adds an End Loop activity and a transition to the End Loop. For outgoing transitions, define one with a loop condition and one with an exit condition. Select an outgoing transition and click Condition in the object details.

Table 27–12 describes the parameters of the For Loop activity.

***Table 27–12 For Loop Activity Parameters***

| Parameter | Description |
|---|---|
| Condition | An expression which when evaluated to true runs the loop transition; otherwise it runs the exit transition |
| Variable | Bound to a variable or parameter, its value is incremented every iteration. |
| Initial_Value | The initial value of the variable on entering the loop. By default, you must enter an expression. |
| Next_Value | The next value of the variable. By default, you must enter an expression. |

# FTP

Use the FTP activity to transfer files from one file location to another based on a script of FTP commands that you provide. The FTP activity is a specialization of the User Defined activity. The difference between these two is that the FTP activity should be configured with the remote file location.

For the process flow to be valid, the FTP commands must involve transferring data either from or to the server with the Control Center Service installed. To move data between two computers, neither of which hosts the Control Center Service, first transfer the data to the Control Center Service host computer and then transfer the data to the second computer.

Before you design a process flow with an FTP activity, ensure that the sources and destinations have defined locations.

The FTP activity relies on a script of FTP commands that you provide. You have a choice of either writing that script within Warehouse Builder or directing Warehouse Builder to a file containing the script. Choose one of the following methods:

- Writing a Script Within Warehouse Builder
- Calling a Script Outside of Warehouse Builder

## Writing a Script Within Warehouse Builder

Choose this method when you want to maintain the script of FTP commands in Warehouse Builder or when password security to servers is a requirement.

For this method, in the COMMAND parameter of the FTP activity, enter the path to the FTP executable. The parameters for the FTP parameter are displayed in the Structure tab of the Design Center. Also, for file transfer protocols other than UNIX, enter additional parameters for the protocol in the PARAMETER_LIST parameter. Enter a script in the VALUE property of the SCRIPT parameter.

Table 27–13 lists the parameters that you set for the FTP activity when writing the script within Warehouse Builder.

***Table 27–13 FTP Activity Parameters for a Script in Warehouse Builder***

| Parameter | Description |
|---|---|
| COMMAND | Enter the path to the file transfer protocol command such as `c:\WINNT\System32\ftp.exe` for Windows operating systems. |
| PARAMETER_LIST | This is a list of parameters that will be passed to the command. Parameters are separated from one another by a token. The token is taken as the first character on the parameter list string, and the string must also end in that token. Warehouse Builder recommends the '?' character, but any character can be used. For example, to pass 'abc,' 'def,' and 'ghi' you can use the following equivalent:<br>?abc?def?ghi?<br>or<br>!abc!def!ghi!<br>or<br>\|abc\|def\|ghi\|<br>If the token character or '\' needs to be included as part of the parameter, then it must be preceded with '\'. For example '\\'. If '\' is the token character, then '/' becomes the escape character.<br>Enter any additional parameters necessary for the file transfer protocol.<br>For Windows, enter *?"-s:${Task.Input}"?* The ${Task.Input} token prompts Warehouse Builder to store the script in a temporary file and replaces the token with the name of the temporary file. The script is therefore not passed on as standard input.<br>Note: The -s parameter is set for the Windows FTP command because it cannot be used with standard input except from a file.<br>For UNIX, you should leave this value blank. In general, UNIX FTPs read from standard input and therefore do not require any other parameters. |
| RESULT_CODE | An integer output of the activity type that indicates if the activity completed successfully. |
| SUCCESS_THRESHOLD | Designates the FTP command completion status.Enter the highest return value from the operating system that indicates a successful completion. When the operating system returns a higher value, it indicates that the command failed.<br>The default value is `0`. |
| SCRIPT | You can type the required script for FTP in this parameter.<br>To enter or paste text, select the SCRIPT parameter in the Structure tab and, in the Property Inspector, click the arrow on the property **Value**. The Edit Property dialog box is displayed, in which you enter the script. The Process Flow Editor does not limit the amount of text you can enter.<br>Each carriage return in the script is equivalent to pressing the Enter key. The script should end with bye or quit followed by a carriage return to ensure that the FTP command is terminated. |

The following is an example script that is entered in the Value property of the SCRIPT parameter in an FTP activity.

```
open ${Remote.Host}
${Remote.User}
```

```
${Remote.Password}
lcd ${Working.RootPath}
cd ${Remote.RootPath}
get salesdata.txt
quit
```

Notice that the example script includes ${Remote.User} and ${Remote.Password}. These are substitution variables. See "Using Substitution Variables" on page 27-16 for more details.

## Using Substitution Variables

Substitution variables are available only when you write and store the FTP script in Warehouse Builder.

Use substitution variables to prevent having to update FTP activities when server files, accounts, and passwords change. For example, consider that you create 10 process flows that utilize FTP activities to access a file on *salessrv1* under a specific directory. If the file is moved, without the use of substitution variables, then you must update each FTP activity individually. With the use of substitution variables, you need only update the location information.

Substitution variables are also important for maintaining password security. When an FTP activity is run with substitution variables for the server passwords, it resolves the variable to the secure password that you entered for the associated location.

Table 27–14 lists the substitute variables that you can enter for the FTP activity. *Working* refers to the computer hosting the Control Center Service, the *local* computer in this case study. Remote refers to the other server involved in the data transfer. You designate which server is remote and local, when you configure the FTP activity, as described in "Configuring Process Flows Reference" on page 24-15.

***Table 27–14 Substitute Variables for the FTP Activity***

| Variable | Value |
|---|---|
| ${Working.RootPath} | The root path value for the location of the Control Center Service host |
| ${Remote.Host} | The host value for the location involved in transferring data to or from the Control Center Service host |
| ${Remote.User} | The user value for the location involved in transferring data to or from the Control Center Service host |
| ${Remote.Password} | The password value for the location involved in transferring data to or from the Control Center Service host |
| ${Remote.RootPath} | The root path value for the location involved in transferring data to or from the Control Center Service host |
| ${Task.Input} | The Working and Remote location are set for the FTP activity when configuring a Process Flow. |
| ${parameter_name} | The values of custom parameters can be substituted into the script and parameter using ${parameter_name} syntax. |

All custom parameters are imported into the command's environment space. For example, by defining a custom parameter called PATH it is possible to change the search path used to locate operating system executables (some JAVA VMs may prevent this).

## Calling a Script Outside of Warehouse Builder

If password security is not an issue, you can direct Warehouse Builder to a file containing a script including the FTP commands and the user name and password.

To call a file on the file system, enter the appropriate command in PARAMETERS_LIST to direct Warehouse Builder to the file. For a Windows operating system, enter the following:

```
?"-s:<file path\file name>"?
```

For example, to call a file named *move.ftp* located in a temp directory on the C drive, enter the following:

```
?"-s:c:\temp\move.ftp"?
```

Leave the SCRIPT parameter blank for this method.

Table 27–15 lists the parameters that you set for the FTP activity when the FTP script resides in a file on your system.

***Table 27–15 FTP Activity Parameters for Script Outside of Warehouse Builder***

| Parameter | Description |
|---|---|
| Command | Leave this parameter blank. |
| Parameter List | Enter the path and name of the file for the FTP script. The Process Flow Editor interprets the first character that you type to be the separator. For example, the Process Flow Editor interprets the following entry as two parameters, `/c` and `dir`:<br>`?/c?dir?`<br>Use the backslash as the escape character. For example, the Process Flow Editor interprets the following entry as three parameters: `-l` and `-s` and `/`.<br>`/-l/-s/\//` |
| RESULT_CODE | An integer output of the activity type that indicates if the activity completed successfully. |
| Success Threshold | Designates the FTP command completion status.Enter the highest return value from the operating system that indicates a successful completion. When the operating system returns a higher value, it indicates that the command failed.<br>The default value is `0`. |
| Script | Leave this parameter blank. |

# Java Class

Use the Java Class activity type to represent a Java class or a Java Bean within a process flow. Java Beans are reusable software components that you can manipulate visually in a builder tool.

The Java Class activity enables you to leverage functionality that was defined using Java Beans or as a Java class.

Table 27–16 describes the parameters for the Java Class activity.

***Table 27–16 Java Class Activity Parameters***

| Parameter Name | Description |
|---|---|
| CLASSPATH | Represents the `classpath` that will be specified while executing the Java Class activity. |

*Table 27–16 (Cont.) Java Class Activity Parameters*

| Parameter Name | Description |
|---|---|
| CLASS_NAME | Name of the class that you want to invoke from the process flow. |
| JAVA_OPTIONS | Represents any options to be passed to the JVM. |
| PARAMETER_LIST | Represents the parameters that you want to pass to the Java class or Java Bean. |
| RESULT_CODE | Represents the value returned by the exit code for this Java class. |
| RUN_DIRECTORY | Represents the name of the working directory when the Java Virtual Machine (JVM) is invoked. |

You do not need a special location to deploy process flows that contain a Java Class activity.

**Note:** Due to the following reasons, it is recommended not to use the Java Class activity type:

- A single Java Virtual Machine (JVM) is used to execute each activity.
- There could be security issues with passwords because Warehouse Builder does not provide a secure way to pass parameters to activities.

## Example of Using a Java Class Activity in a Process Flow

The reporting functionality described in "Example: Using an Enterprise Java Bean Activity to Leverage Existing Business Logic from EJBs" on page 27-7 can be implemented by a Java Class rather than by an EJB. In this case, you add a Java Class activity and set the following values for its parameters:

- CLASSPATH: home/reports/reports.jar
- CLASS_NAME: ordersystem.reports
- JAVA_OPTIONS: Xmx768M -DDIR=d:\\temp\\
- PARAMETER_LIST: ReportName,PrintDevice
- RUN_DIRECTORY: /home/work

ReportName and PrintDevice are custom parameters that you create with the following properties:

- Direction is set to IN for both parameters.
- Literal is set to True for both parameters.
- The value for the parameter ReportName is set to DailyOrders.
- The value for the parameter PrintDevice is lpt1.

## Example of Customizing the Java Class Activity Executable

By default, a Java Class activity is executed by an operating system process that invokes the Java executable from the Control Center Service path. You can override this by setting the following property:

```
property.RuntimePlatform.0.NativeExecution.JavaOSProcess.executa
ble
```

Set this property in the file *OWB_ORACLE_HOME*/bin/admin/Runtime.properties.

For example, use the following steps to execute the Java activities by a specific JDK.

1. In the *OWB_ORACLE_HOME*/owb directory, create my_java.sh to contain the following:

   ```
   #!/bin/sh
   echo $* >> /tmp/out.log
   /usr/local/packages/jdk14/jre/java $*
   ```

2. To the OWB_ORACLE_HOME/bin/admin/Runtime.properties file, add the following:

   ```
   property.RuntimePlatform.0.NativeExecution.JavaOSProcess.exec
   utable=/oracle/owb/my_java.sh
   ```

3. Make my_java.sh executable by the oracle user.

## Manual

Use the Manual activity to halt a process flow.

Once the process flow halts, a user must intervene via the Control Center or Repository Browser to resume the process flow.

Consider using this activity to enable you to design a process to restart or recover ETL processes.

The Manual activity is similar to the Notification activity except that it does not require you to implement Oracle Workflow and therefore does not send an email. To achieve the same results as the Notification activity without interacting with Oracle Workflow, consider using the Email activity followed by a Manual activity.

Table 27–17 describes the parameters of the Manual activity.

***Table 27–17 Manual Activity Parameters***

| Parameter | Description |
|---|---|
| Performer | The name of the person or group that can resume the process flow |
| Subject | Enter the subject of the activity |
| Text_body | Enter special instructions to be performed before resuming the process flow |
| Priority | Set a priority. The options are: 1= high, 50=medium, and 99=low. |

## Mapping

Use the Mapping activity to add an existing mapping that you defined and configured in the Mapping Editor.

You can assign multiple incoming transitions to a Mapping activity. For outgoing transitions, assign one unconditional transition or up to one of each of the unconditional transitions.

When you add a mapping to a process flow, you can view its configuration properties in the Activities panel. The Mapping activity in the Process Flow Editor inherits its

properties from the mapping in the Mapping Editor. In the Process Flow Editor, you cannot change a property data type or direction.

You can, however, assign new values that affect the process flow only and do not change the settings for the mapping in the Mapping Editor. For example, if you change the operating mode from set-based to row-based in the Process Flow Editor, the process flow runs in row-based mode. The original mapping retains set-based mode as its operating mode. To change the properties for the underlying mapping, see "Configuring Mappings Reference" on page 24-1.

If a mapping contains a Mapping Input Parameter operator, specify a value according to its data type. The Process Flow Editor expects to receive a PL/SQL expression when you add a Mapping Input Parameter operator to a mapping. If the Mapping Input Parameter is a string, then enclose the string in double quotation marks.

If you want to update a process flow with changes that you made to a mapping in the Mapping Editor, delete the Mapping activity from the process flow and add the Mapping activity again.

Table 27–18 and Table 27–19 list the different mapping parameters in PL/SQL and SQL*Loader.

Table 27–18 lists the PL/SQL mapping parameters.

***Table 27–18 Mapping parameters for PL/SQL***

| Parameter | Valid Values |
|---|---|
| AUDIT_LEVEL | NONE<br>STATISTICS<br>ERROR_DETAILS<br>COMPLETE |
| BLUK_SIZE | 1+ |
| COMMIT_FREQUENCY | 1+ |
| MAX_NO_OF_ERRORS | Maximum number of errors allowed after which the mappings will terminate with an error |
| OPERATING_MODE | SET_BASED<br>ROW_BASED<br>ROW_BASED_TARGET_ONLY<br>SET_BASED_FAIL_OVER_TO_ROW_BASED<br>SET_BASED_FAIL_OVER_TO_ROW_BASED_TARGET_ONLY |

Table 27–19 lists the SQL*Loader mapping parameters.

***Table 27–19 Mapping parameters for SQL*Loader***

| Parameter | Description |
|---|---|
| BAD_FILE_NAME | The name of the SQL*Loader "BAD" file |
| DATA_FILE_NAME | The name of the SQL*Loader "DATA" file |
| DISCARD_FILE_NAME | The name of the SQL*Loader "DISCARD"file |

# Notification

The Notification activity enables you to design a process to restart or recover ETL processes. This activity works in conjunction with Oracle Workflow. To implement notifications, you must also implement Workflow notifications in Oracle Workflow. Alternatively, you could use an Email activity followed by a Manual activity. Oracle Workflow subsystem decides how the message is sent.

To use the Notification activity, first define the parameters listed in Table 27–20. Define a conditional outgoing transition based on each response that you define. For example, if the value of response_type is *yes*, *no* and default_response is *yes*, define two outgoing transitions. Right-click each transition and select Condition to view a list of conditions. In this example, you create one outgoing transition with condition set to *yes* and another set to *no*.

***Table 27–20 Parameters for the Notification Activity***

| Parameter | Description |
|---|---|
| Performer | Enter the name of a role defined by the Oracle Workflow administrator. |
| Subject | Enter the subject of the e-mail. |
| Text_body | Enter instructions for the performer. Explain how their response affects the process flow and perhaps explain the default action if they do not respond. |
| Html_body | Use html in addition to or instead of text. Content that you enter in html_body is appended to text_body. |
| Response_type | Enter a comma-separated list of values from which the performer selects a response. Each entry corresponds to one outgoing transition from the activity. |
| Default_response | Enter the default response. |
| Priority | Set a priority for the e-mail of either 1 (high), 50 (medium), or 99 (low). |
| Timeout | The number of seconds to wait for response. If this is set, then a #TIMEOUT transition is required. |
| Response_processor | Oracle Workflow notification response processor function. For more information, see the Oracle Workflow documentation. |
| Expand_roles | Used for notification voting. Set this value to TRUE or FALSE. When set to TRUE, a notification is sent to each member of a group rather then a single shared message to the group. For more information, see the Oracle Workflow documentation. |

**Note:** Due to an Oracle Workflow restriction, only the performer, priority, timeout, and customer parameter values can be changed at runtime.

## Notification Message Substitution

Custom parameters can be added to the Notification activity to pass and retrieve data from the user through the notification. IN parameters can be substituted into the message using SQL and appropriate syntax. For example, for a custom parameter called NAME, the text `&NAME` will be replaced with the parameter's value. You will also be prompted to enter values for the OUT parameters.

# OMBPlus

Use the OMBPlus activity to represent an OMB*Plus script in a process flow. This enables you to invoke OMB*Plus while running a process flow, to perform an OMB function or invoke an expert.

This is particularly useful when you use mappings within a process flow. You no longer need to deploy maps before you start a process flow. You can now deploy them using the OMBPlus activity as part of the process flow.

For example, you create a process flow that runs two mappings, each of which loads a target table. You can now deploy the mappings as part of the process flow.

Figure 27–6 displays a mapping that provides this functionality.

***Figure 27–6 OMBPlus Activity in a Mapping***

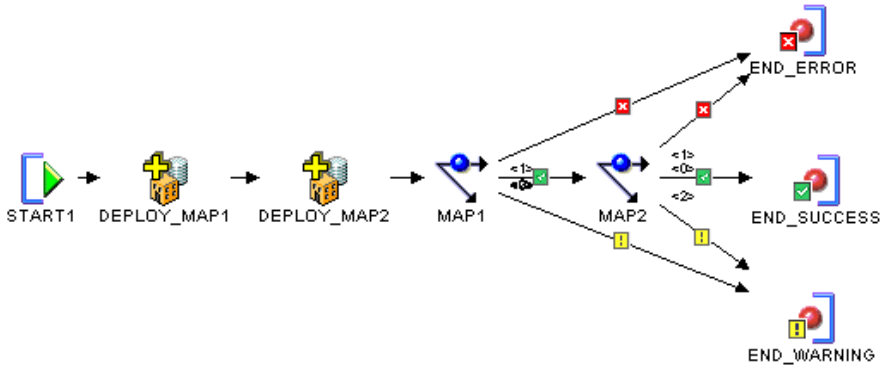



Table 27–21 describes the parameters of the OMBPlus activity.

***Table 27–21 OMBPlus Activity Parameters***

| Parameter Name | Description |
|---|---|
| PARAMETER_LIST | Defines a list of parameters, separated by a repetition of the first character.<br>For example, `/VALUE1/VALUE2/VALUE3/`, where the "/" character is used as a separator. The separator must appear at the end of the list as well as at the front. |
| RESULT_CODE | An integer output of the activity types that indicates if the activity completed successfully. |
| SCRIPT | Represents the OMB*Plus script to be executed.<br>This parameter can only be used to enter the script body that is to be executed. If you want to refer to an existing script, then specify the script in the PARAMETER_LIST. For example, `/my_script.tcl/value1/value2/`. |
| SUCCESS_THRESHOLD | Designates the OMB*Plus script completion status.Enter the highest return value from the script execution that indicates a successful completion. When a higher value is returned, it indicates that the command failed. The default value is `0`. |

You can enter an OMB*Plus script to execute or point to an existing script on the file system. To enter a script, expand the activity node in the Structure panel and select Script. In the Value field of the Property Inspector, click the Ellipsis button, enter the script in the Edit Property dialog box, and click **OK**. To point to an existing script on a

file system, go to the parameter_list parameter and enter the at sign, @, followed by the full path.

### Execution Mode for a Process Flow Containing an OMBPlus Activity

The OMBPlus activity can be run in one of several modes:

- Control Center internal (NATIVE JAVA)
- Database (Scheduler)
- Disabled

The property setting "property.RuntimePlatform.0.NativeExecution.OMBPlus.security_constraint" which is set in `owb/bin/admin/Runtime.properties` controls this behavior.

## OR

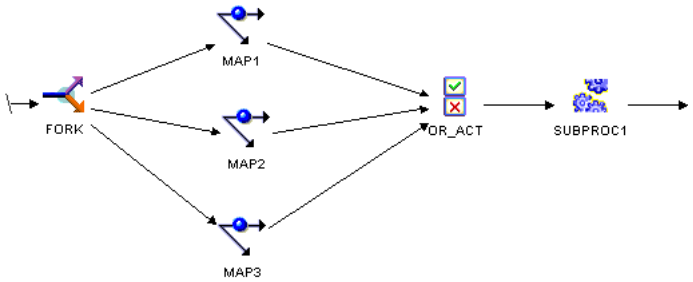
Use the OR activity to start an activity based on the completion of one or multiple number of upstream activities. You can assign multiple incoming transitions and only one unconditional outgoing transition to an OR activity.

The OR activity has similar semantics to the AND activity, except that the OR activity propagates the SUCCESS, WARNING, or ERROR outcome of the first upstream activity that is completed.

An OR activity in a process flow ensures that downstream activities are triggered only once for each run of a process flow.

Figure 27–7 displays the process flow containing an OR activity.

***Figure 27–7 The OR activity in a Process Flow***




The Process Flow Editor enables you to omit the OR activity and assign transitions from each of the three Mapping activities to Subprocess activity SUBPROC1. However, this logic would start SUBPROC1 three times within the same run of a process flow. Avoid this by using an OR activity.

## Route

Use the Route activity to route the outcome of an activity to specific results based on a condition that you define. This enables you to define exclusive OR and if-the-else scenarios.

A Route activity has no operation and therefore can be used to place a bend in a transition. Like any other activity, you can add outgoing complex condition transitions to the Route activity. But because the activity has no operation, the condition may only refer to the process flow's parameters and variables.

The inclusion of a Route activity can affect the outcome of an AND or OR activity. Because the Route activity has no outcome of its own, it will be considered to have completed as SUCCESS.

This activity does not have any parameters.

## Set Status

Use the Set Status activity to interject a success, warning, or error status.

You can use the Set Status activity as a means of overriding the behavior of the AND activity. Recall that if any of the activities immediately preceding an AND return an error, the AND activity resolves to an error. If you want the AND to resolve to success regardless of the result of a preceding activity, then insert between that activity and the AND activity a Set Status activity.

## SQL*PLUS

Use a SQL*PLUS activity to introduce a script into the process flow.

To paste or type in a script, select the activity on the canvas. In the Structure panel, expand the process flow node, then the Activities node, then the SQL*PLUS node, and select **SCRIPT**. The Property Inspector displays the properties of the Script parameter. In the **Value** field of the Property Inspector, paste or enter the script. Or, to point to an existing script on a file system, go to parameter_list and type the at sign, @, followed by the full path.

Although you can use this activity to accomplish a broad range of goals, one example is to use a SQL*PLUS activity to control how multiple mappings are committed in a process flow as described in "Committing Mappings through the Process Flow Editor" on page 10-12.

### Using SQL*PLUS Activities in Process Flows

The process flow in SQL*PLUS activity is performed by the configuration item in the Deployed Location.

To set the location that will run the SQL*PLUS activity:

1. In the Projects Navigator, expand the Process Flow module.
2. Right-click the process flow and select **Configure**.

   The Configuration tab for the process flow is displayed.
3. In the Configuration tab, expand the SQL*PLUS Activities node.
4. Select **SQLPLUS**.
5. Under Path Settings, set the **Deployed Location** option to the location that will run the SQL*PLUS activity.

The SQL*PLUS activity is similar to the User Defined activity with the following differences:

- The COMMAND parameter cannot be specified as it is automatically derived.
- If the ${Task.Input} substitution variable is used then the temporary file that is created will end in .sql.
- It has a different set of substitution variables. The activity should be configured with a Deployed database location.

*Table 27–22 SQL*PLUS Activity Parameters*

| Parameter | Description |
|---|---|
| Parameter_List | Type @ followed by the full path of the location of the file containing the script. |
| Script | As an alternative to typing the path in parameter_list, type or paste in a script. |

## Using Substitution Variables

The substitution variables are similar to FTP. It uses the following location instead of the remote location as it is connecting to an Oracle Database and not a FTP server:

- Working location as the local location
- Deployed location as the target location

*Table 27–23 SQL*PLUS Substitution Variables*

| Substitution Variable | Description |
|---|---|
| ${Working.RootPath} | The local working directory |
| ${Task.Input} | A temporary file create from the SCRIPT parameter |
| ${Target.Host} | The target location's host name |
| ${Target.Port} | The target location's post number |
| ${Target.Service} | The target location's service name |
| ${Target.TNS} | The target location's TNS address |
| ${Target.Schema} | The target location's schema name |
| ${Target.User} | The target location's user name |
| ${Target.Password} | The target location's user password |
| ${Target.URL} | The target location's connection descriptor |

If the PARAMTER_LIST is empty then one of the following parameter list is used depending on the Deployed location parameters:

- ?${Target.User}/${Target.Password}@${Target.TNS}?@${Task.Input}?
- ?${Target.User}/${Target.Password}@${Target. URL}?@${Task.Input}?
- ?${Target. Schema}/${Target.Password}@${Target.TNS}?@${Task.Input}?
- ?${Target. Schema}/${Target.Password}@${Target. URL}?@${Task.Input}?

## SQL *Plus Command

The SQL*Plus command cannot be entered directly to the FTP User Defined activities. It is either loaded from the home directory or its location is predefined by the workspace administrator.

The Sql*Plus execution location is determined from the following platform properties in the following order:

1. property.RuntimePlatform.0.NativeExecution.SQLPlus.sqlplus_exe_10g
2. property.RuntimePlatform.0.NativeExecution.SQLPlus.sqlplus_exe_9i
3. property.RuntimePlatform.0.NativeExecution.SQLPlus.sqlplus_exe_8i

4. property.RuntimePlatform.0.NativeExecution.SQLPlus.sqlplus_exe_default

The Oracle home is determined in a similar way from the following platform properties:

1. property.RuntimePlatform.0.NativeExecution.SQLPlus.oracle_home_10g
2. property.RuntimePlatform.0.NativeExecution.SQLPlus.oracle_home_9i
3. property.RuntimePlatform.0.NativeExecution.SQLPlus.oracle_home_8i
4. property.RuntimePlatform.0.NativeExecution.SQLPlus.oracle_home_default

## Start

By default, each process flow includes one Start activity. You can set input parameters for the Start activity that become the input parameters for the complete process flow.

**To add parameters to a Start activity:**

1. In the Projects Navigator, double-click the Process Flow to open the Process Flow Editor.
2. In the Structure panel, expand the Activities node.

   If the Structure tab is not displayed, select **Structure** from the View menu.
3. Select the Start activity and click **New Process Activity Parameter** (the tiny green "Plus" button at the top) on the Structure tab.

   A new parameter is added under the Start activity.
4. Select the new parameter and, in the Property Inspector, set the properties for this parameter.

   Change the parameter name and data type as necessary. You cannot alter its direction. The direction is IN, indicating that the parameter is an input parameter only. For value, type the parameter value. You can overwrite this value at runtime.
5. You can now use the parameter as input to other activities in the process flow.

## Subprocess

Use a Subprocess activity to start a previously created process flow. From one process flow, you can start any other process flow that is contained within the same or any other process flow package.

Once you add a Subprocess activity to a process flow, use it in your design in a way similar to any other activity. You can assign multiple incoming transitions. For outgoing transitions, assign either one unconditional outgoing transition or up to three outgoing conditional transitions.

The END activities within the subprocess apply to the Subprocess activity only and do not function as a termination point in the process flow.

An important difference between a Subprocess activity and other activities is that you can view the contents of a subprocess, but you cannot edit its contents in the parent process flow. To edit a subprocess, open its underlying process flow from the Projects Navigator. With the exception of renaming a process flow, the Process Flow Editor propagates changes from child process flows to its parent process flows.

**Note:** Use caution when renaming process flows. If you rename a process flow referenced by another process flow, then the parent process flow becomes invalid. You must delete the invalid subprocess and add a new subprocess associated with the new name for the child process flow.

**To add Subprocess activity to a process flow:**

1. From the palette in the Process Flow Editor, drag and drop the Subprocess activity icon onto the canvas.

   Warehouse Builder displays a dialog box to select and add a process flow as a subprocess.

2. Expand the process flow module and select a process flow from the same process flow package as the parent process flow.

   Warehouse Builder displays the process flow as a Subprocess activity on the parent process flow.

3. To view the contents of the subprocess, right-click the subprocess and select **Expand Node.**

   The Process Flow Editor displays the graph for the subprocess surrounded by a blue border.

## Transform

When a function transform is dropped onto the canvas, the return parameter is created as a new parameter with the same name as the transform. When you add transformations from the transformation library to a process flow using the Transform activity, the Process Flow Editor displays the parameters for the transformation in the Activity panel.

You can specify one or more incoming transitions to start a Transform activity. For outgoing transitions, you can either specify one unconditional transition or one of each of the three conditional transitions.

If you specify conditional outgoing transitions, you can configure the activity to base its status on its return value. For more information about **Use Return as Status,** see "Configuring Process Flows Reference" on page 24-15.

To update a process flow with changes that you made to a transformation, delete the Transform activity from the process flow and add the Transform activity again.

For transforms that are not deployed, such as the public transformations, the activity must be configured with a Deployed location value.

## User Defined

The User Defined activity enables you to incorporate into a process flow an activity that is not defined within Warehouse Builder.

You can specify one or more incoming transitions to start a User Defined process activity. For outgoing transitions, you can either specify one unconditional transition or one of each of the three conditional transitions.

If you specify conditional outgoing transitions, you can configure the activity to base its status on its return value. For more information about **Use Return as Status,** see "Configuring Process Flows Reference" on page 24-15.

Table 27–24 lists the parameters you set for the User Defined activity.

***Table 27–24 User Defined Activity Parameters***

| Parameter | Description |
|---|---|
| Command | The command to perform the user defined process that you defined. Enter the path and file name such as `c:\winnt\system32\cmd.exe`. |
| Parameter List | The list of parameters to be passed to the user defined process. Enter the path and file name such as `?/c?c:\\temp\\run.bat`.<br>The Process Flow Editor interprets the first character you type to be the separator. For example, the Process Flow Editor interprets the following entry as `/c and dir`.<br>`?/c?dir?`<br>Use the backslash as the escape character. For example, the Process Flow Editor interprets the following entry as `-l` and `-s` and `/`.<br>`/-l/-s/\//`<br>You can also enter the substitution variables listed in Table 27–25. |
| Success Threshold | Designates the completion status.Enter the highest return value from the operating system that indicates a successful completion. When the operating system returns a higher value, it indicates that the command failed. The default value is `0`. |
| Script | You can enter a script here or enter a file name for a script. If you enter a file name, then use the ${Task.Input} variable in the parameter list to pass the file name.<br>To enter or paste text, select **Value** at the bottom of the Activity panel. The Process Flow Editor does not limit the amount of text you can enter.<br>Each carriage return in the script is equivalent to pressing the **Enter** key. Therefore, end the script with a carriage return to ensure that the last line is sent. |

Table 27–25 lists the substitute variables you can enter for the FTP activity.

***Table 27–25 Substitute Variables for the User Defined Process Activity***

| Variable | Value |
|---|---|
| ${Working.Host} | The host value for the location of the Control Center Service host |
| ${Working.User} | The user value for the location of the Control Center Service host |
| ${Working.Password} | The password value for the location of the Control Center Service host |
| ${Working.RootPath} | The local working directory |
| ${Task.Input} | A temporary file created from the SCRIPT parameter<br>Enter the Task.Input variable to direct Warehouse Builder to the script that you write in the SCRIPT parameter.<br>For Windows, enter into Parameter_List *?"-s:${Task.Input}"?*<br>and for UNIX, enter into Parameter_List *?"${Task.Input}"?*<br>where the question mark as the separator. |

## Wait

Use the Wait activity to interject a delay in the process flow.

Table 27–26 describes the parameters of the Wait activity.

***Table 27–26 Wait Activity Parameters***

| Parameter | Description |
|---|---|
| Minimum_Delay | Enter the minimum time to wait. Specify the time in units of seconds. |
| Until_Date | Specify the date to wait until in the default format for your local region. |

## While Loop

Use the While Loop to run one or more activities only when a condition that you define evaluates to true.

Typically, you associate a While Loop with Assign activities that enable you to define the while condition. At least one Assign activity initializes the data and at least one Assign activity increments or modifies the data again to the end of a loop iteration.

When you add a While Loop activity, the editor also adds an End Loop activity and a transition to the End Loop. Create transitions from the While Loop activity to each activity you want to include in the loop. For each outgoing transition that you add, apply either an EXIT or LOOP condition to the transition by selecting the transition and clicking on Condition in the object details.

To define the while condition that governs whether or not to run the loop, in the Structure panel, expand the process flow node, then the Activities node, then the WHILE_LOOP node, and select **Condition**. The Property Inspector displays the parameters for the Condition.

Table 27–27 describes the parameters of the While Loop activity.

***Table 27–27 While Loop Activity Parameters***

| Parameter | Description |
|---|---|
| Condition | Define with a LOOP or EXIT condition. |

## Web Service

Use the Web Service activity to add an existing Web service to a process flow. The Web services must be defined under the Application Servers node of the Projects Navigator or the Public application Server node of the Globals Navigator.

The Web Service activity enables you use the operations defined in the Web service in your process flow. Since a Web service can contain multiple operations, when you add a Web Service activity to a process flow, you are prompted to select the operation to be used.

The parameters for a Web Service activity depend on the type of operations performed by the Web service. Thus, different operations can have different parameters. Table 27–28 describes the parameters of the runCCJob operation of the default Web service `AgentWebService`.

*Table 27–28 Parameters for Web Service Activities*

| Parameter | Description |
|---|---|
| Username | The name of the workspace user executing the process flow |
| Password | The password of the user specified in the username field |
| Workspace | The name of the workspace in which the Web service execution job should be run.<br>If the user executing the Web service is not the workspace owner, then prefix the workspace name with the user name. For example, `test_user.my_workspace`. |
| Location | The physical name of the location to which the operation is deployed |
| Task_type | The type of operation. Use one of the following values: PLSQL, SQL_LOADER, PROCESS, SAP, or DATA_AUDITOR. |
| Task_name | The physical name of the process flow. Qualify the process flow name with the name of the process flow package to which it belongs. For example, `MY_PROCESS_FLOW_PACK.MY_PROCESS_FLOW`. |
| Connection_string | The connection information of the system that runs the Control Center Manager |
| System_params | The values of the mapping execution parameters, if any, such as Bulk Size, Audit Level, or Operating Mode. |
| Custom_params | The values for the input parameters for the mapping on which the Web service is based |

**To use a Web Service activity in a process flow:**

1. Open the process flow in which you want to add Web Service activity by double-clicking the process flow in the Projects Navigator.

   The process flow is displayed in the editor.

2. From the Projects Navigator, drag and drop the Web service you want to add.

   The Web Service Operation dialog box is displayed.

3. If the Web service selected in the previous step contains more than one operation, select the operation within the Web service that you want to add to the process flow and click **OK**.

   The Web service operation is added to the process flow.

4. Set the parameters for the Web Service activity.

# 28

# Warehouse Builder Transformations Reference

This chapter describes the predefined transformations provided by Warehouse Builder to transform data.

## Predefined Transformations in the Public Oracle Predefined Library

Predefined transformations in the public Oracle Predefined library are categorized as follows:

- Administrative Transformations
- Character Transformations
- Control Center Transformations
- Conversion Transformations
- Date Transformations
- Number Transformations
- OLAP Transformations
- Other Transformations
- Spatial Transformations
- Streams Transformations
- XML Transformations

### Administrative Transformations

Administrative transformations provide prebuilt functionality to perform actions that are regularly performed in ETL processes. The main focus of these transformations is in the DBA related areas or to improve performance. For example, it is common to disable constraints when loading tables and then to reenable them after loading has completed.

The administrative transformations in Warehouse Builder are custom functions. The Administrative transformation that Warehouse Builder provides are:


- WB_ABORT on page 28-2
- WB_COMPILE_PLSQL on page 28-2
- WB_DISABLE_ALL_CONSTRAINTS on page 28-3

- WB_DISABLE_ALL_TRIGGERS on page 28-3
- WB_DISABLE_CONSTRAINT on page 28-4
- WB_DISABLE_TRIGGER on page 28-5
- WB_ENABLE_ALL_CONSTRAINTS on page 28-6
- WB_ENABLE_ALL_TRIGGERS on page 28-6
- WB_ENABLE_CONSTRAINT on page 28-7
- WB_ENABLE_TRIGGER on page 28-8
- WB_TRUNCATE_TABLE on page 28-9


## WB_ABORT

### Syntax

```
WB_ABORT(p_code, p_message)
```

where *p_code* is the abort code, and must be between -20000 and -29999; and *p_message* is an abort message you specify.

### Purpose

`WB_ABORT` enables you to terminate the application from a Warehouse Builder component. You can run it from a post-mapping process or as a transformation within a mapping.

### Example

Use this administration function to terminate an application. You can use this function in a post-mapping process to terminate deployment if there is an error in the mapping.

## WB_COMPILE_PLSQL

### Syntax

```
WB_COMPILE_PLSQL(p_name, p_type)
```

where *p_name* is the name of the object that is to be compiled; *p_type* is the type of object to be compiled. The legal types are:

```
'PACKAGE'
'PACKAGE BODY'
'PROCEDURE'
'FUNCTION'
'TRIGGER'
```

### Purpose

This program unit compiles a stored object in the database.

### Example

The following hypothetical example compiles the procedure called `add_employee_proc`:

```
EXECUTE WB_COMPILE_PLSQL('ADD_EMPLOYEE_PROC', 'PROCEDURE');
```

## WB_DISABLE_ALL_CONSTRAINTS

### Syntax

```
WB_DISABLE_ALL_CONSTRAINTS(p_name)
```

where *p_name* is the name of the table on which constraints are disabled.

### Purpose

This program unit disables all constraints that are owned by the table as stated in the call to the program.

For faster loading of data sets, you can disable constraints on a table. The data is now loaded without validation. This is mainly done on relatively clean data sets.

### Example

The following example shows the disabling of the constraints on the table `OE.CUSTOMERS`:

```
SELECT constraint_name
,      DECODE(constraint_type,'C','Check','P','Primary') Type
,      status
FROM user_constraints
WHERE table_name = 'CUSTOMERS';

CONSTRAINT_NAME                TYPE    STATUS
------------------------------ ------- --------
CUST_FNAME_NN                  Check   ENABLED
CUST_LNAME_NN                  Check   ENABLED
CUSTOMER_CREDIT_LIMIT_MAX      Check   ENABLED
CUSTOMER_ID_MIN                Check   ENABLED
CUSTOMERS_PK                   Primary ENABLED
```

Perform the following in SQL*Plus or Warehouse Builder to disable all constraints:

```
EXECUTE WB_DISABLE_ALL_CONSTRAINTS('CUSTOMERS');

CONSTRAINT_NAME                TYPE    STATUS
------------------------------ ------- --------
CUST_FNAME_NN                  Check   DISABLED
CUST_LNAME_NN                  Check   DISABLED
CUSTOMER_CREDIT_LIMIT_MAX      Check   DISABLED
CUSTOMER_ID_MIN                Check   DISABLED
CUSTOMERS_PK                   Primary DISABLED
```

> **Note:** This statement uses a cascade option to allow dependencies to be broken by disabling the keys.

## WB_DISABLE_ALL_TRIGGERS

### Syntax

```
WB_DISABLE_ALL_TRIGGERS(p_name)
```

where *p_name* is the table name on which the triggers are disabled.

### Purpose

This program unit disables all triggers owned by the table as stated in the call to the program. The owner of the table must be the current user (in variable USER). This action stops triggers and improves performance.

### Example

The following example shows the disabling of all triggers on the table `OE.OC_ORDERS`:

```
SELECT trigger_name
,      status
FROM user_triggers
WHERE table_name = 'OC_ORDERS';

TRIGGER_NAME                   STATUS
------------------------------ --------
ORDERS_TRG                     ENABLED
ORDERS_ITEMS_TRG               ENABLED
```

Perform the following in SQL*Plus or Warehouse Builder to disable all triggers on the table `OC_ORDERS`.

```
EXECUTE WB_DISABLE_ALL_TRIGGERS ('OC_ORDERS');

TRIGGER_NAME                   STATUS
------------------------------ --------
ORDERS_TRG                     DISABLED
ORDERS_ITEMS_TRG               DISABLED
```

## WB_DISABLE_CONSTRAINT

### Syntax

```
WB_DISABLE_CONSTRAINT(p_constraintname, p_tablename)
```

where *`p_constraintname`* is the constraint name to be disabled; *`p_tablename`* is the table name on which the specified constraint is defined.

### Purpose

This program unit disables the specified constraint that is owned by the table as stated in the call to the program. The user is the current user (in variable `USER`).

For faster loading of data sets, you can disable constraints on a table. The data is then loaded without validation. This reduces overhead and is mainly done on relatively clean data sets.

### Example

The following example shows the disabling of the specified constraint on the table `OE.CUSTOMERS`:

```
SELECT constraint_name
, DECODE(constraint_type
, 'C', 'Check'
, 'P', 'Primary'
) Type
, status
FROM user_constraints
WHERE table_name = 'CUSTOMERS';
```

```
CONSTRAINT_NAME                TYPE    STATUS
------------------------------ ------- --------
CUST_FNAME_NN                  Check   ENABLED
CUST_LNAME_NN                  Check   ENABLED
CUSTOMER_CREDIT_LIMIT_MAX      Check   ENABLED
CUSTOMER_ID_MIN                Check   ENABLED
CUSTOMERS_PK                   Primary ENABLED
```

Perform the following in SQL*Plus or Warehouse Builder to disable the specified constraint.

```
EXECUTE WB_DISABLE_CONSTRAINT('CUSTOMERS_PK','CUSTOMERS');

CONSTRAINT_NAME                TYPE    STATUS
------------------------------ ------- --------
CUST_FNAME_NN                  Check   ENABLED
CUST_LNAME_NN                  Check   ENABLED
CUSTOMER_CREDIT_LIMIT_MAX      Check   ENABLED
CUSTOMER_ID_MIN                Check   ENABLED
CUSTOMERS_PK                   Primary DISABLED
```

**Note:** This statement uses a cascade option to allow dependencies to be broken by disabling the keys.

## WB_DISABLE_TRIGGER

### Syntax

```
WB_DISABLE_TRIGGER(p_name)
```

where *p_name* is the trigger name to be disabled.

### Purpose

This program unit disables the specified trigger. The owner of the trigger must be the current user (in variable USER).

### Example

The following example shows the disabling of a trigger on the table `OE.OC_ORDERS`:

```
SELECT trigger_name, status
FROM user_triggers
WHERE table_name = 'OC_ORDERS';

TRIGGER_NAME                   STATUS
------------------------------ --------
ORDERS_TRG                     ENABLED
ORDERS_ITEMS_TRG               ENABLED
```

Perform the following in SQL*Plus or Warehouse Builder to disable the specified constraint.

```
ECECUTE WB_DISABLE_TRIGGER ('ORDERS_TRG');

TRIGGER_NAME                   STATUS
------------------------------ --------
```

```
ORDERS_TRG                     DISABLED
ORDERS_ITEMS_TRG               ENABLED
```

## WB_ENABLE_ALL_CONSTRAINTS

### Syntax

```
WB_ENABLE_ALL_CONSTRAINTS(p_name)
```

where *p_name* is the name of the table for which all constraints should be enabled.

### Purpose

This program unit enables all constraints that are owned by the table as stated in the call to the program.

For faster loading of data sets, you can disable constraints on a table. After the data is loaded, you must enable these constraints again using this program unit.

### Example

The following example shows the enabling of the constraints on the table OE.CUSTOMERS:

```
SELECT constraint_name
, DECODE(constraint_type
, 'C', 'Check'
, 'P', 'Primary)
Type
, status
FROM user_constraints
WHERE table_name = 'CUSTOMERS';

CONSTRAINT_NAME                TYPE    STATUS
------------------------------ ------- --------
CUST_FNAME_NN                  Check   DISABLED
CUST_LNAME_NN                  Check   DISABLED
CUSTOMER_CREDIT_LIMIT_MAX      Check   DISABLED
CUSTOMER_ID_MIN                Check   DISABLED
CUSTOMERS_PK                   Primary DISABLED
```

Perform the following in SQL*Plus or Warehouse Builder to enable all constraints.

```
EXECUTE WB_ENABLE_ALL_CONSTRAINTS('CUSTOMERS');

CONSTRAINT_NAME                TYPE    STATUS
------------------------------ ------- --------
CUST_FNAME_NN                  Check   ENABLED
CUST_LNAME_NN                  Check   ENABLED
CUSTOMER_CREDIT_LIMIT_MAX      Check   ENABLED
CUSTOMER_ID_MIN                Check   ENABLED
CUSTOMERS_PK                   Primary ENABLED
```

## WB_ENABLE_ALL_TRIGGERS

### Syntax

```
WB_ENABLE_ALL_TRIGGERS(p_name)
```

where `p_name` is the table name on which the triggers are enabled.

### Purpose

This program unit enables all triggers owned by the table as stated in the call to the program. The owner of the table must be the current user (in variable USER).

### Example

The following example shows the enabling of all triggers on the table `OE.OC_ORDERS`:

```
SELECT trigger_name
,      status
FROM user_triggers
WHERE table_name = 'OC_ORDERS';

TRIGGER_NAME                   STATUS
------------------------------ --------
ORDERS_TRG                     DISABLED
ORDERS_ITEMS_TRG               DISABLED
```

Perform the following in SQL*Plus or Warehouse Builder to enable all triggers defined on the table `OE.OC_ORDERS`.

```
EXECUTE WB_ENABLE_ALL_TRIGGERS ('OC_ORDERS');

TRIGGER_NAME                   STATUS
------------------------------ --------
ORDERS_TRG                     ENABLED
ORDERS_ITEMS_TRG               ENABLED
```

## WB_ENABLE_CONSTRAINT

### Syntax

```
WB_ENABLE_CONSTRAINT(p_constraintname, p_tablename)
```

where *`p_constraintname`* is the constraint name to be disabled and *`p_tablename`* is the table name on which the specified constraint is defined.

### Purpose

This program unit enables the specified constraint that is owned by the table as stated in the call to the program. The user is the current user (in variable `USER`). For faster loading of data sets, you can disable constraints on a table. After the loading is complete, you must reenable these constraints. This program unit shows you how to enable the constraints one at a time.

### Example

The following example shows the enabling of the specified constraint on the table `OE.CUSTOMERS`:

```
SELECT constraint_name
,      DECODE(constraint_type
       , 'C', 'Check'
       , 'P', 'Primary'
       ) Type
,      status
FROM user_constraints
WHERE table_name = 'CUSTOMERS';
```

```
CONSTRAINT_NAME                TYPE    STATUS
------------------------------ ------- --------
CUST_FNAME_NN                  Check   DISABLED
CUST_LNAME_NN                  Check   DISABLED
CUSTOMER_CREDIT_LIMIT_MAX      Check   DISABLED
CUSTOMER_ID_MIN                Check   DISABLED
CUSTOMERS_PK                   Primary DISABLED
```

Perform the following in SQL*Plus or Warehouse Builder to enable the specified constraint.

```
EXECUTE WB_ENABLE_CONSTRAINT('CUSTOMERS_PK', 'CUSTOMERS');

CONSTRAINT_NAME                TYPE    STATUS
------------------------------ ------- --------
CUST_FNAME_NN                  Check   DISABLED
CUST_LNAME_NN                  Check   DISABLED
CUSTOMER_CREDIT_LIMIT_MAX      Check   DISABLED
CUSTOMER_ID_MIN                Check   DISABLED
CUSTOMERS_PK                   Primary ENABLED
```

## WB_ENABLE_TRIGGER

### Syntax

```
WB_ENABLE_TRIGGER(p_name)
```

where `p_name` is the trigger name to be enabled.

### Purpose

This program unit enables the specified trigger. The owner of the trigger must be the current user (in variable `USER`).

### Example

The following example shows the enabling of a trigger on the table `OE.OC_ORDERS`:

```
SELECT trigger_name
,      status
FROM user_triggers
WHERE table_name = 'OC_ORDERS';

TRIGGER_NAME                   STATUS
------------------------------ --------
ORDERS_TRG                     DISABLED
ORDERS_ITEMS_TRG               ENABLED
```

Perform the following in SQL*Plus or Warehouse Builder to enable the specified constraint.

```
EXECUTE WB_ENABLE_TRIGGER ('ORDERS_TRG');

TRIGGER_NAME                   STATUS
------------------------------ --------
ORDERS_TRG                     ENABLED
ORDERS_ITEMS_TRG               ENABLED
```

### WB_TRUNCATE_TABLE

#### Syntax

```
WB_TRUNCATE_TABLE(p_name)
```

where *p_name* is the table name to be truncated.

#### Purpose

This program unit truncates the table specified in the command call. The owner of the trigger must be the current user (in variable `USER`). The command disables and reenables all referencing constraints to enable the truncate table command. Use this command in a pre-mapping process to explicitly truncate a staging table and ensure that all data in this staging table is newly loaded data.

#### Example

The following example shows the truncation of the table `OE.OC_ORDERS`:

```
SELECT COUNT(*) FROM oc_orders;

  COUNT(*)
----------
       105
```

Perform the following in SQL*Plus or Warehouse Builder to enable the specified constraint.

```
EXECUTE WB_TRUNCATE_TABLE ('OC_ORDERS');

  COUNT(*)
----------
         0
```

## Character Transformations

Character transformations enable Warehouse Builder users to perform transformations on Character objects. The custom functions provided with Warehouse Builder are prefixed with `WB_`.

The character transformations available in Warehouse Builder are listed below. Most of them are implementations of basic SQL functions or procedures. No descriptions are provided for such transformations.

Table 28–1 lists the character transformations that are based on Database SQL functions. The transformations are listed in a columnar table that reads down the columns from left to right to conserve space.

***Table 28–1 Character Transformations Based on SQL character functions***

| Character Transformation Name | Character Transformation Name (Contd.) | Character Transformation Name (Contd.) |
|---|---|---|
| ▪ ASCII | ▪ CHR | ▪ CONCAT |
| ▪ INITCAP | ▪ INSTR | ▪ INSTR2 |
| ▪ INSTR4 | ▪ INSTRB | ▪ INSTRC |
| ▪ LENGTH | ▪ LENGTH2 | ▪ LENGTH4 |
| ▪ LENGTHB | ▪ LENGTHC | ▪ LOWER |

*Table 28–1 (Cont.) Character Transformations Based on SQL character functions*

| Character Transformation Name | Character Transformation Name (Contd.) | Character Transformation Name (Contd.) |
|---|---|---|
| ▪ LPAD | ▪ LTRIM | ▪ NLSSORT |
| ▪ NLS_INITCAP | ▪ NLS_LOWER | ▪ NLS_UPPER |
| ▪ REPLACE | ▪ REGEXP_INSTR | ▪ REGEXP_REPLACE |
| ▪ REGEXP_SUBSTR | ▪ RPAD | ▪ RTRIM |
| ▪ SOUNDEX | ▪ SUBSTR | ▪ SUBSTR2 |
| ▪ SUBSTR4 | ▪ SUBSTRB | ▪ SUBSTRC |
| ▪ TRANSLATE | ▪ TRIM | ▪ UPPER |

For descriptions and examples of these functions, refer to section "Character Functions" in the *Oracle Database SQL Language Reference*.

Following is the list of custom character transformations.

- WB_LOOKUP_CHAR (number) on page 28-10
- WB_LOOKUP_CHAR (varchar2) on page 28-11
- WB_IS_SPACE on page 28-11

## WB_LOOKUP_CHAR (number)

### Syntax

```
WB.LOOKUP_CHAR (table_name
, column_name
, key_column_name
, key_value
)
```

where `table_name` is the name of the table to perform the lookup on and `column_name` is the name of the `VARCHAR2` column that will be returned. For example, the result of the lookup `key_column_name` is the name of the `NUMBER` column used as the key to match on in the lookup table, `key_value` is the value of the key column mapped into the `key_column_name` with which the match will be done.

### Purpose

To perform a key lookup on a number that returns a `VARCHAR2` value from a database table using a `NUMBER` column as the matching key.

### Example

Consider the following table as a lookup table LKP1:

```
KEY_COLUMN    TYPE    COLOR
10            Car     Red
20            Bike    Green
```

Using this package with the following call:

```
WB.LOOKUP_CHAR ('LKP1'
, 'TYPE'
, 'KEYCOLUMN'
, 20
```

```
)
```

returns the value of 'Bike' as output of this transform. This output would then be processed in the mapping as the result of an inline function call.

> **Note:** This function is a row-based key lookup. Set-based lookups are supported when you use the Lookup operator.

## WB_LOOKUP_CHAR (varchar2)

### Syntax

```
WB.LOOKUP_CHAR (table_name
, column_name
, key_column_name
, key_value
)
```

where `table_name` is the name of the table to perform the lookup on; `column_name` is the name of the `VARCHAR2` column that will be returned, for instance, the result of the lookup; `key_column_name` is the name of the `VARCHAR2` column used as the key to match on in the lookup table; `key_value` is the value of the key column, for instance, the value mapped into the `key_column_name` with which the match will be done.

### Purpose

To perform a key lookup on a `VARCHAR2` character that returns a `VARCHAR2` value from a database table using a `VARCHAR2` column as the matching key.

### Example

Consider the following table as a lookup table `LKP1`:

```
KEYCOLUMN  TYPE  COLOR
ACV        Car   Red
ACP        Bike  Green
```

Using this package with the following call:

```
WB.LOOKUP_CHAR ('LKP1'
, 'TYPE'
, 'KEYCOLUMN'
, 'ACP'
)
```

returns the value of 'Bike' as output of this transformation. This output is then processed in the mapping as the result of an inline function call.

> **Note:** This function is a row-based key lookup. Set-based lookups are supported when you use the Lookup operator.

## WB_IS_SPACE

### Syntax

```
WB_IS_SPACE(attibute)
```

#### Purpose

Checks whether a string value only contains spaces. This function returns a Boolean value. In mainframe sources, some fields contain many spaces to make a file adhere to the fixed length format. This function provides a way to check for these spaces.

#### Example

`WB_IS_SPACE` returns TRUE if attribute contains only spaces.

## Control Center Transformations

Control Center transformations are used in a process flow or in custom transformations to enable you to access information about the Control Center at execution time. For example, you can use a Control Center transformation in the expression on a transition to help control the flow through a process flow at execution time. You can also use Control Center transformations within custom functions. These custom functions can in turn be used in the design of your process flow.

All Control Center transformations require an audit ID that provides a handle to the audit data stored in the Control Center workspace. The audit ID is a key into the public view `ALL_RT_AUDIT_EXECUTIONS`. The transformations can be used to obtain data specific to that audit ID at execution time. When run in the context of a process flow, you can obtain the audit ID at execution time using the pseudo variable `audit_id` in a process flow expression. This variable is evaluated as the audit ID of the currently executing job. For example, for a map input parameter, this represents the map execution and for a transition this represents the job at the source of the transition.

The Control Center transformations are:


- WB_RT_GET_ELAPSED_TIME on page 28-12
- WB_RT_GET_JOB_METRICS on page 28-13
- WB_RT_GET_LAST_EXECUTION_TIME on page 28-14
- WB_RT_GET_MAP_RUN_AUDIT on page 28-14
- WB_RT_GET_NUMBER_OF_ERRORS on page 28-15
- WB_RT_GET_NUMBER_OF_WARNINGS on page 28-15
- WB_RT_GET_PARENT_AUDIT_ID on page 28-16
- WB_RT_GET_RETURN_CODE on page 28-16
- WB_RT_GET_START_TIME on page 28-17


### WB_RT_GET_ELAPSED_TIME

#### Syntax

```
WB_RT_GET_ELAPSED_TIME(audit_id)
```

#### Purpose

This function returns the elapsed time, in seconds, for the job execution given by the specified `audit_id`. It returns null if the specified audit ID does not exist. For example, you can use this function on a transition if you want to make a choice dependent on the time taken by the previous activity.

### Example

The following example returns the time elapsed since the activity represented by `audit_id` was started:

```
declare
   audit_id NUMBER := 1812;
   l_time NUMBER;
begin
   l_time:= WB_RT_GET_ELAPSED_TIME(audit_id);
end;
```

## WB_RT_GET_JOB_METRICS

### Syntax

```
WB_RT_GET_JOB_METRICS(audit_id, no_selected, no_deleted, no_updated, no_inserted,
no_discarded, no_merged, no_corrected)
```

where `no_selected` represents the number of rows selected, `no_deleted` represents the number of rows deleted, `no_updated` represents the number of rows updated, `no_inserted` represents the number of rows inserted, `no_discarded` represents the number of rows discarded, `no_merged` represents the number of rows merged, and `no_corrected` represents the number of rows corrected during the job execution.

### Purpose

This procedure returns the metrics of the job execution represented by the specified `audit_id`. The metrics include the number of rows selected, deleted, updated, inserted, discarded, merged, and corrected.

### Example

The following example retrieves the job metrics for the audit ID represented by `audit_id`.

```
declare
   audit_id NUMBER := 16547;
   l_nselected NUMBER;
   l_ndeleted NUMBER;
   l_nupdated NUMBER;
   l_ninserted NUMBER;
   l_ndiscarded NUMBER;
   l_nmerged NUMBER;
   l_ncorrected NUMBER;
begin
   WB_RT_GET_JOB_METRICS(audit_id, l_nselected, l_ndeleted, l_nupdated,
                         l_ninserted, l_ndiscarded, l_nmerged, l_ncorrected);
   dbms_output.put_line('sel=' || l_nselected || ', del=' l_ndeleted ||
                        ', upd=' || l_nupdated);
   dbms_output.put_line('ins='|| l_ninserted || ' , dis=' || l_ndiscarded );
   dbms_output.put_line('mer=' || l_nmerged || ', cor=' ||l_ncorrected);
 end;
```

## WB_RT_GET_LAST_EXECUTION_TIME

### Syntax

```
WB_RT_GET_LAST_EXECUTION_TIME(objectName, objectType, objectLocationName)
```

where `objectName` represents the name of the object, `objectType` represents the type of the object (for example MAPPING, DATA_AUDITOR, PROCESS_FLOW, SCHEDULABLE), and `objectLocationName` represents the location to which the object is deployed.

### Purpose

This transformation gives you access to time-based data. Typically, you can use this in a Process Flow to model some design aspect that is relevant to "time". For example you can design a path that may execute different maps if the time since the last execution is more than 1 day.

You can also use this transformation to determine time-synchronization across process flows that are running concurrently. For example, you can choose a path in a process flow according to whether another Process Flow has completed.

### Example

The following example retrieves the time when the mapping TIMES_MAP was last executed and the if condition determines whether this time was within 1 day of the current time. Based on this time, it can perform different actions.

```
declare
    last_exec_time DATE;
begin
    last_exec_time:=WB_RT_GET_LAST_EXECUTION_TIME('TIMES_MAP','MAPPING','WH_
LOCATION');
    if last_exec_time < sysdate - 1 then
--       last-execution was more than one day ago
--       provide details of action here
          NULL;
    Else
--       provide details of action here
          NULL;
    end if;
end;
```

## WB_RT_GET_MAP_RUN_AUDIT

### Syntax

```
WB_RT_GET_MAP_RUN_AUDIT(audit_id)
```

### Purpose

This function returns the map run ID for a job execution that represents a map activity. It returns null if `audit_id` does not represent the job execution for a map. For example, you can use the returned ID as a key to access the ALL_RT_MAP_RUN_ `<name>` views for more information.

### Example

The following example retrieves the map run ID for a job execution whose audit ID is 67265. It then uses this map run ID to obtain the name of the source from the ALL_RT_MAP_RUN_EXECUTIONS public view.

```
declare
  audit_id NUMBER := 67265;
  l_sources VARCHAR2(256);
  l_run_id NUMBER;
begin
  l_run_id := WB_RT_GET_MAP_RUN_AUDIT_ID(audit_id);
  SELECT source_name INTO l_sources FROM all_rt_map_run_sources
         WHERE map_run_id = l_run_id;
end;
```

## WB_RT_GET_NUMBER_OF_ERRORS

### Syntax

```
WB_RT_GET_NUMBER_OF_ERRORS(audit_id)
```

### Purpose

This function returns the number of errors recorded for the job execution given by the specified `audit_id`. It returns null if the specific `audit_id` is not found.

### Example

The following example retrieves the number of errors generated by the job execution whose audit ID is 8769. You can then perform different actions based on the number of errors.

```
declare
   audit_id NUMBER := 8769;
   l_errors NUMBER;
begin
   l_errors := WB_RT_GET_NUMBER_OF_ERRORS(audit_id);
   if l_errors < 5 then
      .....
   else
      .....
   end if;
end;
```

## WB_RT_GET_NUMBER_OF_WARNINGS

### Syntax

```
WB_RT_GET_NUMBER_OF_WARNINGS(audit_id)
```

### Purpose

This function returns the number of warnings recorded for the job executions represented by `audit_id`. It returns null if `audit_id` does not exist.

### Example

The following example returns the number of warnings generated by the job execution whose audit ID is 54632. You can then perform different actions based on the number of warnings.

```
declare
   audit_is NUMBER := 54632;
   l_warnings NUMBER;
begin
   l_ warnings:= WB_RT_GET_NUMBER_OF_WARNINGS (audit_id);
   if l_warnings < 5 then
      .....
   else
      .....
   end if;
end;
```

## WB_RT_GET_PARENT_AUDIT_ID

### Syntax

```
WB_RT_GET_PARENT_AUDIT_ID(audit_id)
```

### Purpose

This function returns the audit id for the process that owns the job execution represented by audit_id. It returns null if audit_id does not exist. You can then use the returned audit id as a key into other public views such as ALL_RT_AUDIT_EXECUTIONS, or other Control Center transformations if further information is required.

### Example

The following example retrieves the parent audit ID for a job execution whose audit ID is 76859. It then uses this audit ID to determine the elapsed time for the parent activity. You can perform different actions based on the elapsed time of the parent activity.

```
declare
   audit_id NUMBER := 76859;
   l_elapsed_time NUMBER;
   l_parent_id NUMBER;
begin
   l_parent_id := WB_RT_GET_PARENT_AUDIT_ID(audit_id);
   l_elapsed_time := WB_RT_GET_ELAPSED_TIME(l_parent_id);
   if l_elpased_time < 100 then
      .....
   else
      .....
   end if;
end;
```

## WB_RT_GET_RETURN_CODE

### Syntax

```
WB_RT_GET_RETURN_CODE(audit_id)
```

#### Purpose

This function returns the return code recorded for the job execution represented by `audit_id`. It returns null if `audit_id` does not exist. For a successful job execution, the return code is greater than or equal to 0. A return code of less than 0 signifies that the job execution has failed.

#### Example

The following example retrieves the return code for the job execution whose audit ID is represented by `audit_id`.

```
declare
   audit_id NUMBER:=69;
   l_code NUMBER;
begin
   l_code:= WB_RT_GET_RETURN_CODE(audit_id);
end;
```

### WB_RT_GET_START_TIME

#### Syntax

```
WB_RT_GET_START_TIME(audit_id)
```

#### Purpose

This function returns the start time for the job execution represented by `audit_id`. It returns null if `audit_id` does not exist. For example, you can use this in a transition if you wanted to make a choice dependent on when the previous activity started.

#### Example

The following example determines the start time of the job execution whose audit ID is 354.

```
declare
   audit_id NUMBER:=354;
   l_date TIMESTAMP WITH TIME ZONE;
begin
   l_date := WB_RT_GET_START_TIME(audit_id);
end;
```

## Conversion Transformations

The conversion transformations enable Warehouse Builder users to perform functions that allow conditional conversion of values. These functions achieve "if -then" constructions within SQL.

The conversion transformations that Warehouse Builder implements from the SQL conversion functions are as follows:

- ASCIISTR
- COMPOSE
- CONVERT
- HEXTORAW
- NUMTODSINTERVAL

- NUMTOYMINTERVAL
- RAWTOHEX
- RAWTONHEX
- SCN_TO_TIMESTAMP
- TIMESTAMP_TO_SCN
- TO_BINARY_DOUBLE
- TO_BINARY_FLOAT
- TO_CHAR (character), TO_CHAR (datetime), TO_CHAR (number)
- TO_CLOB
- TO_DATE
- TO_DSINTERVAL
- TO_MULTIBYTE
- TO_NCHAR (character), TO_NCHAR (datetime), TO_NCHAR (number)
- TO_NCLOB
- TO_NUMBER
- TO_SINGLE_BYTE
- TO_TIMESTAMP
- TO_TIMESTAMP_TZ
- TO_YMINTERVAL
- UNISTR

For descriptions and examples of these transformations, see "Conversion Functions" in the *Oracle Database SQL Language Reference*.

## Date Transformations

Date transformations provide Warehouse Builder users with functionality to perform transformations on date attributes. These transformations include SQL functions that are implemented by Warehouse Builder and custom functions provided with Warehouse Builder. The custom function are in the format `WB_<function name>`.

Following are the date transformations that are implementations of Database SQL functions:

- ADD_MONTHS
- CURRENT_DATE
- DBTIMEZONE
- FROM_TZ
- LAST_DAY
- MONTHS_BETWEEN
- NEW_TIME
- NEXT_DAY
- ROUND

- SESSIONTIMEZONE
- SYSDATE
- SYSTIMESTAMP
- SYS_EXTRACT_UTC
- TRUNC

For descriptions and examples of these transformations, refer to the section "Datetime Functions" in the *Oracle Database SQL Language Reference*.

The custom Date transformations are:


- WB_CAL_MONTH_NAME on page 28-19
- WB_CAL_MONTH_OF_YEAR on page 28-20
- WB_CAL_MONTH_SHORT_NAME on page 28-20
- WB_CAL_QTR on page 28-21
- WB_CAL_WEEK_OF_YEAR on page 28-21
- WB_CAL_YEAR on page 28-22
- WB_CAL_YEAR_NAME on page 28-22
- WB_DATE_FROM_JULIAN on page 28-23
- WB_DAY_NAME on page 28-23
- WB_DAY_OF_MONTH on page 28-24
- WB_DAY_OF_WEEK on page 28-24
- WB_DAY_OF_YEAR on page 28-25
- WB_DAY_SHORT_NAME on page 28-25
- WB_DECADE on page 28-26
- WB_HOUR12 on page 28-26
- WB_HOUR12MI_SS on page 28-27
- WB_HOUR24 on page 28-27
- WB_HOUR24MI_SS on page 28-28
- WB_IS_DATE on page 28-28
- WB_JULIAN_FROM_DATE on page 28-29
- WB_MI_SS on page 28-29
- WB_WEEK_OF_MONTH on page 28-30


## WB_CAL_MONTH_NAME

### Syntax

```
WB_CAL_MONTH_NAME(attribute)
```

### Purpose

The function call returns the full-length name of the month for the date specified in `attribute`.

### Example

The following example shows the return value on the `sysdate` and on a specified date string:

```
SELECT WB_CAL_MONTH_NAME(sysdate)
   FROM DUAL;

WB_CAL_MONTH_NAME(SYSDATE)
----------------------------
March

SELECT WB_CAL_MONTH_NAME('26-MAR-2002')
   FROM DUAL;

WB_CAL_MONTH_NAME('26-MAR-2002')
----------------------------------
March
```

## WB_CAL_MONTH_OF_YEAR

### Syntax

```
WB_CAL_MONTH_OF_YEAR(attribute)
```

### Purpose

`WB_CAL_MONTH_OF_YEAR` returns the month (1 to 12) of the year for date in `attribute`.

### Example

The following example shows the return value on the `sysdate` and on a specified date string:

```
SELECT WB_CAL_MONTH_OF_YEAR(sysdate) month
   FROM DUAL;

     MONTH
----------
         3

SELECT WB_CAL_MONTH_OF_YEAR('26-MAR-2002') month
FROM DUAL;

     MONTH
----------
         3
```

## WB_CAL_MONTH_SHORT_NAME

### Syntax

```
WB_CAL_MONTH_SHORT_NAME(attribute)
```

### Purpose

`WB_CAL_MONTH_SHORT_NAME` returns the short name of the month (for example 'Jan') for date in `attribute`.

### Example

The following example shows the return value on the `sysdate` and on a specified date string:

```
SELECT WB_CAL_MONTH_SHORT_NAME (sysdate) month
FROM DUAL;

MONTH
---------
Mar

SELECT WB_CAL_MONTH_SHORT_NAME ('26-MAR-2002') month
FROM DUAL;

MONTH
---------
Mar
```

## WB_CAL_QTR

### Syntax

```
WB_CAL_QTR(attribute)
```

### Purpose

`WB_CAL_QTR` returns the quarter of the Gregorian calendar year (for example Jan - March = 1) for the date in `attribute`.

### Example

The following example shows the return value on the `sysdate` and on a specified date string:

```
SELECT WB_CAL_QTR (sysdate) quarter
FROM DUAL;

   QUARTER
----------
         1

SELECT WB_CAL_QTR ('26-MAR-2002') quarter
FROM DUAL;

   QUARTER
----------
         1
```

## WB_CAL_WEEK_OF_YEAR

### Syntax

```
WB_CAL_WEEK_OF_YEAR(attribute)
```

### Purpose

`WB_CAL_WEEK_OF_YEAR` returns the week of the year (1 to 53) for the date in `attribute`.

### Example

The following example shows the return value on the `sysdate` and on a specified date string:

```
SELECT WB_CAL_WEEK_OF_YEAR (sysdate) w_of_y
FROM DUAL;

    W_OF_Y
----------
        13

SELECT WB_CAL_WEEK_OF_YEAR ('26-MAR-2002') w_of_y
FROM DUAL;

    W_OF_Y
----------
        13
```

## WB_CAL_YEAR

### Syntax

```
WB_CAL_YEAR(attribute)
```

### Purpose

`WB_CAL_YEAR` returns the numerical year component for the date in `attribute`.

### Example

The following example shows the return value on the `sysdate` and on a specified date string:

```
SELECT WB_CAL_YEAR (sysdate) year
FROM DUAL;

      YEAR
----------
      2002

SELECT WB_CAL_YEAR ('26-MAR-2002') w_of_y
FROM DUAL;

      YEAR
----------
      2002
```

## WB_CAL_YEAR_NAME

### Syntax

```
WH_CAL_YEAR_NAME(attribute)
```

### Purpose

`WB_CAL_YEAR_NAME` returns the spelled out name of the year for the date in `attribute`.

### Example

The following example shows the return value on the `sysdate` and on a specified date string:

```
select WB_CAL_YEAR_NAME (sysdate) name
from dual;

NAME
---------------------------------------------
Two Thousand Two

select WB_CAL_YEAR_NAME ('26-MAR-2001') name
from dual;

NAME
---------------------------------------------
Two Thousand One
```

## WB_DATE_FROM_JULIAN

### Syntax

```
WB_DATE_FROM_JULIAN(attribute)
```

### Purpose

`WB_DATE_FROM_JULIAN` converts Julian date `attribute` to a regular date.

### Example

The following example shows the return value on a specified Julian date:

```
select to_char(WB_DATE_FROM_JULIAN(3217345),'dd-mon-yyyy') JDate
from dual;

JDATE
-----------
08-sep-4096
```

## WB_DAY_NAME

### Syntax

```
WB_DAY_NAME(attribute)
```

### Purpose

`WB_DAY_NAME` returns the full name of the day for the date in `attribute`.

### Example

The following example shows the return value on the `sysdate` and on a specified date string:

```
select WB_DAY_NAME (sysdate) name
from dual;

NAME
--------------------------------------------
Thursday

select WB_DAY_NAME ('26-MAR-2002') name
```

```
from dual;

NAME
--------------------------------------------
Tuesday
```

## WB_DAY_OF_MONTH

### Syntax

```
WB_DAY_OF_MONTH(attribute)
```

### Purpose

`WB_DAY_OF_MONTH` returns the day number within the month for the date in `attribute`.

### Example

The following example shows the return value on the `sysdate` and on a specified date string:

```
select WB_DAY_OF_MONTH (sysdate) num
from dual;

       NUM
----------
        28

select WB_DAY_OF_MONTH ('26-MAR-2002') num
from dual

       NUM
----------
        26
```

## WB_DAY_OF_WEEK

### Syntax

```
WB_DAY_OF_WEEK(attribute)
```

### Purpose

`WB_DAY_OF_WEEK` returns the day number within the week for date `attribute` based on the database calendar.

### Example

The following example shows the return value on the `sysdate` and on a specified date string:

```
select WB_DAY_OF_WEEK (sysdate) num
from dual;

       NUM
----------
         5

select WB_DAY_OF_WEEK ('26-MAR-2002') num
```

```
from dual;


       NUM
----------
         3
```

## WB_DAY_OF_YEAR

### Syntax

```
WB_DAY_OF_YEAR(attribute)
```

### Purpose

`WB_DAY_OF_YEAR` returns the day number within the year for the date `attribute`.

### Example

The following example shows the return value on the `sysdate` and on a specified date string:

```
select WB_DAY_OF_YEAR (sysdate) num
from dual;

       NUM
----------
        87

select WB_DAY_OF_YEAR ('26-MAR-2002') num
from dual;

       NUM
----------
        85
```

## WB_DAY_SHORT_NAME

### Syntax

```
WB_DAY_SHORT_NAME(attribute)
```

### Purpose

`WB_DAY_SHORT_NAME` returns the three letter abbreviation or name for the date `attribute`.

### Example

The following example shows the return value on the `sysdate` and on a specified date string:

```
select WB_DAY_SHORT_NAME  (sysdate) abbr
from dual;

ABBR
------------------------------------
Thu

select WB_DAY_SHORT_NAME  ('26-MAR-2002') abbr
```

```
from dual;

NUM
------------------------------------
Tue
```

## WB_DECADE

### Syntax

```
WB_DECADE(attribute)
```

### Purpose

WB_DECADE returns the decade number within the century for the date attribute.

### Example

The following example shows the return value on the sysdate and on a specified date string:

```
select WB_DECADE  (sysdate) dcd
from dual;

       DCD
----------
         2

select WB_DECADE  ('26-MAR-2002') DCD
from dual;

       DCD
----------
         2
```

## WB_HOUR12

### Syntax

```
WB_HOUR12(attribute)
```

### Purpose

WB_HOUR12 returns the hour (in a 12-hour setting) component of the date corresponding to attribute.

### Example

The following example shows the return value on the sysdate and on a specified date string:

```
select WB_HOUR12 (sysdate) h12
from dual;

       H12
----------
         9

select WB_HOUR12 ('26-MAR-2002') h12
from dual;
```

```
       H12
----------
        12
```

> **Note:** For a date not including the timestamp (in the second example), Oracle uses the 12:00 (midnight) timestamp and therefore returns 12 in this case.

## WB_HOUR12MI_SS

### Syntax

`WB_HOUR12MI_SS(attribute)`

### Purpose

`WB_HOUR12MI_SS` returns the timestamp in `attribute` formatted to HH12:MI:SS.

### Example

The following example shows the return value on the `sysdate` and on a specified date string:

```
select WB_HOUR12MI_SS (sysdate) h12miss
from dual;

H12MISS
------------------------------------
09:08:52

select WB_HOUR12MI_SS ('26-MAR-2002') h12miss
from dual;

H12MISS
------------------------------------
12:00:00
```

> **Note:** For a date not including the timestamp (in the second example), Oracle uses the 12:00 (midnight) timestamp and therefore returns 12 in this case.

## WB_HOUR24

### Syntax

`WB_HOUR24(attribute)`

### Purpose

`WB_HOUR24` returns the hour (in a 24-hour setting) component of date corresponding to `attribute`.

### Example

The following example shows the return value on the `sysdate` and on a specified date string:

```
select WB_HOUR24 (sysdate) h24
from dual;

       H24
----------
         9

select WB_HOUR24 ('26-MAR-2002') h24
from dual;

       H24
----------
         0
```

**Note:** For a date not including the timestamp (in the second example), Oracle uses the 00:00:00 timestamp and therefore returns the timestamp in this case.

## WB_HOUR24MI_SS

### Syntax

```
WB_HOUR24MI_SS(attribute)
```

### Purpose

`WB_HOUR24MI_SS` returns the timestamp in `attribute` formatted to HH24:MI:SS.

### Example

The following example shows the return value on the `sysdate` and on a specified date string:

```
select WB_HOUR24MI_SS (sysdate) h24miss
from dual;

H24MISS
------------------------------------
09:11:42

select WB_HOUR24MI_SS ('26-MAR-2002') h24miss
from dual;

H24MISS
------------------------------------
00:00:00
```

**Note:** For a date not including the timestamp (in the second example), Oracle uses the 00:00:00 timestamp and therefore returns the timestamp in this case.

## WB_IS_DATE

### Syntax

```
WB_IS_DATE(attribute, fmt)
```

### Purpose

To check whether `attribute` contains a valid date. The function returns a Boolean value which is set to true if `attribute` contains a valid date. `Fmt` is an optional date format. If `fmt` is omitted, then the date format of your database session is used.

You can use this function when you validate your data before loading it into a table. This way the value can be transformed before it reaches the table and causes an error.

### Example

`WB_IS_DATE` returns true in PL/SQL if `attribute` contains a valid date.

## WB_JULIAN_FROM_DATE

### Syntax

```
WB_JULIAN_FROM_DATE(attribute)
```

### Purpose

`WB_JULIAN_FROM_DATE` returns the Julian date of date corresponding to `attribute`.

### Example

The following example shows the return value on the `sysdate` and on a specified date string:

```
select WB_JULIAN_FROM_DATE (sysdate) jdate
from dual;

     JDATE
----------
   2452362

select WB_JULIAN_FROM_DATE ('26-MAR-2002') jdate
from dual;

     JDATE
----------
   2452360
```

## WB_MI_SS

### Syntax

```
WB_MI_SS(attribute)
```

### Purpose

`WB_MI_SS` returns the minutes and seconds of the time component in the date corresponding to `attribute`.

### Example

The following example shows the return value on the `sysdate` and on a specified date string:

```
select WB_MI_SS (sysdate) mi_ss
from dual;
```

```
MI_SS
-------------------------------------------
33:23

select WB_MI_SS ('26-MAR-2002') mi_ss
from dual;

MI_SS
-------------------------------------------
00:00
```

**Note:** For a date not including the timestamp (in the second example), Oracle uses the 00:00:00 timestamp and therefore returns the timestamp in this case.

### WB_WEEK_OF_MONTH

#### Syntax

```
WB_WEEK_OF_MONTH(attribute)
```

#### Purpose

`WB_WEEK_OF_MONTH` returns the week number within the calendar month for the date corresponding to `attribute`.

#### Example

The following example shows the return value on the `sysdate` and on a specified date string:

```
select WB_WEEK_OF_MONTH (sysdate) w_of_m
from dual;

    W_OF_M
----------
         4

select WB_WEEK_OF_MONTH ('26-MAR-2002') w_of_m
from dual;

    W_OF_M
----------
         4
```

## Number Transformations

Number transformations provide Warehouse Builder users with functionality to perform transformations on numeric values. These include Database SQL functions that are implemented by Warehouse Builder and custom functions defined by Warehouse Builder. The custom functions are prefixed with `WB_`.

Table 28–2 lists the number transformations that are based on Database SQL numeric functions. The transformations are listed in a columnar table that reads down the columns from left to right to conserve space.

*Table 28–2 List of Number Transformations Based on Database SQL Functions*

| Number Transformation Name | Number Transformation Name (Contd.) | Number Transformation Name (Contd.) |
|---|---|---|
| ABS | ACOS | ASIN |
| ATAN | ATAN2 | BITAND |
| CEIL | COS | COSH |
| EXP | FLOOR | LN |
| LOG | MOD | NANVL |
| POWER | REMAINDER | ROUND (number) |
| SIGN | SIN | SINH |
| SQRT | TAN | TANH |
| TRUNC (number) | WIDTH_BUCKET | |

For descriptions and examples of these transformations, refer to the section titled "Numeric Functions" in the *Oracle Database SQL Language Reference*.

The custom numeric transformations are:

- WB_LOOKUP_NUM (on a number) on page 28-31
- WB_LOOKUP_NUM (on a varchar2) on page 28-32
- WB_IS_NUMBER on page 28-33

## WB_LOOKUP_NUM (on a number)

### Syntax

```
 WB_LOOKUP_NUM (table_name
, column_name
, key_column_name
, key_value
)
```

where `table_name` is the name of the table to perform the lookup on; `column_name` is the name of the `NUMBER` column that will be returned, for instance, the result of the lookup; `key_column_name` is the name of the `NUMBER` column used as the key to match on in the lookup table; `key_value` is the value of the key column, for example, the value mapped into the `key_column_name` with which the match will be done.

### Purpose

To perform a key look up that returns a `NUMBER` value from a database table using a `NUMBER` column as the matching key.

### Example

Consider the following table as a lookup table `LKP1`:

```
KEYCOLUMN  TYPE_NO  TYPE
10         100123   Car
20         100124   Bike
```

Using this package with the following call:

```
WB_LOOKUP_CHAR('LKP1'
```

```
, 'TYPE_NO'
, 'KEYCOLUMN'
, 20
)
```

returns the value of 100124 as output of this transformation. This output is then processed in the mapping as the result of an inline function call.

> **Note:** This function is a row-based key lookup. Set-based lookups are supported when you use the Lookup operator.

### WB_LOOKUP_NUM (on a varchar2)

#### Syntax:

```
WB_LOOKUP_CHAR(table_name
, column_name
, key_column_name
, key_value
)
```

where `table_name` is the name of the table to perform the lookup on; `column_name` is the name of the `NUMBER` column that will be returned (such as the result of the lookup); `key_column_name` is the name of the `NUMBER` column used as the key to match on in the lookup table; `key_value` is the value of the key column, such as the value mapped into the `key_column_name` with which the match will be done.

#### Purpose:

To perform a key lookup which returns a `NUMBER` value from a database table using a `VARCHAR2` column as the matching key.

#### Example

Consider the following table as a lookup table `LKP1`:

```
KEYCOLUMN  TYPE_NO  TYPE
ACV        100123   Car
ACP        100124   Bike
```

Using this package with the following call:

```
WB_LOOKUP_CHAR ('LKP1'
, 'TYPE'
, 'KEYCOLUMN'
, 'ACP'
)
```

returns the value of `100124` as output of this transformation. This output is then processed in the mapping as the result of an inline function call.

> **Note:** This function is a row-based key lookup. Set-based lookups are supported when you use the Lookup operator described in "Lookup Operator" on page 26-23.

### WB_IS_NUMBER

#### Syntax

```
WB_IS_NUMBER(attibute, fmt)
```

#### Purpose

To check whether `attribute` contains a valid number. The function returns a Boolean value, which is set to true if `attribute` contains a valid number. `Fmt` is an optional number format. If `fmt` is omitted, then the number format of your session is used.

You can use this function when you validate the data before loading it into a table. This way the value can be transformed before it reaches the table and causes an error.

#### Example

`WB_IS_NUMBER` returns `true` in PL/SQL if `attribute` contains a valid number.

## OLAP Transformations

OLAP transformations enable Warehouse Builder users to load data stored in relational dimensions and cubes into an analytic workspace.

The OLAP transformations provided by Warehouse Builder are:

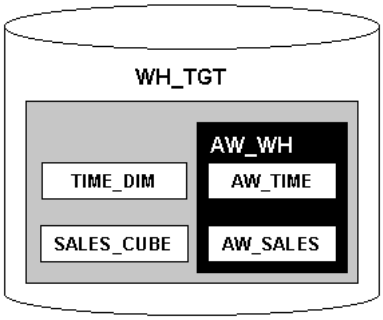
- WB_OLAP_AW_PRECOMPUTE on page 28-34
- WB_OLAP_LOAD_CUBE on page 28-34
- WB_OLAP_LOAD_DIMENSION on page 28-35
- WB_OLAP_LOAD_DIMENSION_GENUK on page 28-35

The `WB_OLAP_LOAD_CUBE`, `WB_OLAP_LOAD_DIMENSION`, and `WB_OLAP_LOAD_DIMENSION_GENUK` transformations are used for cube cloning in Warehouse Builder. Use these OLAP transformations only if your database version is Oracle Database 9*i* or Oracle Database 10*g* Release 1. Starting with Oracle 10*g* Release 2, you can directly deploy dimensions and cubes into an analytic workspace.

The `WB_OLAP_AW_PRECOMPUTE` only works with the Oracle Warehouse Builder 10*g* Release 2.

The examples used to explain these OLAP transformations are based on the scenario depicted in Figure 28–1.

***Figure 28–1 Example of OLAP Transformations***



The relational dimension TIME_DIM and the relational cube SALES_CUBE are stored in the schema WH_TGT. The analytic workspace AW_WH, into which the dimension and cube are loaded, is also created in the WH_TGT schema.

## WB_OLAP_AW_PRECOMPUTE

### Syntax

```
WB_OLAP_AW_PRECOMPUTE(p_aw_name, p_cube_name, p_measure_name, p_allow_parallel_
solve, p_max_job_queues_allocated)
```

where p_aw_name is the name of the AW where cube is deployed, p_cube_name is the name of the cube to solve, p_measure_name is the optional name of a specific measure to solve (if no measure is specified, then all measures will be solved), p_allow_parallel_solve is the boolean to indicate parallelization of solve based on partitioning (performance related parameter), p_max_job_queues_allocated is the number of DBMS jobs to execute in parallel (default value is 0). If 5 is defined and there are 20 partitions, then a pool of 5 DBMS jobs will be used to perform the data load.

There is a subtle different between parallel and non-parallel solving. With non-parallel solve, the solve happens synchronously, so when the API call is completed the solve is complete. Parallel solve executes asynchronously, the API call will return with a job id of the job started. The job will control parallel solving using the max job queues parameter to control its processing. The user may then use the job id to query the all_scheduler_* views to check on the status of the activity.

### Purpose

WB_OLAP_AW_PRECOMPUTE is used for solving a non-compressed cube (compressed cubes are auto-solved). The load and solve steps can be done independently. By default, the cube map loads data, then solves (precomputes) the cube. You can load data using the map, then perform the solve at a different point of time (since the solve/build time is the costliest operation).

### Example

The following example loads data from the relational cubes MART and SALES_CUBE into a cube called SALES and performs a simple solve execution working serially. This example has parameters for parallel solve and max number of job queues. If parallel solve is performed, then an ASYNCHRONOUS solve job is started and the master job ID is returned via the return function.

```
declare
  rslt varchar2(4000);
begin
…
  rslt :=wb_olap_aw_precompute('MART','SALES_CUBE','SALES');
…
end;
/
```

## WB_OLAP_LOAD_CUBE

### Syntax

```
wb_olap_load_cube::=WB_OLAP_LOAD_CUBE(olap_aw_owner, olap_aw_name, olap_cube_
owner, olap_cube_name, olap_tgt_cube_name)
```

where olap_aw_owner is the name of the database schema that owns the analytic workspace; olap_aw_name is the name of the analytic workspace that stores the cube data; olap_cube_owner is the name of the database schema that owns the related relational cube; olap_cube_name is the name of the relational cube; olap_tgt_cube_name is the name of the cube in the analytic workspace.

### Purpose

WB_OLAP_LOAD_CUBE loads data from the relational cube into the analytic workspace. This allows further analysis of the cube data. This is for loading data in an AW cube from a relational cube which it was cloned from. This is a wrapper around some of the procedures in the DBMS_AWM package for loading a cube.

### Example

The following example loads data from the relational cube SALES_CUBE into a cube called AW_SALES in the AW_WH analytic workspace:

```
WB_OLAP_LOAD_CUBE('WH_TGT', 'AW_WH', 'WH_TGT', 'SALES_CUBE', 'AW_SALES')
```

## WB_OLAP_LOAD_DIMENSION

### Syntax

```
wb_olap_load_dimension::=WB_OLAP_LOAD_DIMENSION(olap_aw_owner, olap_aw_name, olap_
dimension_owner, olap_dimension_name, olap_tgt_dimension_name)
```

where olap_aw_owner is the name of the database schema that owns the analytic workspace; olap_aw_name is the name of the analytic workspace that stores the dimension data; olap_dimension_owner is the name of the database schema in which the related relational dimension is stored; olap_dimension_name is the name of the relational dimension; olap_tgt_dimension_name is the name of the dimension in the analytic workspace.

### Purpose

WB_OLAP_LOAD_DIMENSION loads data from the relational dimension into the analytic workspace. This allows further analysis of the dimension data. This is for loading data in an AW dimension from a relational dimension which it was cloned from. This is a wrapper around some of the procedures in the DBMS_AWM package for loading a dimension.

### Example

The following example loads the data from the relational dimension TIME_DIM into a dimension called AW_TIME in the analytic workspace AW_WH:

```
WB_OLAP_LOAD_DIMENSION('WH_TGT', 'AW_WH', 'WH_TGT', 'TIME_DIM', 'AW_TIME')
```

## WB_OLAP_LOAD_DIMENSION_GENUK

### Syntax

```
wb_olap_load_dimension_genuk::=WB_OLAP_LOAD_DIMENSION_GENUK(olap_aw_owner, olap_
aw_name, olap_dimension_owner, olap_dimension_name, olap_tgt_dimension_name)
```

where `olap_aw_owner` is the name of the database schema that owns the analytic workspace; `olap_aw_name` is the name of the analytic workspace that stores the dimension data; `olap_dimension_owner` is the name of the database schema in which the related relational dimension is stored; `olap_dimension_name` is the name of the relational dimension; `olap_tgt_dimension_name` is the name of the dimension in the analytic workspace.

#### Purpose

`WB_OLAP_LOAD_DIMENSION_GENUK` loads data from the relational dimension into the analytic workspace. Unique dimension identifiers will be generated across all levels. This is for loading data in an AW dimension from a relational dimension which it was cloned from. This is a wrapper around some of the procedures in the DBMS_AWM package for loading a dimension.

If a cube has been cloned and if you select YES for the Generate Surrogate Keys for Dimensions option, then when you want to reload the dimensions, you should use the `WB_OLAP_LOAD_DIMENSION_GENUK` procedure. This procedure generates surrogate identifiers for all levels in the AW, because the AW requires all level identifiers to be unique across all levels of a dimension.

#### Example

Consider an example in which the dimension `TIME_DIM` has been deployed to the OLAP server by cloning the cube. The parameter generate surrogate keys for Dimension was set to true. To now reload data from the relational dimension `TIME_DIM` into the dimension `AW_TIME` in the analytic workspace `AW_WH`, use the following syntax.

```
WB_OLAP_LOAD_CUBE('WH_TGT', 'AW_WH', 'WH_TGT', 'TIME_DIM', 'AW_TIME')
```

## Other Transformations

Other transformations included with Warehouse Builder enable you to perform various functions which are not restricted to certain data types. This section describes those types.

Other transformations provided by Warehouse Builder are:

- DEPTH
- DUMP
- EMPTY_BLOB
- EMPTY_CLOB
- NLS_CHARSET_DECL_LEN
- NLS_CHARSET_ID
- NLS_CHARSET_NAME
- NULLIF
- NVL
- NVL2
- ORA_HASH
- PATH
- SYS_CONTEXT

- SYS_GUID
- SYS_TYPEID
- UID
- USER
- USERENV
- VSIZE

For descriptions and examples of these transformations, see *Oracle Database SQL Language Reference*.

## Spatial Transformations

Spatial Transformation is an integrated set of functions and procedures that enables spatial data to be stored, accessed, and analyzed quickly and efficiently in an Oracle Database.

Spatial transformations included with Warehouse Builder are:

- SDO_AGGR_CENTROID
- SDO_AGGR_CONVEXHULL
- SDO_AGGR_MBR
- SDO_AGGR_UNION

For descriptions and examples of these transformations, refer to the *Oracle Spatial Developer's Guide*.

## Streams Transformations

The Streams transformations category contains one transformation called REPLICATE. The following section describes this transformation.

### REPLICATE

#### Syntax

```
REPLICATE(lcr, conflict_resolution)
```

where `lcr` stands for Logical Change Record and encapsulates the DML change. Its data type is SYS.LCR$_ROW_RECORD. `conflict_resolution` is a Boolean variable. If its value is TRUE, then any conflict resolution defined for the table will be used to resolve conflicts resulting from the execution of the LCR. For more information about conflict resolution, see *Oracle Streams Replication Administrator's Guide*.

#### Purpose

REPLICATE is used to replicate a DML change (INSERT, UPDATE, or DELETE) that has occurred on a table in the source system on an identical table in the target system. The table in the target system should be identical to the table in the source system in the following respects:.

- The name of the schema that contains the target table should be the same as the name of the schema that contains the source table.
- The name of the target table should the same as the name of the source table.

- The structure of the target table should be the same as that of the source table. The structure includes the number, name, and data type of the columns in the table.

**Example**

Consider a table T1(c1 varchar2(10), c2 number primary key) in schema S on the source system and an identical table in the target system. Consider the following insert operation on the table T1 on the source system

```
insert into T1 values ('abcde', 10)
```

An LCR representing the change following the above insert of a row on the table T1 in the source system will have the following details

```
LCR.GET_OBJECT_OWNER will be 'S'
LCR.GET_OBJECT_NAME will be 'T1'
LCR.GET_COMMAND_TYPE will be 'INSERT'
LCR.GET_VALUE('c1', 'new') will have the value for the column 'c1' - i.e. 'abcde'
LCR.GET_VALUE('c2', 'new') will have the value for the column 'c2' - i.e. 10
```

Such an LCR will be created and enqueued by a Streams Capture Process on the source system that captures changes on table S.T1

`REPLICATE(lcr, true)` - will result in a row ('abcde', 10) being inserted into the table `T1` on the target system.

> **Note:** Using this approach will not provide lineage information. If lineage is important, then do not use this function. Use the more direct approach of using an LCR Cast operator bound to the source table and a Table operator bound to the target table and connecting the attributes of these two operators with the same name ('Match by name'). Further information about LCR (Logical Change Record) is available in Oracle Database 10*g* Documentation.

## XML Transformations

XML transformations provide Warehouse Builder users with functionality to perform transformations on XML objects. These transformations enable Warehouse Builder users to load and transform XML documents and Oracle AQs.

To enable loading of XML sources, Warehouse Builder provides access to the database XML functionality by implementing database XML functions and by defining custom functions.

Following are the XML transformations that are implemented based on database XML functions:

- EXISTSNODE
- EXTRACT
- EXTRACTVALUE
- SYS_XMLAGG
- SYS_XMLGEN
- XMLCONCAT
- XMLSEQUENCE
- XMLTRANSFORM

**See Also:**

- *Oracle Database SQL Language Reference* for descriptions for these transformations
- *Oracle Spatial Developer's Guide* for examples on using these transformations

The custom XML transformations are:

- WB_XML_LOAD on page 28-39
- WB_XML_LOAD_F on page 28-39

## WB_XML_LOAD

### Syntax:

```
WB_XML_LOAD(control_file)
```

### Purpose

This program unit extracts and loads data from XML documents into database targets. The `control_file`, an XML document, specifies the source of the XML documents, the targets, and any runtime controls. After the transformation has been defined, a mapping in Warehouse Builder calls the transformation as a pre-map or post-map trigger.

### Example

The following example illustrates a script that can be used to implement a Warehouse Builder transformation that extracts data from an XML document stored in the file products.xml and loads it into the target table called books:

```
begin
wb_xml_load('<OWBXMLRuntime>'
||
'<XMLSource>'
||
' <file>\ora817\GCCAPPS\products.xml</file>'
||
'</XMLSource>'
||
'<targets>'
||
' <target XSLFile="\ora817\XMLstyle\GCC.xsl">books</target>'
||
'</targets>'
||
'</OWBXMLRuntime>'
);
end;
```

For more information about control files, see the *Oracle Warehouse Builder User's Guide.*

## WB_XML_LOAD_F

### Syntax

```
WB_XML_LOAD_F(control_file)
```

### Purpose

`WB_XML_LOAD_F` extracts and loads data from XML documents into database targets. The function returns the number of XML documents read during the load. The `control_file`, itself an XML document, specifies the source of the XML documents, the targets, and any runtime controls. After the transformation has been defined, a mapping in Warehouse Builder calls the transformation as a pre-map or post-map trigger.

### Example

The following example illustrates a script that can be used to implement a Warehouse Builder transformation that extracts data from an XML document stored in the file `products.xml` and loads it into the target table books:

```
begin
wb_xml_load_f('<OWBXMLRuntime>'
||
'<XMLSource>'
||
' <file>\ora817\GCCAPPS\products.xml</file>'
||
'</XMLSource>'
||
'<targets>'
||
' <target XSLFile="\ora817\XMLstyle\GCC.xsl">books</target>'
||
'</targets>'
||
'</OWBXMLRuntime>'
);
end;
```

For more information about the types handled and detailed information about `control_files`, see the *Oracle Warehouse Builder Installation and Administration Guide for Windows and UNIX*.

# 29

# Warehouse Builder Code Template Tools and Substitution Methods Reference

This chapter contains the following topics:


- Code Template Tools Supported in Warehouse Builder
- Code Template Tools Not Supported in Oracle Warehouse Builder
- Code Template Substitution Methods Supported in Warehouse Builder
- Code Template Substitution Methods Not Supported in Warehouse Builder


## Code Template Tools Supported in Warehouse Builder

Oracle Warehouse Builder includes a subset of the ODI tools used to make code templates. For the full reference for all tools, see *Oracle Data Integrator Tools Reference Manual for ODI 10.1.3.5* at:

`http://www.oracle.com/technology/products/oracle-data-integrator`

The following tables list the tools supported in Oracle Warehouse Builder and the tools that were omitted.

***Table 29–1 Code Template Tools Supported by Oracle Warehouse Builder***

| Code Template | Description |
|---|---|
| OdiAnt | Calls the building tool Ant of the Apache Software Foundation. This tool can be used to run several commands such as archiving, compiling, deploying, documenting, EJBs. It runs OS processes, manipulates files, .NET tasks, remote tasks, and so on. |
| OdiBeep | Plays a default beep or sound file on the computer hosting the agent. |
| OdiOSCommand | Invokes an operating system command shell to carry out a command and redirects the output to files. |
| OdiFtpGet | Retrieves a local file from a remote FTP server computer using the FTP protocol GET command. |
| OdiFtpPut | Copies a local file to a remote FTP server computer using the FTP protocol PUT command. |

***Table 29–1 (Cont.) Code Template Tools Supported by Oracle Warehouse Builder***

| Code Template | Description |
|---|---|
| OdiInvokeWebService (Replaces OdiExecuteWebService) | Invokes a web service using the HTTP protocol and writes the returned response to an XML file.<br>This tool invokes a specific operation on the port of a web service whose description file (WSDL) URL is provided.<br>If this operation requires a SOAP request, it is provided either in a request file or directly written out in the tool call (XMLRequest). The response of the web service request is written to an XML file that can be used in Oracle Data Integrator.<br>Note: This Tool replaces OdiExecuteWebService.<br>Note: This Tool supports both HTTP and HTTPS protocols. |
| OdiReadMail | Can be used to read emails as well as their attachments from a POP or IMAP internet mail service.<br>The use of this command allows the connection to the mail hosted on the server MAILHOST using the connection parameters specified by USER and PASS. The execution agent is then in waiting mode until MAX_MSG messages are received or the maximum waiting time specified by TIMEOUT is reached. The extracted messages must match the filters such as those specified by the parameters SUBJECT and SENDER. When a message satisfies these criteria, its content as well as its Internet 53 attachment are extracted in a directory specified by the parameter FOLDER. If the parameter KEEP is set to No, the retrieved message is then suppressed from the mailbox. |
| OdiScpGet | Retrieves a file from a remote SSH server. |
| OdiScpPut | Copies a local file to a directory on the remote SSH server computer. |
| OdiSendMail | Can send an email through an SMTP server. |
| OdiSftpGet | Retrieves a file from a remote SSH server with an enabled SFTP subsystem. |
| OdiSftpPut | Copies a local file to a remote SSH server computer that has the SFTP subsystem enabled. |
| OdiFileAppend | Concatenates a set of files into a single file.<br>The HEADER parameter is used to suppress *n* number of first lines of the input files (for instance, headers of csv files). The KEEP_FIRST_HEADER parameter is used to keep or remove the first occurrence of the header lines in the resulting file.<br>When the HEADER parameter is omitted, the concatenation is binary (no *text* reading of the file is performed). |
| OdiFileCopy | Copies files or a folder into files or a folder on the computer of the execution agent. |
| OdiFileDelete | Deletes, based on the file modification time, files or a directory on the computer running the agent which executes it.<br>If -FROMDATE is omitted, all files with a modification date earlier than the -TODATE date will be deleted.<br>If -TODATE is omitted, all files with a modification date later than the -FROMDATE date will be deleted.<br>If both parameters are omitted, all files matching the -FILE parameter value will be deleted. |

*Table 29–1 (Cont.) Code Template Tools Supported by Oracle Warehouse Builder*

| Code Template | Description |
|---|---|
| OdiFileMove | Moves files or a directory into files or a directory on the computer of the execution agent.<br>This command can also be used to rename files or directories. |
| OdiFileWait | Scans the directory DIR and waits for files matching the mask given in PATTERN. When the specified files are found, the action indicated by the parameter ACTION is triggered.<br>The execution agent is in standby mode waiting to receive FILECOUNT files unless the timeout limit, given by TIMEOUT is reached. The directory is scanned every POLLINT milliseconds. |
| OdiMkDir | Creates a directory.<br>If the parent directory does not exist, this command will recursively create the parent directories. |
| OdiOutFile | Generates a text file on the computer where the execution agent is running.<br>The file is stored in the path defined by <file_name> and contains the text specified by <Text>.<br>The optional parameter -APPEND can be used to append the <Text> at the end of the file if it already contains data. The parameters -FILE and -APPEND must be specified on the same line. |
| OdiSqlUnload | Generates a data file by executing the SQL query <sql_query> on the data server whose connection parameters are provided by <driver>, <url>, <user> and <encoded_pass>. The file is written to the path defined by <file_name> (relative to the agent). |
| OdiUnZip | Used to unzip an archive file to a directory. |
| OdiZip | Creates ZIP files from a directory or several files. |
| OdiXMLConcat | Extracts all instances of a given element from a set of source XML files and concatenates them into one target XML file.<br>This tool parses and generates well-formed XML.<br>It does not modify or generate a DTD for the generated files. A reference to an existing DTD can be specified in the HEADER parameter or preserved from the original files using the -KEEP_XML_PROLOGUE.<br>**Note:** XML Namespaces are not supported by this tool. You must provide the local part of element name (without namespace or prefix value) in the -ELEMENT_NAME parameter. |
| OdiXMLSplit | Extracts all instances of a given element stored in a source XML file and splits it over several target XML files.<br>This tool parses and generates well-formed XML.<br>It does not modify or generate a DTD for the generated files. A reference to an existing DTD can be specified in the HEADER parameter or preserved from the original files using the -KEEP_XML_PROLOGUE.<br>Note: XML Namespaces are not supported by this tool. Please provide the local part of element name (without namespace or prefix value) in the -ELEMENT_NAME parameter. |

*Table 29–1 (Cont.) Code Template Tools Supported by Oracle Warehouse Builder*

| Code Template | Description |
|---|---|
| OdiFileWait | Scans the directory DIR and waits for files matching the mask given in PATTERN. When the specified files are found, the action indicated by the parameter ACTION is triggered.<br>The execution agent is in standby mode waiting to receive FILECOUNT files unless the timeout limit, given by TIMEOUT is reached. The directory is scanned every POLLINT milliseconds |
| OdiReadMail | Read emails as well as their attachments from a POP or IMAP internet mail service.<br>The use of this command allows the connection to the mail hosted on the server MAILHOST using the connection parameters specified by USER and PASS. The execution agent is then in waiting mode until MAX_MSG messages are received or the maximum waiting time specified by TIMEOUT is reached. The extracted messages must match the filters such as those specified by the parameters SUBJECT and SENDER. When a message satisfies these criteria, its content as well as its attachments are extracted in a directory specified by the parameter FOLDER. If the parameter KEEP is set to No, the retrieved message is then suppressed from the mailbox. |
| OdiSleep | Waits for <delay> milliseconds. |

## Code Template Tools Not Supported in Oracle Warehouse Builder

The following code template tools are not support in Oracle Warehouse Builder.

*Table 29–2 Unsupported Code Template Tools*

| Unsupported Tools |
|---|
| OdiReverseGetMetaData |
| OdiReverseResetTable |
| OdiReverseSetMetaData |
| OdiDeleteScen |
| OdiExportAllScen |
| OdiExportEnvironmentInformation |
| OdiExportLog |
| OdiExportMaster |
| OdiExportObject |
| OdiExportScen |
| OdiExportWork |
| OdiGenerateAllScen |
| OdiImportObject |
| OdiImportScen |
| OdiDataQuality |
| OdiKillAgent |
| OdiPingAgent |

*Table 29–2 (Cont.) Unsupported Code Template Tools*

| Unsupported Tools |
| --- |
| OdiPurgeLog |
| OdiReinitializeSeq |
| OdiStartScen |
| OdiUpdateAgentSchedule |
| OdiSAPALEClient |
| OdiSAPALEServer |
| OdiWaitForChildSession |
| OdiWaitForData |
| OdiWaitForLogData |
| OdiWaitForTable |
| OdiRefreshJournalCount |
| OdiRetrieveJournalData |
| OdiWaitForData |
| OdiWaitForLogData |
| OdiWaitForTable |

## Code Template Substitution Methods Supported in Warehouse Builder

For the full reference for all tools, see Oracle Data Integrator Tools Reference Manual for ODI 10.1.3.5. Visit the ODI home page on Oracle Technology Network and follow the documentation link at:

`http://www.oracle.com/technology/products/oracle-data-integrator`

Oracle Warehouse Builder supports the following substitution methods, listed with their supported parameters and values.

- getAK
- getAKColList
- getCK
- getColList
- getDataType
- getFK
- getFKColList
- getFilter
- getFilterList
- getFrom
- getGrpBy
- getGrpByList
- getHaving
- getHavingList

- getInfo
- getJDBCConnection
- getJoin
- getJoinList
- getJrnFilter
- getJrnInfo
- getNotNullCol
- getObjectName
- getObjectNameDefaultPSchema
- getOption
- getPK
- getPKColList
- getPop
- GetSchemaName
- getSchemaNameDefaultPSchema
- getSrcTablesList
- getSrcColList
- getSubscriberList
- getTable
- getTargetColList
- getTargetTable
- getUserExit

In the following tables, in the Supported column, Yes indicates that the parameter is supported in Oracle Warehouse Builder, and Yes indicates it is not used in Oracle Warehouse Builder.

### getAK

| Parameter | Supported | Values Supported | Values Not Supported |
|---|---|---|---|
| pPropertyName | Yes | KEY_NAME FULL_NAME MESS | ID flexfield_code |

### getAKColList

| Parameter | Supported | Values Supported | Values Not Supported |
|---|---|---|---|
| Parameter | Supported | Values Supported | Values Not Supported |
| pStart | Yes | | |

| Parameter | Supported | Values Supported | Values Not Supported |
|---|---|---|---|
| pPattern | Yes | COL_NAME POS LONGC SCALE FILE_POS BYTES FILE_END_POS CHECK_ FLOW CHECK_STAT COL_FORMAT EXPRESSION CX_COL_ NAME ALIAS_SEP SOURCE_DT SOURCE_ CRE_DT SOURCE_WRI_ DT DEST_CRE_DT DEST_ WRI_DT DEST_DT | I_COL COL_HEADING COL_DESC IND_WRITE COL_MANDATORY COL_ DEC_SEP REC_CODE_ LIST COL_NULL_IF_ERR DEF_VALUE SCD_COL_ TYPE flexfield_code |
| pSeparator | Yes | | |
| pEnd | Yes | | |

### getCK

| Parameter | Supported | Values Supported | Values Not Supported |
|---|---|---|---|
| Parameter | Supported | Values Supported | Values Not Supported |
| pPropertyName | Yes | COND_ALIAS COND_ NAME COND_TYPE COND_SQL FULL_NAME COND_SQL_DDL MESS | ID |

### getColList

| Parameter | Supported | Values Supported | Values Not Supported |
|---|---|---|---|
| Parameter | Supported | Values Supported | Values Not Supported |
| pStart | Yes | | |
| pPattern | Yes | COL_NAME POS LONGC SCALE FILE_POS BYTES FILE_END_POS CHECK_ FLOW CHECK_STAT COL_FORMAT EXPRESSION CX_COL_ NAME ALIAS_SEP SOURCE_DT SOURCE_ CRE_DT SOURCE_WRI_ DT DEST_DT DEST_CRE_ DT DEST_WRI_DT JDBC_ TYPE | I_COL COL_HEADING COL_DESC IND_WRITE COL_MANDATORY COL_ DEC_SEP REC_CODE_ LIST COL_NULL_IF_ERR DEF_VALUE SCD_COL_ TYPE MANDATORY_ CLAUSE DEFAULT_ CLAUSE flexfield_code |
| pSeparator | Yes | | |
| pEnd | Yes | | |
| pSelector | Yes | INS UPD TRG NULL PK UK REW UD1 UD2 UD3 UD4 UD5 MAP SCD_SK SDK_NK SCD_UPD SCD_ INS SCD_FLAG SCD_ START SCD_END | NEW OLD WS_INS WS_ UPD WS_SEL |

### getDataType

| Parameter | Supported | Values Supported | Values Not Supported |
|---|---|---|---|
| pDataTypeName | Yes | SRC_VARCHAR SRC_NUMERIC SRC_DATE DEST_VARCHAR DEST_NUMERIC DEST_DATE | |
| pDataTypeLength | Yes | | |
| pDataTypePrecision | Yes | | |

### getFK

| Parameter | Supported | Values Supported | Values Not Supported |
|---|---|---|---|
| pPropertyName | Yes | FK_NAME FK_TYPE FK_ALIAS PK_ALIAS PK_CATALOG PK_SCHEMA PK_TABLE_NAME FULL_NAME MESS | ID ID_TABLE_PK COMPLEX_SQL PK_I_MOD flexfield_code |

### getFKColList

| Parameter | Supported | Values Supported | Values Not Supported |
|---|---|---|---|
| pStart | Yes | | |
| pPattern | Yes | COL_NAME POS LONGC SCALE FILE_POS BYTES FILE_END_POS CHECK_FLOW CHECK_STAT COL_FORMAT EXPRESSION CX_COL_NAME ALIAS_SEP SOURCE_DT SOURCE_CRE_DT SOURCE_WRI_DT DEST_CRE_DT DEST_WRI_DT DEST_DT PK_COL_NAME PK_COL_HEADING PK_COL_DESC PK_LONGC PK_SCALE PK_FILE_POS PK_BYTES PK_FILE_END_POS PK_CHECK_FLOW PK_CHECK_STAT PK_COL_FORMAT | I_COL COL_HEADING COL_DESC IND_WRITE COL_MANDATORY COL_DEC_SEP REC_CODE_LIST COL_NULL_IF_ERR DEF_VALUE PK_I_COL PK_POS PK_IND_WRITE PK_COL_MANDATORY PK_COL_DEC_SEP PK_REC_CODE_LIST PK_COL_NULL_IF_ERR PK_DEF_VALUE SCD_COL_TYPE flexfield_code |
| pSeparator | Yes | | |
| pEnd | Yes | | |

### getFilter

| Parameter | Supported | Values Supported | Values Not Supported |
|---|---|---|---|
| no_params | Yes | | |

### getFilterList

| Parameter | Supported | Values Supported | Values Not Supported |
|---|---|---|---|
| pStart | Yes | | |
| pPattern | Yes | EXPRESSION | ID |
| pSeparator | Yes | | |
| pEnd | Yes | | |

### getFrom

| Parameter | Supported | Values Supported | Values Not Supported |
|---|---|---|---|
| no_params | Yes | | |

### getGrpBy

| Parameter | Supported | Values Supported | Values Not Supported |
|---|---|---|---|
| no_params | Yes | | |

### getGrpByList

| Parameter | Supported | Values Supported | Values Not Supported |
|---|---|---|---|
| pStart | Yes | | |
| pPattern | Yes | EXPRESSION | ID |
| pSeparator | Yes | | |
| pEnd | Yes | | |

### getHaving

| Parameter | Supported | Values Supported | Values Not Supported |
|---|---|---|---|
| no_params | Yes | | |

### getHavingList

| Parameter | Supported | Values Supported | Values Not Supported |
|---|---|---|---|
| pStart | Yes | | |
| pPattern | Yes | EXPRESSION | ID |
| pSeparator | Yes | | |
| pEnd | Yes | | |

### getInfo

| Parameter | Supported | Values Supported | Values Not Supported |
|---|---|---|---|
| pPropertyName | Yes | COLL_NAME INT_NAME ERR_NAME TARG_ NAME SRC_CATALOG SRC_SCHEMA SRC_ WORK_CATALOG SRC_ WORK_SCHEMA DEST_ CATALOG DEST_ SCHEMA DEST_WORK_ CATALOG DEST_WORK_ SCHEMA SRC_DSERV_ NAME SRC_JAVA_ DRIVER SRC_JAVA_URL SRC_USER_NAME SRC_ ENCODED_PASS SRC_ TAB_ALIAS_WORD SRC_ DATE_FCT SRC_DDL_ NULL SRC_MAX_COL_ NAME_LEN SRC_MAX_ TAB_NAME_LEN DEST_ DSERV_NAME DEST_ JAVA_DRIVER DEST_ JAVA_URL DEST_USER_ NAME DEST_ENCODED_ PASS DEST_COL_ALIAS_ WORD DEST_TAB_ ALIAS_WORD DEST_ DATE_FCT DEST_DDL_ NULL DEST_MAX_COL_ NAME_LEN DEST_MAX_ TAB_NAME_LEN CT_ ERR_TYPE CT_ORIGIN JRN_NAME JRN_VIEW JRN_DATA_VIEW JRN_ TRIGGER JRN_ITRIGGER JRN_UTRIGGER JRN_ DTRIGGER SUBSCRIBER_ TABLE CDC_SET_TABLE CDC_TABLE_TABLE CDC_SUBS_TABLE CDC_ OBJECTS_TABLE | |

### getJDBCConnection

| Parameter | Supported | Values Supported | Values Not Supported |
|---|---|---|---|
| pPropertyName | Yes | SRC DEST | |

### getJoin

| Parameter | Supported | Values Supported | Values Not Supported |
|---|---|---|---|
| no_params | Yes | | |

### getJoinList

| Parameter | Supported | Values Supported | Values Not Supported |
|---|---|---|---|
| pStart | Yes | | |
| pPattern | Yes | EXPRESSION | ID |
| pSeparator | Yes | | |
| pEnd | Yes | | |

### getJrnInfo

| Parameter | Supported | Values Supported | Values Not Supported |
|---|---|---|---|
| pPropertyName | Yes | FULL_TABLE_NAME JRN_FULL_NAME JRN_FULL_VIEW JRN_FULL_DATA_VIEW JRN_FULL_TRIGGER JRN_FULL_ITRIGGER JRN_FULL_UTRIGGER JRN_FULL_DTRIGGER JRN_SUBSCRIBER JRN_NAME JRN_VIEW JRN_DATA_VIEW JRN_TRIGGER JRN_ITRIGGER JRN_UTRIGGER JRN_DTRIGGER JRN_SUBSCRIBER JRN_COD_MODE JRN_METHOD CDC_SET_TABLE CDC_TABLE_TABLE CDC_SUBS_TABLE CDC_OBJECTS_TABLE | |

### getJrnFilter

No parameters.

### getNotNullCol

| Parameter | Supported | Values Supported | Values Not Supported |
|---|---|---|---|
| pPropertyName | Yes | COL_NAME MESS | ID flexfield_code |

### getObjectName

| Parameter | Supported | Values Supported | Values Not Supported |
|---|---|---|---|
| pMode | Yes | L R | |
| pObjectName | Yes | | |
| pLogicalSchemaName | No | | |
| pContextName | No | | |
| pLocation | Yes | W D | |

### getObjectNameDefaultPSchema

| Parameter | Supported | Values Supported | Values Not Supported |
|---|---|---|---|
| `pPropertyName` | | | |

### getOption

| Parameter | Supported | Values Supported | Values Not Supported |
|---|---|---|---|
| pPropertyName | Yes | | |

### getPK

| Parameter | Supported | Values Supported | Values Not Supported |
|---|---|---|---|
| pPropertyName | Yes | KEY_NAME FULL_NAME MESS | ID flexfield_code |

### getPKColList

| Parameter | Supported | Values Supported | Values Not Supported |
|---|---|---|---|
| pStart | Yes | | |
| pPattern | Yes | COL_NAME POS LONGC SCALE FILE_POS BYTES FILE_END_POS CHECK_ FLOW CHECK_STAT COL_FORMAT EXPRESSION CX_COL_ NAME ALIAS_SEP SOURCE_DT SOURCE_ CRE_DT SOURCE_WRI_ DT DEST_CRE_DT DEST_ WRI_DT DEST_DT PK_ COL_NAME PK_COL_ HEADING PK_COL_DESC PK_LONGC PK_SCALE PK_FILE_POS PK_BYTES PK_FILE_END_POS PK_ CHECK_FLOW PK_ CHECK_STAT PK_COL_ FORMAT | I_COL COL_HEADING COL_DESC IND_WRITE COL_MANDATORY COL_ DEC_SEP REC_CODE_ LIST COL_NULL_IF_ERR DEF_VALUE SCD_COL_ TYPE flexfield_code |
| pSeparator | Yes | | |
| pEnd | Yes | | |

### getPop

| Parameter | Supported | Values Supported | Values Not Supported |
|---|---|---|---|
| pPropertyName | Yes | POP_NAME TABLE_ NAME KEY_NAME DISTINCT_ROWS MAX_ ERR MAX_ERR_PRCT IKM LKM CKM HAS_JRN | I_POP FOLDER IND_ WORK_TARG LSCHEMA_ NAME DESCRIPTION WSTAGE OPT_CTX TARG_CTX flexfield_code |

### GetSchemaName

| Parameter | Supported | Values Supported | Values Not Supported |
|---|---|---|---|
| `pLogicalSchemaName` | | | |
| `pContextCode` | | | |
| `pLocation` | | | |

### getSchemaNameDefaultPSchema

| Parameter | Supported | Values Supported | Values Not Supported |
|---|---|---|---|
| `pLogicalSchemaName` | | | |
| `pContextCode` | | | |
| `pLocation` | | | |

### getSrcTablesList

| Parameter | Supported | Values Supported | Values Not Supported |
|---|---|---|---|
| pStart | Yes | | |
| pPattern | Yes | TABLE_NAME RES_NAME CATALOG WORK_CATALOG SCHEMA WORK_SCHEMA TABLE_ALIAS FILE_FORMAT FILE_SEP_FIELD SFILE_SEP_FIELD FILE_ENC_FIELD FILE_SEP_ROW SFILE_SEP_ROW FILE_FIRST_ROW IND_JRN JRN_ORDER | I_TABLE MODEL_NAME SUB_MODEL_NAME TECHNO_NAME LSCHEMA_NAME POP_TAB_ALIAS TABLE_TYPE DESCRIPTION R_COUNT XFILE_SEP_FIELD XFILE_SEP_ROW FILE_DEC_SEP METADATA OLAP_TYPE flexfield_code |
| pSeparator | Yes | | |
| pEnd | Yes | | |

### getSrcColList

| Parameter | Supported | Values Supported | Values Not Supported |
|---|---|---|---|
| pStart | | | |
| pUnMappedPattern | | | |
| pMappedPattern | | | |
| pSeparator | | | |
| pEnd | | | |

### getSubscriberList

| Parameter | Supported | Values Supported | Values Not Supported |
|---|---|---|---|
| pStart | Yes | | |
| pPattern | Yes | SUBSCRIBER | |
| pSeparator | Yes | | |
| pEnd | Yes | | |

### getTable

| Parameter | Supported | Values Supported | Values Not Supported |
|---|---|---|---|
| pMode | Yes | L R A | |
| pProperty | Yes | TARG_NAME COLL_NAME INT_NAME ERR_NAME CHECK_NAME CT_NAME FK_PK_TABLE_NAME JRN_NAME JRN_VIEW JRN_DATA_VIEW JRN_TRIGGER JRN_ITRIGGER JRN_UTRIGGER JRN_DTRIGGER SUBSCRIBER_TABLE CDC_SET_TABLE CDC_TABLE_TABLE CDC_SUBS_TABLE CDC_OBJECTS_TABLE | ID flexfield_code |
| pLocation | Yes | W D A | |

### getTargetColList

| Parameter | Supported | Values Supported | Values Not Supported |
|---|---|---|---|
| pStart | Yes | | |
| pPattern | Yes | COL_NAME POS LONGC SCALE FILE_POS BYTES FILE_END_POS CHECK_FLOW CHECK_STAT COL_FORMAT ALIAS_SEP SOURCE_DT SOURCE_CRE_DT SOURCE_WRI_DT DEST_DT DEST_CRE_DT DEST_WRI_DT JDBC_TYPE | I_COL COL_HEADING COL_DESC IND_WRITE COL_MANDATORY COL_DEC_SEP REC_CODE_LIST COL_NULL_IF_ERR DEF_VALUE SCD_COL_TYPE MANDATORY_CLAUSE DEFAULT_CLAUSE COL_DESC flexfield_code |
| pSeparator | Yes | | |
| pEnd | Yes | | |
| pSelector | Yes | | |

### getTargetTable

| Parameter | Supported | Values Supported | Values Not Supported |
|---|---|---|---|
| pPropertyName | Yes | TECHNO_NAME TABLE_NAME RES_NAME CATALOG WORK_CATALOG SCHEMA WORK_SCHEMA TABLE_ALIAS FILE_FORMAT FILE_SEP_FIELD SFILE_SEP_FIELD FILE_ENC_FIELD FILE_SEP_ROW SFILE_SEP_ROW FILE_FIRST_ROW IND_JRN JRN_ORDER | MODEL_NAME SUB_MODEL_NAME LSCHEMA_NAME TABLE_TYPE DESCRIPTION TABLE_DESC R_COUNT FILE_DEC_SEP XFILE_SEP_FIELD XFILE_SEP_ROW METADATA_DESC OLAP_TYPE TABLE_DESC WS_NAME WS_NAMESPACE WS_JAVA_PACKAGE WS_ENTITY_NAME WS_DATA_SOURCE flexfield_code |

### getUserExit

| Parameter | Supported | Values Supported | Values Not Supported |
|---|---|---|---|
| pOptionName | | | |

## Code Template Substitution Methods Not Supported in Warehouse Builder

This section lists code template substitution methods not supported in Warehouse builder.

- getCatalogName
- getCatalogNameDefaultPSchema
- getColDefaultValue
- getColumn
- getContext
- getFlexFieldValue
- getIndex
- getIndexColList
- getJRNFilter
- getModel
- getNewColComment
- getNewTableComment
- getPrevStepLog
- getSession
- getSessionVarList
- getStep
- getSysDate

- hasPK
- isColAttrChanged
- nextAK
- nextCond
- nextFK
- setNbDelete
- setNbErrors
- setNbInsert
- setNbRows
- setNbUpdate

# Index

## A


about
  consuming web services, 16-3
  public web services, 16-4
  publishing web services, 16-3
accessing
  transformation libraries, 4-9
ACTIVE_DATE attribute
  about, 6-12
  in cubes, 6-12
activities
  AND, 27-5
  assign, 27-6
  control, 27-3
  data auditor monitor, 27-6
  email, 27-10
  Enterprise Java Beans, 27-7
  file exists, 27-12
  For Loop, 27-14
  FORK, 27-13
  ftp, 27-14
  in process flows, 8-8
  Java Class, 27-17
  manual, 27-19
  mapping, 27-19
  OMBPlus, 27-22
  OR, 27-23
  OWB-specific, 27-1
  route, 27-23
  Set Status, 27-24
  sqlplus, 27-24
  start, 27-26
  user-defined, 27-27
  utility, 27-2
  wait, 27-29
  web service, 27-29
  While Loop, 27-29
activity templates, 8-11
adding
  groups to mappings, 26-2
  mapping operators, 5-13
adding operators
  Add Operator dialog box, 5-15
addresses, cleansing, 22-2
administrative transformations, 28-1
advanced queues
  about, 2-43
  configuring, 2-54
  defining, 2-44
  editing, 2-44
advantages
  web services, 16-2
Aggregate function, 26-8
aggregating data, 26-5
Aggregator operator, 26-5
  ALL, 26-8
  DISTINCT, 26-8
analytic workspace, 3-13
AND activity, 27-5
Anydata Cast operator, 26-9
applying
  data rules, 19-7
assign activity, 27-6
attribute properties, setting, 25-7
attribute sets, 2-33
  about, 2-33
  creating, 2-34
  editing, 2-34
  tables, 2-13
attributes
  connecting, 5-17
  defining, 2-13
audit details, removing, 13-23
auditing
  CT mappings, 7-31
  CT mappings, prerequisites, 7-32
  deployments, 13-1 to 13-24
  executions, 13-1 to 13-24
auto binding
  dimensional objects, 3-10
  rules, 3-29
  steps, 3-10
auto solving
  MOLAP cubes, 3-52


## B


best practices
  naming data objects, 2-8
BINARY_DOUBLE data type, 2-3
BINARY_FLOAT data type, 2-3

binding
about, 3-10
auto binding, 3-10
auto binding, rules, 3-29
auto binding, steps, 3-10
manual binding, 3-11
manual binding, steps, 3-11
unbinding, 3-12
when to perform, 3-10
BLOB data type, 2-3
building expressions, 26-3

## C

calculated measures
about, 3-39
Calculated Measures Wizard
about, 3-50
CASS reporting, 22-5
CCA
starting, 7-23
stopping, 7-23
change data capture
commands, 7-34
performing using CT mappings, 7-33
changes
rolling out to the target schema, 12-12
CHAR data type, 2-4, 26-46
character transformations, 28-9
check key constraints, 2-21
chunking, 25-27
cleansing
addresses, 22-2
names, 22-2
CLOB data type, 2-4
code generation
configuring target directories, 2-46
options, 24-4
code template mappings
*See* CT mappings
code template substitution methods
not supported in OWB, 29-15
supported in OWB, 29-5
code template tools
not supported in OWB, 29-4
supported in OWB, 29-1
code templates
prebuilt, 7-13
commit strategies
committing multiple mappings, 10-10
committing to multiple targets, 10-7
comparing process runs, 13-21
composite partitions about, 2-30
configuration parameters
ABAP extension, 2-47
ABAP run parameter file, 2-47
ABAP spool directory, 2-47
advanced queues, 2-54
archive directory, 2-48
base tables, 2-51
buffer cache, 2-50, 2-52
build, 2-51
data profiles, 18-8
data segment compression, 2-50, 2-52
DDL directory, 2-47
DDL extension, 2-47
DDL spool directory, 2-47
default index tablespace, 2-48
default object tablespace, 2-48
default rollback segment, 2-51
deployable, 2-53, 3-37, 3-55
deployment options, 3-55
dimension, 3-37
end of line, 2-47
error table name, 2-49, 2-52
error table only, 2-49, 2-53
for update, 2-51
hash partition tablespace list, 2-53
input directory, 2-48
invalid directory, 2-48
lib directory, 2-47
lib extension, 2-47
lib spool directory, 2-47
loader directory, 2-47
loader extension, 2-47
loader run parameter file, 2-47
local rollback segment, 2-51
location, 2-48
log directory, 2-48
logging mode, 2-50, 2-52
master rollback segment, 2-51
materialized view index tablespace, 3-56
materialized view tablespace, 3-56
next date, 2-51
overflow tablespace list, 2-50
parallel access mode, 2-50, 2-53
parallel degree, 2-50, 2-53
partition tablespace list, 2-50
PL/SQL directory, 2-47
PL/SQL extension, 2-47
PL/SQL Generation Mode, 2-46
PL/SQL run parameter file, 2-47
PL/SQL spool directory, 2-47
query rewrite, 2-51
queue propagations, 2-55
queue tables, 2-55
receive directory, 2-48
refresh, 2-52
refresh on, 2-52
row movement, 2-50
row-level dependency, 2-50
sequences, 2-54
sort directory, 2-48
source file operator, 24-13
SQLPlus directory, 2-47
SQLPlus extension, 2-47
SQLPlus run parameter file, 2-47
staging file directory, 2-47
start with, 2-52
statistics collection, 2-50

tablespace, 2-49, 2-50, 2-52
Tcl Directory, 2-47
using constraints, 2-52
work directory, 2-48
configuring
advanced queues, 2-54
cubes, 3-55
data auditors, 20-4
data objects, 1-4
data profiles, 18-7
dimensions, 3-37
flat file operators, 24-13
mapping sources and targets, 26-41
mappings, 5-27
master-detail mappings, 10-17
master-detail mappings, direct path load, 10-20
materialized views, 2-51
Name and Address server, 22-23
PL/SQL mappings, 10-1
queue propagations, 2-55
queue tables, 2-55
runtime parameters, 2-46
runtime parameters, SAP files, 7-6
SAP, loading type parameter, 7-5
sequences, 2-54
tables, 2-48
target modules, 2-46
transportable modules, 17-12
views, 2-54
configuring CT mappings
source file operator, 7-31
connecting
attributes, 5-17
groups, 5-17
operators, 5-16
connecting groups
connection options, 5-20
Mapping Connection Dialog box, 5-18
connecting operators
connection options, 5-20
Mapping Connection Dialog box, 5-18
connections, updating, 13-24
Constant operator, 25-9
constants, defining, 25-7
constraints
about, 2-20
check constraints, creating, 2-23
check key, about, 2-21
editing, 2-23
foreign key, about, 2-21
foreign key, creating, 2-22
primary key, about, 2-21
primary key, creating, 2-22
types, 2-21
unique key, about, 2-21
unique key, creating, 2-22
Construct Object operator, 25-9
control activities, 27-3
Control Center Agent
starting, 7-23
stopping, 7-23
Control Center reports, 13-12
control center transformations, 28-12
Control CTs
in CT mappings, 7-40
control rows
about, 3-30
conventional path loading
for master-detail relationships, 10-15
master-detail flat files, 10-16
conversion transformations, 28-17
correlated commit, design considerations, 10-8
Create Cube Wizard
default values, 3-45
Create Dimension wizard
defaults used, 3-20
creating
attribute sets, 2-34
constraints, check constraints, 2-23
constraints, foreign key constraints, 2-22
constraints, unique key constraints, 2-22
CT mapping modules, 7-18
CT mappings, 7-12
cubes, using Cube Editor, 3-45
cubes, using wizard, 3-42
data auditors, 20-3
data profiles, 18-6
data rules, 19-5
dimensions, using Dimension Editor, 3-22
dimensions, using wizard, 3-15
display sets, 5-24
expressions, 26-3
indexes, 2-24
mappings, 5-1
physical objects, 2-46
PL/SQL types, 9-7, 9-9
pluggable mapping folders, 5-41
pluggable mappings, 5-39
primary key constraints, 2-22
process flows, 8-8
time dimensions, using the Time Dimension Wizard, 3-57
type 2 SCDs, 3-19
type 3 SCDs, 3-19
web service packages, 16-6
web services, based on URL, 16-10
web services, based on Warehouse Builder objects, 16-6
CT mapping modules
creating, 7-18
CT mappings, 5-3
about, 7-12
auditing, 7-31
auditing, prerequisites, 7-32
creating, 7-12
deploying, 7-26
executing, 7-27
generated scripts, 7-24
generating, 7-24
moving data from heterogeneous databases, 7-45

performing change data capture, 7-33
performing ETL, 7-17
sample generated code, 7-25
types, 7-13
usage, 5-3, 7-12
using Control CTs, 7-40
using Oracle Target CTs, 7-43
validating, 7-23
viewing execution results, 7-27
Cube operator, 25-10
cubes
about, 3-7
ACTIVE_DATE attribute, 6-12
auto solving, MOLAP cubes, 3-52
calculated measures, 3-39
Calculated Measures Wizard, 3-50
calculated measures, types, 3-39
compression, defining, 3-48
configuring, 3-55
creating, using Cube Editor, 3-45
creating, using wizard, 3-42
deployment options, 3-13
deployment options, Deploy Data Object Only, 3-13
deployment options, Deploy to Catalog Only, 3-13
dimensionality, 3-44
editing, 3-55
example, 3-42
loading data into, 6-13
loading, using type 2 slowly changing dimensions, 6-13
measures, creating, 3-44
orphan management policy, 3-51
parallel solving, 3-54
partitioning, along dimension, 3-49
performing ETL, 6-12
physical bindings, 3-51
ragged data, 3-52
solving cube measures, 3-53
solving, independent of loading, 3-53
sparsity, defining, 3-48
sparsity, guidelines, 3-48
storing, 3-42
custom transformations
about, 4-7
defining, 9-2
editing, 9-11

## D

data
aggregating, 26-5
cleansing, 22-2
test, 5-51
viewing, 2-9
data auditor monitor activities, 27-6
data auditors
configuring, 20-4
creating, 20-3
granting privileges on error tables, 20-9
using, 20-6
viewing error tables, 13-23, 20-8
data flow operators, 26-1
Data Generator operator, 25-12
data objects
about, 2-2
advanced queues, 2-41
best practices for naming, 2-8
data type for columns, 2-3
defining, 2-1 to 2-54
dimensional objects, implementing, 3-9
generating, 1-6
identifying deployment location, 13-22
list, 2-2
monitoring data quality, 20-2
naming conventions, 2-8
overview, 2-1
sequences, 2-35
SQL Server and IBM DB2, 2-57
used in map run, 13-22
validating, 1-4
viewing, 2-9
viewing data, 2-9
data objects, generating, saving scripts, 1-7
data profile
adding objects, 18-22
Data Profile Editor
components, 18-3
data profiles
adding data objects, 18-22
configuration parameters, 18-8
configuring, 18-7
creating, 18-6
data profiling
generating corrections, 21-2
performance tuning, 18-22
performing, 18-4
steps, 18-6
viewing corrections, 21-7
viewing results, 18-11
data quality
Match Merge operator, 23-1
data rules, 2-13
about, 19-1
applying, 19-7
applying to ETL mappings, 19-8
creating, 19-5
deriving, 19-4
editing, 19-6
types, 19-2
using, 19-3
data transformation
about, 4-1
data types
list of supported, 2-3
Data Viewer, 2-9
Data Watch and Repair for MDM, performing, 18-24
DATE data type, 2-4
date transformations, 28-18

DB2
extracting data using CT mappings, 7-45
debugging
map runs, 13-21
mappings, 5-49
processes, 13-20
starting point, 5-55
Deduplicator operator, 26-10
DISTINCT, 26-10
default deployment time setting, 13-22
defining
advanced queues, 2-44
constants, 25-7
cube sparsity, 3-48
data objects, 2-1 to 2-54
dimensional objects, 3-1 to 3-63
error tables, 2-9
ETL process for SAP objects, 7-4
execution units, 7-19
hash by quantity partitions, 2-28
hash partitions, 2-27
indexes, 2-13
list partitions, 2-28
mappings, 5-1
materialized views, 2-18
process flows, 8-5
queue propagations, 2-45
queue tables, 2-42
range partitions, 2-26
schedules, 11-2
sequences, 2-35
SQL Server and IBM DB2 data objects, 2-57
tables, 2-11
test data, 5-51
type 2 SCDs, 3-32
type 3 SCDs, 3-35
views, 2-15, 2-17
defining indexes, 2-13
defining tables, 2-11
definitions
transportable modules, 17-11
deleting
groups from mappings, 26-2
Deploy All, 3-13
deploying
about, 12-1
CT mappings, 7-26
data objects, 12-6
deployment actions, 12-2
deployment errors, 15-1
deployment results, 12-8
deployment status, 12-3
process flows, 8-2
reporting on, 13-12, 13-13
tables in transportable modules, 17-12
transportable module, 17-15
web services, 16-9
web services, prerequisites, 16-9
deployment actions, 12-2
deployment and execution
steps, 12-5
deployment options
Deploy All, 3-13
Deploy to Catalog Only, 3-13
deployment reports, 13-12, 13-13
deployment time settings, 13-22
deployments
auditing, 13-1 to 13-24
identifying, 13-21
deriving
data rules, 19-4
Design Center
in Repository Browser, 13-7
designing
process flows, 8-1
target schemas, 1-1
target schemas, dimensional, 1-3
target schemas, relational, 1-2
diagrams
impact analysis, 14-3 to 14-6
lineage, 14-3 to 14-6
Dimension operator, 25-14
dimensional object
deployment options, Deploy All, 3-13
dimensional objects
about, 3-1
binding, 3-10
creating, about, 3-1
defining, 3-1 to 3-63
deployment options, Data Objects Only, 3-13
deployment options, Deploy to Catalog Only, 3-13
implementing, about, 3-9
orphan management policy, 3-7
unbinding, 3-12
dimensions
about, 3-2
binding, 3-29
configuring, 3-37
control rows, 3-30
creating, using Dimension Editor, 3-22
creating, using wizard, 3-15
default settings, using wizard, 3-20
deployment options, 3-13
deployment options, Deploy Data Object Only, 3-13
deployment options, Deploy to Catalog Only, 3-13
determining number of rows, 3-31
dimension attributes
creating, 3-17
editing, 3-36
error tables, 3-8
example, 3-14
extracting data from, 6-8
hierarchies, creating, 3-18, 3-26
level attributes, creating, 3-18
levels, creating, 3-18
linking to fact data, 3-30
loading data, 6-1

orphan row management, 3-8
loading, example, 6-2
orphan management policy, specifying, 3-28
performing ETL, 6-1
removing data, 6-10
orphan row management, 3-8
removing data, example, 6-11
ROLAP dimension limitations, 3-30
specifying default parent, 3-28
storing, 3-15
surrogate identifiers, 3-3
direct path loading
for master-detail relationships, 10-18
master-detail flat files, 10-20
display sets
creating, 5-24
defined, 5-24
DISTINCT
in the Aggregator operator, 26-8
in the Deduplicator operator, 26-10
DML error logging
about, 15-4
enabling, 15-5
in ETL, 15-5
limitations, 15-6
dynamic population
time dimensions, 3-64
dynamically populating, 3-64

## E

editing
advanced queues, 2-44
attribute sets, 2-34
constraints, 2-23
cubes, 3-55
data rules, 19-6
dimensions, 3-36
invalid objects, 1-6
materialized views, 2-20
PL/SQL types, 9-12
queue propagations, 2-45
queue tables, 2-43
schedules, 11-3
sequences, 2-35
table definitions, 2-13
time dimensions, 3-61
transformation properties, 9-11
transportable modules, 17-17
views, 2-17
effective date
about, 3-3
mapping source attributes, 6-4
email activity, 27-10
enabling
DML error logging, 15-5
hierarchy versioning, 3-6
Enterprise Java Beans activity, 27-7
error logs
interpreting error logs, 15-1
error tables, 3-8
about, 2-9, 15-4
columns, 2-10
defining, 2-9
granting privileges, 20-9
ETL
improving runtime performance, 10-1
ETL objects
scheduling, 11-1
example
cubes, 3-42
dimensions, 3-14
type 2 slowly changing dimension, 3-4
type 3 slowly changing dimension, 3-6
examples
checking data constraints using CT mappings, 7-41
consuming web services in process flows, 16-22
extracting data from IBM DB2 using CT mappings, 7-45
integrating web services with Oracle BPEL Process Manager, 16-23
loading data into Type 2 SCDs, 6-6
loading data into Type 3 SCDs, 6-6
loading dimensions, 6-2
loading transaction data, 5-33
performing change data capture using CT mappings, 7-33
publishing mappings as web services, 16-21
removing data from dimensions, 6-11
using Oracle Target CTs, 7-43
executing
CT mappings, 7-27
mappings from SQL*Plus, 10-11
reports, 13-12, 13-16
web services, using browser, 16-12
web services, using Control Center Manager, 16-11
execution
about, 12-4
auditing, 13-1 to 13-24
errors, 15-1
execution reports, 13-12, 13-16
execution units
adding operators, 7-21
creating default execution units, 7-22
default code templates, 7-22
defining, 7-19
removing, 7-21
removing operators, 7-21
execution view
mapping editor, 5-6
menu, 7-20
toolbar, 7-20
Expand Object operator, 25-18
expiration date
about, 3-4
Expression Builder
about, 26-3
opening, 26-3

Expression operator, 26-11
expressions, creating, 26-3
External Table operator, 25-19
extracting
  data from DB2 into Oracle Database, 7-45
  dimension data, 6-8
  type 2 slowly changing dimension data, 6-8
  type 3 slowly changing dimension data, 6-9
extracting from master-detail flat files, 10-13, 10-15

## F

fast refresh, 2-53
File Exists activity, 27-12
file transfer
  in process flows, 8-22
Filter operator, 26-12
filters, with a transform, 9-13
first class objects
  about, 2-7
Flat File operator, 25-32
flat files
  configuration, 24-13
  configuring master-detail mappings, 10-20
  extracting master and detail records, 10-15
  importing master-detail flat files, 10-15
  mapping, 7-1
  master-detail mappings, post-update scripts for direct path loads, 10-20
  master-detail, example, 10-14
  master-detail, extracting from, 10-13
  master-detail, operations after initial load, 10-18
  variable names, in process flows, 8-18
FLOAT data type, 2-4
For Loop activity, 27-14
foreign key constraints, 2-21
foreign keys, ensuring referential integrity, 10-13
FORK activity, 27-13
FTP
  using in process flows, 8-22
ftp activity, 27-14
full outer joins, 26-15, 26-19
functions
  Aggregate, 26-8
  as transformations, 4-7
  defining, 9-2
  editing, 9-11

## G

generating
  data objects, 1-6
  transportable module, 17-15
  web services, 16-8
generating corrections, 21-2
generation
  errors, 15-1
  saving scripts, 1-7
  viewing results, 1-7
  viewing scripts, 1-7
Group By clause, 26-6
groups
  adding to mappings, 26-2
  connecting, 5-17
  in LIA diagrams, 14-5
  removing from mappings, 26-2

## H

hash by quantity partitions
  about, 2-28
  defining, 2-28
hash partitions
  about, 2-27
  defining, 2-27
Having clause, 26-7
hierarchies
  creating, 3-18, 3-26
hierarchy versioning
  about, 3-5
  enabling, 3-6
householding, 23-1, 23-25

## I

IBM DB2 data objects
  naming rules, 2-57
impact analysis
  rolling out changes to target schema, 12-12
impact analysis diagrams, 14-3 to 14-6
implementation
  dimensional objects, MOLAP, 3-13
  dimensional objects, relational, 3-9
  dimensional objects, ROLAP, 3-12
  dimensional objects, ROLAP with MVs, 3-12
implementing
  dimensional objects, 3-9
  reporting on, 13-11
importing
  master-detail flat files, 10-15
  transformations, 9-13
improving runtime performance, 10-1
index partitioning, 2-24, 2-32
  local index, 2-32
index partitions
  about, 2-32
indexes
  about, 2-24
  creating, 2-24
  defining, 2-13
  types, 2-24
input signature, 5-40
installation
  errors, 15-1
INTEGER data type, 2-4
INTERVAL DAY TO SECOND data type, 2-4
INTERVAL YEAR TO MONTH data type, 2-4

## J

Java Class activity, 27-17

join conditions
default, 26-14
foreign keys, 26-14
join input roles, 26-19
Joiner operator, 26-13, 26-14, 26-15, 26-19
join conditions, 26-14
with SAP Sources, 26-19
joining multiple row sets, 26-13
joins, full outer, 26-15, 26-19
joins, left outer, 26-19
joins, outer, 26-15
joins, right outer, 26-19

## K

Key Lookup operator, 26-23

## L

Language parameter
SAP, 7-6
LCR Cast operator, 26-22
LCR Splitter operator, 26-23
limitations
prebuilt code templates, 7-16
lineage diagrams, 14-3 to 14-6
list partitions
about, 2-28
defining, 2-28
example, 2-29
loading
conventional path for master-detail targets, 10-16
cubes, 6-13
cubes, using type 2 slowly changing dimensions, 6-13
data from materialized view, 25-22
dimension data, 6-1
dimension data, example, 6-2
direct path for master-detail targets, 10-20
master and detail records, 10-15
master-detail relationships, 10-15, 10-18
master-detail relationships, direct path, 10-18
transaction data, 5-33
type 2 slowly changing dimensions, 6-3
type 2 slowly changing dimensions, example, 6-6
type 3 slowly changing dimensions, 6-5
type 3 slowly changing dimensions, example, 6-6
loading types, 25-3
for SAP, 7-5
locations
data objects deployed to, 13-22
of transportable modules, 17-7
unregistering, 13-24
updating connection details, 13-24
logs
interpreting error logs, 15-1
LONG data type, 2-4
LONG RAW data type, 2-4
Lookup operator, 26-23

## M

main procedure, 10-11
management reports, 13-13, 13-19
manual activity, 27-19
manual binding
dimensional objects, 3-11
steps, 3-11
map runs, 13-21, 13-22
mapping activity, 27-19
mapping debugger
restrictions, 5-49
mapping editor
execution view, 5-6
logical view, 5-6
mapping operators
about, 4-2
adding, 5-13
Aggregator operator, 26-5
Anydata Cast, 26-9
connecting, 5-16
Constant, 25-9
Construct Object, 25-9
Cube, 25-10
Data Generator, 25-12
Deduplicator, 26-10
Dimension, 25-14
editing, 5-21
Expand Object, 25-18
Expression, 26-11
External Table, 25-19
Filter, 26-12
Flat File, 25-32
Joiner, 26-13
LCR Cast, 26-22
LCR Splitter, 26-23
Lookup, 26-23
Mapping Output Parameter, 25-21
Match Merge, 23-1
Materialized View, 25-22
Name and Address, 22-2
Pivot, 26-28
Post-Mapping Process, 26-34
Pre-Mapping Process, 26-35
Queue, 25-23
Sequence, 25-25
Set Operation, 26-36
Sorter, 26-37
Splitter operator, 26-39
Subquery Filter, 26-41
Table, 25-26
Table Function, 26-43
Transformation, 26-46
types of, 4-3
Unpivot, 26-47
Varray Iterator, 25-29
View, 25-30
Mapping Output Parameter operator, 25-21
mapping output parameters, 25-21
mappings
about, 4-2, 5-1

accessing data via transportable modules, 17-17
adding self joins, 26-13
applying data rules, 19-8
configuring, 5-27, 10-1
configuring master-detail, 10-17
creating, 5-1
debugging, 5-49
defining, 5-1
executing from SQL*Plus, 10-11
for flat files, 7-1
for PEL, 10-22
groups, 26-2
loading targets, order in which, 5-26
master-detail mappings, 10-13
naming conventions, 5-12, 8-11
operators, 4-2
performing ETL, 5-9
PL/SQL mappings, 10-1
PL/SQL mappings, example, 5-7
runtime parameters, 24-1
searching, attributes, 5-46
searching, groups, 5-46
searching, operators, 5-46
sources and targets, configuring, 26-41
spotlighting operators, 5-45
target load order, 5-26
types, 5-2
types. SAP ABAP mappings, 5-3
types, CT mappings, 5-3
types, PL/SQL mappings, 5-3
types, SQL*Loader, 5-3
ungrouping operators, 5-45
using web services, 16-17
master-detail flat files
as sources, about, 10-13
configuring mappings, 10-17
configuring mappings, direct path load, 10-20
example of a master-detail flat file, 10-14
extracting from, 10-15
extracting from, using conventional path load, 10-15
extracting from, using direct path load, 10-18
importing and sampling, 10-15
operations after initial load, 10-18
performance, 10-15, 10-18
post-update scripts for direct path loads, 10-20
RECNUM, 10-19
sample mapping, conventional path loading, 10-16
sample mapping, direct path loading, 10-20
Match Merge operator, 23-1
Match rules
multiple match rules, 23-9
transitive match rules, 23-10
match rules
address match rules, 23-16
conditional match rules, 23-5
custom match rules, 23-18
firm match rules, 23-14
person match rules, 23-12
weight match rules, 23-10
matching
transitive, 23-10
Match-Merge operator
custom rules, 23-21
design considerations, 23-24
example, 23-8
match rules, 23-5
merge rules, 23-19
restrictions, 23-24
using, 23-22
Materialized View operator, 25-22
materialized views
about, 2-18
attribute sets, adding, 2-20
attribute sets, deleting, 2-20
attribute sets, editing, 2-20
columns, adding, 2-20
columns, deleting, 2-20
columns, editing, 2-20
configuring, 2-51
constraints, adding, 2-20
constraints, deleting, 2-20
constraints, editing, 2-20
defining, 2-18
defining attribute sets, 2-19
defining columns, 2-19
defining constraints, 2-19
defining data rules, 2-20
defining indexes, 2-19
defining partitions, 2-19
defining query, 2-19
editing, 2-20
fast refresh, 2-53
loading data from, 25-22
loading data into, 25-22
renaming, 2-20
update definitions, 2-20
MDSYS.SDO_DIM_ARRAY data type, 2-4
MDSYS.SDO_DIM_ELEMENT data type, 2-4
MDSYS.SDO_ELEM_INFO_ARRAY data type, 2-4
MDSYS.SDO_GEOMETRY data type, 2-4
MDSYS.SDO_ORDINATE_ARRAY data type, 2-4
MDSYS.SDO_POINT_TYPE data type, 2-4
MDSYS.SDOAGGRTYPE data type, 2-4
Merge rules, 23-21
metadata
dependencies, 14-1 to 14-8
import and export errors, 15-1
metadata dependencies, diagrams of, 14-1
Metadata Dependency Manager, 14-1
minus, in the Set Operation operator, 26-37
modules
configuring target modules, 2-46
process flows, 8-2, 8-6
MOLAP implementation
about, 3-13
dimensional objects, 3-13
monitoring
data objects, using auditors, 20-6

data quality, 20-2
monitoring process runs, 13-22
moving data from heterogeneous databases
using CT mappings, 7-45
multiple-record-type flat files
master-detail structure, 10-13
master-detail structure, example of, 10-14
multitable INSERT, 26-40

## N

Name and Address
country postal certifications, Australia, 22-6
country postal certifications, Canada, 22-5
country postal certifications, United States, 22-5
operator, 22-2
purchasing license, 22-1
Name and Address operator, 22-2
best practices, 22-18
CASS reporting, 22-5
enabling, 22-2
input roles, 22-6
output components, 22-8
Name and Address server, 22-23
configuring, 22-23
errors, 15-1
starting, 22-24
stopping, 22-24
names
cleansing, 22-2
flat files with variable names in process flows, 8-18
names and addresses, processing libraries, 22-23
naming conventions
data objects, 2-8
naming rules
IBM DB2 data objects, 2-57
SQL Server data objects, 2-58
navigating
Repository Browser, 13-7
NCHAR data type, 2-5
NCLOB data type, 2-5
nested tables
creating, 2-40
editing, 2-41
overview, 2-40
NUMBER data type, 2-5
number transformations, 28-30
NVARCHAR2 data type, 2-5

## O

object class definition
about, 2-7
object properties, report, 13-8
object types
creating, 2-37
editing, 2-38
overview, 2-36
objects
invalid objects, editing, 1-6
reports, 13-9
OLAP transformations, 28-33
OMBPlus activity, 27-22
opening
Expression Builder, 26-3
Repository Browser, 13-5
operating modes
row-based, 10-5
row-based (target only), 10-6
selecting a default mode, 10-4
set-based, 10-5
operator attributes, 5-17
operator editor
Input tab, 26-2
Input/Output tab, 26-2
Output tab, 26-2
Operator wizard, 26-2
operators
about, 4-1, 4-2
Aggregator, 26-5
Anydata Cast, 26-9
connecting, 5-16
Constant, 25-9
Construct Object, 25-9
Cube, 25-10
data flow, 26-1
Data Generator, 25-12
Deduplicator, 26-10
Dimension, 25-14
editing, 5-21
Expand Object, 25-18
Expression, 26-11
External Table, 25-19
Filter, 26-12
Flat File, 25-32
flat file, 24-13
Joiner, 26-13
LCR Cast, 26-22
LCR Splitter, 26-23
Lookup, 26-23
Mapping Output Parameter, 25-21
Match Merge, 23-1
Materialized View, 25-22
Name and Address, 22-2
Pivot, 26-28
pluggable mapping, 4-5
Post-Mapping Process, 26-34
Pre-Mapping Process, 26-35
pre/post processing, 4-5
Queue, 25-23
Sequence, 25-25
Set Operation, 26-36
Sorter, 26-37
source, 25-1
Splitter operator, 26-39
Subquery Filter, 26-41
Table, 25-26
Table Function, 26-43
target, 25-1

Transformation, 26-46
transformation, 4-4
Unpivot, 26-47
Varray Iterator, 25-29
View, 25-30
operators, mapping
adding, 5-13
connecting, 5-16
editing, 5-21
types of, 4-3
OR activity, 27-23
Oracle RAC, managing service nodes, 13-19
Oracle Target CTs
using in CT mappings, 7-43
ORDER BY
in the Sorter operator, 26-38
ordering
multiple targets, 10-13
orphan management policy, 3-7
cubes, 3-51
dimensions, 3-28
other (non-SQL) transformations, 28-36
outer joins, 26-15
output components
Name and Address operator, 22-8
output signature, 5-40

## P

packages
as transformations, 4-7
defining, 9-2
editing, 9-11
process flows, 8-2, 8-7
parallel solving
cubes, 3-54
parameters
mapping output, 25-21
Partition Exchange Loading (PEL), 10-21
about, 10-22
configuring targets for, 10-25
mappings for, 10-22
performance considerations, 10-24
restrictions on, 10-25, 10-26
partitioning, index, 2-24, 2-32
partitions
about, 2-25
composite, 2-30
defining, 2-13
defining, hash by quantity partitions, 2-28
defining, list partitions, 2-28
defining, range partitions, 2-26
hash by quantity, 2-28
hash partitions, 2-27
index, 2-32
list, 2-28
subpartitions, creating, 2-31
types, 2-25
performing
Data Watch and Repair for MDM, 18-24
performing ETL
steps, 5-9
using cubes, 6-12
using dimensions, 6-1
using mappings, 5-9
Pivot operator, 26-28
editing, 26-30
example, 26-28
expressions for, 26-33
groups, 26-31
input attributes, 26-32
output attributes, 26-32
row locators, 26-30, 26-33
using, 26-30
PL/SQL mappings, 5-3, 10-1
example, 5-7
PL/SQL types
about, 9-7
as transformations, 4-8
creating, 9-7, 9-9
editing, 9-12
pluggable mapping folders
creating, 5-41
pluggable mapping operators, 4-5
pluggable mappings
about, 5-38
creating, 5-39
embedded, 5-39
reusable, 5-39
spotlighting operators, 5-45
ungrouping operators, 5-45
Post-Mapping Process operator, 26-34
prebuilt code templates, 7-13
limitations, 7-16
predefined transformations, 4-6
Pre-Mapping Process operator, 26-35
pre/post processing operators, 4-5
primary key constraints, 2-21
procedures
as transformations, 4-7
defining, 9-2
editing, 9-11
process flows
about, 8-1
activities in, 8-8
adding transformations to, 27-27
complex conditions in, 8-17
creating, 8-8
debugging, 13-20
defining, 8-5
deploying, 8-2
designing, 8-1
halting, 27-19
handling flat files with variable names, 8-18
modules, 8-2, 8-6
packages, 8-2, 8-7
scripting in, 27-24
starting, 27-26
subprocesses, 27-26
transferring remote files with FTP, 8-22

transitions, 8-13
using Enterprise Java Beans, 27-7
using Java classes, 27-17
using OMB*Plus, 27-22
using web services, 16-16
process runs
comparing, 13-21
debugging, 13-20
identifying recent, 13-20
monitoring, 13-22
rerunning, 13-22
terminating, 13-23
processes
debugging, 13-20
identifying recent, 13-20
running, 13-22
properties
for source operators, 25-2
for target operators, 25-2
object, 13-8
proxy settings
creating web services based on external URL, 16-10
public Oracle Custom library, 4-8
public Oracle Predefined library, 4-8
publishing objects
as web services, 16-4

## Q

Queue operator, 25-23
queue propagations
about, 2-45
configuring, 2-55
defining, 2-45
editing, 2-45
queue tables
about, 2-42
configuring, 2-55
defining, 2-42
editing, 2-43
payload type, 2-42

## R

range partitions
defining, 2-26
example, 2-27
RAW data type, 2-5
RECNUM attribute, 10-19
RECNUM columns, 10-19
records
extracting and loading master and detail records, 10-15
relationships between masters and details in flat files, 10-14
referential integrity, ensuring in mappings, 10-13
relating master and detail records, 10-14
relational implementation
about, 3-9
dimensional objects, 3-9
REMAINING_ROWS output group
in the Splitter operator, 26-39
remote files
transferring, 8-22
removing data
from dimensions, 6-10
from dimensions, example, 6-11
from slowly changing dimensions, 6-10
renaming
materialized views, 2-20
sequences, 2-36
tables, 2-14
views, 2-17
reordering table columns, 2-15
repeating schedules, 11-4
REPLICATE, 28-37
reporting
Control Center, 13-12
execution, 13-12, 13-16
implementation, 13-11
management, 13-13, 13-19
object properties, 13-8
on deployment, 13-12, 13-13
on objects, 13-9
reports
Control Center, 13-12
deployment, 13-12, 13-13
execution, 13-12, 13-16
implementation, 13-11
management, 13-13, 13-19
object, 13-9
object properties, 13-8
Repository Browser
about, 13-2
Control Center, 13-12
Design Center, 13-7
implementation reports, 13-11
logging in, 13-6
navigating, 13-7
object reports, 13-9
opening, 13-5
starting, 13-5
stopping, 13-5
Repository navigator, 13-7
restrictions
mapping debugger, 5-49
ROLAP implementation
about, 3-12
dimensional objects, 3-12
ROLAP with MVs implementation
about, 3-12
route activity, 27-23
row locators
in the Pivot operator, 26-30, 26-33
in the Unpivot operator, 26-48, 26-49
row-based, 10-5
row-based (target only), 10-6
row-based versus set-based
loading transaction data, 5-33

rows, filtering out, 26-12
RTRIM function
in the Transformation operator, 26-46
running
processes, 13-22
runtime parameters
SAP, 7-6
runtime parameters, configuring, 24-1
runtime performance, improving, 10-1

## S

sampling
master-detail flat files, 10-15
SAP
and joins, 26-19
defining ETL process for SAP objects, 7-4
Language parameter, setting, 7-6
runtime parameters, setting, 7-6
SAP ABAP mappings, 5-3
SAP file physical properties
Data File Name, 7-6
File Delimiter for Staging File, 7-6
Nested Loop, 7-7
SAP System Version, 7-7
SQL Join Collapsing, 7-6
Staging File Directory, 7-7
Use Single Select, 7-7
SAP parameters
Language, 7-6
loading type, 7-5
runtime, 7-6
schedules
creating, 11-2
defining, 11-2
duration, 11-4
editing, 11-3
example, 11-8
repeating, 11-4
using, 11-2
scheduling
about, 11-1
ETL jobs, 12-11
ETL objects, 11-1
scripting
in process flows, 27-24
scripts
for FTP commands, 27-14, 27-17
searching
groups, in mappings, 5-46
operators, in mappings, 5-46
second class objects
about, 2-7
self joins, 26-13
Sequence operator, 25-25
sequences
configuring, 2-54
Create Sequence Wizard, 2-35
defining, 2-35
editing, 2-35
renaming, 2-36
service nodes, managing, 13-19
set based update, 10-5
Set Operation operator, 26-36
intersect, 26-37
minus, 26-37
union, 26-36
union all, 26-37
Set Status activity, 27-24
set-based mode, 10-5
set-based versus row-based
loading transaction data, 5-33
set-based versus row-based modes, 10-4
setting
a starting point, 5-55
attribute properties, 25-7
signatures
input, 5-40
output, 5-40
slowly changing dimensions
about, 3-3
additional attributes, 3-3
effective date, 3-3
expiration date, 3-4
hierarchy versioning, about, 3-5
hierarchy versioning, enabling, 3-6
previous attribute, about, 3-4
removing data, 6-10
triggering attributes, 3-3
type 2, 3-19
type 2, defining, 3-32
type 2, example, 3-4
type 2, requirements, 3-4
type 2, updating, 3-33
type 3, 3-19
type 3, defining, 3-35
type 3, example, 3-6
type 3, requirements, 3-6
solving
cube measures, 3-53
solving cubes
independent of loading, 3-53
Sorter operator, 26-37, 26-38
source file operator, 7-31
source operators, 25-1
sources
master-detail flat file sources, 10-13
master-detail flat files, 10-13
master-detail flat files, example, 10-14
Spatial Transformations, 28-37
specifying
orphan management policy, dimensions, 3-28
Splitter operator, 26-39
spotlighting operators
mappings, 5-45
pluggable mappings, 5-45
SQL expressions, 26-11
SQL Server data objects
naming rules, 2-58
SQL*Loader mappings, 5-3

sqlplus activity, 27-24
Start activity, 27-26
starting
  CCA, 7-23
  Control Center Agent, 7-23
  Name and Address server, 22-24
  Repository Browser, 13-5
starting point, setting, 5-55
stopping
  CCA, 7-23
  Control Center Agent, 7-23
  Name and Address server, 22-24
  Repository Browser, 13-5
streams transformations, 28-37
subpartitions
  creating, 2-31
subprocesses, to start process flows, 27-26
Subquery Filter operator, 26-41
substitution variables, 27-16
summarizing
  data with a transformation, 26-5
surrogate identifiers
  about, 3-3
  usage, 3-3
surrogate keys
  in the Key Lookup operator, 26-23
synchronization
  mapping objects, advanced options, 5-32
synchronizing
  operators, 5-28
  operators, matching strategies, 5-32
  web services, 16-17
  workspace objects, 5-28
SYS.ANYDATA data type, 2-5
SYS.LCR$_ROW_RECORD data type, 2-6

## T

table definitions
  creating, 2-11
  editing, 2-13
Table Function operator, 26-43
  prerequisites, 26-45
  using, 26-44
table functions
  as transformations, 4-7
Table operator, 25-26
tables, 2-13
  attribute sets, 2-33
  attribute sets, adding, 2-14
  attribute sets, defining, 2-13
  attribute sets, deleting, 2-14
  attribute sets, editing, 2-14
  chunking, 25-27
  columns, adding, 2-14
  columns, defining, 2-12
  columns, deleting, 2-14
  columns, editing, 2-14
  configuring, 2-48
  constraints, adding, 2-14
  constraints, defining, 2-13
  constraints, deleting, 2-14
  constraints, editing, 2-14
  data rules, 2-13
  defining, 2-11
  defining partitions, 2-13
  naming, 2-11
  renaming, 2-14
  reordering columns, 2-15
tables (defined), 2-10
target load ordering, 10-13
target modules
  configuring, 2-46
target operators, 25-1
  loading types, 25-3
  properties for, 25-2
target schemas
  designing, 1-1
  designing, dimensional, 1-3
  designing, relational, 1-2
  rolling out changes, 12-12
targets
  defining load orders, 10-13
  multiple targets in a mapping, 10-7
  multiple targets in mappings, 26-39, 26-40
templates
  activity, 8-11
test data, defining, 5-51
Time Dimension Wizard
  defaults, 3-60
time dimensions, 3-64
  creating, using the Time Dimension Wizard, 3-57
  data range, 3-58
  editing, 3-61
  levels, creating, 3-59
  overlapping data population, 3-65
  populating, 3-64
  storing, 3-58
time settings, 13-22
TIMESTAMP data type, 2-6
TIMESTAMP WITH LOCAL TIMEZONE data
    type, 2-6
TIMESTAMP WITH TIMEZONE data type, 2-6
LIA *See* lineage and impact analysis
lineage and impact analysis *See also* impact analysis
lineage and impact analysis *See also* lineage
UK *See* constraints
  unique key
transaction data
  loading, 5-33
transferring
  remote files, 8-22
transformation filter data, 9-13
transformation libraries
  about, 4-8
  accessing, 4-9
  public Oracle Custom library, 4-8
  public Oracle Predefined library, 4-8
  types, 4-8
Transformation operator, 26-46

CHAR, 26-46
CHAR result RTRIM, 26-46
data type, 26-46
RTRIM function, 26-46
transformation operators, 4-4
about, 4-1
transformation properties, 9-11
transformations
about, 4-6
adding to process flows, 27-27
administrative, 28-1
character, 28-9
control center, 28-12
conversion, 28-17
custom, 4-7
custom example, 9-13
date, 28-18
group by operation, 26-5
importing, 9-13
introduction to, 9-1 to 9-14
number, 28-30
OLAP, 28-33
other (non-SQL), 28-36
predefined, 4-6
streams, 28-37
types, 4-6
XML, 28-38
transforming data
about, 4-1
transition conditions, 8-17
Transition Editor, 8-17
transitions
conditions of, 8-17
in process flows, 8-13
transportable modules
about, 17-4
configuring, 17-12
definitions, 17-11
deploying, 17-15
editing, 17-17
generating, 17-15
locations for, 17-7
mapping, 17-17
using, 17-5
tree walking, 26-13
triggering attributes
about, 3-3
tuning
data profiling performance, 18-22
type 2 slowly changing dimensions
creating, using the Dimension Wizard, 3-19
effective date, mapping source attributes, 6-4
extracting data from, 6-8
loading data, 6-3
loading data, example, 6-6
updating, 3-33
type 3 slowly changing dimensions
creating, using the Dimension Wizard, 3-19
extracting data from, 6-9
loading data, 6-5
loading data, example, 6-6
types
calculated measures, 3-39
mappings, 5-2
transformations, 4-6
types of
transformation libraries, 4-8

## U

unbinding
dimensional objects, 3-12
ungrouping operators
mappings, 5-45
pluggable mappings, 5-45
UNION ALL set operation, 26-37
UNION set operation, 26-36
unique
key constraints, 2-21
Unpivot operator, 26-47
editing, 26-48
example, 26-47
expressions for, 26-51
groups, 26-48
input attributes, 26-49
input connections, 26-49
output attributes, 26-50
row locators, 26-48, 26-49
using, 26-48
updating
target schema, 12-12
UROWID data type, 2-6
user-defined activities, 27-27
user-defined types
overview, 2-36
using
transportable modules, 17-5
utilities
activities, 27-2

## V

validating
about, 1-4
data objects, 1-5
editing invalid objects, 1-6
web services, 16-8
validation
about, 1-4
editing invalid objects, 1-6
errors, 15-1
viewing results, 1-5
VARCHAR data type, 2-6
VARCHAR2 data type, 2-7
variables
substitution, 27-16
Varray Iterator operator, 25-29
Varrays
creating, 2-39
editing, 2-40

overview, 2-38
View operator, 25-30
viewing
data, 2-9
data auditor error tables, 13-23
data objects, 2-9
data stored in data objects, 2-9
generation results, 1-7
generation scripts, 1-7
validation results, 1-5
views
about, 2-15
attribute sets, adding, 2-18
attribute sets, deleting, 2-18
attribute sets, editing, 2-18
columns, adding, 2-17
columns, defining, 2-16
columns, deleting, 2-17
columns, editing, 2-17
configuring, 2-54
constraints, adding, 2-17
constraints, deleting, 2-17
constraints, editing, 2-17
defining, 2-15
editing, 2-17
materialized, 25-22
renaming, 2-17

## W

wait activity, 27-29
WB_ABORT function, 28-2
WB_CAL_MONTH_NAME function, 28-19
WB_CAL_MONTH_OF_YEAR function, 28-20
WB_CAL_MONTH_SHORT_NAME function, 28-20
WB_CAL_QTR function, 28-21
WB_CAL_WEEK_OF_YEAR function, 28-21
WB_CAL_YEAR function, 28-22
WB_CAL_YEAR_NAME function, 28-22
WB_COMPILE_PLSQL transformation, 28-2
WB_DATE_FROM_JULIAN function, 28-23
WB_DAY_NAME function, 28-23
WB_DAY_OF_MONTH function, 28-24
WB_DAY_OF_WEEK function, 28-24
WB_DAY_OF_YEAR function, 28-25
WB_DAY_SHORT_NAME function, 28-25
WB_DECADE function, 28-26
WB_DISABLE_ALL_CONSTRAINTS, 28-3
WB_DISABLE_ALL_TRIGGERS, 28-3
WB_DISABLE_CONSTRAINT, 28-4
WB_DISABLE_TRIGGER, 28-5
WB_ENABLE_ALL_CONSTRAINTS, 28-6
WB_ENABLE_ALL_TRIGGERS, 28-6
WB_ENABLE_CONSTRAINT, 28-7
WB_ENABLE_TRIGGER, 28-8
WB_HOUR12 function, 28-26
WB_HOUR12MI_SS function, 28-27
WB_HOUR24 function, 28-27
WB_HOUR24MI_SS function, 28-28
WB_IS_DATE function, 28-28
WB_IS_NUMBER function, 28-33
WB_IS_SPACE function, 28-11
WB_JULIAN_FROM_DATE function, 28-29
WB_LOOKUP_CHAR function, 28-10, 28-11
WB_LOOKUP_NUM function, 28-31, 28-32
WB_MI_SS function, 28-29
WB_OLAP_AW_PRECOMPUTE, 28-34
WB_OLAP_LOAD_CUBE, 28-34
WB_OLAP_LOAD_DIMENSION, 28-35
WB_OLAP_LOAD_DIMENSION_GENUK, 28-35
WB_RT_GET_ELAPSED_TIME function, 28-12
WB_RT_GET_JOB_METRICS function, 28-13
WB_RT_GET_LAST_EXECUTION_TIME, 28-14
WB_RT_GET_MAP_RUN_AUDIT function, 28-14
WB_RT_GET_NUMBER_OF_ERRORS
function, 28-15
WB_RT_GET_NUMBER_OF_WARNINGS
function, 28-15
WB_RT_GET_PARENT_AUDIT_ID function, 28-16
WB_RT_GET_RETURN_CODE function, 28-16
WB_RT_GET_START_TIME function, 28-17
WB_TRUNCATE_TABLE, 28-9
WB_WEEK_OF_MONTH function, 28-30
WB_XML_LOAD, 28-39
WB_XML_LOAD_F, 28-39
web service packages
creating, 16-6
web services
about, 16-1
accessing securely, 16-19
advantages, 16-2
consuming, about, 16-3
creating, based on URL, 16-10
creating, based on Warehouse Builder
objects, 16-6
deploying, 16-9
deployment locations, 16-9
executing, using browser, 16-12
executing, using Control Center Manager, 16-11
generating, 16-8
prerequisites for deploying, 16-9
public web services, about, 16-4
publishing, about, 16-3
setting up secure access to external servers, 16-19
synchronizing, 16-17
using in mappings, 16-17
using in process flows, 16-16
validating, 16-8
web services based on external URL
proxy settings, 16-10
WHERE (in the Filter operator), 26-12
While Loop activity, 27-29
wizards
Create Cube Wizard, 3-42
Create Data Auditor Wizard, 20-3
Create Data Rule Folder Wizard, 19-4
Create Data Rule Wizard, 19-5
Create Dimension Wizard, 3-15
Create Sequence Wizard, 2-35
Operator, 26-2

Pivot Wizard, 26-30
Time Dimension Wizard, 3-57
Unpivot Wizard, 26-48
writing SQL expressions, 26-11

## X

XML Transformations, 28-38
XMLFORMAT data type, 2-7
XMLTYPE data type, 2-7