

**Oracle® Warehouse Builder**

User's Guide

11g Release 1 (11.1)

**B31278-01**

July 2007

Oracle Warehouse Builder User's Guide, 11g Release 1 (11.1)

B31278-01

Copyright © 2000, 2007, Oracle. All rights reserved.

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States 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, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software--Restricted Rights (June 1987). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and we disclaim liability for any damages caused by such use of the Programs.

Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

The Programs may provide links to Web sites and access to content, products, and services from third parties. Oracle is not responsible for the availability of, or any content provided on, third-party Web sites. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

This program contains Batik version 1.6.

Apache License

Version 2.0, January 2004

<http://www.apache.org/licenses/>

For additional information about the terms and conditions, search for "Apache License" in Oracle Warehouse Builder online help.

---

---

# Contents

<b>Preface</b> .....	xvii
Audience .....	xvii
Documentation Accessibility .....	xvii
Conventions .....	xviii
Getting Help .....	xviii
Related Publications .....	xix
<b>What's New</b> .....	xxi
New in Oracle Warehouse Builder 11g Release 1 (11.1) .....	xxi
<b>Part I Introduction and Concepts</b>	
<b>1 Introduction to Oracle Warehouse Builder</b>	
<b>Overview of Oracle Warehouse Builder</b> .....	1-1
Data Consolidation and Integration .....	1-1
<b>Product Options and Licensing</b> .....	1-2
Warehouse Builder Core Functionality .....	1-3
Warehouse Builder Enterprise ETL Option .....	1-3
Warehouse Builder Data Quality Option .....	1-5
Warehouse Builder Connector - E-Business Suite .....	1-5
Warehouse Builder Connector - PeopleSoft .....	1-6
Warehouse Builder Connector - SAP R/3 Connector .....	1-6
Warehouse Builder Connector - Siebel .....	1-7
<b>2 Getting Started with Oracle Warehouse Builder</b>	
<b>Understanding the Basic Concepts</b> .....	2-1
<b>Implementing a Data Integration Solution</b> .....	2-2
Before You Begin .....	2-3
Preparing the Warehouse Builder Design Center .....	2-4
Importing the Source Metadata .....	2-5
Profiling Data and Ensuring Data Quality .....	2-6
Designing the Target Schema .....	2-6
Designing ETL Logic .....	2-7
Deploying the Design and Executing the Data Integration Solution .....	2-7

Monitoring and Reporting on the Data Warehouse .....	2-8
--	-----

### 3 Setting Up Warehouse Builder

<b>Organizing Design Objects into Projects</b> .....	3-1
<b>Setting Preferences</b> .....	3-2
Appearance Preferences .....	3-2
Control Center Monitor Preferences .....	3-2
Data Profiling Preferences .....	3-3
Deployment Preferences .....	3-4
Environment Preferences .....	3-5
Generation/Validation Preferences .....	3-5
Logging Preferences.....	3-6
Naming Preferences.....	3-6
About Naming Modes.....	3-7
Security Preferences.....	3-8
<b>Defining Collections</b> .....	3-9
Creating a Collection .....	3-9
Name and Description Page.....	3-9
Contents Page .....	3-9
Summary Page.....	3-10
Editing Collection Definitions .....	3-10
Name Tab .....	3-10
Contents Tab .....	3-10
<b>Alternative Interfaces</b> .....	3-11

### 4 Identifying Data Sources and Importing Metadata

<b>About Source Data and Metadata</b> .....	4-1
<b>Supported Sources and Targets</b> .....	4-2
<b>General Steps for Importing Metadata from Sources</b> .....	4-3
Example: Importing Metadata from Flat Files.....	4-4
About Modules.....	4-4
Creating Modules.....	4-4
<b>Using the Import Metadata Wizard</b> .....	4-5
Importing Definitions from a Database .....	4-6
Filter Information Page .....	4-6
Object Selection Page.....	4-6
Summary and Import Page .....	4-7
Import Results Page.....	4-7
Importing Definitions from Flat Files .....	4-7
<b>Reimporting Definitions from an Oracle Database</b> .....	4-9
Advanced Import Options.....	4-9
Advanced Import Options for Views and External Tables.....	4-10
Advanced Import Options for Tables .....	4-10
Advanced Import Options for Object Types.....	4-11
Advanced Import Options for SQL Collections .....	4-11
Updating Oracle Database Source Definitions .....	4-11
<b>Integrating with E-Business Suite</b> .....	4-11

Importing E-Business Suite Metadata Definitions .....	4-12
Filtering E-Business Suite Metadata.....	4-12
Filtering E-Business Suite Metadata by Business Domain .....	4-13
Filtering E-Business Suite Metadata by Text String.....	4-13
Selecting the Objects .....	4-14
Reviewing Import Summary.....	4-14
<b>Integrating with PeopleSoft</b> .....	4-15
Importing PeopleSoft Metadata Definitions .....	4-15
Filtering PeopleSoft Metadata.....	4-15
Filtering PeopleSoft Metadata by Business Domain .....	4-16
Filtering PeopleSoft Metadata by Text String.....	4-16
Selecting the Objects .....	4-16
Reviewing Import Summary.....	4-17
<b>Integrating with Siebel</b> .....	4-18
Importing Siebel Metadata Definitions .....	4-18
Creating a Siebel Source Module.....	4-18
Importing Siebel Metadata .....	4-18
<b>Integrating with SAP R/3</b> .....	4-20
About SAP Business Domains .....	4-21
SAP Table Types .....	4-21
Required Files For SAP Connector .....	4-21
Creating SAP Module Definitions .....	4-22
Connecting to an SAP Source Application.....	4-23
Importing SAP Metadata Definitions .....	4-24
Filtering SAP Metadata .....	4-24
Filtering SAP Metadata by Business Domain.....	4-25
Filtering SAP Metadata by Text String.....	4-25
Selecting the Objects .....	4-26
Reviewing Import Summary.....	4-26
Reimporting SAP Objects .....	4-27
Updating SAP Source Modules .....	4-27
Defining the ETL Process for SAP Objects .....	4-27
Defining Mappings Containing SAP Objects .....	4-28
Configuring Code Generation for SAP Objects.....	4-28
Generating SAP Definitions .....	4-30
Loading SAP Data into the Workspace.....	4-30
Deploying and Executing an SAP Mapping .....	4-31
Deploying PL/SQL Scripts for Transparent Tables .....	4-32
<b>Integrating with Business Intelligence Tools</b> .....	4-32
Introduction to Business Intelligence Objects in Warehouse Builder .....	4-32
Introduction to Business Definitions.....	4-33
About Business Definitions .....	4-33

## 5 Understanding Data Quality Management

<b>About the Data Quality Management Process</b> .....	5-1
Phases in the Data Quality Lifecycle .....	5-2
Quality Assessment .....	5-2

Quality Design.....	5-3
Quality Transformation .....	5-3
Quality Monitoring.....	5-3
<b>About Data Profiling .....</b>	<b>5-3</b>
Benefits of Data Profiling .....	5-4
Types of Data Profiling.....	5-4
Attribute Analysis .....	5-5
Functional Dependency .....	5-6
Referential Analysis.....	5-6
Data Rule Profiling .....	5-7
About Six Sigma .....	5-8
What is Six Sigma?.....	5-8
Six Sigma Metrics for Data Profiling.....	5-8
<b>About Data Correction and Augmentation .....</b>	<b>5-9</b>
About the Match-Merge Operator.....	5-9
Example of Matching and Merging Customer Data .....	5-10
Overview of the Matching and Merging Process.....	5-10
Matching and Merging Records .....	5-11
Constructing Match Bins .....	5-12
Constructing Match Record Sets .....	5-12
Constructing Merge Records .....	5-12
Match Rules.....	5-12
Conditional Match Rules.....	5-13
Comparison Algorithms .....	5-13
Creating Conditional Match Rules .....	5-15
Weight Match Rules.....	5-15
Example of Weight Match Rules .....	5-15
Creating Weight Match Rules .....	5-16
Person Match Rules.....	5-16
Person Roles.....	5-17
Person Details .....	5-17
Creating Person Match Rules.....	5-18
Firm Match Rules .....	5-18
Firm Roles .....	5-19
Firm Details.....	5-19
Creating Firm Match Rules.....	5-19
Address Match Rules.....	5-20
Address Roles .....	5-20
Address Details .....	5-21
Creating Address Match Rules .....	5-22
Custom Match Rules.....	5-22
Creating Custom Match Rules .....	5-23
Merge Rules .....	5-23
Using a Match-Merge Operator .....	5-24
About the Name and Address Operator .....	5-26
Example: Correcting Address Information .....	5-27
Example Input .....	5-27

Example Steps.....	5-27
Example Output.....	5-28
Handling Errors in Name and Address Data .....	5-29
About Postal Reporting.....	5-30
United States Postal Service CASS Certification .....	5-30
Canada Post SERP Certification.....	5-30
Australia Post AMAS Certification .....	5-31
<b>About Data Rules .....</b>	<b>5-31</b>
<b>About Quality Monitoring .....</b>	<b>5-31</b>
About Data Auditors .....	5-32
<b>Performing Data Profiling.....</b>	<b>5-32</b>
Import or Select the Metadata .....	5-33
Create a Data Profile .....	5-33
Profile the Data .....	5-34
Configuring Data Profiles.....	5-34
Steps to Profile Data .....	5-35
View Profile Results.....	5-35
Derive Data Rules .....	5-36
Generate Corrections .....	5-37
Define and Edit Data Rules Manually .....	5-39
Generate, Deploy, and Execute .....	5-39
<b>Tuning the Data Profiling Process .....</b>	<b>5-40</b>
Tuning the Data Profile for Better Data Profiling Performance .....	5-40
Tuning the Oracle Database for Better Data Profiling Performance .....	5-40
Multiple Processors .....	5-40
Memory .....	5-41
I/O System.....	5-41
<b>Using Data Rules.....</b>	<b>5-41</b>
Creating Data Rules .....	5-42
Applying Data Rules to Objects.....	5-43
<b>Monitoring Data Quality Using Data Auditors .....</b>	<b>5-43</b>
Creating Data Auditors.....	5-44
Auditing Data Objects Using Data Auditors .....	5-45
Manually Running Data Auditors.....	5-45
Automatically Running Data Auditors .....	5-45
Data Auditor Execution Results .....	5-46

## 6 Designing Target Schemas

<b>About Data Objects.....</b>	<b>6-1</b>
Supported Data Types.....	6-3
Naming Conventions for Data Objects .....	6-6
<b>About the Data Object Editor .....</b>	<b>6-6</b>
Data Viewer .....	6-7
Using the Data Object Editor to Create Data Objects .....	6-8
Creating Data Objects Using the Menu Bar .....	6-8
Creating a Data Object Using the Canvas .....	6-9
Creating a Data Object Using the Data Object Editor Palette.....	6-9

<b>About Dimensional Objects</b> .....	6-10
Defining Dimensional Objects.....	6-11
Implementing Dimensional Objects .....	6-11
Relational Implementation of Dimensional Objects .....	6-12
Binding .....	6-12
ROLAP Implementation of Dimensional Objects .....	6-14
MOLAP Implementation of Dimensional Objects .....	6-14
Analytic Workspace .....	6-14
OLAP Catalog .....	6-14
Deploying Dimensional Objects .....	6-15
Loading Dimensional Objects .....	6-16
<b>About Dimensions</b> .....	6-16
Rules for Dimension Objects .....	6-16
Limitations of Deploying Dimensions to the OLAP Catalog .....	6-17
Defining a Dimension.....	6-17
Defining Dimension Attributes.....	6-18
Defining Levels.....	6-18
Surrogate Identifiers.....	6-18
Business Identifiers.....	6-18
Parent Identifier .....	6-19
Defining Level Attributes .....	6-19
Defining Hierarchies .....	6-19
Dimension Roles .....	6-19
Level Relationships.....	6-20
Dimension Example.....	6-20
Control Rows .....	6-21
Value-based Hierarchies .....	6-21
Implementing a Dimension .....	6-22
Relational and ROLAP Implementation of a Dimension.....	6-22
Star Schema.....	6-22
Snowflake Schema .....	6-23
Binding .....	6-24
MOLAP Implementation .....	6-25
<b>About Slowly Changing Dimensions</b> .....	6-25
About Type 1 Slowly Changing Dimensions.....	6-26
About Type 2 Slowly Changing Dimensions.....	6-26
Defining a Type 2 Slowly Changing Dimension.....	6-26
Updating Type 2 Slowly Changing Dimensions.....	6-28
About Type 3 Slowly Changing Dimensions.....	6-30
Defining a Type 3 Slowly Changing Dimension.....	6-30
<b>About Time Dimensions</b> .....	6-30
Best Practices for Creating a Time Dimension.....	6-31
Defining a Time Dimension.....	6-31
Levels .....	6-31
Dimension Attributes .....	6-32
Level Attributes.....	6-32
Hierarchies .....	6-33



Implementing a Time Dimension .....	6-33
Using a Time Dimension in a Cube Mapping .....	6-34
Populating a Time Dimension.....	6-34
Overlapping Data Populations .....	6-35
<b>About Cubes</b> .....	6-35
Defining a Cube.....	6-36
Cube Measures .....	6-36
Cube Dimensionality .....	6-36
Cube Example.....	6-37
Implementing a Cube .....	6-37
Relational and ROLAP Implementation of a Cube.....	6-37
Binding .....	6-38
MOLAP Implementation of a Cube .....	6-39
Solve Dependency Order of Cube .....	6-39
<b>Designing the Target Schema</b> .....	6-39
Designing a Relational Target Schema.....	6-39
Designing a Dimensional Target Schema.....	6-40
<b>Creating Oracle Data Objects</b> .....	6-41
Creating Relational Data Objects .....	6-41
Creating Dimensions .....	6-41
Creating Time Dimensions .....	6-43
Creating Cubes .....	6-44
<b>Configuring Data Objects</b> .....	6-45
<b>Validating Data Objects</b> .....	6-45
Editing Invalid Objects.....	6-46
<b>Generating Data Objects</b> .....	6-47
Viewing Generated Scripts .....	6-47
Saving Generated Scripts to a File .....	6-48
<b>Deriving Business Intelligence Metadata</b> .....	6-48

## 7 Creating Mappings

<b>About Mappings and Operators</b> .....	7-1
<b>Instructions for Defining Mappings</b> .....	7-2
Instructions for Using Flat File Sources or Targets in a Mapping .....	7-3
<b>Creating a Mapping</b> .....	7-4
About the Mapping Editor .....	7-5
Mapping Editor Windows .....	7-6
Explorer .....	7-6
Properties Inspector.....	7-6
Palette .....	7-6
Bird's Eye View .....	7-6
Data Viewer .....	7-7
Generation.....	7-7
Mapping Editor Toolbars.....	7-7
Mapping Editor Display Options .....	7-7
Types of Operators.....	7-8
Oracle Source/Target Operators .....	7-8

Data Flow Operators .....	7-9
Pre/Post Processing Operators.....	7-10
Pluggable Mapping Operators.....	7-10
<b>Adding Operators</b> .....	7-11
Adding Operators that Bind to Workspace Objects.....	7-11
Add Operator Dialog Box.....	7-12
Create Unbound Operator with No Attributes .....	7-12
Select from Existing Repository Object and Bind.....	7-12
<b>Editing Operators</b> .....	7-12
Name Tab .....	7-13
Groups Tab.....	7-13
Input and Output Tabs.....	7-13
Mapping Naming Conventions .....	7-14
Using Display Sets .....	7-16
Defining Display Sets .....	7-16
Selecting a Display Set .....	7-17
<b>Connecting Operators</b> .....	7-17
Connecting Attributes .....	7-17
Connecting Groups.....	7-18
Example: Using the Mapping Editor to Create Staging Area Tables .....	7-18
Using the Connect Operators Dialog Box .....	7-19
Copy Source Attributes to Target Group and Match .....	7-19
Match by Position of Source and Target Attributes.....	7-20
Match by Name of Source and Target Attributes .....	7-20
<b>Using Pluggable Mappings</b> .....	7-20
Creating a Pluggable Mapping .....	7-21
Standalone Pluggable Mapping.....	7-21
Pluggable Mapping Folders .....	7-21
Signature Groups .....	7-22
Input Signature.....	7-22
Output Signature .....	7-22
Pluggable Mapping Editor.....	7-23
<b>Setting Mapping Properties</b> .....	7-23
Target Load Order .....	7-23
Reset to Default .....	7-24
<b>Setting Operator, Group, and Attribute Properties</b> .....	7-24
<b>Synchronizing Operators and Workspace Objects</b> .....	7-25
Synchronizing An Operator .....	7-25
Synchronizing From a Workspace Object to an Operator .....	7-26
Synchronizing Operators based on Workspace Objects .....	7-26
Synchronizing from an Operator to a Workspace Object .....	7-27
Advanced Options for Synchronizing .....	7-28
Matching Strategies .....	7-28
Match by Object Identifier.....	7-29
Match by Bound Name .....	7-29
Match by Position .....	7-29
<b>Using DML Error Logging</b> .....	7-29

About Error Tables.....	7-30
Error Tables and DML Error Logging .....	7-30
Error Tables and Data Rules.....	7-31
Using Error Tables for DML Error Logging and Data Rules.....	7-31
Enabling DML Error Logging .....	7-31
DML Error Logging and ETL.....	7-32
DML Error Logging Limitations .....	7-32
<b>Debugging a Mapping</b> .....	7-33
Starting a Debug Session.....	7-33
The Debug Panels of the Mapping Editor .....	7-33
Debug Info Panel.....	7-33
Debug Data Panel .....	7-34
Defining Test Data .....	7-34
Creating New Tables to Use as Test Data .....	7-35
Editing the Test Data .....	7-35
Setting Breakpoints .....	7-35
Setting Watches .....	7-35
Running the Mapping .....	7-36
Selecting the First Source and Path to Debug.....	7-36
Debugging Mappings with Correlated Commit .....	7-37
Setting a Starting Point.....	7-37
Debugging Pluggable Submap Operators .....	7-37
Re-Initializing a Debug Session .....	7-38
Scalability .....	7-38

## 8 Designing Process Flows

<b>About Process Flows</b> .....	8-1
About Process Flow Modules and Packages.....	8-2
<b>Instructions for Defining Process Flows</b> .....	8-2
Creating Process Flow Modules.....	8-3
Creating Process Flow Packages.....	8-3
Creating Process Flows .....	8-4
<b>About the Process Flow Editor</b> .....	8-4
Standard Editor Components.....	8-4
Process Flow Editor Windows .....	8-5
Opening the Process Flow Editor .....	8-6
Navigating the Process Flow Editor.....	8-6
<b>Adding Activities to Process Flows</b> .....	8-7
About Activities.....	8-7
Adding Activities .....	8-7
Parameters for Activities.....	8-8
<b>Creating and Using Activity Templates</b> .....	8-10
Name and Description Page.....	8-10
Parameters Page .....	8-11
Using Activity Templates .....	8-11
<b>About Transitions</b> .....	8-13
Rules for Valid Transitions .....	8-13

Connecting Activities .....	8-14
Configuring Activities .....	8-14
Using Parameters and Variables .....	8-15
Using Namespace .....	8-15
Using Bindings .....	8-16
<b>Expressions .....</b>	<b>8-16</b>
Global Expression Values .....	8-16
<b>Defining Transition Conditions .....</b>	<b>8-17</b>

## 9 Understanding Performance and Advanced ETL Concepts

<b>Best Practices for Designing PL/SQL Mappings .....</b>	<b>9-1</b>
Set Based Versus Row Based Operating Modes .....	9-4
Set Based .....	9-5
Row Based .....	9-5
Row Based (Target Only) .....	9-6
About Committing Data in Warehouse Builder .....	9-6
Committing Data Based on Mapping Design .....	9-7
Committing Data from a Single Source to Multiple Targets .....	9-7
Automatic Commit versus Automatic Correlated Commit .....	9-8
Embedding Commit Logic into the Mapping .....	9-9
Committing Data Independently of Mapping Design .....	9-10
Running Multiple Mappings Before Committing Data .....	9-10
Committing Data at Runtime .....	9-11
Committing Mappings through the Process Flow Editor .....	9-12
Ensuring Referential Integrity in PL/SQL Mappings .....	9-13
<b>Best Practices for Designing SQL*Loader Mappings .....</b>	<b>9-14</b>
Using Conventional Loading to Ensure Referential Integrity in SQL*Loader Mappings .....	9-14
Maintaining Relationships Between Master and Detail Records .....	9-14
Extracting and Loading Master-Detail Records .....	9-15
Error Handling Suggestions .....	9-17
Subsequent Operations .....	9-19
Using Direct Path Loading to Ensure Referential Integrity in SQL*Loader Mappings .....	9-19
<b>Improved Performance through Partition Exchange Loading .....</b>	<b>9-22</b>
About Partition Exchange Loading .....	9-23
Configuring a Mapping for PEL .....	9-23
Direct and Indirect PEL .....	9-24
Using Indirect PEL .....	9-24
Example: Using Direct PEL to Publish Fact Tables .....	9-25
Using PEL Effectively .....	9-25
Configuring Targets in a Mapping .....	9-26
Step 1: Create All Partitions .....	9-26
Step 2: Create All Indexes Using the LOCAL Option .....	9-26
Step 3: Primary/Unique Keys Use "USING INDEX" Option .....	9-27
Restrictions for Using PEL in Warehouse Builder .....	9-27
<b>High Performance Data Extraction from Remote Sources .....</b>	<b>9-27</b>

## 10 Introducing Oracle Warehouse Builder Transformations

<b>About Transforming Data Using Warehouse Builder</b> .....	10-1
Benefits of Using Warehouse Builder for Transforming Data .....	10-2
<b>About Transformations</b> .....	10-2
Types of Transformations .....	10-2
Predefined Transformations.....	10-2
Custom Transformations .....	10-3
<b>About Transformation Libraries</b> .....	10-4
Types of Transformation Libraries .....	10-4
Accessing Transformation Libraries.....	10-4
<b>Defining Custom Transformations</b> .....	10-5
Defining Functions and Procedures .....	10-7
Name and Description Page.....	10-7
Parameters Page .....	10-7
Implementation Page .....	10-7
Summary Page.....	10-8
Defining PL/SQL Types .....	10-8
About PL/SQL Types.....	10-8
Usage Scenario for PL/SQL Types.....	10-9
Creating PL/SQL Types .....	10-10
Name and Description Page.....	10-10
Attributes Page.....	10-11
Return Type Page.....	10-11
Summary Page.....	10-12
<b>Editing Custom Transformations</b> .....	10-12
Editing Function or Procedure Definitions .....	10-12
Name Tab .....	10-12
Parameters Tab.....	10-12
Implementation Tab .....	10-13
Editing PL/SQL Types.....	10-13
Name Tab .....	10-13
Attributes Tab.....	10-13
Return Type Tab.....	10-13
<b>Importing PL/SQL</b> .....	10-14
Restrictions on Using Imported PL/SQL .....	10-14

## 11 Deploying to Target Schemas and Executing ETL Logic

<b>About Deployment and Execution in Warehouse Builder</b> .....	11-1
About Deployment .....	11-1
Deployment Actions.....	11-2
Deployment Status.....	11-3
About Execution.....	11-3
About the Warehouse Builder Implementation Environment.....	11-3
About Control Centers .....	11-4
Creating a Control Center.....	11-4
Activating a Control Center .....	11-4

About Locations .....	11-5
Creating Locations .....	11-5
Registering and Unregistering Locations .....	11-6
Deleting Locations .....	11-6
About Connectors .....	11-6
<b>The Deployment and Execution Process .....</b>	<b>11-7</b>
Deploying Objects .....	11-8
Deploying Business Definitions to Oracle Discoverer.....	11-8
Deploying Business Definitions Directly to Oracle Discoverer .....	11-9
Deploying Business Definitions to Earlier Versions of Oracle Discoverer.....	11-9
Deploying Business Definitions Using the Core Functionality.....	11-10
Reviewing the Deployment Results .....	11-11
Starting ETL Jobs.....	11-11
Viewing the Data.....	11-12
<b>Scheduling ETL Jobs .....</b>	<b>11-12</b>
<b>Configuring the Physical Details of Deployment .....</b>	<b>11-12</b>
About Configurations.....	11-12
Creating New Configurations.....	11-13
Activating Configurations .....	11-13
Creating Additional Configurations .....	11-14
Scenario Requiring Multiple Configurations.....	11-14
Setting Configuration Properties for a Named Configuration.....	11-15
Deploying a Design to Multiple Target Systems.....	11-15
Benefit of Creating Additional Configurations .....	11-16
<b>About Schedules.....</b>	<b>11-16</b>
<b>Process for Defining and Using Schedules.....</b>	<b>11-17</b>
Example Schedules .....	11-18

## Part II Example Cases

### 12 Loading Data Stored in a Microsoft Excel File

Case Study .....	12-1
Troubleshooting.....	12-4

### 13 Connecting to SQL Server and Importing Metadata

Creating an ODBC Data Source .....	13-1
Configuring the Oracle Database Server.....	13-2
Creating a Heterogeneous Service Configuration File .....	13-2
Editing the listener.ora file.....	13-2
Adding the SQL Server as a Source in Warehouse Builder .....	13-3
What's Next.....	13-3
Troubleshooting.....	13-3

<b>14</b>	<b>Loading Transaction Data</b>	
<b>15</b>	<b>The Fastest Way to Load Data from Flat Files</b>	
	SQL *Loader .....	15-1
	When To Use SQL*Loader .....	15-1
	External Tables .....	15-2
	Benefits of Using External Tables .....	15-2
	When To Use External Tables .....	15-2
	Solution 1: Using SQL*Loader.....	15-2
	Solution 2: Using External Tables .....	15-3
<b>16</b>	<b>Importing from CA ERwin and Other Third-Party Design Tools</b>	
<b>17</b>	<b>Reusing Existing PL/SQL Code</b>	
<b>18</b>	<b>Sourcing from Flat Files with Variable Names</b>	
	Creating the Process Flow.....	18-2
	Setting Parameters for the External Process Activity .....	18-2
	Method 1: Write a script within Warehouse Builder .....	18-2
	Method 2: Call a script maintained outside of Warehouse Builder .....	18-4
	Configuring the External Process Activity .....	18-6
	Designing the Mapping .....	18-7
	Deploying and Executing .....	18-7
	Subsequent Steps .....	18-7
	Creating a Schedule .....	18-7
<b>19</b>	<b>Transferring Remote Files</b>	
	Creating the Process Flow.....	19-2
	Setting Parameters for the FTP Activity.....	19-2
	Example: Writing a Script in Warehouse Builder for the FTP Activity .....	19-3
	Using Substitution Variables .....	19-4
	Configuring the FTP Activity.....	19-5
	Registering the Process Flow for Deployment.....	19-5
	Defining Locations.....	19-6
<b>20</b>	<b>Inspecting Error Logs in Warehouse Builder</b>	
<b>21</b>	<b>Updating the Target Schema</b>	
<b>22</b>	<b>Managing Multiple Versions of a BI Implementation</b>	
	Approach.....	22-2
	Initial Phase.....	22-2
	Case Study.....	22-3
	Mature Phase .....	22-5

Case Study..... 22-5

**Index**



---

---

# Preface

This preface includes the following topics:

- [Audience](#)
- [Documentation Accessibility](#)
- [Conventions](#)
- [Getting Help](#)
- [Related Publications](#)

## 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, with good usability, to the disabled community. 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.

## TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

## 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
.	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.
<b>boldface text</b>	Boldface type in text refers to interface buttons and links. Boldface type also serves as emphasis to set apart main ideas.
<i>italicized text</i>	Italicized text applies to new terms introduced for the first time. Italicized text also serves as an emphasis on key concepts.
<code>unicode text</code>	Unicode text denotes exact code, file directories and names, and literal commands.
<i>italicized unicode text</i>	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.

- **Popup menus:** Click the arrow icon on the right side of the title bar for a window. Then choose **Help** from the popup 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](http://www.oracle.com/technology/products/warehouse/selfserv_edu/self_service_education.html)

## Related Publications

The Warehouse Builder documentation set includes these manuals:

- *Oracle Warehouse Builder User's Guide*
- *Oracle Warehouse Builder Installation and Administration Guide*
- *Oracle Warehouse Builder API and Scripting Reference*

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



---

---

# What's New

This preface includes the following topics:

- [New in Oracle Warehouse Builder 11g Release 1 \(11.1\)](#) on page xxi

## New in Oracle Warehouse Builder 11g Release 1 (11.1)

### Changes in the Installation Requirements and Instructions

Previously, if you wanted to utilize Oracle Workflow to manage job dependencies or if you wanted to deploy process flows, it was necessary to install Oracle Workflow. Beginning in this release, these additional installation steps are no longer required as Oracle Workflow components are embedded within Warehouse Builder.

Previously, each Oracle Database utilized as a repository for this product required a user with `SYSDBA` privileges. Beginning in this release, this is no longer required. A schema `OWBSYS` is created while installing Oracle Database 11g Release 1 (11.1). `OWBSYS` holds the metadata which is divided into workspaces. To start using Warehouse Builder, you just need to create a new workspace. You do not need `SYSDBA` privileges.

Previously, users accessed the repository as a whole. Therefore, users were referred to as *repository users* and *repository owners*. Beginning with this release, repository is replaced with workspace. Thus, instead of granting access to a repository, you grant access to a workspace. Because of the usage of workspaces in a single schema, creating workspaces is simplified.

Beginning with Warehouse Builder 11g Release 1 (11.1), the preferred method of implementing metadata security is through the user interface available in the Design Center and described in the *Oracle Warehouse Builder User's Guide*. If, in a previous release, you implemented security using a PL/SQL package, Warehouse Builder 11g Release 1 (11.1) does support that implementation.

For additional information, refer to the *Oracle Warehouse Builder Installation and Administration Guide*.

### Connectivity to Siebel

With Oracle Warehouse Builder 11g Release 1 (11.1), an application connector to Siebel is added. This connector allows you to connect to the Siebel metadata as could be done in previous versions with Oracle E-Business Suite, PeopleSoft, and SAP. For more information, see "[Integrating with Siebel](#)" on page 4-18.

### **Additions to Dimensional Objects**

There are some modifications to the functionality for updating records in a Type 2 Slowly Changing Dimension (SCD). For more information, see "[Updating Type 2 Slowly Changing Dimensions](#)" on page 6-28.

You can now version hierarchies in a Type 2 SCD. For information about enabling hierarchy versioning, see "[Hierarchy Versioning](#)" on page 6-27.

### **Improvements to the Documentation Set**

In this release, the documentation set has been reorganized and revised.

The book formerly entitled the Oracle Warehouse Builder Installation and Configuration Guide is now entitled the Oracle Warehouse Builder Installation and Administration Guide and includes administration information such as implementing security.

The Oracle Warehouse Builder User's Guide now includes enhanced introductory and conceptual information. Related reference material organized by subject matter is now contained in the Oracle Warehouse Builder Online Help.

The Oracle Warehouse Builder API and Scripting Reference now includes information on using experts and the Expert Editor, which was formerly contained in the Oracle Warehouse Builder User's Guide.

# Part I

---

## Introduction and Concepts

This part contains the following chapters:

- Chapter 1, "Introduction to Oracle Warehouse Builder"
- Chapter 2, "Getting Started with Oracle Warehouse Builder"
- Chapter 3, "Setting Up Warehouse Builder"
- Chapter 4, "Identifying Data Sources and Importing Metadata"
- Chapter 5, "Understanding Data Quality Management"
- Chapter 6, "Designing Target Schemas"
- Chapter 7, "Creating Mappings"
- Chapter 8, "Designing Process Flows"
- Chapter 9, "Understanding Performance and Advanced ETL Concepts"
- Chapter 10, "Introducing Oracle Warehouse Builder Transformations"
- Chapter 11, "Deploying to Target Schemas and Executing ETL Logic"





---

---

# Introduction to Oracle Warehouse Builder

Oracle Warehouse Builder provides enterprise solutions for end-to-end data integration. This chapter introduces you to the range of functionality provided by Warehouse Builder.

This chapter includes the following topics:

- [Overview of Oracle Warehouse Builder](#)
- [Product Options and Licensing](#)

## Overview of Oracle Warehouse Builder

Oracle Warehouse Builder is a single, comprehensive tool for all aspects of data integration. Warehouse Builder leverages Oracle Database to transform data into high-quality information. It provides data quality, data auditing, fully integrated relational and dimensional modeling, and full lifecycle management of data and metadata. Warehouse Builder enables you to create data warehouses, migrate data from legacy systems, consolidate data from disparate data sources, clean and transform data to provide quality information, and manage corporate metadata.

## Data Consolidation and Integration

Many global corporations have data dispersed on different platforms using a wide variety of data reporting and analysis tools. Customer and supplier data may be stored in applications, databases, spreadsheets, flat files, and legacy systems. This diversity may be caused by organizational units working independently over a period of time, or it may be the result of business mergers. Whatever the cause of diversity, this diversity typically results in poor quality data that provides an incomplete and inconsistent view of the business.

Transforming poor quality data into high quality information requires:

- **Access to a wide variety of data sources**  
Warehouse Builder leverages Oracle Database to establish transparent connections to numerous third-party databases, applications, files, and data stores as listed in ["Supported Sources and Targets"](#) on page 4-2.
- **Ability to profile, transform, and cleanse data**  
Warehouse Builder provides an extensive library of data transformations for data types such as text, numeric, date, and others. Use these transformations to reconcile the data from many different sources as described in ["Introducing Oracle Warehouse Builder Transformations"](#) on page 10-1.

Before loading data into a new data store, you can optionally profile the data to evaluate its quality and appropriateness. Subsequently, you can match and merge records using rules that you devise. You can validate name and address data against postal databases. This process of changing poor quality data into high quality information is introduced in "[About the Data Quality Management Process](#)" on page 5-1.

- **Ability to implement designs for diverse applications**

Using Warehouse Builder, you can design and implement any data store required by your applications, whether relational or dimensional. The process of designing your data store is described in "[Designing Target Schemas](#)" on page 6-1.

- **Audit trails**

After consolidating data from a variety of sources into a single data store, you are likely to face the challenge of verifying the validity of the output information. For instance, can you track and verify how a particular number was derived? This is a question often posed by decision makers within your organization and by government regulators.

## Product Options and Licensing

A significant portion but not all of the Warehouse Builder features are included in Oracle Database editions at no additional cost and enable you to design, deploy, and manage a basic Oracle data warehouse. If you intend to extract from applications or intend to perform data profiling or advanced Extraction, Transform, and Load (ETL) processes, consider licensing additional options available only with the Oracle Database Enterprise Edition.

[Table 1–1](#) can help you understand the difference between the options and determine the combination of database edition and Warehouse Builder options that addresses your requirements. The table lists the features available in Oracle Database Standard Edition One (SE1), Standard Edition (SE), and Enterprise Edition (EE). The Y value in a column indicates that the feature is available in the specified release; N indicates that it is not available.

---



---

**Note:** Depending on how you utilize Warehouse Builder, you may require licenses for additional database options and, or technologies such as Oracle Partitioning, Oracle OLAP, and Oracle Transparent Gateways.

---



---

**Table 1–1 Warehouse Builder Options Availability in Oracle Database Editions**

Option/ Feature	SE1	SE	EE	Comments
<a href="#">Warehouse Builder Core Functionality</a>	Y	Y	Y	Enables the design, deployment, execution, and management of common data integration or data warehouse projects.
<a href="#">Warehouse Builder Enterprise ETL Option</a>	N	N	Y	Enables higher developer productivity in (larger) projects. Also allows for reuse of transformation logic and for certain fast extraction methods in large volume data movements.

**Table 1–1 (Cont.) Warehouse Builder Options Availability in Oracle Database Editions**

Option/ Feature	SE1	SE	EE	Comments
<a href="#">Warehouse Builder Data Quality Option</a>	N	N	Y	Enables profiling of data to detect information quality issues in the source. Once the issues are documented, developers can generate business rules and automatically cleanse data using these business rules in the data integration process. In addition to this, the Data Quality option allows monitoring of quality on a regular basis using methods such as Six Sigma.
<a href="#">Warehouse Builder Connector - E-Business Suite</a>	N	N	Y	Enables access to technical and business metadata within Oracle E-Business Suite. Facilitates deployment to Oracle Concurrent Manager and access to Oracle E-Business Suite at execution-time.
<a href="#">Warehouse Builder Connector - PeopleSoft</a>	N	N	Y	Enables access to data and metadata in PeopleSoft applications.
<a href="#">Warehouse Builder Connector - SAP R/3 Connector</a>	N	N	Y	Enables uploading of generated ABAP code to the SAP system and executing ABAP programs from the Control Center Manager. For production systems, it allows the execution of registered ABAP programs from process flows.
<a href="#">Warehouse Builder Connector - Siebel</a>	N	N	Y	Enables access to data and metadata in Siebel applications.

## Warehouse Builder Core Functionality

The core Oracle Warehouse Builder functionality enables Extraction, Transformation, and Loading (ETL) of data from heterogeneous sources into heterogeneous targets. You can load data into relational, multidimensional, flat file, and XML storage systems.

If you licensed and used earlier versions of this product, note that the core functionality equates to the functionality available in Oracle Warehouse Builder 10g Release 1.

The core Warehouse Builder functionality is included in the Oracle Database license at no additional cost. If a feature is not specifically mentioned in one of the following options, you can safely assume that the feature is part of the core functionality:

[Warehouse Builder Enterprise ETL Option](#)

[Warehouse Builder Data Quality Option](#)

[Warehouse Builder Connector - E-Business Suite](#)

[Warehouse Builder Connector - PeopleSoft](#)

[Warehouse Builder Connector - SAP R/3 Connector](#)

[Warehouse Builder Connector - Siebel](#)

## Warehouse Builder Enterprise ETL Option

The Enterprise ETL option enables large-scale, complex ETL deployments. Developers can incorporate advanced functionality, such as retaining history for dimensions, reusing mapping code, performing interactive lineage and impact analysis, and defining custom types of objects in a repository. This option also enables the rapid movement of large amounts of data, and the construction of advanced process flows.

[Table 1–2](#) lists the functionality available with the Enterprise ETL option. The functionality is grouped into areas. For example, the area *Schema Modeling* includes functionality for slowly changing dimensions and business intelligence.

**Table 1–2 Warehouse Builder Enterprise ETL Option**

Area and Functionality	Comments
Schema Modeling	Available in the Data Object Editor.
Slowly changing dimensions	Includes support for Slowly Changing Dimension (SCD) types 2 and 3.
Sources and Targets	Available in the Design Center
XML file as target	Supported through the flat file operator.
ETL Design	Available in the Mapping, Process Flow, and Schedule Editors
Advanced ETL features	Includes the following ETL features: transportable modules, multiple configurations, and pluggable mappings.  Includes the following operators associated with reusing mapping code: pluggable mapping, pluggable mapping input signature, pluggable mapping output signature.
Real Applications Cluster (RAC) support	Includes maintaining the Warehouse Builder design environment in a Real Applications Cluster environment. Without the Enterprise ETL option, you can install the Warehouse Builder repository in a Real Applications Cluster environment for the limited purposes of executing in that environment.
Mapping operators	Includes the operators for handling complex types: varray iterator, construct object, and expand object.
Target load ordering	For mappings with multiple targets, includes functionality to specify the order in which the targets are loaded.
Transformations	Seeded Spatial and Streams transformations.
Process flows	Includes the following advanced process flow functionality: <ul style="list-style-type: none"> <li>■ Activity templates</li> <li>■ Variables support: Using variables in process flows to pass information across activities, including the Assign and Set Status activities.</li> <li>■ Looping activities such as For Loop and While Loop</li> <li>■ Route and Notification activities, but not the Email activity</li> </ul> The Data Auditor activity requires the <a href="#">Warehouse Builder Data Quality Option</a> .
Metadata Management	Available in the Design Center
Lineage and impact analysis	Includes interactive analysis available in the Design Center.
Change propagation	Includes automatic propagation of property changes to impacted objects through the Lineage and Impact Analyzer.
Extensibility	Includes project based and public based user-defined objects, user-defined associations, and user-defined modules. Includes creating icon sets and assigning custom icons to objects.
Deployment and Execution	Available in the Control Center Manager
Schedules	Includes functionality to model schedules for mappings and process flows.

**Table 1–2 (Cont.) Warehouse Builder Enterprise ETL Option**

Area and Functionality	Comments
Business intelligence deployment targets	Includes direct deployment to the Discoverer End User Layer (EUL).

## Warehouse Builder Data Quality Option

The Data Quality option enables you to convert raw data into quality information. Developers and data librarians can gain insight into their data and identify previously unknown data quality problems. Subsequently, developers can define rules and generate mappings that correct the data. Based on the data rules, developers can also create data auditors to ensure the quality of incoming data on a repeated basis.

[Table 1–3](#) lists the Warehouse Builder functionality available in the Data Quality option.

**Table 1–3 Warehouse Builder Functionality in the Data Quality Option**

Area and Functionality	Comments
<b>Data Profiling</b>	<b>Available in the Data Profile Editor and the Mapping Editor</b>
Data profiling	Includes functionality for data profiling and data drill-down.
Data rules	Includes functionality for data rule derivation and data rule profiling. Includes support for custom and predefined data rules and support for apply data rules to data objects.
Data corrections	Includes the generation of mappings that correct data based on data rules.
<b>ETL Design</b>	<b>Available in the Process Flow Editor</b>
Process flows	Includes the use of the Data Auditor Monitor activity in the Process Flow Editor.

## Warehouse Builder Connector - E-Business Suite

The Warehouse Builder Connector to E-Business Suite provides access to the technical and business metadata within Oracle E-Business Suite. Subsequently, you can build mappings and process flows that either source or target Oracle E-Business Suite. The connector also facilitates deployment to Oracle Concurrent Manager and access to Oracle E-Business Suite at execution-time.

With the E-Business Suite Connector for Warehouse Builder, you can use the functionality listed in [Table 1–4](#) in addition to the [Warehouse Builder Core Functionality](#).

**Table 1–4 Warehouse Builder Functionality in the E-Business Suite Connector**

Area and Functionality	Comments
<b>Metadata Management</b>	<b>Available in the Design Center</b>
Oracle E-Business Suite	Includes access to technical and business metadata in E-Business Suite.
<b>ETL Design</b>	<b>Available in the Mapping Editor and Process Flow Editor</b>
ETL support	Enables the inclusion of E-Business Suite data objects into mappings and process flows.
<b>Deployment and Execution</b>	<b>Available in the Control Center Manager</b>
Deploying ETL objects	Includes deploying mappings and process flows designed with E-Business Suite objects.

**Table 1–4 (Cont.) Warehouse Builder Functionality in the E-Business Suite Connector**

Area and Functionality	Comments
Deployment targets	Includes deployment to Oracle Concurrent Manager. This also available in the <a href="#">Warehouse Builder Enterprise ETL Option</a> .

## Warehouse Builder Connector - PeopleSoft

With Warehouse Builder Connector to PeopleSoft, you can connect to and then extract data and metadata from PeopleSoft applications. The connection to the PeopleSoft application using database users with the appropriate privileges set by the DBA.

After you import metadata from PeopleSoft applications, you can work with packaged applications as you would with other SQL based systems. You can include PeopleSoft objects as sources or targets in Warehouse Builder mappings, create process flows, and generate SQL code.

This connector can operate with non-Oracle databases after you establish a connection to those databases. [Table 1–5](#) lists the functionality available in the Warehouse Builder Connector to PeopleSoft.

**Table 1–5 Warehouse Builder Functionality in the PeopleSoft Connector**

Area and Functionality	Comments
<b>Metadata Management</b>	<b>Available in the Design Center</b>
PeopleSoft	Includes access to technical and business metadata in PeopleSoft.
<b>ETL Design</b>	<b>Available in the Mapping Editor and Process Flow Editor</b>
ETL support	Enables the inclusion of PeopleSoft objects into mappings and process flows.
<b>Deployment and Execution</b>	<b>Available in the Control Center Manager</b>
Deploying ETL objects	Includes deploying mappings and process flows designed with PeopleSoft objects.

## Warehouse Builder Connector - SAP R/3 Connector

With the Warehouse Builder Connector to SAP R/3, you can connect to and then extract data and metadata from SAP R/3. You can access both the technical and business metadata in the SAP R/3 application. The connector masks the complexities of the SAP metadata by displaying pool tables and cluster tables as regular tables. To access SAP metadata, you use an RFC call with a SAP GUI account as authentication.

After you import SAP metadata and understand relationships, you can use the SAP objects like any other objects in Warehouse Builder. You can include SAP R/3 objects in Warehouse Builder mappings and process flows and generate ABAP code. The connector allows direct deployment and execution of ABAP in SAP and execution of generated and uploaded ABAP from production process flows. The connector also supports the use of substitution variables to facilitate transporting of ABAP code between development and production systems by the SAP administrator. Variable support in ABAP also allows you to easily create change data capture flows, either based on dates or based on ID ranges.

[Table 1–6](#) lists the functionality available in the Warehouse Builder Connector to SAP R/3.

**Table 1–6 Warehouse Builder Functionality in the SAP R/3Connector**

Area and Functionality	Comments
Metadata Management	Available in the Design Center
SAP R/3	Includes access to technical and business metadata in SAP R/3.
ETL Design	Available in the Mapping Editor and Process Flow Editor
ETL support	Enables the inclusion of SAP R/3 objects in mappings and process flows. Generates ABAP code. Includes parameterization and tuning of ABAP code. To enable migration between SAP R/3 environments such as development versus production environments, this connector supports the use of substitution variables to facilitate transporting ABAP code.
Deployment and Execution	Available in the Control Center Manager
Deploying ETL objects	Includes deploying mappings and process flows designed with SAP R/3 objects.

## Warehouse Builder Connector - Siebel

The Warehouse Builder Connector to Siebel enables you to connect to and extract data and metadata from Siebel applications. The connection to the Siebel applications is using database users with the appropriate privileges set by the DBA.

After you import metadata from Siebel applications, you can use Siebel objects in mappings, create process flows containing Siebel objects, and generate SQL code.

You can use this connector with non-Oracle databases after you establish a connection to those databases. [Table 1–7](#) lists the functionality available in the Warehouse Builder Connector to Siebel.

**Table 1–7 Warehouse Builder Functionality in the Siebel Connector**

Area and Functionality	Comments
Metadata Management	Available in the Design Center
Siebel	Includes access to technical and business metadata in Siebel.
ETL Design	Available in the Mapping Editor and Process Flow Editor
ETL support	Enables the inclusion of Siebel objects into mappings and process flows.
Deployment and Execution	Available in the Control Center Manager
Deploying ETL objects	Includes deploying mappings and process flows designed with Siebel objects.





---

---

# Getting Started with Oracle Warehouse Builder

Oracle Warehouse Builder is a flexible tool that enables you to design and deploy various types of data integration strategies. Projects commonly implemented using Warehouse Builder involve mission critical operational systems, migration scenarios, integration of disparate operational systems, and traditional data warehousing. This chapter provides an introduction to using Warehouse Builder. It provides a starting point for using Warehouse Builder for the first time user and serves as a road map to the documentation.

If you have already read the *Oracle Database 2 Day + Data Warehousing Guide*, you may recognize some of the same content repeated here in an expanded format and with additional information for long-term planning and maintenance of not only data warehouses but data integration solutions in general.

This chapter includes the following topics:

- [Understanding the Basic Concepts](#)
- [Implementing a Data Integration Solution](#)

## Understanding the Basic Concepts

Oracle Warehouse Builder is comprised of a set of graphical user interfaces to assist you in implementing solutions for integrating data. In the process of designing solutions, you create various objects that are stored as metadata in a centralized repository, known as a workspace.

The **workspace** is hosted on an Oracle Database. As a general user, you do not have full access to the workspace. Instead, you can access those workspaces to which you have been granted access.

You log in to a workspace by starting the **Design Center**, which is the primary graphical user interface. Use the Design Center to import source objects, design ETL processes such as mappings, and ultimately define the integration solution.

A **mapping** is an object in which you define the flow of data from sources to targets. Based on a mapping design, Warehouse Builder generates the code required to implement the ETL logic. In a data warehousing project, for example, the integration solution is a target warehouse. In that case, the mappings you create in the Design Center ultimately define a target warehouse.

After you complete the design of a mapping and prompt Warehouse Builder to generate the code, the next step is to deploy the mapping. **Deployment** is the process of copying the relevant metadata and code you generated in the Design Center to a

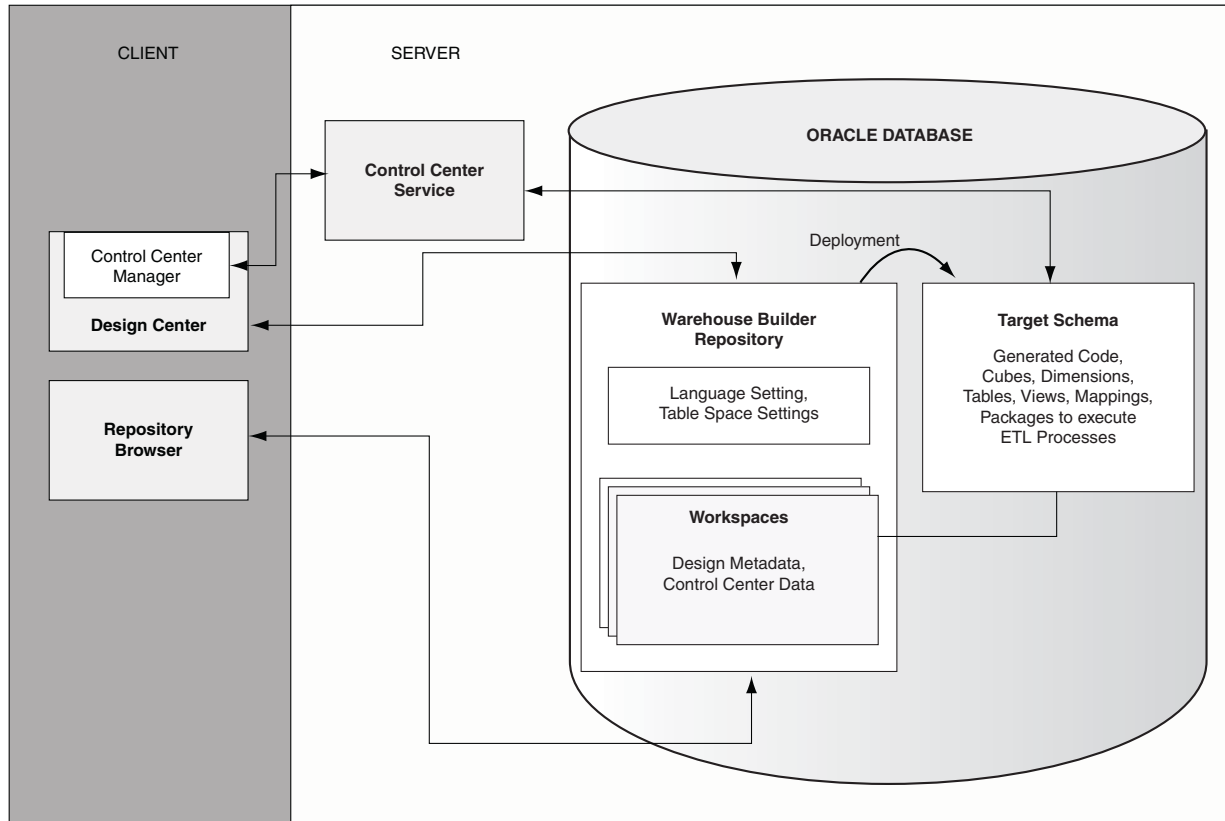
target schema. The **target schema** is generically defined as the Oracle Database which will execute the ETL logic you designed in the Design Center. Specifically, in a traditional data warehousing implementation, the data warehouse is the target schema and the two terms are interchangeable.

Figure 2–1 illustrates the Warehouse Builder components.

As previously noted, the Design Center is the primary user interface. It is also a centralized interface in that you can start from it all the client based tools, including the Control Center Manager. A secondary user interface is the web-based **Repository Browser**. In addition to browsing design metadata and auditing execution data, you can view and create reports.

For the purposes of this illustration, the target schema and the repository exist on the same Oracle Database; however, in practice, target schemas often exist on separate databases. To deploy design objects and subsequently execute the generated code, use the **Control Center Manager**, which is the client interface that interacts with the target schema through the control center service.

Figure 2–1 Warehouse Builder Components



## Implementing a Data Integration Solution

Use Warehouse Builder to create a data warehouse in the following recommended order:

1. [Before You Begin](#)
2. [Preparing the Warehouse Builder Design Center](#)
3. [Importing the Source Metadata](#)

4. [Profiling Data and Ensuring Data Quality](#)
5. [Designing the Target Schema](#)
6. [Designing ETL Logic](#)
7. [Deploying the Design and Executing the Data Integration Solution](#)
8. [Monitoring and Reporting on the Data Warehouse](#)

## Before You Begin

Before you can use any of the Warehouse Builder client components, first ensure you have access to a Warehouse Builder workspace.

**To begin using Warehouse Builder, take the following steps:**

1. Install the Warehouse Builder software and create the necessary workspaces as described in the *Oracle Warehouse Builder Installation and Administration Guide*.

If an administrator has previously completed the installation, contact that person for the required connection information.

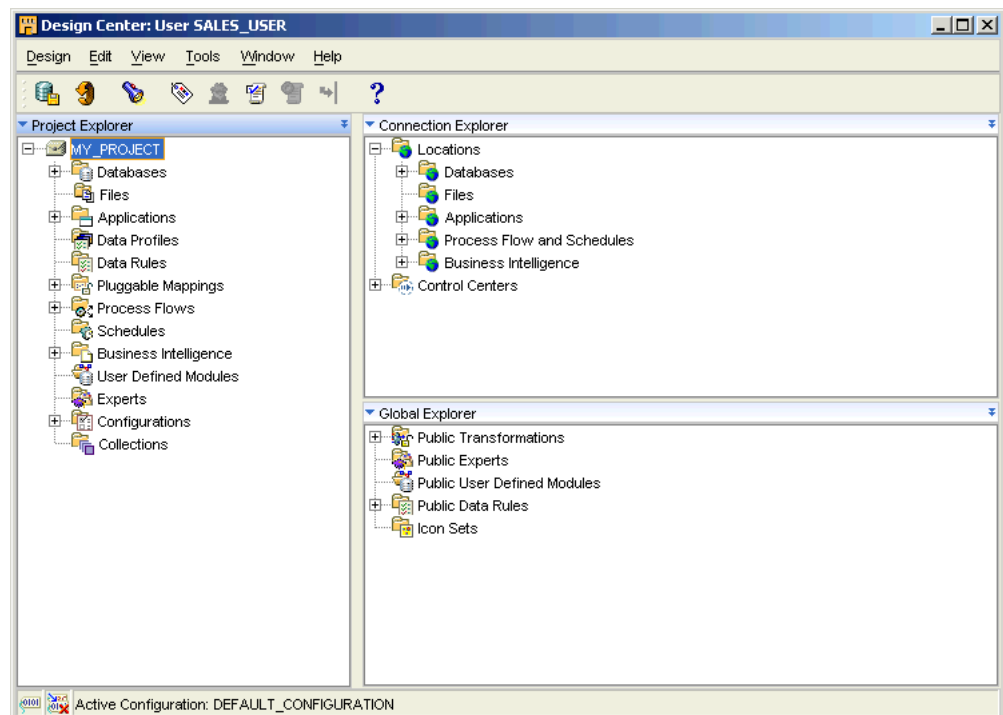
2. Start the Design Center.

On a Windows platform, from the **Start** menu, select **Programs**. Select the Oracle home in which Warehouse Builder is installed, then **Warehouse Builder**, and then **Design Center**.

On a Linux platform, run `owbclient.sh` located in the `owb/bin/unix` directory in the Oracle home for Warehouse Builder.

Figure 2–2 shows the Design Center with the top level folders in each of its three explorers expanded.

**Figure 2–2 The Design Center**



Use the Project Explorer to manage design objects for a given workspace. The design objects are organized into **projects** which provide a means for structuring the objects for security and reusability. Each project contains nodes for each type of design object that you can create or import.

Use the Connection Explorer to establish connections between the Warehouse Builder workspace and databases, data files, and applications.

Use the Global Explorer to manage objects that are common to all projects in a workspace and to administer security. Note that the Security node is visible to users who have an administrator role as discussed in the Oracle Warehouse Builder Installation and Administration Guide.

## Preparing the Warehouse Builder Design Center

To prepare the Design Center, complete the following steps:

1. In the Project Explorer, identify the project to be used.

If you are satisfied with the single default project, MY\_PROJECT, continue with the next step.

Alternatively, you can rename MY\_PROJECT or define more projects. Each project you define is organized in the same fashion with nodes for databases, files, applications, and so on. For a different organization, consider creating optional collections as described in "[Defining Collections](#)" on page 3-9.

2. Connect to source and target data objects.

In the Connection Explorer, establish these connections by defining **locations**. Expand the Location node and the nodes within it to gain a general understanding of the types of source and targets you can access from Warehouse Builder.

To create a location, right-click the appropriate node and select **New**. Fill in the requested connection information and select **Test Connection**. In this step, you merely establish connections to sources and targets. You do not move data or metadata until subsequent steps.

For more information about locations see "[About Locations](#)" on page 11-5.

3. Identify the target schema.

Although you can use a flat file as a target, the most common and recommended scenario is to use the Oracle Database as the target schema.

To define the target schema, begin by creating a module. **Modules** are grouping mechanisms in the Project Explorer that correspond to locations in the Connection Explorer. The Oracle target module is the first of several modules you create in Warehouse Builder.

In the Project Explorer, expand the Databases node. Right-click **Oracle** and select **New**. The Create Module wizard displays. Set the module type to Warehouse Target and specify whether the module will be used in development, quality assurance, or production. This module status is purely descriptive and has no bearing on subsequent steps you take.

When you complete the wizard, the target module displays with nodes for mappings, transformations, tables, cubes and the various other types of objects you utilize to design the target warehouse.

4. Create a separate Oracle module for the data sources. (Optional)

At your discretion, you can either create another Oracle module to contain Oracle source data or proceed to the next step.

5. Identify the execution environment.

Under the Connection Explorer, notice the Control Centers node. A control center is an Oracle Database schema that manages the execution of the ETL jobs you design in the Design Center in subsequent steps.

During installation, Warehouse Builder creates the `DEFAULT_CONTROL_CENTER` schema on the same database as the workspace.

If you choose to utilize the default execution environment, continue to the next step. Alternatively, you can define new control centers at any time. For more information and instructions, see "[Deploying to Target Schemas and Executing ETL Logic](#)" on page 11-1.

6. Prepare development, test, and production environments. (Optional)

Thus far, these instructions describe the creation of a single project corresponding to a single execution environment. You can, however, reuse the logical design of this project in different physical environments such as testing or production environments.

Deploy a single data system to several different host systems or to various environments, by "[Creating Additional Configurations](#)" on page 11-14.

7. Adjust the client preference settings as desired or accept the default preference settings and proceed to the next step.

From the main menu in the Design Center, select **Tools** and then **Preferences**.

As a new user, you may be interested in setting the [Environment Preferences](#), the locale under [Appearance Preferences](#), and the naming mode under [Naming Preferences](#). For information on all the preferences, see "[Setting Preferences](#)" on page 3-2.

## Importing the Source Metadata

1. Create modules for each type of design object you intend to import metadata.

In the Project Explorer, select a node such as Files. For that node, determine the locations from which you intend to ultimately extract data. Now create a module for each relevant location by right-clicking on the node and select **New**.

2. Import metadata from the various data sources.

Right-click the module and select **Import** to extract metadata from the associated location. Warehouse Builder displays a wizard to guide you through the process of importing data.

For an example and additional information on importing data objects, see "[Identifying Data Sources and Importing Metadata](#)" on page 4-1.

3. For the metadata you imported, profile its corresponding data. (Optional)

Before continuing to the next step, consider using the data profiling option to ensure data quality as described in "[Understanding Data Quality Management](#)" on page 5-1.

## Profiling Data and Ensuring Data Quality

Data can only be transformed into actionable information when you are confident of its reliability. Before you load data into your target system, you must first understand the structure and the meaning of your data, and then assess the quality.

Consider using the data profiling option to better understand the quality of your source data. Next, correct the source data and establish a means to detect and correct errors that may arise in future loads. For more information, on data profiling and data quality, see "[Understanding Data Quality Management](#)" on page 5-1.

## Designing the Target Schema

1. Create and design the data objects for the Oracle target module.

In previous steps, you may have already imported existing target objects. For new target objects, design any of the dimensional or relational objects listed in [Table 6-1](#) on page 6-2.

To create data objects, you can either start the appropriate wizard or use the Data Object Editor. To use a wizard, right-click the node for the desired object and select **New**. After using a wizard, you may want to modify the object in the editor. In that case, right-click the object and select **Open Editor**.

For additional information, see "[Designing the Target Schema](#)" on page 6-39.

2. As you design objects, be sure to frequently validate the design objects.

You can validate objects as you create them, or validate a group of objects together. In the Project Explorer, select one or more objects or modules, then click the Validate icon.

Examine the messages in the Validation Results window. Correct any errors and try validating again.

To redisplay the most recent validation results at a later time, select **Validation Messages** from the View menu.

For additional information, see "[Validating Data Objects](#)" on page 6-45.

3. Configure the data objects.

Configuring data objects sets the physical properties of the object. You must not generate and deploy data objects without specifying the physical property values.

When you create data objects, Warehouse Builder assigns default configuration property values based on the type of object. In most cases, these default values are appropriate. You can edit and modify the configuration property values of objects according to your requirement. For example, you configure a table to specify the name of the tablespace in which it is created.

To configure a data object, select the data object in the Project Explorer and click the Configure icon. Or right-click the data object in the Project Explorer and select **Configure**.

4. When satisfied with the design of the target objects, generate the code.

Generation produces a DDL or PL/SQL script to be used in subsequent steps to create the data objects in the target schema. For more information about generation, see "[Generating Data Objects](#)" on page 6-47.

In the Data Object Editor, you can generate code for a single object by clicking the Generate icon.

In the Project Explorer, select one or more objects or modules, then click the Generate icon. Examine the messages in the Generation Results window. To redisplay the most recent generation results at a later time, select **Generated Scripts** from the View menu.

You can save the generated script as a file and optionally deploy it outside Warehouse Builder.

## Designing ETL Logic

1. Design mappings that define the flow of data from a source to target objects.

In the Project Explorer, expand the Oracle target module, right-click the Mappings node and select **New**.

The Mapping Editor enables you to define the flow of data visually. You can drag-and-drop operators onto the canvas, and draw lines that connect the operators. Operators represent both data objects and functions such as filtering, aggregating, and so on.

Follow the [Instructions for Defining Mappings](#), concluding with generating the code for the mapping.

2. To manage dependencies between mappings, see "[Designing Process Flows](#)" on page 8-1.

## Deploying the Design and Executing the Data Integration Solution

Recall that deployment is the process of copying the relevant metadata and code you generated in the Design Center to a target schema. This step is necessary to enable the target schema to execute ETL logic such as mappings.

**To deploy and execute, complete the following steps:**

1. Deploy objects from either the Design Center or Control Center Manager.

In this step, you define the objects in the target schema. You need do this only once.

The simplest approach is to deploy directly from the Design Center by selecting an object and clicking the Deploy icon. In this case, Warehouse Builder deploys the objects with the default deployment settings.

Alternatively, if you want more control and feedback on how Warehouse Builder deploys objects, from the Design Center menu select **Tools**, then **Control Center Manager**.

Whether you deploy objects from the Design Center or the Control Center Manager, be sure to deploy all associated objects. For example, when deploying a mapping, also deploy the target data objects such as tables that you defined and any associated process flows or other mappings.

For more information, see "[Deploying to Target Schemas and Executing ETL Logic](#)" on page 11-1.

2. Execute the ETL logic to populate the target warehouse.

In this step, you move data for the first time. Repeat this step each time you want to refresh the target with new data.

You have two options for executing the ETL logic in mappings and process flows. You can create and deploy a schedule as described in "[Process for Defining and](#)

[Using Schedules](#)" on page 11-17. Or you can execute jobs manually as described in "[Starting ETL Jobs](#)" on page 11-11.

## Monitoring and Reporting on the Data Warehouse

It is essential to ensure the quality of data entering your data warehouse over time. Data auditors enable you to monitor the quality of incoming data by validating incoming data against a set of data rules and determining if the data confirms to the business rules defined for your data warehouse. For more information about data auditors and data rules, see "[Understanding Data Quality Management](#)" on page 5-1.

Although the Control Center Manager displays histories for both deployment and execution, the Repository Browser is the preferred interface for monitoring and reporting on Warehouse Builder operations.



---

---

## Setting Up Warehouse Builder

This chapter includes additional and optional steps that you may take when initially designing your data system. This chapter covers the following topics:

- [Organizing Design Objects into Projects](#)
- [Setting Preferences](#)
- [Defining Collections](#)
- [Alternative Interfaces](#)

### Organizing Design Objects into Projects

Projects are the largest storage objects within a Warehouse Builder workspace. Projects store and organize related metadata definitions. You should include all the objects in a project that you think can or will share information. These definitions include data objects, mappings, and transformation operations. The definitions are organized into folders within the project. By creating multiple projects, you can organize the design and deployment of a large system.

#### To create a project:

1. In the Project Explorer, right-click a project, such as `MY_PROJECT`, and select **New**.

The Create Project dialog box is displayed.

2. Click **Help** for additional instructions.

Each Warehouse Builder workspace has a default project called `MY_PROJECT`. You can rename `MY_PROJECT`, or you can delete it after you create other projects. However, a workspace must contain at least one project at all times.

Because projects are the main design component in Warehouse Builder, some restrictions are enforced to prevent you from deleting them unintentionally. You cannot delete:

- The currently active or expanded project.
- The only project in a workspace.

#### To delete a project:

1. In the **Project Explorer**, collapse the project that you want to delete. You cannot delete the project when it is expanded.
2. Select and expand any other project.
3. Highlight the project you want to delete and, from the **Edit** menu, select **Delete**.

*or*

Right-click the project and select **Delete**.

The Warehouse Builder Warning dialog box provides the option of putting the project in the recycle bin.

4. Click **OK** to delete the project.

## Setting Preferences

Warehouse Builder provides a set of user preferences that enable you to customize your user interface environment. To set user preferences, select **Tools** and then **Preferences** from the Design Center menu. The Preferences dialog box used to set preferences is displayed.

The Preferences dialog box contains two sections. The section on the left lists the categories of preferences. The section on the right displays the preferences and their corresponding values. Click a category on the left panel to display the preferences it contains and its value in the right panel.

Warehouse Builder enables you to set the following types of preferences:

- [Appearance Preferences](#)
- [Control Center Monitor Preferences](#)
- [Data Profiling Preferences](#)
- [Deployment Preferences](#)
- [Environment Preferences](#)
- [Generation/Validation Preferences](#)
- [Logging Preferences](#)
- [Naming Preferences](#)
- [Security Preferences](#)

### Appearance Preferences

The Appearance category contains the Locale preference. Use the Locale list to set the language you want the client text to display. This list displays the language options. Warehouse Builder prompts you to restart the computer in order to use the new language setting.

The Locale selection does not define the character set of your repository; it only affects the text and menu options on the client user interface. The repository character set is determined by the database.

### Control Center Monitor Preferences

Use the Control Center Monitor category to set preferences that control the display of components in the control center. When you use the control center to deploy or execute objects, the Job Details window displays the results of deployment or execution. The Control Center Monitor preferences enable you to control the display of components in the object tree of the Job Details window.

---

**Note:** Warehouse Builder displays the Job Details window only if you select the **Show Monitor** preference under the Process node of the Deployment preferences category.

If this option is not selected, view the Job Details window by double-clicking the row representing the deployment or execution job in the Control Center Jobs panel of the Control Center.

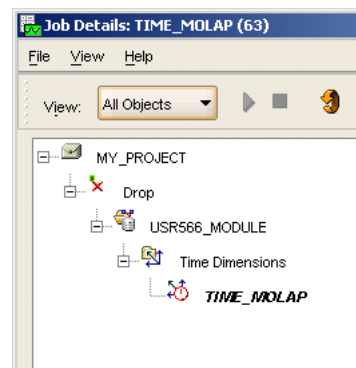
---

You can set the following control center monitor preferences:

- **Show Project:** Select this option to display the project name in the Job Details window object tree. When this option is selected, the object tree displays a node for the project name. All the objects are displayed under the project node.
- **Show Module:** Select this option to display the name of the module to which the object being deployed or executed belongs in the Job Details window. When this option is selected, the object tree displays a node for the module. Expand the module node to view the object details.
- **Show Location:** Select this option to display the location name in the object tree of the Job Details window.
- **Show Action:** Select this option to display the action performed on the object in the object tree of the Job Details window. The actions performed include Create, Drop, Deploy, and Upgrade.
- **Show Type:** Select this option to display the type of object in the object tree of the Job Details window. When you select this option, a node is displayed for the type of object and the objects are listed under the respective nodes.

Figure 3–1 displays the object tree of the Job Details window in which the following Control Center preferences were selected: Show Project, Show Module, Show Action, and Show Type.

**Figure 3–1 Job Details Window with Control Center Monitor Preferences**



## Data Profiling Preferences

Use the Data Profiling category to set the preferences for data profiling. This section contains the following preferences:

- **Data Rule Folder Name:** Use this option to set the name of the folder that contains the data rules as a result of data profiling.
- **Default Profile Location:** Use this option to set the default location that is used to store the data profiling results. You can override this setting by selecting a

different location as your profile location. In the Data Profile Editor, from the Edit menu, select **Properties**. Use the Data Locations tab to change the default profile location.

## Deployment Preferences

The Deployment category enables you to set deployment preferences such as displaying the deployment monitor, prompting for execution parameters, and showing completion messages. This enables you to control some of the popup windows that are displayed by the Control Center Manager during object deployment.

Deployment preferences are divided into two sections: Process and Tools. Expand the **Deployment** node in the Preferences dialog box. Two nodes called Process and Tools are displayed. Click the node for which you want to set preferences.

### Process

Set the following deployment options in this section:

- **Allow Undo/Redo:** Select this option to allow the user to undo and redo a deployment upgrade job. You can undo or redo a deployment upgrade job using the Job Details window. To display the Job Details window for a job, double-click the job in the Control Center Jobs panel of the Control Center Manager.
- **Pause After Compile:** Select this option to pause deployment after script generation. This means that you must explicitly deploy an object after it is successfully generated.
- **Prompt for Commit:** Select this option to prompt the user to commit design time changes before a deployment. When you deploy objects from the Design Center, if there are any unsaved design changes, Warehouse Builder prompts you to save these changes by displaying the Warehouse Builder Warning dialog box. Click **Save** to commit unsaved design changes. Click **Cancel** to terminate the deployment.

If you do not set this option, Warehouse Builder saves any design changes before the deployment job.

- **Prompt for Job Name:** Select this option to prompt the user for the name of a deployment job. When this option is not selected, Warehouse Builder assigns a default job name.
- **Prompt for Execution Parameters:** Select this option to prompt the user for the values of execution parameters. If you do not select this option, Warehouse Builder uses the default value of parameters during the execution. The user is not prompted to provide the parameter values.
- **Show Monitor:** Select this option to display the Job Details window when you deploy or execute an object. This dialog box displays details of the objects being deployed, deployment progress, and deployment status.
- **Show Deployment Completion Message:** Select this option to display an alert indicating that the deployment job has completed.
- **Show Design Center Deployment Job:** Select this option to display the Control Center Jobs dialog box when you deploy an object from the Design Center. The Control Center Jobs dialog box, which is similar to the Jobs panel of the Control Center Manager, contains the Deployment, Execution, and Scheduled tabs. Use this option to view the status of a deployment job while deploying using the Design Center.

## Tools

Set the following deployment options:

- **Show Monitor Tree:** Select this option to show the Job Details window when you perform a deployment or execution.
- **Show Monitor Results:** Select this option to display the deployment or execution results in Control Center Manager.
- **Show Monitor Logfile:** Select this option to display the log file in the Control Center Manager.

## Environment Preferences

The Environment category enables you to set generic preferences that control the client environment such as displaying welcome pages for wizards and recycle bin preferences.

Set the following environment preferences:

- **Personality:** For the standard installation, set the value of this preference to Default. For a customized installation, this preference tailors the types of objects shown in the Project Explorer tree. Oracle recommends that you change the value of this preference, from Default, only after discussion with your Oracle system administrator. This feature is reserved for future use.
- **Allow Optimize Repository Warning on Startup:** Select this option to collect schema statistics when you log in to Warehouse Builder. Collecting schema statistics improves repository performance. If this option is selected, at log on, Warehouse Builder determines if statistics must be gathered for the repository schema. If statistics must be gathered, a warning dialog box is displayed asking if you want to gather statistics now. Click **Yes** to collect schema statistics and optimize the repository.

If you do not select this option, you can still collect schema statistics from the Design Center by selecting **Optimize Repository** from Tools menu.

- **Hide All Wizard Welcome pages:** Select this option to hide the welcome page of all wizards. Every wizard in Warehouse Builder starts with a Welcome page that summarizes the steps to follow to complete that task. To display the Welcome page for all wizards, deselect this preference.
- **Show Delete Confirmation Dialog Box:** Select this option to display a dialog box that asks for a confirmation before deleting an object. When this option is selected, if you delete an object, the Warehouse Builder Warning dialog box is displayed. Click **Yes** to delete the object. Click **No** to cancel the Delete operation and retain the object.
- **Recycle Deleted Objects:** Select this option to move deleted objects to the recycle bin. If this option is not selected, any objects you delete are lost and you have no way of recovering them.
- **Empty Recycle Bin on Exit:** Select this option to empty the contents of the recycle bin when you exit the Warehouse Builder client. Deselect this option to save the recycle bin objects across sessions.

## Generation/Validation Preferences

The Generation/Validation category enables you to set preferences related to generation and validation of Warehouse Builder design objects. Use these preferences to control what is displayed in the Generation Results window or Validation Results

window. These dialog boxes are displayed when you generate or validate an object from the design center. You can set the following preferences:

- **Show Project:** Select this option to display the project node in Validation Results window or the Generation Results window.
- **Show Module:** Select this option to display the module node in Validation Results window or the Generation Results window.
- **Show Location:** Select this option to display the location node in Validation Results window or the Generation Results window.
- **Show Action:** Select this option to display the action node in Validation Results window or the Generation Results window.
- **Show Type:** Select this option to display the type node in Validation Results window or the Generation Results window.

## Logging Preferences

The Logging Preferences category enables you to set log file options such as file location, file size, and types of messages saved to any log file. The log file contains messages relating to your design process. By default a message log is saved to the default location. Following are the logging preferences that you can set:

- **File Path:** Represents the location where the log files are saved. Type the complete path or use the Browse button to select the location. The default location is `OWB_ORACLE_HOME\owb\bin\admin`.
- **File Name:** Represents the name of the log file. Do not include a file extension when you specify a file name.
- **Maximum Size (Kb):** Indicate the maximum file size for the log file(s) in KB. There are two log files: `<logfilename>.0`, and `<logfilename>.1`. When the maximum size of the first log file `<logfilename>.0` is reached, Warehouse Builder starts writing to the second, `<logfilename>.1`. When the maximum size of the second one is reached, Warehouse Builder starts overwriting the first.
- **Log Error Messages:** Select this option to write all error messages to the log file.
- **Log Warning Messages:** Select this option to write all warning messages to the log file.
- **Log Information Messages:** Select this option to write all information messages to the log file.

## Naming Preferences

The Naming Preferences category enables you to set naming preferences by selecting whether you want to view objects in business or physical name mode. You can also set up how you want to propagate object name changes.

Set the following naming preferences:

- **Naming Mode:** Select whether to display objects using their physical or business names.
- **Propagate Name Changes:** Propagate Name Changes from the current naming mode to the other naming mode.

## About Naming Modes

Warehouse Builder maintains a business and a physical name for each object stored in the repository. A business name is a descriptive logical name for an object.

When you generate DDL scripts for a named object, the physical names are used. Physical names must conform to the syntax rules for basic elements as defined in the *Oracle Database SQL Language Reference*. Names must be unique within their category:

- Module names must be unique within a project.
- Warehouse object names must be unique within a warehouse module. This includes the names of tables, dimensions, cubes, mappings, materialized views, sequences, views and indexes.
- Transformation names must be unique within a transformation package.

A business name must conform to these rules:

**Business Name Mode** You can create a business name for an object or change the business name of an existing object when Warehouse Builder is in business name mode. Warehouse Builder editors, wizards, and property sheets display the business names of objects in this mode.

- The length of a name cannot exceed 200 characters.
- The name must be unique within its category.
- All source modules reflect the case of the imported source and are subject to the double-quotes rules as defined in the *Oracle Database SQL Language Reference*.

Copy operations from a source to a target in a mapping do not propagate case.

When you create a business name, Warehouse Builder generates a valid physical name that resembles the business name. If you create a business name that duplicates an existing physical name, Warehouse Builder appends an underscore and a number in order to create a unique name.

**Physical Name Mode** You can create a physical name for an object or change the physical name of an existing object when Warehouse Builder is in the Physical name mode. Warehouse Builder editors, wizards, and property sheets display physical names of objects in this mode. Physical names are converted to uppercase.

A physical name must:

- Contain no more than 30 characters.
- Conform with the basic syntax rules for schema objects defined by the *Oracle Database SQL Language Reference*.

---



---

**Note:** A collection can have a physical name containing up to 200 characters.

---



---

Warehouse Builder prevents you from entering an invalid physical name. For example, you cannot enter a duplicate name, a name with too many characters, or a name that is a reserved word.

**Setting the Name Mode** To create or change a business name for an object, Warehouse Builder must be in business name mode. To create or change a physical name for an object, Warehouse Builder must be in physical name mode.

The default naming preferences for Warehouse Builder are as follows:

- **Mode:** The default setting for the mode is physical name mode.

- **Propagation:** The default propagation setting is to propagate physical name to business name.

Icons for the name mode and name propagation settings are located in the lower-left corner of the editors. These icons indicate the current naming preference setting.

Warehouse Builder saves your naming preferences across sessions. The name mode preference is stored in a file on the client workstation. If you use Warehouse Builder from another workstation, your preferences may be different.

## Security Preferences

Only administrators can edit the security preferences. Administrators can set the following preferences:

### **Persist Location Password in Metadata**

This option determines whether or not location passwords are persisted across Warehouse Builder design sessions.

By default, this option is deselected, which is the more secure option. Warehouse Builder retains location passwords for the length of the design session only. That is, the first time you start tools such as the Data Viewer or Debugger, you must enter the appropriate location passwords.

If selected, Warehouse Builder persists encrypted versions of location passwords in the workspace. You can start tools such as the Data Viewer and Debugger without entering passwords each time.

### **Share Location Password During Runtime**

This preference determines whether or not the location passwords users enter during the design phase can be shared with other users. For example, assume that user `Dev1` designs mapping `MAP1`. To access the sources and targets for this mapping, `Dev1` defines the locations to each source and target including a username and password. When other users subsequently attempt to execute `MAP1` or view data associated with it, the [Share Location Password During Runtime](#) preference determines whether or not each user must enter the password each time in the Design Center or the first time in the Control Center.

[Share Location Password During Runtime](#) works in conjunction with [Persist Location Password in Metadata](#). The most secure mode, and therefore the default behavior, is for both options to be deactivated. In this case, each user including `Dev1` must enter their password once for each Design Center session and the first time they attempt to use that location in the Control Center. Depending on your security requirements, you may want each user to define their own location for a given source or target

If both [Share Location Password During Runtime](#) and [Persist Location Password in Metadata](#) are activated, then any user can access a schema given that any user previously defined the location. Therefore, user `Oper2` can execute `MAP1` given that `Dev1` or any other user previously defined the location with valid credentials.

### **Default Metadata Security Policy**

Specify the default security policy to be applied. Minimum security allows all users full control over objects any newly registered user creates. Maximum security, however, restricts access to the newly registered user that created the object and Warehouse Builder administrators.



This setting is not retroactive. That is, if you change this setting in an existing Warehouse Builder implementation, the setting does not affect existing users and existing objects. You must change the security on existing objects manually.

## Defining Collections

Collections are structures in Warehouse Builder that store the metadata you want to export to other tools and systems. Collections enable you to perform the following tasks:

- Organize a large logical warehouse.
- Validate and generate a group of objects.

When you create a collection, you do not create new objects or copies of existing objects. You create shortcuts pointing to objects already existing in the project. You can use a collection to quickly access a base object and make changes to it.

You can define more than one collection within a project and an object can be referenced by more than one collection. For example, each user that accesses a project can create his own collection of frequently used objects. The users can also add the same objects (such as mappings, tables, or process flows) to their separate collections.

Each user can also delete either the shortcut or the base object. Shortcuts to deleted objects are deleted in the collection.

When you open an object in a collection, you obtain a lock on that object. Warehouse Builder prevents other users from editing the same object from another collection.

## Creating a Collection

Use the Create Collection Wizard to define a collection.

### To define a new collection:

1. Select and expand a project node on the Project Explorer.
2. Right-click the Collections node and select **New**.

Warehouse Builder displays the Welcome page for the Create Collections Wizard. This page lists the steps to create a collection. Click **Next** to proceed.

3. Provide information on the following pages of the Create Collection wizard:
  - [Name and Description Page](#)
  - [Contents Page](#)
  - [Summary Page](#)

### Name and Description Page

Use the Name and Description page to provide a name and an optional description for the collection. The name should be unique within the module. In physical naming mode, type a name between 1 to 200 characters. Spaces are not allowed. In logical mode, the maximum number of characters is 200 and spaces are allowed.

### Contents Page

The Contents page enables you to select the data objects that you want to refer to in the collection. Use the following steps:

1. Select and expand the project node in the left panel.

The wizard displays a list of objects you can add to the collection.

2. Select objects from Available section in the left panel.

Use the **Ctrl** key to select multiple objects. You can select objects at the object level or the module level. For example, under the Files node, you can add a specific file or add all the files in a given flat file module.

If you add a module or another collection, Warehouse Builder creates references to the module or collection and also creates references to objects contained in the module or collection.

3. Click the right arrow.

The wizard displays the list of objects under the Selected section on the right panel. You can remove objects from the list by selecting objects and clicking the left arrow.

### Summary Page

The Summary page displays the objects selected for the collection. Review the objects and click **Back** to make changes to your selections. Click **Finish** to complete the collection definition. Warehouse Builder creates the collection and adds it to the Project Explorer.

## Editing Collection Definitions

Use the Edit Collection dialog box to edit a collection. You can perform the following actions when you edit a collection definition:

- Rename the collection
- Add data objects to the collection
- Remove data objects that the collection references.

From the Project Explorer, right-click the collection and select **Open Editor**.

Warehouse Builder displays the Edit Collection dialog box that contains the following two tabs: [Name Tab](#) and [Contents Tab](#). Use these tabs to edit the collection definition.

### Name Tab

Use the Name tab to rename a collection or modify its description. To rename a collection, select the name in the Name field and enter the new name. The name must be unique within the project. In physical naming mode, type a name between 1 to 200 characters. Spaces are not allowed. In logical mode, the maximum number of characters is 200 and spaces are allowed.

You can also modify the description of the collection using the Description field.

### Contents Tab

Use the Contents tab to modify the contents of the collection. Use the following steps:

1. Select and expand the project node in the left panel.

The wizard displays a list of objects you can add to the collection.

2. Select and expand the collection node in the right panel.

The list of objects that are referenced in the collection are displayed.

3. Add new objects to the collection by selecting the objects in the Available section and clicking the right arrow.

4. Remove objects referenced in the collection by selecting the objects in the Selected section and clicking the left arrow.

## Alternative Interfaces

In addition to the Design Center, Warehouse Builder provides other interfaces to create and implement your data integration solution. One such interface is OMB Plus.

OMB Plus, an extension of the Tcl programming language, is the scripting language provided by Warehouse Builder. It is a flexible, high-level command line metadata access tool for Warehouse Builder. With OMB Plus, you can write the syntactic constructs such as variable support, conditional and looping control structures, error handling, and standard library procedures.

Use OMB Plus to create, modify, delete, and retrieve object metadata in Warehouse Builder repository. You can use this scripting interface to:

- Perform complex actions directly in Warehouse Builder, without launching the client user interface.
- Define sets of routine operations to be executed in Warehouse Builder.
- Perform batch operations in Warehouse Builder.
- Automate a series of conditional operations in Warehouse Builder.

### To access OMB Plus:

Select **Start, Programs, Oracle - OWB\_HOME, Warehouse Builder, then OMB Plus.**

*or*

From the Design Center, select **Window, then OMB\*Plus.**

The Design Center displays the OMB\*Plus panel.



---

---

## Identifying Data Sources and Importing Metadata

In Oracle Warehouse Builder you can access data from a variety of sources. You can interpret and extract metadata from custom as well as packaged applications and databases. As a precursor to extracting any data set, you first import its metadata.

This chapter includes the following topics:

- [About Source Data and Metadata](#)
- [Supported Sources and Targets](#)
- [General Steps for Importing Metadata from Sources](#)
- [Using the Import Metadata Wizard](#)
- [Reimporting Definitions from an Oracle Database](#)
- [Integrating with E-Business Suite](#)
- [Integrating with PeopleSoft](#)
- [Integrating with Siebel](#)
- [Integrating with SAP R/3](#)
- [Integrating with Business Intelligence Tools](#)

### About Source Data and Metadata

The source systems for a data warehouse are typically transaction processing applications. For example, a sales analysis data warehouse typically extracts data from an order entry system that records current order activities.

Designing the extraction process can be problematic. If the source system is complex and poorly documented, then determining which data to extract can be difficult. Moreover, the source system typically cannot be modified, nor can its performance or availability be adjusted. You can overcome these problems by first importing the metadata.

Metadata is the data that describes the contents of a given object in a data set. For example, the metadata for a table would indicate the data type for each column. After you import the metadata into Warehouse Builder, you can annotate the metadata and design an extraction strategy independently from the transaction processing application.

Before you import source metadata into Warehouse Builder, first create a module that will contain these metadata definitions. The type of module you create depends on the

source from which you are importing metadata. For example, to import metadata definitions from an Oracle database, create an Oracle module. To import metadata definitions from flat files, create a flat file module.

## Supported Sources and Targets

Table 4–1 lists the data storage systems and applications that Warehouse Builder 11.1 can access. The table lists the supported sources and targets for each **Location** node as displayed in the Connection Explorer.

**Table 4–1 Sources and Targets Supported in Warehouse Builder 11.1**

Location Node in the Connection Explorer	Supported Sources	Supported Targets
Databases/Oracle	Oracle DB 8.1, 9.0, 9.2, 10.1, 10.2, 11.1	Oracle DB 9.2, 10.1, 10.2, 11.1
Databases/Non-Oracle	<p>Any database accessible through <a href="#">Oracle Heterogeneous Services</a>, including but not limited to DB2, DRDA, Informix, SQL Server, Sybase, and Teradata.</p> <p>Any data store accessible through the ODBC Data Source Administrator, including but not limited to Excel and MS Access.</p> <p>See "<a href="#">Loading Data Stored in a Microsoft Excel File</a>" on page 12-1 and "<a href="#">Connecting to SQL Server and Importing Metadata</a>" on page 13-1.</p> <p>Oracle E-Business Suite, see "<a href="#">Integrating with E-Business Suite</a>" on page 4-11</p> <p>PeopleSoft 8, 9, see "<a href="#">Integrating with PeopleSoft</a>" on page 4-15</p> <p>Siebel, see "<a href="#">Integrating with Siebel</a>" on page 4-18</p>	<p>Any database accessible through <a href="#">Oracle Heterogeneous Services</a>, including but not limited to DB2, DRDA, Informix, SQL Server, Sybase, and Teradata.</p> <p>Any data store accessible through the ODBC Data Source Administrator, including but not limited to Excel and MS Access.</p> <p>To load data into spreadsheets or third-party databases, first deploy to a comma-delimited or XML format flat file.</p>
Files	<p>Delimited and fixed-length flat files.</p> <p>See "<a href="#">Importing Definitions from Flat Files</a>" on page 4-7.</p>	<p>Comma-delimited and XML format flat files.</p> <p>See "<a href="#">Defining Flat Files and External Tables</a>" in the <i>Warehouse Builder Online Help</i>.</p>
Applications	<p>SAP R/3: 3.x, 4.0x, 4.6x, 4.7, 5.0; mySAP ERP 2004; mySAP ERP 2005 (with SAP NetWeaver 2004, SAP BASIS 700 Components)</p> <p>See "<a href="#">Integrating with SAP R/3</a>" on page 4-20.</p>	None
Process Flows and Schedules/Oracle Workflow	None	Oracle Workflow 2.6.2, 2.6.3, 2.6.4, 11i
Process Flows and Schedules/Concurrent Manager	None	<p>In general, you can deploy a schedule in any Oracle database location, version 10g or later.</p> <p>To deploy a schedule in Concurrent Manager, version 11i or 12i is required. However, for both versions, you must select 11i as the version when you create a location in Warehouse Builder.</p>

**Table 4–1 (Cont.) Sources and Targets Supported in Warehouse Builder 11.1**

Location Node in the Connection Explorer	Supported Sources	Supported Targets
Business Intelligence/Discoverer	None	Discoverer 10.1
Databases/Transportable Module Source	See "Moving Large Volumes of Data" in the <i>Warehouse Builder Online Help</i> .	N/A
Databases/Transportable Module Target	N/A	See "Moving Large Volumes of Data" in the <i>Warehouse Builder Online Help</i> .

### Oracle Heterogeneous Services

Warehouse Builder communicates with non-Oracle systems using Oracle Database Heterogeneous Services and a complementary agent. Heterogeneous Services make a non-Oracle system appear as a remote Oracle Database server. The agent can be an Oracle Transparent Gateway or the generic connectivity agent included with Oracle Database.

- A transparent gateway agent is a system-specific source. For example, for a Sybase data source, the agent is a Sybase-specific transparent gateway. You must install and configure this agent to support the communication between the two systems.
- Generic connectivity is intended for low-end data integration solutions and the transfer of data is subject to the rules of specific ODBC or OLE DB drivers installed on the client computer. In this case, you do not need to purchase a separate transparent gateway; you can use the generic connectivity agent included with the Oracle Database server. You must still create and customize an initialization file for your generic connectivity agent.

## General Steps for Importing Metadata from Sources

Whether you want to import metadata from a table, file, or application, the general process is the same and you always import metadata through a module.

1. Review the list of supported sources and targets in [Table 4–1](#) to determine if the source from which you want to extract data is supported in Warehouse Builder.
2. If you have not already done so, create a location and module for the source metadata as described in "[Creating Modules](#)" on page 4-4.
3. Right-click the module and select **Import**.
4. Follow the prompts in the Metadata Import Wizard.

The wizard prompts you for information based on the type of source you selected. For more information, see "[Using the Import Metadata Wizard](#)" on page 4-5.

5. (Optional) For Oracle data objects, view the data stored in the data object using the Data Viewer. Right-click the object and select **Data**.

### Subsequent Steps

After successfully importing the metadata, you can design ETL logic to extract the data from the source, transform the data, and load it into a target schema.

Over a period of time, the source metadata may change. If this occurs, you can use Warehouse Builder to identify the ETL logic that would be impacted and potentially made invalid due to a change in metadata.

**See Also:**

- "Managing Metadata Dependencies" in the *Warehouse Builder Online Help*
- "Updating the Target Schema" on page 21-1

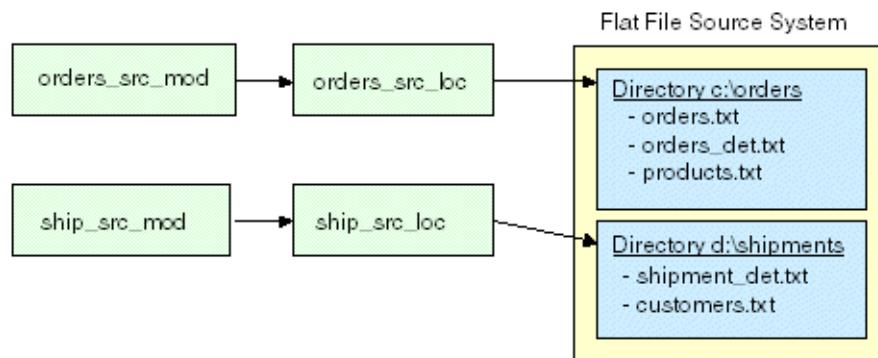
To introduce the changed metadata into Warehouse Builder, right-click the desired module and select **Import**. As described in "Reimporting Definitions from an Oracle Database" on page 4-9, Warehouse Builder recognizes when you are reimporting metadata.

## Example: Importing Metadata from Flat Files

Assume that there are numerous flat files stored across two different drives and directories on your source system. In the Connection Explorer, you create two locations that reference the directories in which the source data is stored. Now in the Project Explorer, right-click the **Files** node and select **New** to create a new module. Repeat this for each of the two directories. For each of the two modules, select **Import**. A wizard directs you on how to import one or more files into each module.

Figure 4-1 provides a diagrammatic representation of accessing flat file data stored in different drives or directories on your source system. Each location maps to a particular directory on your source system.

**Figure 4-1 Importing Data From Flat File Sources**



## About Modules

Modules are grouping mechanisms in the Project Explorer that correspond to locations in the Connection Explorer. A single location can correspond to one or more modules. However, a given module can correspond to only a single location at a time.

The association of a module to a location enables you to perform certain actions more easily in Warehouse Builder. For example, you can reimport metadata by reusing an existing module. Furthermore, when you deploy ETL processes in subsequent steps, modules enable you to deploy related objects together such as process flows.

### Creating Modules

**To create a module:**

1. Expand the Project Explorer until you find the node for the appropriate metadata type.



For example, if the source data is stored in an Oracle Database, then expand the Databases node to view the Oracle node. If the source data is in an SAP R/3 system, expand the Applications node to view the SAP node.

2. Right-click the desired node and select **New**.

The Create Module wizard opens. The wizard determines the correct integrator to use to enable access to the data store you selected.

3. On the Name and Description page, provide a name and an optional description for the module.

4. Click **Next**.

The Connection Information page is displayed.

5. Provide details about the location that is associated with this module.

The contents of the Connection Information page depend on the type of module you create. For more information about providing information on this page, click **Help**.

6. Click **Next** to display the Summary page.

Review the information you provided and click **Back** to modify entered values.

7. Click **Finish**.

During the course of using Warehouse Builder, you may need to associate a module with a new location. For example, assuming your production environment utilizes different locations than your development environment, you need to reassociate the modules.

#### To change the location associated with a module:

1. In the Project Explorer, select the module.

2. Click the Configure icon.

The Configuration Properties dialog box is displayed.

3. In the Identification folder, select a new value for the Locations property.

## Using the Import Metadata Wizard

Importing is also known as reverse engineering. It saves design time by bringing metadata definitions of existing database objects into Warehouse Builder. You use the Import Metadata Wizard to import metadata definitions into modules.

The Import Metadata Wizard supports importing of tables, views, materialized views, dimensions, cubes, external tables, sequences, user-defined types, and PL/SQL transformations directly or through object lookups using synonyms.

Importing a table includes importing its columns, primary keys, unique keys, and foreign keys, which enable import of secondary tables. When you import an external table, Warehouse Builder also imports the associated location and directory information for the associated flat file.

You can import metadata definitions either from the Oracle Database catalog or Designer/2000 (Oracle Designer).

This section contains the following topics:

- [Importing Definitions from a Database](#)
- [Importing Definitions from Flat Files](#)

## Importing Definitions from a Database

Use the Import Metadata Wizard to import metadata from a database into a module. You can import metadata from an Oracle Database, a non-Oracle Database, or a Designer repository.

### To import definitions from an Oracle Data Dictionary:

1. Right-click a data source module name and select **Import**.

The Welcome page of the Import Metadata Wizard is displayed. This page lists the steps to import object metadata. Click **Next** to proceed with the import.

If you did not specify the location details for the Oracle module, Warehouse Builder displays a warning dialog box. This dialog box informs you that you must first specify the location details. Click **OK**. The Edit Oracle Database Location dialog box for the Oracle module is displayed. Use this dialog box to specify the location information. Clicking **OK** on this dialog box displays the Welcome page of Import Metadata Wizard.

2. Complete the following pages:

- [Filter Information Page](#)
- [Object Selection Page](#)
- [Summary and Import Page](#)
- [Import Results Page](#)

### Filter Information Page

Use the Filter Information page to limit the search of the data dictionary. Use one of the following methods to limit the search:

**Selecting the Object Types** The Object Type section displays the types of database objects that you can import. This include tables, dimensions, external tables, sequences, materialized views, cubes, views, PL/SQL transformations, and user-defined types. Select the types of objects you want to import. For example, to import three tables and one view, select **Tables** and **Views**.

**Search Based on the Object Name** Use the **Only select objects that match the pattern** option to type a search pattern. Warehouse Builder searches for objects whose names match the pattern specified. Use % as a wild card match for multiple characters and \_ as a wild card match for a single character. For example, you can type a warehouse project name followed by a % to import objects that begin with that project name.

Click **Next** and Warehouse Builder retrieves names that meet the filter conditions from the data dictionary and displays the Object Selection page.

### Object Selection Page

Select items to import from the Available list and click the right arrow to move them to the Selected list.

To search for specific items by name, click the Find Objects icon that displays as a flashlight.

To move all items to the Selected Objects list, click **Move All**.

**Importing Dependent Objects** The Import Metadata wizard enables you to import the dependent objects of the object being imported. If you are reimporting definitions, previously imported objects appear in bold.

Select one of the following options to specify if dependent objects should be included in the import:

- **None:** Moves only the selected object to the Selected list. No dependencies are imported when you select this option.
- **One Level:** Moves the selected object and the objects it references to the Selected list. This is the default selection.
- **All Levels:** Moves the selected object and all its references, direct or indirect, to the Selected list.

Click **Next** and the Summary and Import page is displayed.

**Importing Dimensions** When you import a dimension that uses a relational implementation, the implementation table that stores the dimension data is not imported. You must explicitly import this table by moving the table from the Available list to the Selected list on the Object Selection page. Also, after the import, you must bind the dimension to its implementation table. For more information on how to perform binding, see "[Binding](#)" on page 6-12.

### Summary and Import Page

This page summarizes your selections in a spreadsheet listing the name, type of object, and whether the object will be reimported or created. Verify the contents of this page and add descriptions, if required, for each of the objects.

If the objects you selected on the Object Selection page already exist in the module into which you are attempting to import them, you can specify additional properties related to the reimport. Click **Advanced Import Options** to specify options related to reimporting objects. The Advanced Import Options dialog box is displayed. For more information on the contents of this dialog box, see "[Advanced Import Options](#)" on page 4-9.

Click **Finish** to import the selected objects. The Importing Progress dialog box shows the progress of the import activity. After the import completes, the Import Results page is displayed.

### Import Results Page

This page summarizes the import and lists the objects and details about whether the object was created or synchronized.

Click **OK** to accept the changes. To save an MDL file associated with this import, click **Save**. Click **Undo** to cancel the import. Warehouse Builder stores the definitions in the database module from which you performed the import.

## Importing Definitions from Flat Files

If you have existing flat files to use as sources, you can import and then sample the metadata from these flat files. Use the Import Metadata Wizard to import metadata from flat files. This metadata must be imported into a file module.

### To import flat file definitions:

1. Establish network connectivity to the files you wish to import.

If you are accessing the data files directly, and if the client and the data files reside on different types of operating systems, contact your system administrator to establish the required connectivity through NFS or other network protocol.

If the client and data files reside on Windows operating systems, store the data files on any drive the client computer can access.

2. Create a flat file module that will contain the imported flat file definitions.

Create a module for each folder in your file system from which you want to import files. See ["Example: Importing Metadata from Flat Files"](#) on page 4-4.

When you create a flat file module, the location corresponding to this module is a folder in the file system from which metadata is being imported. Use the Connection Information Page of the Create Module Wizard to specify the folder that contains the source metadata.

Note that a flat file location does not include subfolders of the specified folder.

3. Right-click the flat file module and select **Import**.

The Import Metadata Wizard is displayed.

4. On the Filter Information page, filter file names by selecting one of the following options:

**All Data Files:** This option returns all the data files available for the directory you specified for the flat file module.

**Data files matching this pattern:** Use this option to select only data files that match the pattern you type. For example, if you select this option and enter (\*.dat), only files with .dat file extensions will be displayed on the next wizard page. If you type % as part of a filter string, it is interpreted as a wild card match for multiple characters. If you type '\_' as part of a filter string, it is interpreted as a wild card match for a single character.

5. On the Object Selection page, move the names of the files to be imported from Available Objects on the left to the Selected Objects section on the right.

Because inbound synchronization for flat files is not permitted, the available objects will never appear in bold like other objects when they are reimported. When you reimport flat files, you always need to sample the flat file objects again.

6. On the Summary and Import page, ensure that metadata is available for the selected flat files is available in the workspace. You cannot complete the import if the metadata is not present.

If the Status field contains a red x, metadata is not available in the workspace. For all such files, either select a file with a matching format in the workspace or sample the file.

Use the **Same As** field to select a file with a matching format.

To sample a file, select the file and click **Sample**. The Flat File Sample Wizard is launched. The Flat File Sample Wizard enables you to view a sample of the flat file and define record organization and file properties. You can sample and define common flat file formats such as string and ascii.

For files with complex record structures, the Flat File Sample Wizard may not be suitable for sampling the data. In such cases, see ["Adding Existing Binary Files to the Workspace"](#) in the *Warehouse Builder Online Help*.

7. Once you provide metadata information for all files you want to import, click **Finish**.

The wizard creates definitions for files, stores the definitions in the flat file module, and inserts the file names under the flat file module in the Project Explorer.

## Reimporting Definitions from an Oracle Database

Reimporting your source database definitions enables you to import changes made to your source metadata since your previous import. You do not have to remove the original definitions from the workspace. Warehouse Builder provides you with options that also enable you to preserve any changes you may have made to the definitions since the previous import. This includes any new objects, foreign keys, relationships, and descriptions you may have created in Warehouse Builder.

### To reimport definitions:

1. Right-click a data source module name and select **Import**.

The Welcome page for the Import Metadata Wizard is displayed.

2. Click **Next**.

The Filter Information page is displayed.

3. Complete the [Filter Information Page](#) and [Object Selection Page](#), selecting the same settings used in the original import to ensure that the same objects are reimported.

4. The Summary and Import page displays. For objects that already exist in the workspace or ones that you are reimporting, the Reimport action is displayed in the Action column.

If the source contains new objects related to the object you are reimporting, the wizard requires that you import the new objects at the same time. For these objects, the Create action displays in the Action column.

5. Click [Advanced Import Options](#) and make selections. (Optional)

6. Click **Finish**.

Warehouse Builder reconciles and creates objects. When this is complete, the Import Results dialog box displays.

The report lists the actions performed by Warehouse Builder for each object.

Click **Save** to save the report. You should use a naming convention that is specific to the reimport.

7. Click **OK** to proceed.

Click **Undo** to undo all changes to your workspace.

## Advanced Import Options

The Advanced Import Options dialog box displays the options that you can configure while importing objects. This dialog box enables you to preserve any edits and additions made to the object definitions in the Warehouse Builder workspace.

By default, all options on this dialog box are checked. Clear boxes to have these objects replaced and not preserved.

For example, after importing tables or views for the first time, you manually add descriptions to the table or view definitions. If you want to make sure that these descriptions are not overwritten while reimporting the table or view definitions, you

must select the Preserve Existing Definitions option. This ensures that your descriptions are not overwritten.

The contents of this dialog box depend on the type of objects being imported. For more information about the advanced import options for each type of objects, refer to the following sections:

- [Advanced Import Options for Views and External Tables](#)
- [Advanced Import Options for Tables](#)
- [Advanced Import Options for Object Types](#)
- [Advanced Import Options for SQL Collections](#)

### **Advanced Import Options for Views and External Tables**

Select these options for reconciling views or external tables:

- **Import descriptions:** The descriptions of the view or external table are imported. Existing descriptions are not preserved.
- **Preserve repository added columns:** The columns you added to the object in the workspace are preserved.

### **Advanced Import Options for Tables**

Select these options for reconciling tables:

- **Preserve repository added columns:** Select this option to retain any columns added to the table in the workspace.
- **Preserve repository added constraints:** The constraints you added to the table in Warehouse Builder are preserved.
- **Import indexes:** Select this option to specify additional details about how indexes should be imported. Importing indexes consist of the following options:
  - **Preserve repository added indexes:** Select this option to retain any indexes added to the workspace table.
  - **Import physical properties of indexes:** Select this option to indicate how indexes should be imported. Select the **Preserve repository added physical properties of indexes** option below this option to specify that any physical properties added to the indexes should be preserved.
  - **Import index partitioning:** Select this option to indicate how index partitions should be imported. Select the **Preserve repository added index partitioning** option to specify that any index partitions added to the workspace table must be preserved.
- **Import Partitioning:** Select this option to specify additional details about how partitions should be imported. Importing partitions contains the following options:
  - **Preserve repository added partitioning:** Select this option to retain all partitions added to the workspace table.
  - **Import physical properties of partitioning:** Use this option to indicate how the physical properties of partitions should be imported. Select **Preserve repository added physical properties of partitioning** to indicate that all physical properties of the partitions in the workspace table should be retained.
- **Import physical properties:** Select this option to indicate how the physical properties of the table should be imported. Select the **Preserve repository added**

**physical properties** option to specify that all physical properties added to the workspace table must be preserved.

- **Import descriptions:** Select this option to import the descriptions of the table.

### Advanced Import Options for Object Types

Select these options for reconciling object types:

- **Import descriptions:** Select this option to import the descriptions of the object type.
- **Preserve repository added attributes:** Select this option to retain the attributes added to the object type in the workspace.

### Advanced Import Options for SQL Collections

SQL collection includes nested tables or Varrays.

**Import descriptions:** Select this option to import the descriptions of the queue table, advanced queue, streams queue, nested table, or Varray.

## Updating Oracle Database Source Definitions

The Edit Module dialog box enables you to edit the name, metadata location, and the data location of a source module.

### To update the database definitions:

1. Double-click any Oracle module.

The Edit Module dialog box displays. You can edit the metadata location as well as the data location of the database.

2. To edit the metadata location, click the Metadata Locations tab and specify the following:
  - **Source Type:** The source type identifies the location of the data and the metadata. It can be either Oracle Data Dictionary or Oracle Designer Repository. Select Oracle Data Dictionary if the metadata is stored in the default workspace of the Oracle Database. Select Oracle Designer Repository if the metadata is stored in an Oracle Designer repository.
  - **Location:** Identifies the location of the module. You can select a location from the list.
3. To edit the data location, click the Data Locations tab. You can either select from the existing locations or create a new location. To create a new location, click **New**. The Edit Oracle Database Location dialog box displays. Specify the details of the data location here.

## Integrating with E-Business Suite

Warehouse Builder enables you to import metadata stored in an E-Business Suite database using the Import Metadata Wizard.

### Before You Begin

Contact the database administrator for the E-Business Suite database and request a user name and password for accessing the APPS schema. The DBA may have previously created a user by running the script `owbebs.sql` as described in the *Oracle Warehouse Builder Installation and Administration Guide*. If not, you will need to provide

the DBA with a list of the tables, views, sequences, and keys from which you want to extract data.

Depending on the preference of the DBA, there may be a single user who extracts both, the metadata as well as the data. Or, there may be two separate users to access the metadata and data respectively.

## Importing E-Business Suite Metadata Definitions

After creating the E-Business Suite source module, you can import metadata definitions from E-Business Suite objects using the Import Metadata Wizard. This wizard enables you to filter the E-Business Suite objects you want to import and verify those objects. You can import metadata for tables, views, and sequences.

### To import E-Business Suite metadata:

1. From the Project Explorer, expand the Applications node.
2. If you have not already done so, create an E-Business Suite module and that will contain the imported metadata.

To create an E-Business Suite module, right-click `ORACLE_EBUSINESS_SUITE` under the Applications node and select **New**. The Create Module Wizard is displayed. Follow the prompts in the wizard. Click **Help** on a wizard page for more information about that page.

Ensure that the location associated with the E-Business Suite module contains information needed to connect to the E-Business Suite source. If you created a location earlier, associate that location with the module being created by selecting that location on the Connection Information page. Or create a new location by clicking Edit on the Connection Information page of the Create Module Wizard. For more information about the details to be entered on this page, click **Help**.

3. Right-click the E-Business Suite source module into which you want to import metadata and select **Import**.

Warehouse Builder displays the Welcome page for the Import Metadata Wizard.

4. Click **Next**.
5. Complete the following tasks:

[Filtering E-Business Suite Metadata](#)

[Selecting the Objects](#)

[Reviewing Import Summary](#)

### Filtering E-Business Suite Metadata

The Import Metadata Wizard includes a Filter Information page that enables you to select the metadata. Warehouse Builder provides two filtering methods:

- Business Domain

This filter enables you to browse E-Business Suite business domains to locate the metadata you want to import. You can view a list of objects contained in the business domain and the names of the objects in the E-Business Suite application. For more information, see "[Filtering E-Business Suite Metadata by Business Domain](#)" on page 4-13.

- Text String Matching



This filter enables you to search tables, views, and sequences by typing text string information in the field provided in the Filter Information page. This is a more specific search method if you are familiar with the contents of your E-Business Suite application database. For more information, see "[Filtering E-Business Suite Metadata by Text String](#)" on page 4-13.

Select a filtering method and click **Next** to proceed with the importing of metadata.

### Filtering E-Business Suite Metadata by Business Domain

1. Select **Business Domain** and click **Browse** to open the Business Component Hierarchy dialog box.
2. The Business Component Hierarchy dialog box lists the available E-Business Suite business domains.

---

**Note:** It may take two to ten minutes to list the business domains depending on the network location of the E-Business Suite application server, the type of LAN used, or the size of the E-Business Suite application database.

---

Use the Business Component Hierarchy dialog box to select the E-Business Suite business domains that contain the metadata objects you want to import.

3. Select a business domain and click **Show Entities**.

The Folder dialog box displays a list of objects available in the selected business domain.

4. Review this dialog box to ensure that you are selecting the required objects and click **OK** to go back to the Business Component Hierarchy dialog box.

Some business domains can contain more than 1000 objects. Importing such a large amount of metadata can take from one to three hours or more, depending on the network connection speed and the processing power of the source and target systems.

5. Click **OK**.

The wizard displays the Filter Information page with the E-Business Suite business domain displayed in the Business Domain field.

### Filtering E-Business Suite Metadata by Text String

1. Select **Text String, where object**.
2. Select the objects you wish to import. You can select Tables, Views, and Sequences.

If you wish to select specific objects, type the object name in the text field. Create a filter for object selection by using the wildcard characters (%) for zero or more matching characters, and (\_) for a single matching character.

For example, if you want to search the business domain for tables whose names contain the word CURRENCY, then type %CURRENCY%. If you want to refine the search to include only tables named CURRENCY and followed by a single digit, then type %CURRENCY\_.

## Selecting the Objects

The Object Selection page contains a description of the objects and enables you to select the objects you want to import into the E-Business Suite module. To select the objects:

1. Move the objects from the available list to the selected list.

The Import Wizard also enables you to choose whether you want to import tables with foreign key relationships for each object that you choose to import. You can select one of the following:

**None:** Import only the objects in the Selected list.

**One Level:** Import the objects in the Selected list and any tables linked to it directly through a foreign key relationship.

**All Levels:** Import the objects in the Selected list and all tables linked to it through foreign key relationships.

The foreign key level you select is the same for all tables selected for importing.

---

---

**Note:** Selecting All Levels increases the time it takes to import the metadata because you are directing the wizard to import tables that are related to each other through foreign key constraints. Select this option only if it is necessary.

---

---

2. Click **Next**.

If you select One Level or All Levels, the Confirm Import Selection dialog box is displayed.

Review this dialog box to ensure that you are selecting the required tables.

3. Click **OK**.

The selected objects appear in the right pane of the Object Selection page.

4. Click **Next**.

The wizard displays the Summary and Import page.

## Reviewing Import Summary

The wizard imports definitions for the selected objects from the E-Business Suite Application Server, stores them in the E-Business Suite source module, and then displays the Summary and Import page.

You can edit the descriptions for each object by selecting the description field and typing a new description.

Review the information on the Summary and Import page and click **Finish**.

The E-Business Suite integrator reads the table definitions from the E-Business Suite application server and creates the metadata objects in the workspace.

The time it takes to import the E-Business Suite metadata to the workspace depends on the size and number of tables and the connection between the E-Business Suite application server and the workspace. Importing 500 or more objects could take one to three hours or more, especially if you are connecting servers in separate Local Area Networks (LANs).

When the Import completes, the Import Results dialog box displays. Click **OK** to finish importing.

## Integrating with PeopleSoft

PeopleSoft applications provide ERP solutions. A PeopleSoft application consists of numerous modules, each pertaining to a specific area in an enterprise, such as Human Resource Management System (HRMS), Financials, and Material Management. You can use the Import Metadata Wizard to import metadata from Peoplesoft applications into Warehouse Builder.

### Importing PeopleSoft Metadata Definitions

After creating the PeopleSoft source module, you can import metadata definitions from PeopleSoft objects using the Import Metadata Wizard. This wizard enables you to filter the PeopleSoft objects you want to import and verify those objects. You can import metadata for tables, views, and sequences.

#### To import PeopleSoft metadata:

1. From the Project Explorer, expand the **Applications** node.
2. If you have not already done so, create a Peoplesoft module that will contain the imported metadata.

Right-click PEOPLESOFT8\_9 and select **New**. The Create Module wizard is displayed. Click **Help** on a wizard page for more information about the page.

Ensure that the location associated with the PeopleSoft module contains information needed to connect to the PeopleSoft source. If you created a location earlier, associate that location with the module being created by selecting the location on the Connection Information page. Or create a new location by clicking Edit on the Connection Information page of the Create Module Wizard. For more information about the details to be entered on this page, click **Help**.

3. Right-click the PeopleSoft source module into which you want to import metadata and select **Import**.

Warehouse Builder displays the Welcome page for the Import Metadata Wizard.

4. Click **Next**.
5. Complete the following tasks:
  - [Filtering PeopleSoft Metadata](#)
  - [Selecting the Objects](#)
  - [Reviewing Import Summary](#)

#### Filtering PeopleSoft Metadata

The Import Metadata Wizard includes a Filter Information page that enables you to select the metadata. Warehouse Builder provides two filtering methods:

- Business Domain

This filter enables you to browse PeopleSoft business domains to locate the metadata you want to import. You can view a list of objects contained in the business domain. For more information, see "[Filtering PeopleSoft Metadata by Business Domain](#)" on page 4-16.

- Text String Matching

This filter enables you to search tables, views, and sequences by typing text string information in the field provided in the Filter Information page. This is a more specific search method if you are familiar with the contents of your PeopleSoft

application database. For more information, see "[Filtering PeopleSoft Metadata by Text String](#)" on page 4-16.

Select a filtering method and click **Next** to proceed with the importing of metadata.

### Filtering PeopleSoft Metadata by Business Domain

1. Select **Business Domain** and click **Browse** to open the Business Component Hierarchy dialog box.

The Import Metadata Wizard displays Loading Progress dialog box while it is retrieving the business domains.

2. The Business Component Hierarchy dialog box lists the available PeopleSoft business domains.

---

**Note:** It may take two to ten minutes to list the business domains depending on the network location of the PeopleSoft application server, the type of LAN used, or the size of the PeopleSoft application database.

---

Use the Business Component Hierarchy dialog box to select the PeopleSoft business domains that contain the metadata objects you want to import.

3. Select a folder and click **Show Entities**.

The Import Wizard displays a list of objects in the selected business domain in the Folder dialog box.

4. Review this dialog box to ensure that you are selecting the required objects.

Some business domains can contain more than 1000 objects. Importing such a large amount of metadata can take from one to three hours or more, depending on the network connection speed and the processing power of the source and target systems.

5. Click **OK**.

The wizard displays the Filter Information page with the PeopleSoft business domain displayed in the Business Domain field.

### Filtering PeopleSoft Metadata by Text String

1. Select **Text String, where object**.
2. In the Object Type section, select the types of objects you wish to import. You can select Tables, Views, and Sequences.

If you wish to select specific objects, type the object name in the text field. Create a filter for object selection by using the wildcard characters (%) for zero or more matching characters, and (\_) for a single matching character.

For example, if you want to search the business domain for tables whose names contain the word CURRENCY, then type %CURRENCY%. If you want to refine the search to include only tables named CURRENCY and followed by a single digit, then type %CURRENCY\_.

### Selecting the Objects

The Object Selection page contains a description of the objects and enables you to select the objects you want to import into the PeopleSoft module. To select the objects:

1. Move the objects from the Available list to the Selected list.

The Import Wizard also enables you to choose whether you want to import tables with foreign key relationships for each object that you choose to import. You can select one of the following:

**None:** Import only the objects in the Selected list.

**One Level:** Import the objects in the Selected list and any tables linked to it directly through a foreign key relationship.

**All Levels:** Import the objects in the Selected list and all tables linked to it through foreign key relationships.

The foreign key level you select is the same for all tables selected for importing.

---

---

**Note:** Selecting **All Levels** increases the time it takes to import the metadata because you are directing the wizard to import tables that are related to each other through foreign key constraints. Select this option only if it is necessary.

---

---

2. Click **Next**.

If you select One Level or All Levels, the Confirm Import Selection dialog box is displayed.

Review this dialog box to ensure that you are selecting an appropriate number of tables.

3. Click **OK**.

The selected objects appear in the Selected pane of the Object Selection page.

4. Click **Next**.

The wizard displays the Summary and Import page.

### Reviewing Import Summary

The wizard imports definitions for the selected tables from the PeopleSoft Application Server, stores them in the PeopleSoft source module, and then displays the Summary and Import page.

You can edit the descriptions for each object by selecting the description field and typing a new description.

Review the information on the Summary and Import page and click **Finish**.

The PeopleSoft Connector reads the table definitions from the PeopleSoft application server and creates the metadata objects in the workspace.

The time taken to import PeopleSoft metadata to the workspace depends on the size and number of tables and the connection between the PeopleSoft application server and the workspace. Importing 500 or more objects could take one to three hours or more, especially if you are connecting to servers in separate Local Area Networks (LANs).

When the import completes, the Import Results dialog box displays. Click **OK** to finish importing metadata.

## Integrating with Siebel

Siebel applications provide Customer Relationship Management (CRM) solutions. Warehouse Builder provides a Connector for Siebel systems that enables you to extract both metadata and data from your Siebel systems.

The Siebel Connector enables you to connect to any Siebel application, read its metadata, import the metadata into Warehouse Builder, and extract data from the system.

### Importing Siebel Metadata Definitions

Before you import metadata definitions from Siebel, you must create a Siebel module. You can then import metadata definitions from Siebel using the Import Metadata Wizard. This wizard enables you to filter the Siebel objects you want to import and verify those objects. You can import metadata for tables, views, and sequences.

#### To import metadata definitions from Siebel:

1. Create a Siebel source module, as described in "[Creating a Siebel Source Module](#)" on page 4-18.
2. Import metadata from Siebel, as described in "[Importing Siebel Metadata](#)" on page 4-18.

#### Creating a Siebel Source Module

1. From the Project Explorer, expand the Applications node.
2. Right-click Siebel and select **New**.  
The Create Module wizard is displayed.
3. Click **Next** to display the Name and Description page.
4. Specify a name and an optional description for the Siebel source module and click **Next**.  
The Connection Information page is displayed.
5. Specify the connection information for the Siebel source module and click **Next**.  
Ensure that the location associated with the Siebel module contains information needed to connect to the Siebel source. If you created a location earlier, associate that location with the module being created by selecting the location on the Connection Information page. Or create a new location by clicking **Edit** on the Connection Information page of the Create Module Wizard.  
For more information about the details to be entered on this page, click **Help**.
6. On the Summary page, review the options entered on the previous wizard pages. Click **Back** to modify any selections. Click **Finish** to create the Siebel source module.

#### Importing Siebel Metadata

1. Right-click the Siebel source module into which you want to import metadata and select **Import**.  
Warehouse Builder displays the Welcome page for the Import Metadata Wizard.
2. Click **Next**.  
The Filter Information page is displayed.

3. Select the objects to be imported and click **Next**.

Warehouse Builder provides two filtering methods:

- **Business Domain**

This filter enables you to browse PeopleSoft business domains to locate the metadata you want to import. You can view a list of objects contained in the business domain. For more information, see "[Filtering Siebel Metadata by Business Domain](#)" on page 4-19.

- **Text String Matching**

This filter enables you to search tables, views, and sequences by typing text string information in the field provided in the Filter Information page. This is a more specific search method if you are familiar with the contents of your PeopleSoft application database. For more information, see "[Filtering Siebel Metadata by Text String](#)" on page 4-20.

4. On the Objects Selection page, select the objects to be imported into the Siebel module and click **Next**.

You can choose whether you want to import tables with foreign key relationships for each object that you choose to import using the following options on this page:

**None:** Import only the objects in the Selected list.

**One Level:** Import the objects in the Selected list and any tables linked to it directly through a foreign key relationship.

**All Levels:** Import the objects in the Selected list and all tables linked to it through foreign key relationships.

The foreign key level you select is the same for all tables selected for importing.

---



---

**Note:** Selecting **All Levels** increases the time it takes to import the metadata because you are directing the wizard to import tables that are related to each other through foreign key constraints. Select this option only if it is necessary.

---



---

5. Review the summary information and click **Finish** to complete the import. To modify any selections, click **Back**.

After you import metadata for tables, views, or sequences from Siebel applications, you can use these objects in mappings.

### **Filtering Siebel Metadata by Business Domain**

1. Select **Business Domain** and click **Browse** to open the Business Component Hierarchy dialog box.

The Import Metadata Wizard displays Loading Progress dialog box while it is retrieving the business domains.

2. The Business Component Hierarchy dialog box lists the available PeopleSoft business domains.

---



---

**Note:** It may take two to ten minutes to list the business domains depending on the network location of the PeopleSoft application server, the type of LAN used, or the size of the PeopleSoft application database.

---



---

Use the Business Component Hierarchy dialog box to select the PeopleSoft business domains that contain the metadata objects you want to import.

3. Select a folder and click **Show Entities**.

The Import Wizard displays a list of objects in the selected business domain in the Folder dialog box.

4. Review this dialog box to ensure that you are selecting the required objects.

Some business domains can contain more than 1000 objects. Importing such a large amount of metadata can take from one to three hours or more, depending on the network connection speed and the processing power of the source and target systems.

5. Click **OK**.

The wizard displays the Filter Information page with the PeopleSoft business domain displayed in the Business Domain field.

### **Filtering Siebel Metadata by Text String**

1. Select **Text String, where object**.
2. In the Object Type section, select the objects you wish to import. You can select Tables, Views, and Sequences.

If you wish to select specific objects, type the object name in the text field. Create a filter for object selection by using the wildcard characters (%) for zero or more matching characters, and (\_) for a single matching character.

For example, if you want to search the business domain for tables whose names contain the word CURRENCY, then type %CURRENCY%. If you want to refine the search to include only tables named CURRENCY and followed by a single digit, then type %CURRENCY\_.

## **Integrating with SAP R/3**

The SAP Connector enables you to connect to SAP application source systems and import the SAP source definitions into a project in the workspace.

You can then generate ABAP or PL/SQL code to extract, transform, and load data from SAP systems to your target system.

The SAP Connector enables you to import metadata object definitions from SAP Application data sources into the workspace. This chapter describes how to use SAP objects in a mapping, generate PL/SQL and ABAP code for the mappings, and deploy them to a target. This section also describes how to extract and load SAP data into your target.

This section contains the following topics:

- [About SAP Business Domains](#)
- [SAP Table Types](#)
- [Required Files For SAP Connector](#)
- [Creating SAP Module Definitions](#)
- [Importing SAP Metadata Definitions](#)
- [Updating SAP Source Modules](#)
- [Defining the ETL Process for SAP Objects](#)



- [Loading SAP Data into the Workspace](#)

## About SAP Business Domains

SAP application systems logically group database and metadata objects under different business domains. In SAP, a business domain is an organizational unit in an enterprise that groups product and market areas. For example, the Financial Accounting (FI) business domain represents data describing financial accounting transactions. These transactions might include General Ledger Accounting, Accounts Payable, Accounts Receivable, and Closing and Reporting.

When you import SAP definitions, you can use a graphical navigation tree in the Business Domain Hierarchy dialog box to search the business domain structure in the SAP source application. This navigation tree enables you to select SAP metadata objects from the SAP application server.

## SAP Table Types

The SAP Connector enables you to import metadata for SAP Business Domains or any of their related ABAP Dictionary objects.

With the SAP Connector, you can import definitions and generate deployment code for the following SAP table types:

- **Transparent:** A transparent table is first defined in the ABAP Dictionary and then created in the database. You can also use transparent tables independently of the R/3 System. You can generate either PL/SQL or ABAP code for transparent tables.
- **Cluster:** A cluster table is an ABAP Dictionary table type. It contains information pertaining to any group of database tables and it is not created in the SAP database. Because cluster tables are data dictionary tables and not database tables, you can only generate ABAP code.
- **Pooled:** The data from several tables is stored together as a table pool in the database. Pooled tables exist in the ABAP Dictionary and are not known to the database. You can only generate ABAP code for pooled tables.

## Required Files For SAP Connector

### Required Files for Windows

The SAP Connector requires a dynamic link library file named `librfc32.dll` to use remote function calls on the client computer. This file is available on the SAP Application Installation CD. You need to copy this file to the following directory on your client system:

```
OWB_ORACLE_HOME\bin\admin
```

If you create an SAP source module and import SAP tables but cannot see the columns in the tables, then you have an incompatible `librfc32.dll` file. Check the version or build number of your `.dll` file from your NT Explorer window.

The following version is currently supported:

File Version: 4640,5,123,2956

Build: Wednesday, August 09 23:46:33 2000

File Size: 1,945,138 bytes

Product Version: 46D,123

You can locate this version of the `.dll` file on the Installation CD.

### Required Files for Unix

The SAP Connector requires a dynamic link library file named `librfccm.so` to use remote function calls on the client computer. This file is available on the SAP Application Installation CD. You need to copy this file to the following directory on your client system:

```
OWB_ORACLE_HOME\owb\bin\admin
```

You also need to add `OWB_ORACLE_HOME\owb\bin\admin` to the Unix environment variable path: `LD_LIBRARY_PATH`.

## Creating SAP Module Definitions

Use the Create Module Wizard to create an SAP source module that stores data from an SAP source. You can choose either SAP R/3 version 3.x or SAP R/3 version 4.x system type as your source. After you select the application version, you need to set the connection information between the workspace and the SAP application server. You can set the connection either by selecting from existing SAP locations or by creating a new SAP location.

---

---

**Note:** To create a connection to an SAP source, you must first obtain the connection information to your SAP Application server from your system administrator.

---

---

When you set the connection information, you can choose the following connection types:

- Remote Function Call (RFC)

This is the default connection type. A remote function call locates a function module running in a system different from that of the caller. The remote function can also be called from within the same system (as a remote call), but usually the caller and the called are located in different systems. This method requires specific IP Address information for the SAP application server.
- SAP Remote Function Call (SAPRFC.INI)

SAP can use its own initialization file to track the IP Address information for you. The `SAPRFC.INI` enables remote calls between two SAP Systems (R/3 or R/4), or between an SAP System and a non-SAP System. This method is useful when you know the SAP-specific connection information and want to automate the IP connection information.

---

---

**Note:** To use the `SAPRFC.INI` connection type, the file `SAPRFC.INI` must be installed in the directory:

```
OWB_ORACLE_HOME\owb\bin\admin
```

This file is available in the SAP Application client installation CD. Consult your system administrator for more information.

---

---

The Create Module Wizard creates the module for you based on the metadata contained in the SAP application server.

## Connecting to an SAP Source Application

1. Select one of the following connection types:
  - Remote Function Call (RFC) is the default connection type.
  - SAP Remote Function Call (SAPRFC.INI).

For more information about these connection types, see "[Creating SAP Module Definitions](#)" on page 4-22.

2. Type the connection information in the appropriate fields. The fields displayed on this page depend on the connection type you choose.

---

**Note:** You must load the `librfc32.dll` file before you can set the connection details. For more information, see "[Required Files For SAP Connector](#)" on page 4-21.

---

You must obtain the connection information to your SAP Application server from your system administrator before you can complete this step.

RFC Connection type requires the following connection information:

**Application Server:** Type the alias name or the IP address of the SAP application server.

**System Number:** Type the SAP system number for SAP user interface login. This number is required in the SAP application configuration and is supplied by the SAP system administrator.

**Client:** Type the SAP client number. This number is required in the SAP application configuration and is supplied by the SAP system administrator.

**User Name:** Type the user name for the SAP user interface. This name is required in the SAP application configuration and is supplied by the SAP system administrator.

**Language:** EN for English or DE for German. If you select DE, the description text displays in German and all other text displays in English.

SAPRFC.INI File connection type requires the following connection information:

**RFC Destination:** Type the alias for the SAP connection information.

**Client:** Type the SAP client number.

**User Name:** Type the SAP user name for the SAP user interface.

**Language:** EN for English or DE for German. If you select DE, the description text displays in German and all other text displays in English.

In addition, both the connection types require the following connection information:

**Host Login User Name:** A valid user name on the system that hosts the SAP application server. This user must have access rights to copy the SAP extraction file using FTP.

**FTP Directory:** The directory where the SAP extraction file is stored. For systems where the ftp directory structure is identical to the operating system directory structure, this field can be left blank. For systems where the file system directory structure is mapped to the ftp directory structure, enter the ftp directory path that is mapped to staging file directory in the file system directory structure. For

example, on a computer that runs Windows, the staging file directory "C:\temp" is mapped to "/" in the FTP directory structure, then enter "/" in this field.

**Execution Function Module:** In a SAP instance, if a remote function module other than the SAP delivered function module: RFC\_ABAP\_INSTALL\_AND\_RUN is used to remotely execute ABAP reports through RFC connections, then enter the remote function module name here.

3. Click **Test Connection** to verify that the connection information you provided are correct.
4. Click **OK** to go back to the Connection Information page of the Create Module wizard.

## Importing SAP Metadata Definitions

After creating the SAP source module, you can import metadata definitions from SAP tables using the Import Metadata Wizard. This wizard enables you to filter the SAP objects you want to import, verify those objects, and reimport them. You can import metadata for transparent tables, cluster tables, or pool tables.

Perform the following steps to import SAP metadata:

1. From the Project Explorer, expand the **Applications** node.
2. If you have not already done so, create an SAP module that will contain the imported metadata.

Right-click the SAP node and select **New**. The Create Module Wizard is displayed. Follow the prompts and create an SAP module. Click **Help** on a wizard page for details about the information you must provide on that page.

Ensure that the location associated with the E-Business Suite module contains information needed to connect to the E-Business Suite source. If you created a location earlier, associate that location with the module being created by selecting that location on the Connection Information page. Or create a new location by clicking Edit on the Connection Information page of the Create Module Wizard. For more information about the details to be entered on this page, click **Help**.

3. Right-click the SAP source module into which you want to import metadata and select **Import**.

Warehouse Builder displays the Welcome page for the Import Metadata Wizard.

4. Click **Next**.
5. Complete the following tasks:
  - [Filtering SAP Metadata](#)
  - [Selecting the Objects](#)
  - [Reviewing Import Summary](#)

### Filtering SAP Metadata

The Import Metadata Wizard includes a Filter Information page that enables you to select the metadata. Warehouse Builder provides two filtering methods:

- **Business Domain**

This filter enables you to browse SAP business domains to locate the metadata you want to import. You can view a list of tables contained in the business domain and

the names of the tables in the SAP application. For more information, see "[Filtering SAP Metadata by Business Domain](#)" on page 4-25.

- **Text String Matching**

This filter enables you to search for tables by typing text string information in fields provided in the Filter Information page. This is a more specific search method if you are familiar with the contents of your SAP application database. For more information, see "[Filtering SAP Metadata by Text String](#)" on page 4-25.

Select a filtering method and click **Next** to proceed with the importing of metadata.

### **Filtering SAP Metadata by Business Domain**

1. Select **Business Domain** and click **Browse** to display the SAP R/3 Business Domain Hierarchy dialog box.

The Import Metadata wizard displays the Loading Progress dialog box while it is retrieving the business domains.

2. The Business Domain Hierarchy dialog box lists the available SAP business domains.

---

**Note:** It may take two to ten minutes to list the business domains depending on the network location of the SAP application server, the type of LAN used, or the size of the SAP application database.

---

Use the Business Domain Hierarchy dialog box to select the SAP business domains that contain the metadata objects you want to import.

3. Select a folder and click **Show Tables** to view the tables available in a business domain.

The Import Wizard displays a list of tables in the selected business domain in the Folder dialog box.

4. Review this dialog box to ensure that you are selecting the required tables.

Some business domains can contain more than 1000 tables. Importing such a large amount of metadata can take from one to three hours or more, depending on the network connection speed and the processing power of the source and target systems.

5. Click **OK**.

The wizard displays the Filter Information page with the SAP business domain displayed in the Business Domain field.

### **Filtering SAP Metadata by Text String**

1. Select **Text String, where object** and choose the Name matches entry field or the Description matches entry field to type a string and obtain matching tables from the SAP data source.

The Name matches field is not case sensitive, while the Description matches field is case sensitive.

You must type a text string in the selected Text String entry field. It cannot be empty.

Create a filter for object selection by using the wildcard characters (%) for zero or more matching characters, and (\_) for a single matching character.

For example, if you want to search the business domain for tables whose descriptions contain the word CURRENCY, then select **Description matches** and type %CURRENCY%. You can also search for tables by their names.

---

---

**Note:** Description searches are case sensitive whereas name searches are not case sensitive.

---

---

2. Specify the number of tables you want to import in the Maximum number of objects displayed field.

### Selecting the Objects

The Object Selection page contains a description of the objects and enables you to select the objects you want to import into the SAP module. To select the objects:

1. Move the objects from the available list to the selected list.

The Import Wizard also enables you to choose whether you want to import tables with foreign key relationships for each object that you choose to import. You can select one of the following:

**None:** Import only the objects in the Selected list.

**One Level:** Import the objects in the Selected list and any tables linked to it directly through a foreign key relationship.

**All Levels:** Import the objects in the Selected list and all tables linked to it through foreign key relationships.

The foreign key level you select is the same for all tables selected for importing.

---

---

**Note:** Selecting All Levels increases the time it takes to import the metadata because you are directing the wizard to import tables that are related to each other through foreign key constraints. Select this option only if it is necessary.

---

---

2. Click **Next**.

If you select One Level or All Levels, the Confirm Import Selection dialog box is displayed.

Review this dialog box to ensure that you are selecting the required tables.

3. Click **OK**.

The selected objects appear in the Selected list of the Object Selection page.

4. Click **Next**.

The wizard displays the Summary and Import page.

### Reviewing Import Summary

The wizard imports definitions for the selected tables from the SAP Application Server, stores them in the SAP source module, and then displays the Summary and Import page.

You can edit the descriptions for each table by selecting the Description field and typing a new description.

Review the information on the Summary and Import page and click **Finish**.

The SAP Connector reads the table definitions from the SAP application server and creates the metadata objects in the workspace.

The time it takes to import the SAP metadata into the workspace depends on the size and number of tables and the connection between the SAP application server and the workspace. Importing 500 or more objects could take one to three hours or more, especially if you are connecting servers in separate Local Area Networks (LANs).

When the import completes, the Import Results dialog box displays. Click **OK** to finish importing metadata.

### Reimporting SAP Objects

To reimport SAP objects, follow the importing procedure using the Import Metadata Wizard. Prior to starting the import, the wizard checks the source for tables with the same name as those you are importing. The tables that have already been imported appear in bold in the Object Selection page. In the Summary and Import page, the Action column indicates that these tables will be reimported. The wizard then activates the Advanced Synchronize Options button so that you can control the reimport options.

## Updating SAP Source Modules

You must update existing SAP source module definitions whenever you upgrade SAP application versions, migrate SAP servers, and change network connection configurations. You also need to check this information when you reimport metadata.

You can update an SAP module by editing its properties using the Edit Module dialog box.

#### To update SAP object definition:

1. From the Project Explorer, expand the Applications node and then the SAP node.
2. Right-click the SAP source object and select **Open Editor**.

The Edit Module dialog box is displayed.

3. Select the appropriate tab to edit the SAP object properties.

**Name:** Use the Name tab to specify a name and an optional description for the table. Use the description field, for example, to note the purpose of the module and how it relates to the information required by the end-users of the project. In addition to the rules listed in "[Naming Conventions for Data Objects](#)" on page 6-6, the name must be unique across the module.

If necessary, change the status of the SAP object. Select Development, Quality Assurance, or Production.

**Data Source:** Use this tab to modify the application type.

**Metadata Location:** Use this tab to change the location of the metadata.

**Data Locations:** Use this tab to change the data location. You can either select from an existing list of available locations or specify a new location.

## Defining the ETL Process for SAP Objects

After you define the SAP source module and import the metadata, you can define the ETL mappings to extract and load the data from your SAP source to the target. The SAP Connector features a special mapping tool for SAP objects. Warehouse Builder

enables you to configure mappings to generate ABAP or PL/SQL code to deploy your metadata.

This section contains the following topics:

- [Defining Mappings Containing SAP Objects](#)
- [Configuring Code Generation for SAP Objects](#)
- [Generating SAP Definitions](#)

### **Defining Mappings Containing SAP Objects**

You can use the Mapping Editor to define mappings for SAP sources. While SAP mappings are similar to other types of mappings, there is one important difference, which is that only Table, Filter, Joiner, and Mapping Input Parameter mapping operators are available for SAP objects.

#### **Adding SAP Objects to a Mapping**

**To add an SAP object to a mapping:**

1. From the Mapping Editor Palette, drag and drop the Table operator onto the Mapping Editor canvas.

The Add Table Operator dialog box displays.

2. Choose **Select from existing repository objects and bind**.

The field at the bottom of the dialog box displays a list of SAP tables whose definitions were previously imported into the SAP source module.

3. Select a source table name and click **OK**.

The editor places a Table operator on the mapping canvas to represent the SAP table.

You can define it as you would with any other type of mapping operator.

#### **Configuring Code Generation for SAP Objects**

Configuring a mapping containing an SAP source is similar to configuring a mapping containing any other source:

- Use the Operator properties panel of the Mapping Editor to set the loading properties.
- Use the Configuration properties dialog box to define the code generation properties.
- If you intend to generate ABAP code, set the directory and initialization file settings in the Configuration properties dialog box.

#### **Setting the Loading Type**

**To set the loading type for an SAP operator:**

1. On the Mapping Editor, select the SAP source operator. The Table Operator Properties panel displays the properties of the SAP table operator.
2. Select a loading type from the Loading Type list. If you specify ABAP code as the language for the mapping, the SQL\*Loader code is generated as indicated in [Table 4–2](#).



**Table 4–2 Loading Types in ABAP Code**

Loading Type	SQL* Loader Code Generated in ABAP Code
INSERT	APPEND
CHECK/INSERT	INSERT
TRUNCATE/INSERT	TRUNCATE
DELETE/INSERT	REPLACE
All other types	APPEND

### Setting the Language Parameter

This parameter enables you to choose the type of code you want to generate for your SAP mappings. If your source includes clustered or pooled tables, then you must select ABAP as the generated code.

#### To choose the language:

1. Right-click the mapping and select **Configure**.  
The Configuration Properties dialog box is displayed.
2. From the list in the **Language** field, select the type of code you want to generate: ABAP, SQL\*LOADER, or PL/SQL scripts (available for transparent tables only).
3. Click **OK**.

### Setting the Runtime Parameters

If you set the language to ABAP, then you can expand the Runtime Parameters node in the Configuration Properties dialog box to display settings specific to ABAP code generation. These settings come with preset properties that optimize code generation and should not be changed. Altering these settings can result in a slowing down of the code generation process.

The following runtime parameters are available for SAP mappings:

- **SAP System Version:** Specifies the SAP system version number to which you want to deploy the ABAP code. For MySAP ERP instances, select SAP R/3 4.7.
- **Staging File Directory:** Specifies the location of the directory where the data generated by ABAP code resides.
- **Data File Name:** Specifies the name of the data file created during code generation.
- **File Delimiter for Staging File:** Specifies the column separator in a SQL data file.
- **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:** Specify the primary key to be used for a join.
- **Nested Loop:** Specifies a hint to generate nested loop code for a join, if possible.
- **Use Select Single:** Indicates whether Select Single is generated, if possible.
- **SAP Location:** The location of the SAP instance from where the data can be extracted.

- **Background Job:** Select this option if you wish to run the ABAP report as a background job in the SAP system.

### Generating SAP Definitions

You can generate PL/SQL code for a mapping containing an SAP transparent table just as you generated code for any other PL/SQL mapping. However, you must generate ABAP code for pooled and cluster tables.

Warehouse Builder validates and generates the scripts required to create and populate the SAP source object.

When you generate code, a single script is generated for each physical object you want to create. For example, there is one script for each index you are creating. This is useful if you need to re-deploy a single object at a later time without re-deploying the entire warehouse.

#### To generate the scripts for SAP mappings:

1. Right-click the SAP mapping and select **Generate**.  
The Generation Results window is displayed.
2. On the Script tab, select the script name and select **View Code**.  
The generated code displays in the Code Viewer.  
You can edit, print, or save the file using the code editor. Close the Code Viewer to return to the Generation Results window.
3. From the Generation Results window, click **Save as File** to save the ABAP code to your hard drive.
4. Click **Save** to save the generated scripts to a file system. You can save the ABAP code with any file extension. You can use the suffix `.abap` (for example, `MAP1.abap`) or any other naming convention.

## Loading SAP Data into the Workspace

When you generate an ABAP code for an SAP mapping, Warehouse Builder creates an ABAP program that loads the data. You must run this program from the SAP user interface. The program uploads the generated code and executes it on your SAP system. You can then load the data into your staging area before using SQL\*Loader to upload the data into your warehouse tables.

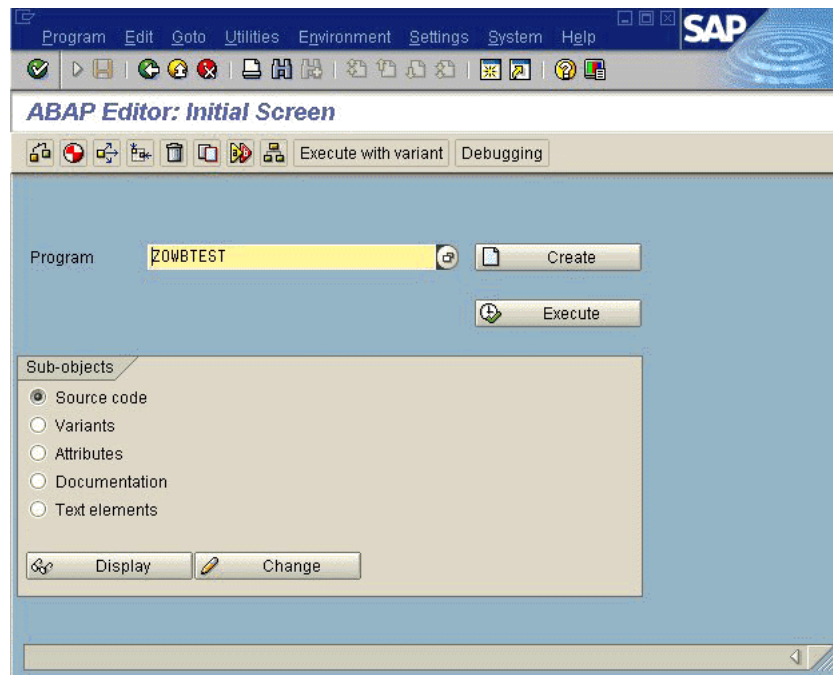
#### To upload and execute the ABAP code on your SAP system using the SAP user interface:

1. Open the SAP user interface and specify op-code `SE38`.
2. Create a program to execute the ABAP code (for example, `ZOWBTEST1`). For detailed instructions on creating a program, refer to your SAP documentation. If you already have a program created for testing purposes, you can use it to execute the ABAP code.

The default selection is set to Source Code.

[Figure 4–2](#) shows the SAP ABAP editor.

Figure 4–2 SAP ABAP Editor



3. Click **Change**.
4. From the ABAP Editor menu, select **Utilities**, then **Upload/Download**, and then **Upload**.
5. In the File Name field, specify the location of the generated ABAP code.
6. Click **Transfer**.
7. Press F8 to execute the ABAP code. Or you can also select **Program** and then **Check** before selecting **Program** and then **Execute** to run the code.

The ABAP code is executed in the SAP application server.

8. Use FTP to fetch data from the SAP application server and send it to the staging area.
9. Use SQL\*Loader to upload data into your warehouse tables. The following is an example of a command line:

```
SQLldr USERID=scott/tiger CONTROL=abap_datactlfile.dat LOG=yourlogfile.log
```

### Deploying and Executing an SAP Mapping

After you create an SAP mapping, you must deploy the mapping to create the logical objects in the target location. Deploying an SAP mapping is similar to deploying any other object. To deploy an SAP mapping, right-click the mapping and select **Deploy**. You can also deploy the mapping from Control Center Manager. For detailed information about deployment, see "[Deploying to Target Schemas and Executing ETL Logic](#)" on page 11-1.

When an SAP mapping is deployed, an ABAP mapping is created and stored in the workspace.

It also saves the .abap file under `OWB_ORACLE_HOME\owb\deployed_files`, where `OWB_ORACLE_HOME` is the location of the oracle home directory of your installation.

Executing an SAP mapping is similar to executing other objects. Before executing the mapping, make sure that the mapping has been deployed successfully.

To execute an SAP mapping, you need to perform the following steps:

1. From Control Center Manager, right-click the deployed SAP mapping and select **Start**.

The ABAP mapping is executed on the remote SAP instance and the resultant file is stored under the file system of the SAP instance.

2. Use FTP to transfer the file from the remote SAP system to the local system. Make sure that you provide the correct user name and password for the FTP connection.
3. Use SQL\*Loader to upload the file into Warehouse Builder.

The auditing information is written onto the workspace and can be viewed from the Repository Browser. For more information about auditing, see "Auditing Deployments and Executions" in the *Warehouse Builder Online Help*.

### Deploying PL/SQL Scripts for Transparent Tables

Deployment of PL/SQL scripts for SAP transparent tables is the same as deployment of PL/SQL scripts for Oracle Database sources. The PL/SQL scripts run in your Oracle data warehouse and perform remote queries to extract table data from the SAP application. For more information about deployment, see "[Deploying to Target Schemas and Executing ETL Logic](#)" on page 11-1.

## Integrating with Business Intelligence Tools

Warehouse Builder provides an end-to-end business intelligence solution by enabling you to integrate metadata from different data sources, designing and deploying it to a data warehouse, and making that information available to analytical tools for decision making and business reporting.

Warehouse Builder introduces Business Intelligence (BI) objects that enable you to integrate with Oracle Business Intelligence tools such as Discoverer. You can define BI objects in Warehouse Builder that enable you to store definitions of business views. You can then deploy these definitions to the Oracle Business Intelligence tools and extend the life-cycle of your data warehouse. The method you use to deploy business definitions depends on the version of Discoverer to which you want to deploy and the Warehouse Builder licensing option you purchased. For more information, see "[Deploying Business Definitions to Oracle Discoverer](#)" on page 11-8.

This section contains the following topics:

- [Introduction to Business Intelligence Objects in Warehouse Builder](#)
- [Introduction to Business Definitions](#)
- [About Business Definitions](#)

### Introduction to Business Intelligence Objects in Warehouse Builder

Warehouse Builder enables you to derive and define Business Intelligence (BI) objects that integrate with analytical business intelligence tools, such as Oracle Discoverer. By deploying these BI definitions to your analytical tools, you can perform ad hoc queries

on top of the relational data warehouse or define a dashboard on top of multidimensional data marts.

The BI objects you derive or define in Warehouse Builder represent equivalent objects in Oracle Discoverer. These definitions are stored under the Business Intelligence node on the Warehouse Builder Project Explorer.

The Business Intelligence node contains an additional node called Business Definitions. You start by first creating a Business Definition module to store the definitions to be deployed to Discoverer. For details, see "[About Business Definitions](#)" on page 4-33.

## Introduction to Business Definitions

Business intelligence is the ability to analyze data to answer business questions and predict future trends. Oracle Discoverer is a BI tool that enables users to analyze data and retrieve information necessary to take business decisions. Discoverer also enables users to share the results of their data analysis in different formats, including charts and Excel spreadsheets.

Discoverer uses the End User Layer (EUL) metadata view to insulate its end users from the complexity and physical structure of the database. You can tailor the EUL to suit your analytical and business requirements and produce queries by generating SQL. The EUL provides a rich set of default settings to aid report building.

Through BI objects, Warehouse Builder enables you to design a data structure that facilitates this data analysis. Business Intelligence objects in Warehouse Builder provide the following benefits:

- Complete and seamless integration with Oracle Discoverer.
- Advanced deployment control of metadata objects using the Warehouse Builder Control Center.
- Complete, end-to-end lineage and impact analysis of Discoverer objects based on information in the Warehouse Builder workspace.
- Ability to utilize Warehouse Builder metadata management features such as snapshots, multilanguage support, and command-line interaction.

## About Business Definitions

You can integrate with Discoverer by deriving business definitions directly from your warehouse design metadata. Alternatively, you can also create your own customized business definitions in Warehouse Builder.

The business definition objects in Warehouse Builder are equivalent to the Discoverer EUL objects. When you derive business definitions from your existing design metadata, Warehouse Builder organizes the definitions in Item Folders that correspond to Folders in Discoverer. You can define joins and conditions for the Items Folders and select the Items they contain using the Warehouse Builder wizards and editors. Additionally, you can define Drill Paths, Alternative Sort Orders, Drills to Detail, and Lists of Values for the Items within the Item Folders.

Warehouse Builder also enables you to define any functions registered with Discoverer. You can also sort your definitions by subject area by defining Business Areas that reference multiple Item Folders. You can then deploy these Business Areas along with the business definitions to a Discoverer EUL using the Control Center.

**See Also:**

- ["Deriving Business Intelligence Metadata"](#) on page 6-48
- ["Defining Business Intelligence Objects"](#) in the *Warehouse Builder Online Help*
- ["Deploying Business Definitions to Oracle Discoverer"](#) on page 11-8

---

---

## Understanding Data Quality Management

Today, more than ever, organizations realize the importance of data quality. By ensuring that quality data is stored in your data warehouse or business intelligence application, you also ensure the quality of information for dependent applications and analytics.

Oracle Warehouse Builder offers a set of features that assist you in creating data systems that provide high quality information to your business users. You can implement a quality process that assesses, designs, transforms, and monitors quality. Within these phases, you will use specific functionality from Warehouse Builder to create improved quality information.

This chapter contains the following topics:

- [About the Data Quality Management Process](#)
- [About Data Profiling](#)
- [About Data Correction and Augmentation](#)
- [About Data Rules](#)
- [About Quality Monitoring](#)
- [Performing Data Profiling](#)
- [Tuning the Data Profiling Process](#)
- [Using Data Rules](#)
- [Monitoring Data Quality Using Data Auditors](#)

### About the Data Quality Management Process

Quality data is crucial to decision-making and planning. The aim of building a data warehouse is to have an integrated, single source of data that can be used to make business decisions. Since the data is usually sourced from a number of disparate systems, it is important to ensure that the data is standardized and cleansed before loading into the data warehouse.

Warehouse Builder provides functionality that enables you to effectively manage data quality by assessing, transforming, and monitoring your data. The benefits of using Warehouse Builder for data management are as follows:

- Provides an end-to-end data quality solution
- Enables you to include data quality and data profiling as an integral part of your data integration process.

- Stores metadata regarding the quality of your data alongside your data definitions.
- Automatically generates the mappings that you can use to correct data. These mappings are based on the business rules that you choose to apply to your data and decisions you make on how to correct data.

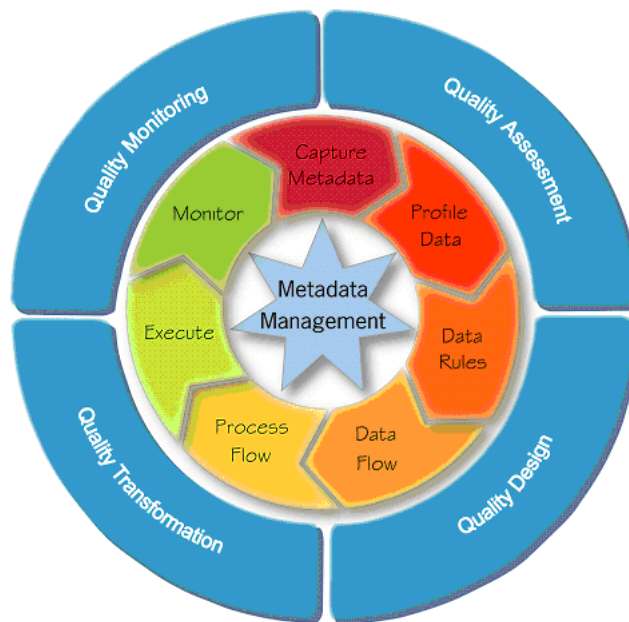
## Phases in the Data Quality Lifecycle

Ensuring data quality involves the following phases:

- [Quality Assessment](#)
- [Quality Design](#)
- [Quality Transformation](#)
- [Quality Monitoring](#)

Figure 5–1 shows the phases involved in providing high quality information to your business users.

**Figure 5–1 Phases Involved in Providing Quality Information**



### Quality Assessment

In the quality assessment phase, you determine the quality of the source data. The first step in this phase is to import the source data, which could be stored in different sources, into Warehouse Builder. You can import metadata and data from both Oracle and non-Oracle sources.

After you load the source data, you use data profiling to assess its quality. Data profiling is the process of uncovering data anomalies, inconsistencies, and redundancies by analyzing the content, structure, and relationships within the data. The analysis and data discovery techniques form the basis for data monitoring.



**See Also:**

- ["About Data Profiling"](#) on page 5-3 for data profiling concepts
- ["Performing Data Profiling"](#) on page 5-32 for the steps to perform data profiling
- ["Tuning the Data Profiling Process"](#) on page 5-40 for information about tuning the profiling process

**Quality Design**

The quality design phase consists of designing your quality processes. You can specify the legal data within a data object or legal relationships between data objects using data rules.

**See Also:**

- ["About Data Rules"](#) on page 5-31 for data rules concepts
- ["Using Data Rules"](#) on page 5-41 for information about creating and applying data rules

You also correct and augment your data. You can use data quality operators to correct and augment data.

**See Also:**

- ["About Data Correction and Augmentation"](#) on page 5-9 for information about the data quality operators
- ["Generate Corrections"](#) on page 5-37 for information about generating corrections based on the results of data profiling

As part of the quality design phase, you also design the transformations that ensure data quality. These transformations could be mappings that are generated by Warehouse Builder as a result of data profiling or mappings you create.

**Quality Transformation**

The quality transformation phase consists of running the correction mappings you designed to correct the source data.

**Quality Monitoring**

Data monitoring is the process of examining warehouse data over time and alerting you when the data violates business rules set for the data.

**See Also:**

- ["About Quality Monitoring"](#) on page 5-31 for concepts about quality monitoring
- ["Monitoring Data Quality Using Data Auditors"](#) on page 5-43 for information about creating and using data auditors to monitor data quality

## About Data Profiling

Data profiling is the first step for any organization to improve information quality and provide better decisions. It is a robust data analysis method available in Warehouse

Builder that you can use to discover and measure defects in your data before you start working with it. Because of its integration with the ETL features in Warehouse Builder and other data quality features, such as data rules and built-in cleansing algorithms, you can also generate data cleansing mappings and schema correction scripts. This enables you to automatically correct any inconsistencies, redundancies, and inaccuracies in both the data and metadata.

Data profiling enables you to discover many important things about your data. Some common findings include the following:

- A domain of valid product codes
- A range of product discounts
- Columns that hold the pattern of an e-mail address
- A one-to-many relationship between columns
- Anomalies and outliers within columns
- Relations between tables even if they are not documented in the database

## Benefits of Data Profiling

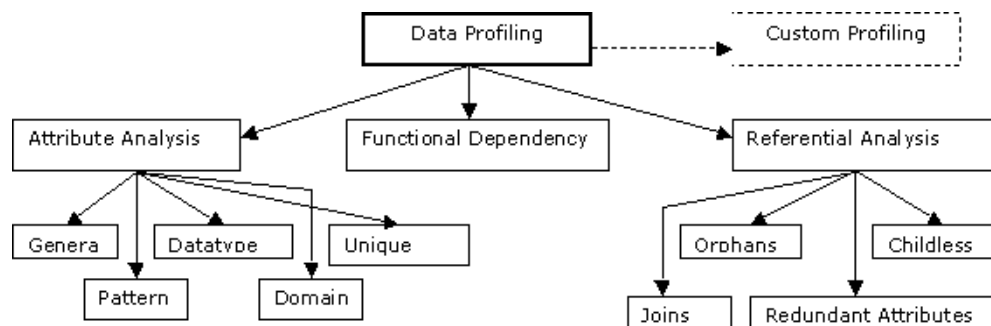
Using the data profiling functionality in Warehouse Builder enables you to:

- Profile data from any source or combination of sources that Warehouse Builder can access.
- Explore data profiling results in tabular or graphical format.
- Drill down into the actual data related to any profiling result.
- Derive data rules, either manually or automatically, based on the data profiling results.
- Attach any data rule to a target object and select an action to perform if the rule fails.
- Create a data auditor from a data rule to continue monitoring the quality of data being loaded into an object.
- Derive quality indices such as six-sigma valuations.
- Profile or test any data rules you want to verify before putting in place.

## Types of Data Profiling

Following the selection of data objects, determine the aspects of your data that you want to profile and analyze. Data profiling offers three main types of analysis: attribute analysis, functional dependency, and referential analysis. You can also create custom profiling processes using data rules, allowing you to validate custom rules against the actual data and get a score of their accuracy.

[Figure 5–2](#) displays a representation of the types of data profiling and how you can perform each type.

**Figure 5–2 Data Profiling Overview**

### Attribute Analysis

Attribute analysis seeks to discover both general and detailed information about the structure and content of data stored within a given column or attribute. Attribute analysis looks for information about patterns, domains, data types, and unique values.

Pattern analysis attempts to discover patterns and common types of records by analyzing the string of data stored in the attribute. It identifies the percentages of your data that comply with a certain regular expression format pattern found in the attribute. Using these pattern results, you can create data rules and constraints to help clean up current data problems. Some commonly identified patterns include dates, e-mail addresses, phone numbers, and social security numbers.

Table 5–1 shows a sample attribute, Job Code, that could be used for pattern analysis.

**Table 5–1 Sample Columns Used for Pattern Analysis**

Job ID	Job Code
7	337-A-55
9	740-B-74
10	732-C-04
20	43-D-4

Table 5–2 shows the possible results from pattern analysis, where D represents a digit and X represents a character. After looking at the results and knowing that it is company policy for all job codes be in the format of DDD-X-DD, you can derive a data rule that requires all values in this attribute to conform to this pattern.

**Table 5–2 Pattern Analysis Results**

Job Code	% Occurred
DDD-X-DD	75%
DD-X-D	25%

Domain analysis identifies a domain or set of commonly used values within the attribute by capturing the most frequently occurring values. For example, the Status column in the Customers table is profiled and the results reveal that 90% of the values are among the following: "MARRIED", "SINGLE", "DIVORCED". Further analysis and drilling down into the data reveal that the other 10% contains misspelled versions of these words with few exceptions. Configuration of the profiling determines when something is qualified as a domain, so review the configuration before accepting

domain values. You can then let Warehouse Builder derive a rule that requires the data stored in this attribute to be one of the three values that were qualified as a domain.

Data type analysis enables you to discover information about the data types found in the attribute. This type of analysis reveals metrics such as minimum and maximum character length values as well as scale and precision ranges. In some cases, the database column is of data type `VARCHAR2`, but the values in this column are all numbers. Then you may want to ensure that you only load numbers. Using data type analysis, you can have Warehouse Builder derive a rule that requires all data stored within an attribute to be of the same data type.

Unique key analysis provides information to assist you in determining whether or not an attribute is a unique key. It does this by looking at the percentages of distinct values that occur in the attribute. You might determine that attributes with a minimum of 70% distinct values should be flagged for unique key analysis. For example, using unique key analysis you could discover that 95% of the values in the `EMP_ID` column are unique. Further analysis of the other 5% reveals that most of these values are either duplicates or nulls. You could then derive a rule that requires that all entries into the `EMP_ID` column be unique and not null.

### Functional Dependency

Functional dependency analysis reveals information about column relationships. This enables you to search for things such as one attribute determining another attribute within an object.

[Table 5–3](#) shows the contents of the Employees table in which the attribute Dept. Location is dependent on the attribute Dept. Number. Note that the attribute Dept. Number is not dependent on the attribute Dept. Location.

**Table 5–3 Employees Table**

ID	Name	Salary	Dept Number	Dept Location
10	Alison	1000	10	SF
20	Rochnik	1000	11	London
30	Meijer	300	12	LA
40	John	500	13	London
50	George	200	13	London
60	Paul	600	13	London
70	Ringo	100	13	London
80	Yoko	600	13	London
90	Jones	1200	10	SF

### Referential Analysis

Referential analysis attempts to detect aspects of your data objects that refer to other objects. The purpose behind this type of analysis is to provide insight into how the object you are profiling is related or connected to other objects. Because you are comparing two objects in this type of analysis, one is often referred to as the parent object and the other as the child object. Some of the common things detected include orphans, childless objects, redundant objects, and joins. Orphans are values that are found in the child object, but not found in the parent object. Childless objects are values that are found in the parent object, but not found in the child object. Redundant attributes are values that exist in both the parent and child objects.

Table 5-4 and Table 5-5 show the contents of two tables that are candidates for referential analysis. Table 5-4 is the child object and Table 5-5 is the parent object.

**Table 5-4 Employees Table (Child)**

ID	Name	Dept. Number	City
10	Alison	17	NY
20	Rochnik	23	SF
30	Meijer	23	SF
40	Jones	15	SD

**Table 5-5 Department Table (Parent)**

Dept. Number	Location
17	NY
18	London
20	SF
23	SF
55	HK

Referential analysis of these two objects would reveal that Dept. Number 15 from the Employees table is an orphan and Dept. Numbers 18, 20, and 55 from the Department table are childless. It would also reveal a join on the Dept. Number column.

Based on these results, you could derive referential rules that determine the cardinality between the two tables.

### Data Rule Profiling

In addition to attribute analysis, functional dependency, and referential analysis, Warehouse Builder offers data rule profiling. Data rule profiling enables you to create rules to search for profile parameters within or between objects.

This is very powerful as it enables you to validate rules that apparently exist and are defined by the business users. By creating a data rule, and then profiling with this rule you can verify if the data actually complies with the rule, and whether or not the rule needs amending or the data needs cleansing.

For example, you could create a rule that  $\text{Income} = \text{Salary} + \text{Bonus}$  for the Employee table shown in Table 5-6. You can then catch errors such as the one for employee Alison.

**Table 5-6 Sample Employee Table**

ID	Name	Salary	Bonus	Income	
10	Alison	1000	50	1075	X
20	Rochnik	1000	75	1075	
30	Meijer	300	35	335	
40	Jones	1200	500	1700	

## About Six Sigma

Warehouse Builder provides Six Sigma results embedded within the other data profiling results to provide a standardized approach to data quality.

### What is Six Sigma?

Six Sigma is a methodology that attempts to standardize the concept of quality in business processes. It achieves this goal by statistically analyzing the performance of business processes. The goal of Six Sigma is to improve the performance of these processes by identifying the defects, understanding them, and eliminating the variables that cause these defects.

Six Sigma metrics give a quantitative number for the number of defects for each 1,000,000 opportunities. The term "opportunities" can be interpreted as the number of records. The perfect score is 6.0. The score of 6.0 is achieved when there are only 3.4 defects for each 1,000,000 opportunities. The score is calculated using the following formula:

- Defects Per Million Opportunities (DPMO) =  $(\text{Total Defects} / \text{Total Opportunities}) * 1,000,000$
- Defects (%) =  $(\text{Total Defects} / \text{Total Opportunities}) * 100\%$
- Yield (%) =  $100 - \% \text{Defects}$
- Process Sigma =  $\text{NORMSINV}(1 - ((\text{Total Defects}) / (\text{Total Opportunities}))) + 1.5$   
where NORMSINV is the inverse of the standard normal cumulative distribution.

### Six Sigma Metrics for Data Profiling

Six Sigma metrics are also provided for data profiling in Warehouse Builder. When you perform data profiling, the number of defects and anomalies discovered are shown as Six Sigma metrics. For example, if data profiling finds that a table has a row relationship with a second table, the number of records in the first table that do not adhere to this row-relationship can be described using the Six Sigma metric.

Six Sigma metrics are calculated for the following measures in the Data Profile Editor:

- **Aggregation:** For each column, the number of null values (defects) to the total number of rows in the table (opportunities).
- **Data Types:** For each column, the number of values that do not comply with the documented data type (defects) to the total number of rows in the table (opportunities).
- **Data Types:** For each column, the number of values that do not comply with the documented length (defects) to the total number of rows in the table (opportunities).
- **Data Types:** For each column, the number of values that do not comply with the documented scale (defects) to the total number of rows in the table (opportunities).
- **Data Types:** For each column, the number of values that do not comply with the documented precision (defects) to the total number of rows in the table (opportunities).
- **Patterns:** For each column, the number of values that do not comply with the common format (defects) to the total number of rows in the table (opportunities).

- **Domains:** For each column, the number of values that do not comply with the documented domain (defects) to the total number of rows in the table (opportunities).
- **Referential:** For each relationship, the number of values that do not comply with the documented foreign key (defects) to the total number of rows in the table (opportunities).
- **Referential:** For each column, the number of values that are redundant (defects) to the total number of rows in the table (opportunities).
- **Unique Key:** For each unique key, the number of values that do not comply with the documented unique key (defects) to the total number of rows in the table (opportunities).
- **Unique Key:** For each foreign key, the number of rows that are childless (defects) to the total number of rows in the table (opportunities).
- **Data Rule:** For each data rule applied to the data profile, the number of rows that fail the data rule to the number of rows in the table.

## About Data Correction and Augmentation

Warehouse Builder enables you to automatically create correction mappings based on the results of data profiling. On top of these automated corrections that make use of the underlying Warehouse Builder architecture for data quality, you can create your own data quality mappings.

Warehouse Builder provides functionality that enables you to correct and augment source data. While transforming the source data, you can use the following operators to ensure data quality:

- Match-Merge Operator  
See "[About the Match-Merge Operator](#)" on page 5-9
- Name and Address Operator  
See "[About the Name and Address Operator](#)" on page 5-26

### About the Match-Merge Operator

Duplicate records can obscure your understanding of who your customers and suppliers really are. Eliminating duplicate records is an important activity in the data correction process. The Match-Merge operator enables you to identify matching records and merge them into a single record. You can define the business rules used by the Match-Merge operator to identify records in a table that refer to the same data. Master data management working on various systems will make use of this operator to ensure that records are created and matched with a master record.

The Match-Merge operator provides the following:

- Enables you to use weights to determine matches between records.
- Uses built-in algorithms, including the Jaro-Winkler and edit distance algorithms, to determine matches.
- Enables cross referencing of data to track and audit matches.
- Enables you to create custom rules combining built-in rules for matching and merging.

## Example of Matching and Merging Customer Data

Consider how you could utilize 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 5-7 shows records that refer to the same person prior to using the Match-Merge operator.

**Table 5-7 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 5-8 shows the single record for Jane Doe after using the Match-Merge operator. Notice that the new record retrieves data from different rows in the sample.

**Table 5-8 Match-Merge Results**

First Name	Last Name	SSN	Address	Unit	Zip
Jane	Doe	111111111	123 Main Street	Apt 4	22222

### Restrictions on Using the 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.
- Because the Match-Merge 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 Sort operator or another Match-Merge operator.

## 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 an exact match to sophisticated algorithms that can discover and correct common data entry errors.

**See Also:** ["Match Rules"](#) on page 5-12 for more information about match rules

Merging consolidates matched records into a single record that is free from duplicate records, omissions, misspellings, and unnecessary variations. You can define merge rules to select the preferred data values for use in the consolidated record.



**See Also:** ["Merge Rules"](#) on page 5-23 for more information about merge rules

Warehouse Builder uses the following in the matching and merging process.

### **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, only records within the same match bin are compared. 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 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.

### **Match Record Sets**

A match record set consists of one or more similar records. After matching records, a match record set is created for each match bin. You can define the 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.

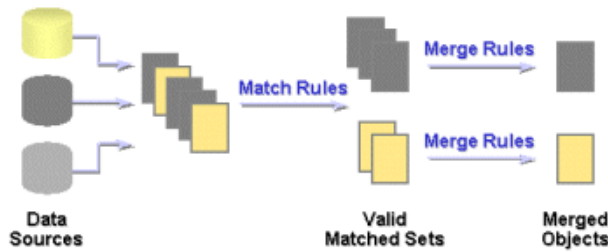
### **Matching and Merging Records**

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.

[Figure 5-3](#) describes the matching and merging process. The high-level tasks involved in the process are:

- [Constructing Match Bins](#)
- [Constructing Match Record Sets](#)
- [Constructing Merge Records](#)

**Figure 5–3 Match-Merge 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. For more information about match rules, see ["Match Rules"](#) on page 5-12.

**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. A match rule is an  $n \times n$  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, this means that record A is equal to record C.

**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. For more information about merge rules, see ["Merge Rules"](#) on page 5-23.

**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 not executed.

[Table 5–9](#) describes the types of match rules.

**Table 5–9 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 <a href="#">Conditional Match Rules</a> on page 5-13.
Weight	Matches row based on scores that you assign to the attributes. For more information about Weight match rules and how to create one, see <a href="#">"Weight Match Rules"</a> on page 5-15.

**Table 5–9 (Cont.) Types of Match Rules**

Match Rule	Description
Person	Matches records based on the names of people. For more information about Person match rules and how to create one, see <a href="#">"Person Match Rules"</a> on page 5-16.
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 <a href="#">"Firm Match Rules"</a> on page 5-18.
Address	Matches records based on postal addresses. For more information about Address match rules and how to create one, see <a href="#">"Address Match Rules"</a> on page 5-20.
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 <a href="#">"Custom Match Rules"</a> on page 5-22.

## Conditional Match Rules

Conditional match rules specify the conditions under which records match.

A conditional match rule allows 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.

You can specify how attributes are compared using comparison algorithms.

### 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 5–10](#) describes the types of comparisons.

**Table 5–10 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.  For data types other than <code>STRING</code> , this is the only type of comparison allowed.
Standardized Exact	Standardizes the values of the attributes before comparing for an exact match. With standardization, the comparison ignores case, spaces, and non-alphanumeric 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.
Edit Distance	A "similarity score" in the range 0-100 is entered. If the similarity of the two attributes is equal or greater to the specified value, the attribute values are considered matched.  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.  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.

**Table 5–10 (Cont.) Types of Comparison Algorithms for Conditional Match Rules**

<b>Algorithm</b>	<b>Description</b>
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 non-alphanumeric 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 non-alphanumeric characters.
Abbreviation	<p>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 non-alphanumeric character.</p> <p>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.</p> <p>For example, "Intl. Business Products" would match "International Bus Prd".</p>
Acronym	<p>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, 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, the names are considered a match.</p> <p>For example, "Chase Manhattan Bank NA" matches "CMB North America". The comparison ignores case and non-alphanumeric characters.</p>
Jaro-Wrinkler	<p>Matches strings based on their similarity value using 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.</p> <p>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.</p>
Standardized Jaro-Wrinkler	Eliminates case, spaces, and non-alphanumeric characters before using the Jaro-Winkler algorithm to determine a match.

**Table 5–10 (Cont.) Types of Comparison Algorithms for Conditional Match Rules**

Algorithm	Description
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 5–10](#) 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.

### Weight Match Rules

A weighted match rule allows 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, 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, percentage value in the range 0-100, 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.

### Example of Weight Match Rules

[Table 5–11](#) displays the attribute values contained in two separate records that are read in the following order.

**Table 5–11 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.

- Since 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 43 (0.83 X 50).
- Since 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 5–12](#) describes the roles for different parts of a name that are used for matching. On the Match Rules page or Match Rules page, use the Roles column on the Person Attributes tab to define person details.

**Table 5–12 Name Roles for Person Match Rules**

Role	Description
Prename	<p>Prenames are compared only if the following are true:</p> <ul style="list-style-type: none"> <li>▪ 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.</li> <li>▪ The "Mrs. Match" option is selected.</li> <li>▪ Either record has a missing First_name_std.</li> </ul>
First Name Standardized	<p>Compares the first names. By default, the first names must match exactly, but you can specify other comparison options as well.</p> <p>First names match if both are blank. A blank first name will not match a non-blank 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, a role of First_name_std must be assigned.</p>
Middle Name Standardized, Middle Name 2 Standardized, Middle Name 3 Standardized	<p>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, attributes assigned to the different roles are cross-compared.</p> <p>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, the First_name_std role must also be assigned.</p>
Last Name	<p>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.</p>
Maturity Post Name	<p>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.</p>

## Person Details

[Table 5–13](#) 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 5–13 Options for Person Match Rule**

Option	Description
Detect switched name order	<p>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.</p>
Match on initials	<p>Matches initials to names such as 'R.' and 'Robert'. You can select this option for first name and middle name roles.</p>
Match on substrings	<p>Matches substrings to names such as 'Rob' to 'Robert'. You can select this option for first name and middle name roles.</p>

**Table 5–13 (Cont.) Options for Person Match Rule**

Option	Description
Similarity Score	Records are considered a match if the similarity is greater than or equal to score. For example, "Susan" will match "Susen" if the score is less than or equal to 80.  Uses a similarity score to determine a match, as calculated by the Edit Distance or Jaro-Winkler algorithms. A value of 100 requires an exact match, and a value of 0 requires no similarity whatsoever.
Match on Phonetic Codes	Determines a match using either the Soundex or the Double Metaphone algorithms.
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 Name Roles Attributes.
3. For each attribute, select the role 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 5–12](#) for the types of roles you can assign.
4. Select the Details tab and select the applicable options as listed in [Table 5–13](#).

### 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 they have been assigned and other comparison options have you set. For a complete list of firm roles and how each role is treated in a firm match rule, see ["Firm Roles"](#) on page 5-19.



## Firm Roles

Firm roles define the parts of a firm name that are used for matching. The options you can select for firm role are Firm1 or Firm2. If you select one attribute, for firm name, select Firm1 as the role. If you selected two attributes, designate one of them as Firm1 and the other as Firm2.

- **Firm1:** If this role is assigned, the business names represented by Firm1 are compared. Firm1 names will not be compared against Firm2 names unless if 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, the values of the attribute assigned to Firm2 will be compared. Firm2 names will not be compared against Firm1 names unless if 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, a Firm2 roles must be assigned.

## Firm Details

Table 5–14 describes the rule options you can specify for each component of the firm name.

**Table 5–14 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 firm 1 and firm 2	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 algorithms. 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.  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 5-19.

## 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 Rules work differently depending on whether the address being processed has been corrected or not. Generally, corrected addresses have already been identified in a postal matching database, and are therefore not only syntactically correct, but are legal and existing addresses according to the Postal Service of the country containing the address. Corrected addresses can be processed more quickly, since 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 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 5-15](#) describes the address roles you can select for each part of an address.

**Table 5-15 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.  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 is set, 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, 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.

**Table 5–15 (Cont.) Address Roles**

<b>Role</b>	<b>Description</b>
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.
City	<p>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.</p> <p>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.</p>
State	<p>Assign this role only when also assigning the City role.</p> <p>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.</p>
Postal Code	<p>For uncorrected address data, the operator does not use Postal Code.</p> <p>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, City and State roles are compared to determine whether the address line roles should be compared.</p>
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 the address has been found in the postal database or not.

## Address Details

Table 5–16 describes the options for determining a match for an address rule.

**Table 5–16 Options for Address Roles**

<b>Option</b>	<b>Description</b>
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	Matches records if either the street address or the post office box match.

**Table 5–16 (Cont.) Options for Address Roles**

Option	Description
Address line similarity	Match if address line similarity $\geq$ the score. All spaces and non-alphanumeric characters are removed before the similarity is calculated.
Last line similarity	Match is the last line similarity $\geq$ score. The last line consists of city and state. All spaces and non-alphanumeric 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, the match rule is invalid.
4. Add other attributes and designate their roles as necessary. See [Table 5–15](#) for the types of roles you can assign.
5. Select the Details tab and select the applicable options as listed in [Table 5–16](#).

### 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 5–1](#) describes the PL/SQL code used to create the custom match rule that implements this example.

#### **Example 5–1 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.

[Table 5–17](#) describes the types of merge rules.

**Table 5–17 Merge Rule Types**

Merge Rule	Description
Any	Uses the first non-blank 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.

**Table 5–17 (Cont.) Merge Rule Types**

Merge Rule	Description
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.
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.

## Using a Match-Merge Operator

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.

### To match and merge source data using the Match-Merge operator:

1. Drag and drop the operator 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 and Address 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, select the attributes that will be used as input to the Match-Merge operator.

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 operators, 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 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. Ensure that you 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 new record.
- Old records will not be compared with each other.
- A matched record set will not be presented to the merge processing unless the matched record set contains a new record.
- An old record will not be 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 5-10.

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 you can define, and the steps used to define them, see "[Match Rules](#)" on page 5-12.

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 type of Merge rules and the steps to create Merge rules, see "[Merge Rules](#)" on page 5-23.

12. On the Summary page, review your selections. Click **Back** to modify any selection 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.

## About the Name and Address Operator

After matching and merging records, you can further validate information about your customers and suppliers, and discover additional errors and inconsistencies. 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, which can augment your records with information such as postal routes and geographic coordinates.

Successful delivery and lower postage rates are not the only reasons to cleanse name and address data. You will get better results from data analysis when the results are not skewed by duplicate records and incomplete information.

Warehouse Builder enables you to perform name and address cleansing on data using the Name and Address operator. The Name and Address operator identifies and corrects errors and inconsistencies in name and address source data by comparing input data to the data libraries supplied by third-party name and address cleansing software vendors. You can purchase the data libraries directly from these vendors.

---

---

**Note:** The Name and Address operator requires separate licensing and installation of third-party name and address cleansing software. Refer to the *Oracle Warehouse Builder Installation and Administration Guide* for more information.

---

---

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 from large cities); marketing only to customers served by a company's stores (within an  $x$ -mile radius from 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 5-30.

## 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 5-18](#).

**Table 5-18 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:

- 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 5-18](#).
  - A Name and Address operator. This action starts the Name and Address Wizard. Follow the steps of the wizard.
  - A Splitter operator. For information on using this operator, see "Splitter Operator" in the *Warehouse Builder Online Help*.

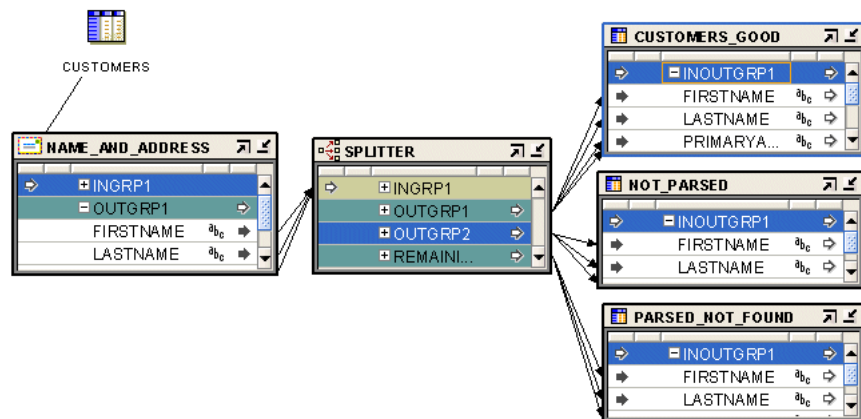
- Three target operators into which you load the successfully parsed records, the records with parsing errors, and the records whose addresses are parsed but not found in the postal matching software.
- 2. Map the attributes from the CUSTOMERS table to the Name and Address operator ingroup. Map the attributes 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 5–4 shows a mapping designed for this example. The data is mapped from the 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 5–4 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 5–19. Compare it with the input record from Table 5–18 on page 5-27.

**Table 5–19 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

**Table 5–19 (Cont.) Sample Output from Name and Address Operator**

<b>Address Column</b>	<b>Address Component</b>
City	BLOOMINGTON
State	MN
Postal Code	55437-3813
Latitude	44.849194
Longitude	-093.356352
Is Parsed	True or False. Indicates whether a record can be separated into individual elements.
Is Good Name	True or False. Indicates whether the name was found in a postal database.
Is Good Address	True or False. Indicates whether the address was found in a postal database or was parsed successfully.
Is Found	True or False. Indicates whether the address was found in a postal database.
Name Warning	True or False. Indicates whether problems occurred in parsing the name.
Street Warning	True or False. Indicates whether problems occurred in parsing the address.
City Warning	True or False. Indicates whether problems occurred in parsing the city name.

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 are directed to a different target from the ones that have problems.

## 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 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.

Since 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, 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, you may still wish 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.

## 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 letter lodgements

PreSort Letter 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.

## About Data Rules

Data rules are definitions for valid data values and relationships that can be created in Warehouse Builder. They determine legal data within a table or legal relationships between tables. Data rules help ensure data quality. They can be applied to tables, views, dimensions, cubes, materialized views, and external tables. Data rules are used in many situations including data profiling, data and schema cleansing, and data auditing.

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 Data Object Editor for the `Employees` table.

There are two ways to create a data rule. A data rule can be derived from the results of data profiling, or it can be created using the Data Rule Wizard. For more information about data rules, see ["Using Data Rules"](#) on page 5-41.

## About Quality Monitoring

Quality monitoring builds on your initial data profiling and data quality initiatives. It enables you to monitor the quality of your data over time. You can define the business rules to which your data should adhere.

To monitor data using Warehouse Builder you need to create data auditors. Data auditors ensure that your data complies with the business rules you defined. You can define the business rules that your data should adhere to using a feature called data rules.

## About 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 have thresholds that allow you to create logic based on the fact that too many non-compliant records can divert the process flow into an error or notification stream. Based on this threshold, the process can choose actions. In addition, the audit results can be captured and stored for analysis purposes.

Data auditors can be deployed and executed ad-hoc, but they are typically run to monitor the quality of the data in an operational environment like a data warehouse or ERP system and, therefore, can be added to a process flow and scheduled.

When executed, the data auditor sets several output values. One of these output values is called the audit result. If the audit result is 0, then there were no errors. If the audit result is 1, at least one error occurred. If the audit result is 2, then at least one data rule failed to meet the specified error threshold. Data auditors also set the actual measured values such as Error Percent and Six Sigma values.

Data auditors are a very important tool in ensuring that data quality levels are up to the standards set by the users of the system. It also helps determine spikes in bad data allowing events to be tied to these spikes.

For information about creating and using data auditors, see "[Monitoring Data Quality Using Data Auditors](#)" on page 5-43.

## Performing Data Profiling

Data profiling is, by definition, a resource-intensive process that requires forethought and planning. It analyzes data and columns and performs many iterations to detect defects and anomalies in your data. So it warrants at least some forethought and planning in order to be as effective as possible.

Before you begin profiling data, first reduce the data set by doing a random sampling. Next identify the data objects that you want to target. Instead of profiling everything, choose objects that are deemed crucial. You should not select an entire source system for profiling at the same time. Not only is it a waste of resources, but it is also often unnecessary. Select areas of your data where quality is essential and has the largest fiscal impact.

For example, you have a data source that contains five tables: CUSTOMERS, REGIONS, ORDERS, PRODUCTS, and PROMOTIONS. You decide that the two most important tables with respect to data quality are CUSTOMERS and ORDERS. The CUSTOMERS table is known to contain many duplicate and erroneous entries that cost your company money on wasted marketing efforts. The ORDERS table is known to contain data about orders in an incorrect format. In this case, you would select only these two tables for data profiling.

### Steps to Perform Data Profiling

After you have chosen the objects you want to profile, use the following steps to guide you through the profiling process:

1. [Import or Select the Metadata](#)
2. [Create a Data Profile](#)
3. [Profile the Data](#)

4. [View Profile Results](#)
5. [Derive Data Rules](#)
6. [Generate Corrections](#)
7. [Define and Edit Data Rules Manually](#)
8. [Generate, Deploy, and Execute](#)

The data profiling process ends at step 4. Steps 5 to 7 are optional and can be performed if you want to perform data correction after the data profiling. Step 8 is required when you perform both data profiling and data correction along with data profiling.

## Import or Select the Metadata

Data profiling requires the profiled objects to be present in the project in which you are performing data profiling. Ensure that these objects are either imported into this project or created in it. Also ensure that the data is loaded into the objects. Having the data loaded is essential to data profiling.

Also, 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.

---

---

**Note:** You can only profile data in the default configuration.

---

---

## Create a Data Profile

After your system is set up, you can create a data profile using the Design Center. A data profile is a metadata object in the workspace. It includes the set of data objects you want profiled, 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 Project Explorer, expand the project node in which you want to create a data profile.
2. Right-click Data Profiles and select **New**.

The Welcome page of the Create Data Profile Wizard is displayed.

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, specify the objects that you want to include in the data profile and click **Next**.

The Available section displays the objects available for profiling. Select the objects to include in the data profile and use the shuttle buttons to move them to the Selected section. 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.

When you select 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. You can select an

attribute set to profile only the columns included in that attribute set. To profile all columns in the data object, select **<all columns>**.

When you select a dimensional object in the Available section, 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.

5. On the Summary page, review the choices you made on the previous wizard pages. Click **Back** to change any selected values. Click **Finish** to create the data profile.

The data profile is added to the Data Profiles node in the navigation tree.

If this is the first data profile you have created in the current project, the Connection Information dialog box for the selected control center is displayed. Enter the connection information and click **OK**. The Data Profile Editor is displayed.

---

---

**Note:** 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.

---

---

## Profile the 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 type of profiling you are running. However, profiling is run as an asynchronous job, and the client can be closed during this process. You will see the job running in the job monitor and Warehouse Builder prompts you when the job is complete.

### Configuring Data Profiles

You can, and should, configure the profile before running it if there are specific types of analysis you do, or do not, want to run. To configure a data profile, you set its configuration parameters in the Property Inspector panel of the Data Profile Editor.

Configuration of the profile and its objects is possible at the following levels:

- The entire profile (all the objects it contains)

Select the data profile in the Profile Objects tab of the Data Profile Editor. In the Property Inspector, set the values of the configuration parameters. These parameters are set for all the objects in the data profile.

- An individual object in the data profile (for example, a table)

Select the object in the Profile Objects tab of the Data Profile Editor. In the Property Inspector, set the configuration parameters. These parameters are set for the selected object.

- An attribute within an object (for example, a column within a table)

In the Profile Objects tab of the Data Profile Editor, expand the object node to display the attributes it contains. For example, you can expand a table node to display its columns. Select the attribute for which you want to specify configuration parameters. In the Property Inspector, set the configuration parameters.

For example, if you know you only have one problematic column in a table and you already know that most of the records should conform to values within a certain domain, then you can focus your profiling resources on domain discovery



and analysis. By narrowing down the type of profiling necessary, you use less resources and obtain the results faster.

For more information about the configuration parameters you can set for data profiles, see "Configuration Parameters for Data Profiles" in the *Warehouse Builder Online Help*.

### 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 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 the data:

1. Expand the Data Profiles node in the Project Explorer, right-click a data profile, and select **Open Editor**.

The Data Profile Editor opens the selected data profile.

2. 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 in this dialog box. 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. Click **OK**.

Once the profiling job starts, the data profiling is asynchronous and you can continue working or even close the client. Your profiling process will continue to run until it is completed.

3. 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.

4. After the profiling is complete, the Retrieve Profile Results dialog box is displayed and you are prompted to refresh the results.

You can use this option if you have previously profiled data in the same data profile. It allows you to control when the new profiling results become visible in the Data Profile Editor.

---



---

**Note:** Data profiling results are overwritten on subsequent profiling executions.

---



---

### View 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 Editor**.

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 results of the selected object are displayed in the Profile Results Canvas. You can switch between objects. The tab that you had selected for the previous object remains selected.

The Profile Results Canvas contains the following tabs that display the results of data profiling:

- Data Profile
- Profile Object
- Aggregation
- Data Type
- Pattern
- Domain
- Unique Key
- Functional Dependency
- Referential
- Data Rule

For more information about the contents of these tabs, click the arrow on the right of the Profile Results Canvas panel and select **Help**.

## Derive Data Rules

Based on the results of data profiling, you can decide to derive data rules that can be used to clean up your 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 non-compliant data.

Although you can create data rules and apply them manually to your data profile, derived data rules allow you to move quickly and seamlessly between data profiling and data correction.

For example, if 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.

**To derive a data rule:**

1. Select a data profile in the navigation tree, right-click, and select **Open Editor**.  
The Data Profile Editor is displayed with the profiling results.
2. Review the profiling results and determine which findings you want derived into 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 you want derived into a data rule and then 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 opens and displays the Welcome page.
5. Click **Next**.  
The Name and Description page is displayed.
6. The **Name** field displays a default name for the data rule. You can either accept the default name or enter a new name.
7. Click **Next**.  
The Define Rule page is displayed.
8. Provide details about the data rule parameters.  
The Type field that represents the type of data rule is populated based on the tab from which you derived the data rule. 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 depends on the type of data rule.
9. Click **Next**.  
The Summary page is displayed. Review the options you set in the wizard using this page. Click **Back** if you want to change any of the selected values.
10. Click **Finish**.  
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 Project Explorer. You can reuse this data rule by attaching it to other data objects.

## Generate Corrections

After you have derived data rules from the 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 source data objects 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.

As part of the correction process, the following are created:

- Corrected tables that adhere to the newly derived data rules  
The correction tables have names that are prefixed with `TMP__`. For example, when you profile the `EMPLOYEES` table, the correction table will be called `TMP__EMPLOYEES`.
- Correction mappings that you use to cleanse the data  
The correction mappings enforce the data rules. While moving data from the old "dirty" tables in the profile source tables into the corrected tables, these mappings correct records that do not comply with the data rules. The name of the correction mapping is the object name prefixed with `M_`. For example, the correction mapping for the `EMPLOYEE` table is called `M_EMPLOYEE`.

### Steps to Create Corrections

Use the Data Profile Editor to create corrections based on the profiling results.

#### To create corrections:

1. If the Data Profile is not already open, open it by right-clicking the data profile in the Project Explorer and selecting **Open Editor**.

2. From the Profile menu, select **Create Correction**.

The Create Correction Wizard is displayed. Click **Help** on any wizard page for more information about the page.

3. On the Select Target Module page, specify the target module that will contain the corrections. You can create a new module or select an existing module.

If you choose to create a new target module, the Create Module Wizard is displayed that enables you to create a new module.

You can remove correction objects created as a result of previous corrections by selecting **Remove previous correction objects**.

4. On the Select Objects page, select the objects that you want to correct by moving them to the Selected list.

5. On the Select Data Rules and Data Types page, specify the corrections that you want to implement for each object. The navigation tree on the left displays the objects. Select each object in the navigation tree and specify corrections for that object on the Data Rules and Data Types tabs.

The Data Rules tab displays the data rules that will be applied to the corrected object. Select the rules to be applied. For each rule, in the Bindings section at the bottom of the page, select the column to which the rule must be bound.

The Data Types tab displays the new data type and the documented data type for each column. To use the new data type determined as a result of the data correction actions, select the column by clicking the box to the right of the column. Columns which are not selected will retain their existing data types.

6. On the Verify and Accept Corrected Tables page, select the objects that you want to correct.

On the top of this page, the objects selected for corrections are listed. Select **Create** to the right of the table name to generate corrected objects for the object.

The bottom part of this page contains the columns, Constraints, and Data Rules tabs. These tabs contain the definitions used for the corrected objects. You can make modifications to these tabs, if required.

7. On the Choose Data Correction Actions page, specify the correction actions to be taken for objects.

Select an object by clicking the **Correct** to the left of the object and use the Choose Data Correction Actions section to specify the correction action and cleansing strategy. For more information about correction actions, click **Help** on this page.

8. On the Summary page, click **Finish** to create the correction objects.

At this stage, the corrections objects are only defined and their metadata is stored in the workspace. To implement the correction objects in your target schema, you must deploy the correction tables and correction mappings.

Before you deploy a correction mapping, ensure that you do the following:

- Deploy the correction tables created as a result of data profiling.
- 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.

### Viewing the Correction Tables and Mappings

You can review the correction tables in the Data Object Editor to see the data rules and constraints created as part of the design of your table.

**To view the correction mappings:**

1. Double-click the table or mapping to open the object in their respective editors.
2. After the mapping is open, select **View** and then **Auto Layout** to view the entire mapping.
3. Select the submapping `ATTR_VALUE_1` and click the Visit Child Graph icon from the toolbar to view the submapping.

The submapping is displayed. The submapping is the element in the mapping that performs the actual correction cleansing you specified in the Create Correction Wizard.

## Define and Edit Data Rules Manually

Data rules can be derived or manually created. Before and after you have created the corrections, you can define additional data rules manually.

For more information about defining and editing data rules manually, see ["Creating Data Rules"](#) on page 5-42.

## Generate, Deploy, and Execute

Finally, you can generate, deploy, and execute the correction mappings and data rules. After you run the correction mappings with the data rules, your data is corrected. The derived data rules remain attached to the objects in the corrected schema for optional use in data monitors.

## Tuning the Data Profiling Process

Data profiling is a highly 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. For more information about configuration parameters, see "Configuration Parameters for Data Profiles" in the *Warehouse Builder Online Help*.

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.
- Analyze lesser amount of data  
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 be located on the same database instance as the profile workspace. You can do this by installing the workspace into the same Oracle 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 the 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, the following considerations are applicable:

- [Multiple Processors](#)
- [Memory](#)
- [I/O System](#)

### Multiple Processors

The computer that runs the Oracle Database needs multiple processors. Data profiling has been designed and tuned to take maximum advantage of the parallelism provided by the Oracle Database. While profiling large tables (more than 10 million rows), it is highly recommended to 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. It is recommended that the size of the System Global Area be at least 500 MB. If possible, configure it to 2 GB or 3 GB.

For advanced database users, it is recommended 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. Since today's CPUs can easily out-drive the I/O system, you must carefully design and configure the I/O subsystem. Keep in mind 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, 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. It is recommended 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:

$$\text{MAX}(1, \text{DB\_FILE\_MULTIBLOCK\_READ\_COUNT}/\text{number\_of\_disks}) \times \text{DB\_BLOCK\_SIZE}$$

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 a volume management software such as Veritas Volume Manager or Sun Storage Manager. If you are using Oracle Database 10g or a higher version and you do not have any volume management software, you can use the Automatic Storage Management feature of the 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. So you can consider placing these tablespaces on separate disks to reduce interference.

## 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, 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 reused in cleansing, and auditing.

### Using Data Rules in Data Cleansing and Schema Correction

Data rules can be used in two ways to cleanse data and correct schemas. The first way is to convert a source schema into a new target schema where the structure of the new tables strictly adheres to the data rules. The new tables would then have the right data types, constraints would be enforced, and schemas would be normalized. The second way 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. For more information about data auditors, see "[About Data Auditors](#)" on page 5-32.

## Creating Data Rules

The Data Rules folder in the Project Explorer contains the data rules. Every data rule must belong to a data rule folder. The subfolder DERIVED\_DATA\_RULES 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 the data rule should be created and select **New**.  
The Create Data Rule Wizard is displayed.
2. On the Name and Description page, enter a name and an optional description for the data rule. Click **Next**.
3. 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.

For more information about the types of rules, see "Types of Data Rules" in the *Warehouse Builder Online Help*.

4. On the Summary page, review the selections 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.



## Applying Data Rules to 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 Data 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 Project Explorer, right-click the object to which the data rule must be applied and select **Open Editor**.

The Data Object Editor for the data object is displayed.

2. Navigate to the Data Rules tab.

If any data rules are bound to the data object, these are displayed on this tab.

3. Click **Apply Rule**.

The Apply Data Rule wizard is displayed.

4. On the Select Rule page, select the data rule that you want to apply to the data object. Click **Next**.

5. On the Name and Description page, enter a name and an optional description for the applied data rule. Click **Next**.

6. On the Bind Rule Parameters page, bind the data rule to the column in the data object to which the data rule must be applied. Click **Next**.

7. On the Summary page, review the sections 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 folder.

## 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. You can create data auditors for tables, views, materialized views, and external tables.

**See Also:** ["About Data Auditors"](#) on page 5-32 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 monitor.

See ["Creating Data Auditors"](#) on page 5-44

2. Run the data auditor to identify the 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 5-45 for information about running data auditors.

---

---

**Note:** You cannot import metadata for data auditors in Merge mode. For more information about import mode options, see "Import Option" in the *Warehouse Builder Online Help*.

---

---

## 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**.  
The Create Data Auditor Wizard is displayed.
3. On the Name and Description page, enter a name and an optional description for the data auditor. Click **Next**.
4. On the Select Objects page, select the data objects that you want to audit. Use the shuttle buttons to move objects to the Selected section and click **Next**.  
You can select multiple objects by holding 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 data rules bound to the selected objects. Click **Next**.

The Choose Actions page contains two sections, Error threshold mode and Data Rules.

### Error threshold mode

Error threshold mode is used to determine the compliance of data to data rules in the objects. Select one of the following options:

- **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, no failures are allowed. This value is ignored if you select Percent in the Error threshold mode section.

6. On the Summary page, review the selections you made. Click **Back** to modify any selected values. Click **Finish** to create the data auditor.

The created 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.

## 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](#)
- [Automatically Running Data Auditors](#)

### 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 5-46.

### Automatically Running Data Auditors

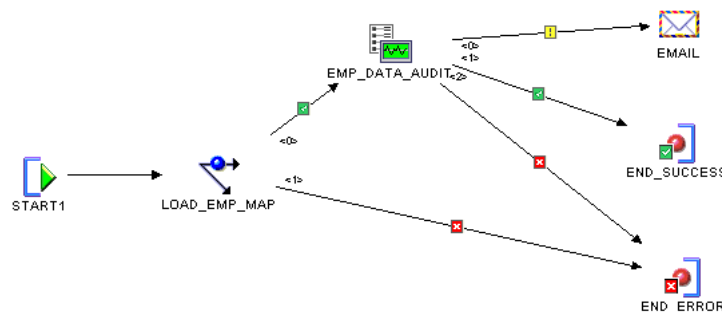
You can automate the process of running a data auditor using the following steps:

1. Create a process flow that contains a Data Auditor Monitor activity.
2. Schedule this process flow to run at a predefined time.

For more information about scheduling objects, see "[Process for Defining and Using Schedules](#)" on page 11-17.

[Figure 5-5](#) 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, 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 5-5 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. For more information about deployment preferences, see ["Deployment Preferences"](#) on page 3-4.

[Figure 5-6](#) displays the Execution Results tab of the Job Details window.

**Figure 5-6 Data Auditor Execution Results**

	Inserted	Updated	Deleted	Merged
"NEW_EMP_DA":E_NOT_NULL (E_NOT_NULL)	1	0	0	0

**Output Parameters**

AUDIT\_RESULT

EO\_NEW\_EMP\_NOT\_NULL

SO\_NEW\_EMP\_NOT\_NULL

The Input Parameters tab contains the values of input parameters used to run the data auditor. 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, multi-table insert may be used in the data auditor. In this case, the count of the number of rows will not be accurate.

In [Figure 5-6](#), 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 5-44.

- **EO\_<data\_rule\_name>:** Represents the calculated error quality for the specified data rule. 0 indicates all errors and 100 indicates no errors.
- **SO\_<data\_rule\_name>:** Represents the Six Sigma quality calculated for the specified data rule.

---

---

## Designing Target Schemas

Warehouse Builder is also a design tool that enables you to design your data warehouse. Target schemas contain all the necessary data objects in your data warehouse such as tables, views, dimensions, and. In a traditional data warehousing implementation, there is typically only one target schema, which is the data warehouse target. You can design target schemas, both relational and dimensional, using the Data Object Editor.

This chapter includes the following topics:

- [About Data Objects](#)
- [About the Data Object Editor](#)
- [About Dimensional Objects](#)
- [About Dimensions](#)
- [About Slowly Changing Dimensions](#)
- [About Time Dimensions](#)
- [About Cubes](#)
- [Designing the Target Schema](#)
- [Creating Oracle Data Objects](#)
- [Configuring Data Objects](#)
- [Validating Data Objects](#)
- [Generating Data Objects](#)
- [Deriving Business Intelligence Metadata](#)

### About Data Objects

The Oracle module contains nodes for each type of data object that you can define in Warehouse Builder. In the Project Explorer, under the Oracle node, expand the module node to view all the supported data objects.

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, and sequences. You can also create optional structures associated with relational objects such as constraints, indexes, partitions, and attribute sets. For more information about these structures, refer to the online help.

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. This chapter provides specific information about each type of dimensional object and how they are used in Warehouse Builder.

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 Project Explorer. 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. For more information about intelligence objects, see "Defining Business Intelligence Objects" in the *Warehouse Builder Online Help*.

[Table 6–1](#) describes the types of data objects you can use in Warehouse Builder.

**Table 6–1 Data Objects in Warehouse Builder**

Data Object	Type	Description
Tables	Relational	<p>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 a table's data, integrity constraints can be defined for a table.</p> <p>See "Using Tables" in the <i>Warehouse Builder Online Help</i> for more information.</p>
External Tables	Relational	<p>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.</p> <p>See "Using External Tables" in the <i>Warehouse Builder Online Help</i> for more information.</p>
Views	Relational	<p>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.</p> <p>See "Using Views" in the <i>Warehouse Builder Online Help</i> for more information.</p>
Materialized Views	Relational	<p>Materialized views are pre-computed 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.</p> <p>See "Using Materialized Views" in the <i>Warehouse Builder Online Help</i> for more information.</p>
Sequences	Relational	<p>Sequences are database objects that generate lists of unique numbers. You can use sequences to generate unique surrogate key values.</p> <p>See "Using Sequences" in the <i>Warehouse Builder Online Help</i> for more information.</p>
Dimensions	Dimensional	<p>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.</p> <p>See "<a href="#">About Dimensions</a>" on page 6-16 for more information.</p>

**Table 6–1 (Cont.) Data Objects in Warehouse Builder**

Data Object	Type	Description
Cubes	Dimensional	Cubes contain measures and links to one or more dimension tables. They are also known as facts.  See " <a href="#">About Cubes</a> " on page 6-35 for more information.
Advanced Queues	Relational	Advanced Queues enable message management and communication required for application integration.  Currently, you cannot create advanced queues using Warehouse Builder. You can only import advanced queues that were exported into an .mdl file using a previous version of the product.
Queue Tables	Relational	Queue tables are tables that store queues. Each queue table contains a payload whose data type can be an object type or RAW.  You cannot create a queue table using Warehouse Builder. A queue table is imported as part of an advanced queue payload.
Object Types	Relational	An object type is made up of one or more user-defined types or scalar types.  See " <a href="#">About Object Types</a> " in the <i>Warehouse Builder Online Help</i> for more information.
Varrays	Relational	A varray is an ordered collection of elements.  See " <a href="#">About Varrays</a> " in the <i>Warehouse Builder Online Help</i> 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 the one object.  See " <a href="#">About Nested Tables</a> " in the <i>Warehouse Builder Online Help</i> for more information.

## Supported Data Types

[Table 6–2](#) displays the data types you can use to create and edit columns.

**Table 6–2 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 <code>d</code> . For example, <code>3.0235d</code> .
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 <code>f</code> . For example, <code>2.07f</code> .
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 four gigabytes.
CHAR	Stores fixed-length character data to a maximum size of 4000 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.

**Table 6–2 (Cont.) Data Types**

<b>Data Type</b>	<b>Description</b>
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, which 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 <code>SYSDATE</code> returns the current date and time.
FLOAT	Stores a single-precision, floating-point, number. <code>FLOAT</code> can be loaded with correct results only between systems where the representation of a <code>FLOAT</code> is compatible and of the same length.
INTEGER	A <code>NUMBER</code> 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 <code>LONG</code> data type is like the <code>VARCHAR2</code> data type, except that the maximum length of a <code>LONG</code> value is 2147483647 bytes (two gigabytes).
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 <code>MDSYS.SDO_DIM_ELEMENT</code> .
MDSYS.SDO_DIM_ELEMENT	Stores the dimension name, lower boundary, upper boundary and tolerance.
MDSYS.SDO_ELEM_INFO_ARRAY	Stores an array of type <code>MDSYS.SDO_ORDINATE_ARRAY</code> .
MDSYS.SDO_GEOMETRY	Stores Geographical Information System (GIS) or spatial data in the database. For more information, refer to the <i>Oracle Spatial Users Guide and Reference</i> .
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.
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 <code>NCHAR</code> data in the database, in-line or out-of-line.



**Table 6–2 (Cont.) Data Types**

<b>Data Type</b>	<b>Description</b>
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.
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.LCR\$_ROW_RECORD	This type 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 initialization parameter NLS_TIMESTAMP_FORMAT.
TIMESTAMP WITH LOCAL TIMEZONE	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 TIMEZONE	Extends the data type TIMESTAMP 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.
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.
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 4000 characters.
XMLFORMAT	This is an object type that is used to specify formatting arguments for SYS_XMLGEN() and SYS_XMLAGG() functions.

**Table 6–2 (Cont.) Data Types**

Data Type	Description
XMLTYPE	An Oracle-supplied type that can be used to store and query XML data in the database. It has member functions you can use to access, extract, and query the XML data using XPath expressions. XPath is another standard developed by the W3C committee to traverse XML documents.

## Naming Conventions for Data Objects

The rules for naming data objects depend on the naming mode you set for Warehouse Builder in the [Naming Preferences](#) section of the Preferences dialog box. 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. See ["Naming Preferences"](#) on page 3-6 for details on how to specify a naming mode.

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

### Naming Data Objects

In the physical naming mode, the name can be from 1 to 30 alphanumeric characters. Blank spaces are not allowed. Do not use any of the reserved words as a name of an object.

In the business naming mode, the limit is 200 characters. The name should be unique across the object category that owns the object. For example, since all tables belong to a module, table names should be unique across the module to which they belong. Similarly, module names should 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 2 and 2,000 alphanumeric characters and can contain blank spaces. Specifying a description for a data object is optional.

## About the Data Object Editor

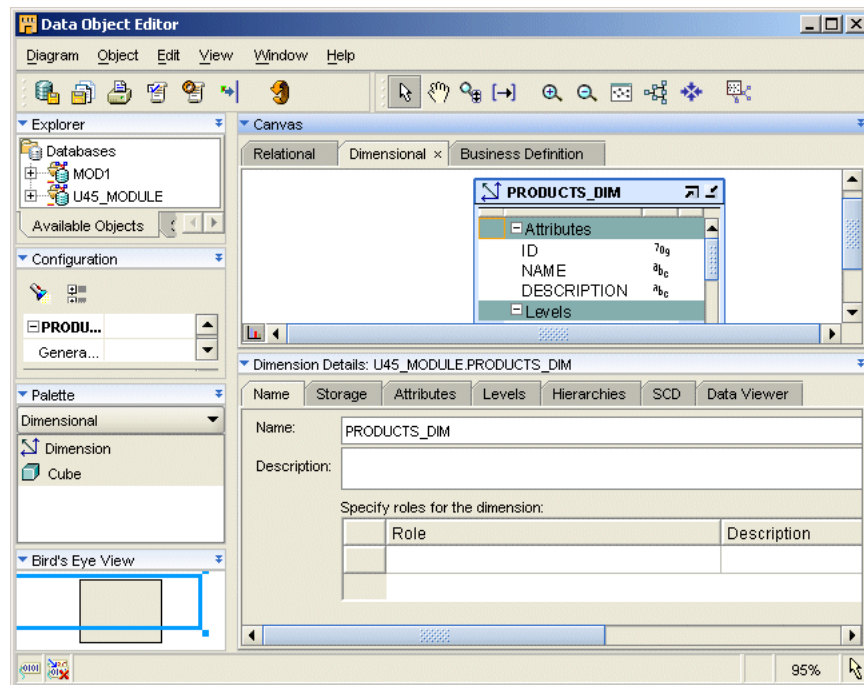
The Data Object Editor provides a centralized interface to create, edit, configure, validate, and deploy Oracle data objects. You can use the Data Object Editor with relational, dimensional, and business intelligence objects. You can also view the data stored in these objects.

The Data Object Editor enables you to build your warehouse schema designs. It also provides an intuitive user interface that supports fast entry of design details. The Data Object Editor contains a menu bar, multiple toolbars, and multiple panels. All the panels are dockable. You can resize the panels or relocate them anywhere in the editor window. You can also choose to display or hide any of the panels. For more information about the Data Object Editor components, refer to the online help.

To relocate a panel, hold down the mouse button on the panel title, drag to the new location and release the mouse button. Resize a panel by placing your mouse on the panel border, pressing the mouse button when the double sided arrow appears, and dragging your mouse to indicate the desired size.

[Figure 6–1](#) displays the Data Object Editor.

Figure 6–1 Data Object Editor Window



Use the Data Object Editor to:

- Create, edit, and delete relational and dimensional objects.
- Create, edit, and delete the following business intelligence objects: Business Areas and Item Folders.
- Define relationships between Oracle data objects.
- Validate, generate, and deploy Oracle data objects.
- Define and edit all aspects of a data object such as its columns, constraints, indexes, partitions, data rules, and attribute sets.
- View impact analysis and lineage information for a data object.
- Define implementation details for dimensional objects with a relational implementation.
- View the data stored in a data object.

### Starting the Data Object Editor

Use one of the following methods to start the Data Object Editor:

- Select a data object in the Project Explorer. From the Design Center menu select **Edit**, then **Open Editor**.
- Right-click a data object in the Project Explorer and select **Open Editor**.
- Double-click a data object in the Project Explorer.

## Data Viewer

The Data Viewer enables you to view the data stored in the data object. For example, the data viewer for a table enables you to view the table data. You can access the Data Viewer using one of the following methods:

- From the Project Explorer, right-click a data object and select **Data**.
- In the Data Object Editor for the data object, navigate to the Data Viewer tab of the Details panel. Click **Execute Query**.

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 Object Editor are taken directly from the location in which the actual table is deployed. If the table definition in the Data Viewer does not match with what you see in the Data Object Editor, it is because the changes you made in the editor have not yet been deployed.

## Using the Data Object Editor to Create Data Objects

Use the Data Object Editor to create relational, dimensional, and certain business intelligence objects. There are multiple methods of creating data objects using the Data Object Editor.

Use one of the following editor components to create a data object:

- Menu Bar  
See [Creating Data Objects Using the Menu Bar](#) on page 6-8.
- Canvas  
See [Creating a Data Object Using the Canvas](#) on page 6-9.
- Data Object Editor Palette  
See [Creating a Data Object Using the Data Object Editor Palette](#) on page 6-9.

### Creating Data Objects Using the Menu Bar

To create a data object using the menu bar:

1. If it is not already open, open the Data Object Editor.
2. Navigate to the tab that corresponds to the type of data object that you want to create.

For example, to create a table, select the Relational tab. To create a business area, select the Business Intelligence tab. To create dimensions and cube, select the Dimensional tab.

3. From the **Diagram** menu, select **Add**, then select the type of data object to create.

Warehouse Builder displays the Add a New or Existing <Object> dialog box. For more information about this dialog box, click **Help**.

Notice that the list of data objects in the Add menu contains some disabled items. Only the data objects that you can create from the current editor context are enabled.

4. Select the **Create a new <object>** option.  
For example, to add a table, select the **Create a new Table** option.
5. Specify the name of the data object using the New <Object> Name field.  
The New <Object> Name field displays a default name for the object. You can choose to retain this default name or specify a different name.
6. Click **OK**.  
Warehouse Builder adds a node for the new data object to the canvas.
7. Use the tabs of the Details panel to define the data object.

### Creating a Data Object Using the Canvas

To create a data object using the canvas:

1. If it is not already open, open the Data Object Editor.
2. Navigate to the tab that corresponds to the type of data object that you want to create.  
For example, to create a materialized view, select the Relational tab. To create a dimension, select the Dimensional tab.
3. Right-click whitespace (blank area) on the canvas.  
Warehouse Builder displays a shortcut menu containing the types of data objects you can create.
4. Select the option corresponding to the type of object you want to create.  
For example, to create a materialized view, select the **Add a Materialized View** option.  
Warehouse Builder displays the Add a New or Existing <Object> dialog box. For more information about this dialog box, click **Help**.
5. Select the **Create a new <object>** option.  
For example, to add a cube, select the **Create a new Cube** option.
6. Specify the name of the data object using the New <Object> Name field.  
The New <Object> Name field displays a default name for the object. You can choose to retain this default name or specify a different name.
7. Click **OK**.  
Warehouse Builder adds a node for the new data object to the canvas.
8. Use the tabs of the Details panel to define the data object.

### Creating a Data Object Using the Data Object Editor Palette

To create a data object using the Palette:

1. If it is not already open, open the Data Object Editor.
2. Navigate to the tab that corresponds to the type of data object that you want to create.  
For example, to create a view, select the Relational tab. To create a cube, select the Dimensional tab.
3. Drag and drop the operator that corresponds to the type of object that you want to create on to the canvas.

For example, to create a view, drag and drop the View operator from the palette on to the canvas.

Warehouse Builder displays the Add a New or Existing <Object> dialog box. For more information about this dialog box, click **Help**.

4. Select the **Create a new <object>** option.

For example, to add a cube, select the **Create a new Cube** option.

5. Specify the name of the data object using the New <Object> Name field.

The New <Object> Name field displays a default name for the object. You can choose to retain this default name or specify a different name.

6. Click **OK**.

Warehouse Builder adds a node for the new data object to the canvas.

7. Use the tabs of the Details panel to define the data object.

## About Dimensional Objects

This section describes the basic concepts related to dimensional objects. If you are familiar with dimensional objects concepts and the types of implementations for dimensional objects in Warehouse Builder, skip the next few sections and continue with "[Designing the Target Schema](#)" on page 6-39.

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, the word 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.

### Design versus Implementation

Warehouse Builder separates the logical design of dimensional objects from their storage. The logical design (business rules) allow you to focus on the structure and the content of the dimensional object first. You can then choose a relational, ROLAP, or MOLAP implementation for the dimensional object.

ROLAP and relational implementations store the dimensional object in a relational schema in the database.

A MOLAP implementation stores the dimensional object in analytic workspaces in the database.

Warehouse Builder enables you to use the same metadata to create and manage both your relational and multidimensional data stores. Separating the design from the implementation has the following advantages:

- Implementation is easier, because you first design and then implement.
- ETL is transparent as it is always the same for any type of implementation.

### Uses of OLAP

Business organizations typically have complex analytic, forecast, and planning requirements. Analytic Business Intelligence (BI) applications provide solutions by answering critical business questions using the data available in your database.

Dimensional objects provide complex analytic power to your data warehouse. After you load data into dimensional objects, you can use tools and applications to run complex analytical queries that answer your business questions. These analytic queries include time-series analysis, inter-row calculations, access to aggregated historical and current data, and forecasts. Multidimensional objects are more effective in answering these types of queries quickly.

### About Creating Dimensional Objects

Creating dimensional objects consists of four high-level tasks:

1. [Defining Dimensional Objects](#)
2. [Implementing Dimensional Objects](#)
3. [Deploying Dimensional Objects](#)
4. [Loading Dimensional Objects](#)

## Defining Dimensional Objects

When you define dimensional objects, you describe 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.

Warehouse Builder provides the following two methods to define dimensional objects:

- **Wizards:** Use wizards to create dimensional objects easily. The wizard creates a fully functional dimensional object along with the implementation objects that store the dimensional object data. Many options are defaulted to the most common settings. You can change these settings later using the editors.

You use the Create Dimension Wizard to create dimensions, the Create Time Dimension Wizard to create time dimensions, and the Create Cube Wizard to create cubes.

- **Editors:** Use editors to create or edit dimensional objects. Use editors to create a dimensional object when you want to specify settings that are different from the default settings used by the wizards. Also use editors to create dimensional objects that use certain advanced options that are not available when you use wizards. For example, to create a relational dimension that uses a snowflake schema implementation, you must use the editor. When you use the wizard, the default implementation method used is the star schema. However, you can edit a dimension that you created using the Create Dimension Wizard and modify it to use a snowflake schema implementation.

## 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 10g Enterprise Edition with the OLAP option
  - OLAP 10.1.0.4 or higher
- 
- 

You set the Deployment Option configuration property to specify the type of implementation for a dimensional object. For more information on setting this property, see "Configuring Dimensions" and "Configuring Cubes" in the *Warehouse Builder Online Help*.

### 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.

When you use the wizard to define dimensional objects, Warehouse Builder creates the database tables that store the dimensional object data. When you define a dimensional object using the Data Object Editor, 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. The following section on binding describes how you specify the relationship between the dimensional object and its implementation objects.

For a relational implementation, you cannot use the Data Viewer to view the data stored in the dimensional object. You can however 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 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
- Manual binding

**Auto Binding** In auto binding, Warehouse Builder creates 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. You can perform auto binding using both the wizards and the editors.

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. For more information on these options, see "[Relational and ROLAP Implementation of a Dimension](#)" on page 6-22.

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 Project Explorer, right-click the dimensional object and select **Open Editor**. The Data Object Editor for this dimensional object is displayed.
2. On the Dimensional tab, right-click the dimensional object node and select **Bind**. Alternatively, select the dimensional object node on the canvas and from the **Object** menu select **Bind**.

If the Bind option is not enabled, 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.

**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.

**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 Project Explorer, right-click the dimensional and select **Open Editor**.

The Data Object Editor for the dimensional object is displayed. On the canvas, the Dimensional tab is active.

3. Right-click the dimensional object and select **Detail View**.

Warehouse Builder opens a new tab that has the same name as the dimensional object.

4. From the palette, drag and drop the operator that represents the implementation object onto the canvas.

Warehouse Builder displays the Add a New or Existing <Object> dialog box. For example, if the dimension data is stored in a table, drag a Table operator from the Palette and drop it onto the canvas. The Add a New or Existing Table dialog box is displayed.

5. Choose the **Select an existing <Object>** option and then select the data object from the list of objects displayed in the selection tree.

6. Click **OK**.

A node representing the object that you just added is displayed on the canvas.

7. For dimensions, if more than one data object is used to store the dimension data, perform steps 4 to 6 for each data implementation object.

8. 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 6-20, 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, right-click the dimensional object on the Relational tab of the Canvas and select **Unbind**. Unbinding removes the bindings between the dimensional object and its implementation objects. However, it does not modify 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. In addition to creating DDL scripts that can be deployed to a database, a ROLAP implementation enables you to create CWM2 metadata for the dimensional object in the OLAP catalog.

### MOLAP Implementation of Dimensional Objects

In a MOLAP implementation, the dimensional object data is stored in an analytic workspace in Oracle Database 10g. This analytic workspace, in turn, is stored in the database.

**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*.

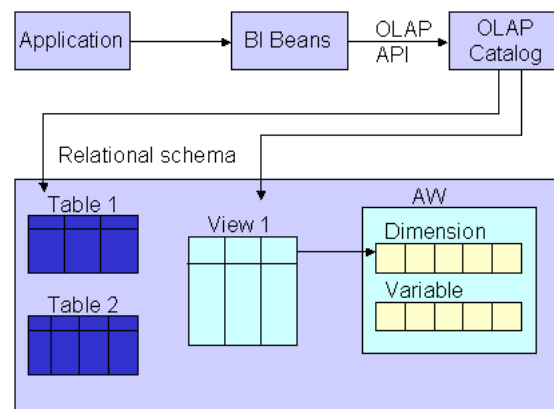
**OLAP Catalog** The OLAP catalog is the metadata repository provided for the OLAP option in the Oracle Database. This metadata describes the data stored in both relational tables and in analytic workspaces.

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\_OLAP2\_AW).

In Oracle Database 10g, 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.

[Figure 6-2](#) describes how the OLAP catalog enables applications to access data stored in relational tables and analytic workspaces.

**Figure 6–2 Using the OLAP Catalog to Access Dimensional Objects**

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.

## Deploying Dimensional Objects

To instantiate the dimensional objects in the database, you must deploy them. To specify the type of implementation for dimensional objects, you set the deployment option. The configuration parameter **Deployment Options** enables you to set the deployment option.

Warehouse Builder provides the following deployment options for dimensional objects.

- **Deploy All:** For a relational or ROLAP implementation, the dimensional object is deployed to the database and a CWM definition to the OLAP catalog. For a MOLAP implementation, the dimensional object is deployed to the analytic workspace.
- **Deploy Data Objects Only:** Deploys the dimensional object only to the database. You can select this option only for dimensional objects that use 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 dimensional object 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 re-deploying the data objects again.
- **Deploy Aggregation:** Deploys the aggregations defined on the cube measures. This option is available only for cubes.

### Deploying Dimensional Objects that Use a MOLAP Implementation

Dimensional objects that use a MOLAP implementation can be deployed just after you define them. You can use the Design Center or the Control Center Manager to deploy a dimensional object.

### Deploying Dimensional Objects that Use a Relational or ROLAP Implementation

Before you deploy a relational or ROLAP dimensional object, ensure that the implementation details are specified. This means that the dimensional object should be

bound to its implementation objects. Also ensure that the dimensional object is valid. For more information on implementing dimensional objects, see "[Relational Implementation of Dimensional Objects](#)" on page 6-12. For more information on performing binding, see "[Binding](#)" on page 6-12.

After you perform binding, deploy the dimensional object. Before you deploy a dimensional object, ensure that all its implementation objects are deployed. For a dimension, this includes the sequence that is used to generate the surrogate identifier of the dimension levels. Alternatively, you can deploy the implementation objects together with the dimensional object.

## Loading Dimensional Objects

After you deploy a dimensional object, you load data into it by creating a mapping. Use the Mapping Editor to create the mapping that loads data from the source objects into the dimensional object. You then deploy and execute this mapping.

For more information on loading dimensions, see "Dimension Operator as a Target" in the *Warehouse Builder Online Help*. For information on loading cubes, see "Cube Operator" in the *Warehouse Builder Online Help*.

## About 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.

Typical relational dimension tables have the following characteristics:

- A single column primary key populated with values called warehouse keys.  
Warehouse keys that provide administrative control over the dimension, support techniques that preserve dimension history, and reduce the size of cubes.
- One or more hierarchies that are explicitly defined as dimension objects.  
Hierarchies maximize the number of query rewrites by the Oracle server.

## Rules for Dimension Objects

When you create a dimension object using Warehouse Builder, the dimension must conform to the following rules:

- A dimension must have a surrogate identifier and a business identifier.
- The surrogate identifier can consist of only one attribute. However, the business identifier can consist of more than one attribute.
- Every dimension level must have at least one attribute.
- A dimension attribute can be either a surrogate identifier, a business identifier, a parent identifier, or a regular attribute.
- A regular attribute can also play only one of the following roles at a time: effective date, expiration date, or triggering attribute.

- A dimension that uses a relational or ROLAP implementation must have at least one level.
- Any database table or view that implements a dimension that uses a relational or ROLAP implementation must have only one LONG, LONG RAW, or NCLOB column.
- For a dimension that uses a relational or ROLAP implementation, all level attributes must bind to database tables or views only.
- A dimension that uses a relational or ROLAP implementation must be associated with a sequence that is used to load the dimension key attribute.
- The dimension key attribute of a dimension that uses a relational or ROLAP implementation must bind to the primary key of a table.
- A Type 2 SCD must have the effective date, expiration date, and at least one triggering attribute.
- A Type 3 SCD must have the effective date and at least one triggering attribute.

## 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 property View Name of the dimension or cube.

- Skip-level hierarchies and ragged hierarchy metadata not deployed to the OLAP catalog.

If you create a dimension that contains skip-level or ragged hierarchies, 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.

## Defining a Dimension

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

- Hierarchies

### Defining Dimension Attributes

A dimension attribute is a descriptive characteristic of a dimension member. It has a name and a data type. A dimension attribute is applicable to one or more levels in the dimension. They are implemented as level attributes to store data.

In Warehouse Builder, you define dimension attributes when you define a dimension. The list of dimension attributes must include all the attributes that you may need for any of the levels in the dimension. Dimension attributes are the only attributes that are visible in Discoverer and other OLAP tools.

For example, the Products dimension has a dimension attribute called Description. This attribute is applicable to all the levels Total, Groups, and Products and stores the description for each of the members of these levels.

### Defining Levels

The levels in a dimension represent the level of aggregation of data. A dimension must contain at least one level, except in the case of a dimension that contains a value-based hierarchy. Every level must have level attributes and a level identifier.

For example, the dimension Products can have the following levels: Total, Groups, and Product.

### Surrogate, Business, and Parent Identifiers

Every level must have two identifiers: a surrogate identifier and a business identifier. When you create a dimension, each level must implement the dimension attributes marked as the surrogate identifier and business identifier (attributes, in the case of a composite business identifier) of the dimension.

**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. Because the value of the surrogate identifier must be unique across all dimension levels, you use the same sequence to generate the surrogate identifier of all the dimension levels.

For a relational implementation, the surrogate identifier serves the following purposes:

- If a child level is stored in a different table from the parent level, each child level record stores the surrogate identifier of the parent record.
- In a fact table, each cube record stores only the surrogate identifier of the dimension record to which it refers. By storing the surrogate identifier, the size of the fact table that implements the cube is reduced.

**Business Identifiers** A business identifier consists of a user-selected list of attributes. The business identifier must be unique across the level and is always derived from the natural key of the data source. The business identifier uniquely identifies the member. For example, the business identifier of a Product level can be its Universal Product Code (UPC), which is a unique code for each product.

---



---

**Note:** For a dimension that has a MOLAP implementation, the business identifier can consist of only one attribute.

---



---

The business identifier does the following:

- Identifies a record in business terms.
- Provides a logical link between the fact and the dimension or between two levels.
- Enables the lookup of a surrogate key.

When you populate a child level in a dimension, you must specify the business identifier of its parent level. When you populate a cube, you must specify the business identifier of the dimension level to which the cube refers.

**Parent Identifier** A parent identifier is used to annotate the parent reference in a value-based hierarchy. For more information on value-based hierarchies, see "[Value-based Hierarchies](#)" on page 6-21.

For example, an EMPLOYEE dimension with a value-based hierarchy, has the following dimension attributes: ID, FIRST\_NAME, LAST\_NAME, EMAIL, PHONE, JOB\_ID, HIRE\_DATE, and MANAGER\_ID. In this dimension, ID is the surrogate identifier and MANAGER\_ID is the parent identifier.

### Defining Level Attributes

A level attribute is a descriptive characteristic of a level member. Each level in the dimension has a set of level attributes. To define level attributes, you select the dimension attributes that the level will implement. A level attribute has a distinct name and a data type. The data type is inherited from the dimension attribute that the level attribute implements. The name of the level attribute can be modified to be different from that of the dimension attribute that it implements.

Every level must implement the attribute marked as the surrogate identifier and the business identifier in the set of the dimension attributes.

### Defining Hierarchies

A dimension hierarchy is a logical structure that uses ordered levels or a set of data values (for a value-based hierarchy) as a means of organizing data. A hierarchy describes parent-child relationships among a set of levels. A level-based hierarchy must have at least one level. A level can be part of more than one hierarchy.

For example, the Time dimension can have the following two hierarchies:

Fiscal Hierarchy: Fiscal Year > Fiscal Quarter > Fiscal Month > Fiscal Week > Day

Calendar Hierarchy: Calendar Year > Calendar Quarter > Calendar Month > Day

All hierarchies must be strict 1:n relationships. One record in a parent level corresponds to multiple records in a child level. But one record in a child level corresponds to only one parent record within a hierarchy.

### Dimension Roles

A dimension role is an alias for a dimension. In a data warehouse, a cube can refer to the same dimension multiple times, without requiring the dimension to be stored multiple times. Multiple references to the same dimension may cause confusion. So you create an alias for each reference to the dimension, thus allowing the joins to be

instantly understandable. In such cases, the same dimension performs different dimension roles in the cube.

For example, a sales record can have the following three time values:

- Time the order is booked
- Time the order is shipped
- Time the order is fulfilled

Instead of creating three time dimensions and populating them with data, you can use dimension roles. Model one time dimension and create the following three roles for the time dimension: order booked time, order shipped time, and order fulfillment time. The sales cube can refer to the order time, ship time, and fulfillment time dimensions.

When the dimension is stored in the database, only one dimension is created and each dimension role references this dimension. But when the dimension is stored in the OLAP catalog, Warehouse Builder creates a dimension for each dimension role. Thus, if a time dimension has three roles, three dimensions are created in the OLAP catalog. However, all three dimensions are mapped to the same underlying table. This is a workaround because the OLAP catalog does not support dimension roles.

---



---

**Note:** Dimension roles can be created for dimensions that have a relational implementation only.

---



---

### Level Relationships

A level relationship is an association between levels in a dimension hierarchy. Level relationships are implemented using level attributes that store the reference to the parent level in the hierarchy.

For example, the Products dimension has the following hierarchy: Total > Groups > Product. Warehouse Builder creates two level relationships: Product to Groups and Groups to Total. Two new attributes implement this level relationship: one in the Product level and one in the Groups level. These attributes store the surrogate ID of the parent level.

### Dimension Example

An example of a dimension is the Products dimension that you use to organize product data. [Table 6-3](#) lists the levels in the PRODUCTS dimension and the surrogate identifier and business identifier for each of the levels in the dimension.

**Table 6-3 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



**Table 6–3 (Cont.) Products Dimension Level Details**

Level	Attribute Name	Identifier
	Name	
	Description	
	Package Type	
	Package Size	

The PRODUCTS dimension contains the following hierarchy:

Hierarchy 1: Total > Groups > Product

### Control Rows

Warehouse Builder creates control rows that 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.

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 6-20. 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 6–4](#). 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 6–4 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	

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.

### Value-based Hierarchies

A value-based hierarchy is a dimension in which hierarchical relationships are defined by a parent dimension attribute and a child dimension attribute. This is different from a level-based hierarchy, referred to as a hierarchy in this chapter, in which the hierarchical relationships are defined between levels.

You create a value-based hierarchy when the parent-child relationships cannot be grouped into meaningful levels. A value-based hierarchy has no levels. When you

create the dimension attributes, you must specify which dimension attribute is the parent attribute.

For example, consider an EMPLOYEE dimension that has the following dimension attributes: ID, FIRST\_NAME, LAST\_NAME, EMAIL, PHONE, JOB\_ID, HIRE\_DATE, DESCRIPTION, and MANAGER\_ID. This dimension contains a parent-child relationship in which the MANAGER\_ID attribute identifies the manager of each employee. But these relationships may not form meaningful levels across the organization. This is because the number of levels between an employee and the CEO is not the same for all employees. There may be four levels between employee A and the CEO, whereas, there may be six levels between employee B and the CEO. In such cases, you create a value-based hierarchy with MANAGER\_ID as the parent identifier.

You can create value-based hierarchies using the Data Object Editor only. For more information about specifying a parent attribute, see "Attributes Tab" in the *Warehouse Builder Online Help*.

---

---

**Note:** Value-based hierarchies can be created only in dimensions that use a MOLAP implementation.

---

---

## Implementing a Dimension

Implementing a dimension consists of specifying how the dimension and its data are physically stored. You can choose either a relational implementation, ROLAP implementation, or MOLAP implementation for a dimension. For more information about setting the implementation method, see "[Implementing Dimensional Objects](#)" on page 6-11.

### Relational and ROLAP Implementation of a Dimension

When you store dimension data in a relational form, you can implement the dimension using one of the following methods:

- [Star Schema](#)
- [Snowflake Schema](#)

**Star Schema** In a star schema implementation, Warehouse Builder stores the dimension data in a single table. Because the same table or view stores data for more than one dimension level, you must specify a dimension key column in the table. The dimension key column is the primary key for the dimension. This column also forms the foreign key reference to the cube.

Each level implements a subset of dimension attributes. By default, the level attribute name is the same as the dimension attribute name. To avoid name conflicts caused by all level data being stored in the same table, Warehouse Builder uses the following guidelines for naming in a star table:

- If the level attribute name is not unique, Warehouse Builder prefixes it with the name of the level.
- If the level attribute name is unique, Warehouse Builder does not use any prefix.

---

---

**Note:** To ensure that no prefixes are used, you must explicitly change the level attribute name in the Create Dimension wizard or the Data Object Editor.

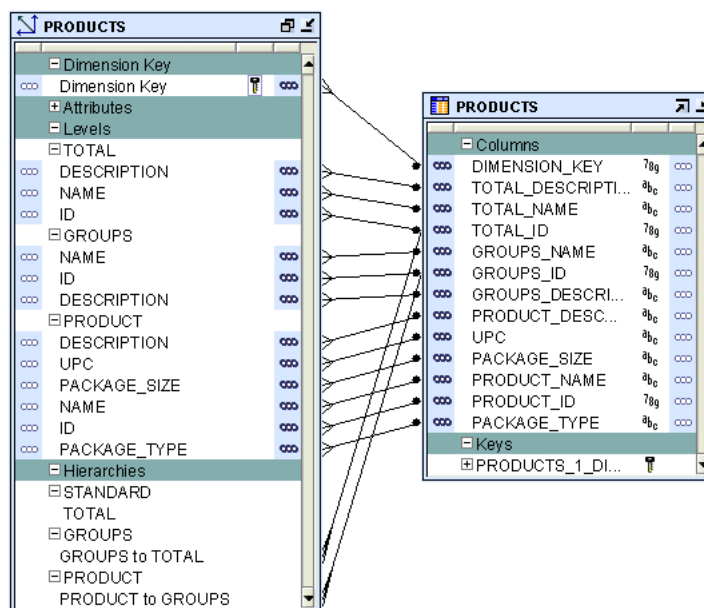
---

---

For example, if you implement the Products dimension using a star schema, Warehouse Builder uses a single table to implement all the levels in the dimension.

Figure 6–3 displays the star schema implementation of the Products dimension. The attributes in all the levels are mapped to different columns in a single table called PRODUCTS. The column called DIMENSION\_KEY stores the surrogate ID for the dimension and is the primary key of the table.

**Figure 6–3 Star Schema Implementation of Products Dimension**

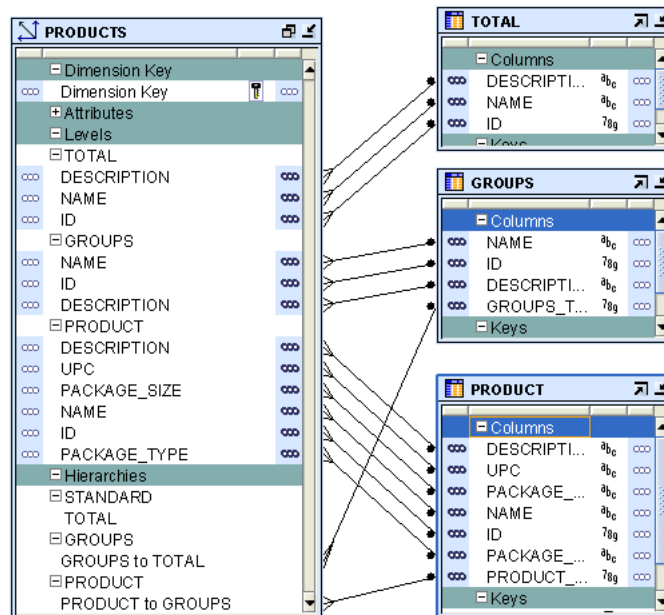


For relational or ROLAP dimensions that use a star implementation, you can bind attributes from more than one levels to the same database column. A database column that is bound to attributes from more than one dimension levels is referred to as a *shared column*. For a Type 2 Slowly Changing Dimension (SCD), you cannot set the level attributes that are bound to a shared column as triggering attributes.

**Snowflake Schema** In a snowflake schema implementation, Warehouse Builder uses more than one table to store the dimension data. Separate database tables or views store the data pertaining to each level in the dimension.

Figure 6–4 displays the snowflake implementation of the PRODUCTS dimension. Each level in the dimension is mapped to a different table.

**Figure 6–4 Snowflake Schema Implementation of the Products Dimension**



## Binding

When you perform binding, you specify the database columns that will store the data of each attribute and level relationship in the dimension. You can perform either auto binding or manual binding for a dimension. For more information about binding, see ["Binding"](#) on page 6-12.

**Auto Binding** When you perform auto binding, Warehouse Builder binds the dimension object attributes 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.

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, Warehouse Builder rebinds the dimensional object to the existing implementation objects. The implementation method can be either Star or Snowflake. For more information on implementation methods, see ["Relational and ROLAP Implementation of a Dimension"](#) on page 6-22.

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, Warehouse Builder deletes the old implementation objects and creates a new set of implementation tables. If you want to retain the old implementation objects, you must first unbind the dimensional object and then perform auto binding. For more information on implementation methods, see ["Relational and ROLAP Implementation of a Dimension"](#) on page 6-22.

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.

For information about how to perform auto binding, see ["Auto Binding"](#) on page 6-12. Auto binding uses the implementation settings described in ["Relational and ROLAP Implementation of a Dimension"](#) on page 6-22.

**Manual Binding** You would typically use manual binding to bind existing tables to a dimension. Use manual binding if no auto binding or rebinding is required.

For information about how to perform manual binding, see ["Manual Binding"](#) on page 6-13.

### MOLAP Implementation

When a dimension is implemented in a MOLAP environment, the dimension definition and data are stored in an analytic workspace. This is done using analytic workspace objects such as dimensions, relationships, and so on. You can store multiple cubes in the same analytic workspace. For more information on MOLAP implementation, see ["MOLAP Implementation of Dimensional Objects"](#) on page 6-14.

## About 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.

With the appropriate licensing, you can use Warehouse Builder to define, deploy, and load all three types of SCDs. You can create slowly changing dimensions only for dimensions that use a relational implementation.

---



---

**Note:** Type 1 does not require additional licensing; however, type 2 and type 3 SCDs require the [Warehouse Builder Enterprise ETL Option](#).

---



---

[Table 6-5](#) describes the three types of SCDs.

**Table 6-5** *Types of Slowly Changing Dimensions*

Type	Use	Description	Preserves History?
Type 1	Overwriting	Only one version of the dimension record exists. When a change is made, the record is overwritten and no historic data is stored.	No
Type 2	Creating a new version of a dimension record	There are multiple versions of the same dimension record, and new versions are created while the old ones are still kept upon modification.	Yes
Type 3	Creating a current value field	There is one version of the dimension record. This record stores the previous value and current value of selected attributes.	Yes

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.

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.

## About Type 1 Slowly Changing Dimensions

In a Type 1 Slowly Changing Dimension (SCD), the new data overwrites the existing data. Typically, this type is not considered an SCD and most dimensions are of this type. Thus the existing data is lost as it is not stored anywhere else. This is the default type of dimension you create. You need not specify any additional information to create a Type 1 SCD. Unless there are specific business reasons, you must assume that a Type 1 SCD is sufficient. For more information on how to define and implement a Type 1 SCD, refer to the following:

- [Defining a Dimension](#)
- [Implementing a Dimension](#)

## About Type 2 Slowly Changing Dimensions

A Type 2 Slowly Changing Dimension (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.

---

---

**Note:** Be aware of the impact that all levels in a dimension not storing historical data has on query tools.

---

---

### Defining a Type 2 Slowly Changing Dimension

To define a Type 2 Slowly Changing Dimension (SCD):

- 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 ID, effective date attribute or expiration date attribute as the triggering attribute.

Each version of a record is assigned a different surrogate identifier. The business ID 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. [Table 6–6](#) lists level attributes of the `Customers` Type 2 SCD.

**Table 6–6 Customers Type 2 SCD Attributes**

Attribute Name	Identifier
ID	Surrogate identifier
BUSN_ID	Business identifier
ADDRESS	
ZIP	
MARITAL_STATUS	
HOME_PHONE	
EFFECTIVE_DATE	Effective Date
EXPIRATION_DATE	Expiration Date

Customer is the leaf level and Household is the non-leaf level.

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 table that implements this Type 2 SCD (for a relational or ROLAP implementation) contains the following columns: DIMENSION\_KEY, H\_ID, H\_BUSN\_ID, H\_ADDRESS, H\_ZIP, H\_EFFECTIVE\_DATE, H\_EXPIRATION\_DATE, C\_ID, C\_BUSN\_ID, C\_MARITAL\_STATUS, C\_HOME\_PHONE, C\_EFFECTIVE\_DATE, and C\_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 Data Object Editor, you must explicitly create these attributes and apply them to the required levels.

### Hierarchy Versioning

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, 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. When you create a Type 2 SCD using the Create Dimension Wizard, hierarchy versioning is disabled. You must use the Data Object Editor to enable hierarchy versioning.

**To enable hierarchy versioning:**

1. Right-click the Type 2 SCD in the Project Explorer and select **Open Editor**.  
The Data Object 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. Open the Data Object Editor for this Type 2 SCD and navigate to the Type 2 slowly changing dimension dialog box as described in steps 1 to 3. 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.

**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 Slowly Changing Dimension (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" in the *Warehouse Builder Online Help*.

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, 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, 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.

## About Type 3 Slowly Changing Dimensions

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.

### Defining a Type 3 Slowly Changing Dimension

To define a Type 3 Slowly Changing Dimension (SCD):

1. For each level, specify which attributes should be versioned. That is, 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 ID 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 6-20 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.

## About Time Dimensions

A time dimension is a dimension that stores temporal data. Time dimensions are used extensively in data warehouses. Warehouse Builder enables you to create and populate time dimensions. You can use Warehouse Builder to create both fiscal and calendar time dimensions.

When you create a time dimension using the wizard, Warehouse Builder creates the mapping for you to execute to populate the time dimension. Also, the data loaded into the time dimension conforms to the best practices recommended by Warehouse Builder for a time dimension.

This section contains the following topics:

- [Best Practices for Creating a Time Dimension](#)

- [Defining a Time Dimension](#)
- [Implementing a Time Dimension](#)
- [Using a Time Dimension in a Cube Mapping](#)
- [Populating a Time Dimension](#)

## Best Practices for Creating a Time Dimension

Warehouse Builder provides an accelerator to create time dimensions. It also specifies a set of rules as best practices for defining a time dimension. Warehouse Builder enforces these rules when you use Create Time Dimension wizard to create a time dimension.

The rules are as follows:

- The time dimension can contain only a subset of the predefined levels specified by Warehouse Builder.
- Each level in a time dimension must have attributes for the time span and ending date.
- A time dimension can have one or more hierarchies. Each hierarchy should be either a fiscal hierarchy or a calendar hierarchy.
- When you deploy a time dimension to the OLAP catalog, you must attach the time span and end date descriptors related to the levels to the dimension and its levels. When you create a time dimension using the Create Time Dimension wizard, Warehouse Builder performs this for you.

If you find these rules too restrictive for your business environment, you can create your own time dimension by setting the time attributes in the Data Object Editor. Ensure that you set the descriptors when you create a time dimension using the Data Object Editor.

## Defining a Time Dimension

A time dimension consists of a set of levels and a set of hierarchies defined over these levels. Dimension roles are used extensively in time dimensions. For more information about dimension roles see "[Dimension Roles](#)" on page 6-19. To create a time dimension you must define the following:

- Levels
- Dimension Attributes
- Level Attributes
- Hierarchies

### Levels

A level represents the level of aggregation of data. A time dimension must contain at least two levels. You can use a level only once in a time dimension. For example, a time dimension can contain only one Calendar Month level. Each level must have a surrogate identifier and a business identifier. The surrogate identifier should be the ID level attribute.

A Warehouse Builder time dimension can contain only a subset of the following levels:

- Day
- Fiscal week

- Calendar week
- Fiscal month
- Calendar month
- Fiscal quarter
- Calendar quarter
- Fiscal year
- Calendar year

### Dimension Attributes

A dimension attribute is an attribute that is implemented by more than one level in the time dimension. [Table 6–7](#) describes the dimension attributes of the Warehouse Builder time dimension.

**Table 6–7 Dimension-level Attributes of the Time Dimension**

Dimension Attribute	Description
ID	The ID attribute is implemented as level ID in all the levels.
Start Date	The start date for the period. It always starts at 00:00:00 of the first day of the period.
End Date	The end date for the period. It always ends on 23:59:59 of the last day of the period.
Time Span	Number of days in the period.
Description	Description of the level record.

### Level Attributes

A level attribute is a descriptive characteristic of a level value. Warehouse Builder creates level attributes for the time dimension based on the levels that you decide to implement for the time dimension.

[Table 6–8](#) lists the attributes of each level in the Warehouse Builder time dimension. For a description of each attribute, refer to Appendix B.

**Table 6–8 Time Dimension Level Attributes**

Level Name	Attribute Name
DAY	ID, DAY, START_DATE, END_DATE, TIME_SPAN, JULIAN_DATE, DAY_OF_CAL_WEEK, DAY_OF_CAL_MONTH, DAY_OF_CAL_QUARTER, DAY_OF_CAL_YEAR, DAY_OF_FISCAL_WEEK, DAY_OF_FISCAL_MONTH, DAY_OF_FISCAL_QUARTER, DAY_OF_FISCAL_YEAR, DESCRIPTION.
FISCAL WEEK	ID, WEEK_NUMBER, WEEK_OF_FISCAL_MONTH, WEEK_OF_FISCAL_QUARTER, WEEK_OF_FISCAL_YEAR, START_DATE, END_DATE, TIME_DATE, DESCRIPTION.
CALENDAR WEEK	ID, START_DATE, END_DATE, TIME_SPAN, DESCRIPTION.
FISCAL MONTH	ID, MONTH_NUMBER, MONTH_OF_QUARTER, MONTH_OF_YEAR, START_DATE, END_DATE, TIME_SPAN, DESCRIPTION.
CALENDAR MONTH	ID, MONTH_NUMBER, MONTH_OF_QUARTER, MONTH_OF_YEAR, START DATE, END_DATE, TIME_SPAN, DESCRIPTION.
FISCAL QUARTER	ID, QUARTER_NUMBER, QUARTER_OF_YEAR, START_DATE, END_DATE, TIME_SPAN, DESCRIPTION

**Table 6–8 (Cont.) Time Dimension Level Attributes**

Level Name	Attribute Name
CALENDAR QUARTER	ID, QUARTER_NUMBER, QUARTER_OF_YEAR, START_DATE, END_DATE, TIME_SPAN, DESCRIPTION.
FISCAL YEAR	ID, YESR_NUMBER, START_DATE, END_DATE, TIME_SPAN, DESCRIPTION.
CALENDAR YEAR	ID, YEAR_NUMBER, START_DATE, END_DATE, TIME_SPAN, DESCRIPTION

### Hierarchies

A hierarchy is a structure that uses ordered levels to organize data. It defines hierarchical relationships between adjacent levels in a time dimension. A time dimension can have one or more hierarchies. Each hierarchy must be either a fiscal hierarchy or a calendar hierarchy. A single time dimension cannot contain both fiscal and calendar hierarchies.

**Calendar Hierarchy** A calendar hierarchy must contain at least two of the following levels: DAY, CALENDAR\_WEEK, CALENDAR\_MONTH, CALENDAR\_QUARTER, CALENDAR\_YEAR.

There is no drill-up path from CALENDAR\_WEEK to any other levels. Thus, if a calendar hierarchy contains CALENDAR\_WEEK level, it cannot contain either the CALENDAR\_MONTH, CALENDAR\_QUARTER, or CALENDAR\_YEAR levels.

**Fiscal Hierarchy** A fiscal hierarchy should contain at least two of the following levels: DAY, FISCAL\_WEEK, FISCAL\_MONTH, FISCAL\_QUARTER, FISCAL\_YEAR.

When you create a fiscal hierarchy, you must specify the following:

- Start month of the fiscal year
- Start date of the fiscal year
- Start day for the fiscal week
- Fiscal Convention used by the time dimension.

The options that you can select for fiscal convention are:

- **455:** Select this option if the first month in the quarter has 4 weeks, the second month in the quarter has 5 weeks, and the third month in the quarter has 5 weeks.
- **544:** Select this option if the first month in the quarter has 5 weeks, the second month in the quarter has 4 weeks, and the third month in the quarter has 4 weeks.

## Implementing a Time Dimension

When you implement a time dimension, you specify how the time dimension and its data are physically stored. You can store the time dimension data either in a relational form or multidimensional form in the database.

The implementation of a time dimension is similar to the implementation of a regular dimension. For more information on implementing a dimension, see "[Implementing a Dimension](#)" on page 6-22.

## Using a Time Dimension in a Cube Mapping

A time dimension created using the Create Time Dimension wizard uses the attribute `ID` as the surrogate identifier and the attribute `CODE` as the business identifier. The data type of both these attributes is `NUMBER`. When you create a cube that references a time dimension, the cube contains attributes that pertain to the surrogate identifier and the business identifier of the lowest level of the time dimension. Both these attributes have a data type of `NUMBER`.

When loading a cube, if you use a Warehouse Builder created time dimension as the source, both the source attributes and the cube attributes are of data type `NUMBER`. For example, consider a cube `ALL_SALES` that references two dimensions `PRODUCTS` and `TIME_FISCAL`. `TIME_FISCAL` is a calendar time dimension created using the Time Dimension wizard and it contains the levels Year, Month, and Day. When you create a map to load the `ALL_SALES` cube, you can directly map the attribute `DAY_CODE` of the Day level of `TIME_FISCAL` to the attribute `ALL_SALES_DAY_CODE` in the cube `ALL_SALES`. The data type of both these attributes is `NUMBER`.

Consider a scenario where you load data into the `ALL_SALES` cube from a source object in which the time data is stored as a `DATE` attribute. In this case, you cannot directly map the `DATE` attribute from the source to the attribute `ALL_SALES_DAY_CODE` of the `ALL_SALES` cube. Instead, you use an Expression operator in the mapping to convert the input `DATE` attribute to a `NUMBER` value and then load it into the `ALL_SALES` cube. In the Expression operator you convert the input using the following expression:

```
TO_NUMBER (TO_CHAR (input, 'YYYYMMDD' ) )
```

where `input` represents the `DATE` attribute from the source object that needs to be converted to a `NUMBER` value. For information on using the Expression operator, see "Expression Operator" in the *Warehouse Builder Online Help*.

## Populating a Time Dimension

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. The time dimension is populated 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 attributes are initialized at the time of creating the time dimension using the Create Time Dimension wizard. You can alter the values of these parameters using the Data Object Editor. To change the values of the start date of the calendar year and the number of calendar years, use the Name tab of the Data Object Editor. To change the values of the parameters pertaining to fiscal time dimensions, use the Fiscal Settings button on the Hierarchies tab of Data Object Editor.

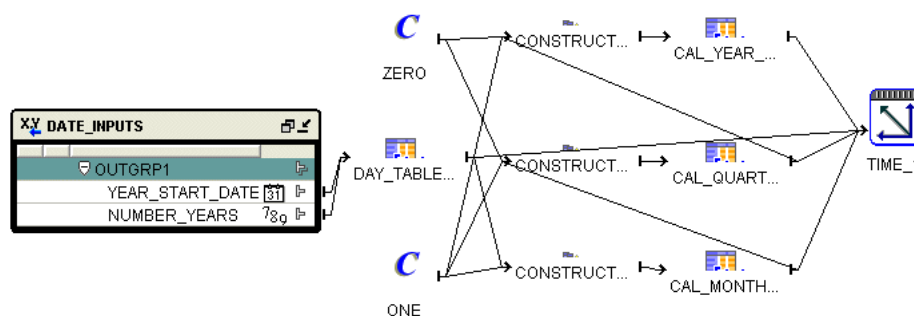
---

**Note:** When you alter the values of any of the parameters pertaining to the data to be loaded into the time dimension, you must re-create the map that loads the time dimension. For more information on re-creating the map, see "Hierarchies Tab" in the *Warehouse Builder Online Help*.

---

Figure 6–5 displays a mapping to load a calendar time dimension. The Mapping Input operator DATE\_INPUTS represents the attributes needed to populate the time dimension.

**Figure 6–5 Mapping that Populates a Time Dimension**



### 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.

## About 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 in a 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.

## Defining a Cube

A cube consists of the set of measures defined over a set of dimensions. To create a cube, you must define the following:

- [Cube Measures](#)
- [Cube Dimensionality](#)

### Cube Measures

A measure is data, usually numeric and additive, that can be examined and analyzed. Examples of measures include sales, cost, and profit. A cube must have one or more measures. You can also perform aggregation of measures. Only numeric measures can be aggregated.

### Cube Dimensionality

A cube is defined by a set of dimensions. A cube can refer to a level that is not the lowest level in a dimension.

For cubes that use a pure relational implementation, you can reuse the same dimension multiple times with the help of dimension roles. For more information on dimension roles, see "[Dimension Roles](#)" on page 6-19.

Before you validate a cube, ensure that all the dimensions that the cube references are valid.

To define a dimension reference, specify the following:

- The dimension and the level within the dimension to which the cube refers.  
For a cube that uses a relational implementation, you can refer to intermediate levels in a dimension. However, for cubes that use a MOLAP implementation, you can only reference the lowest level in the dimension. Warehouse Builder supports a reference to the non surrogate identifier of a level, for example, the business keys.
- For dimensions that use a relational or ROLAP implementation, a dimension role for each dimension to indicate what role the dimension reference is performing in the cube. Specifying the dimension role is optional.

When you define a MOLAP cube, the order in which you define the dimension references is important. The physical ordering of dimensions on disk is the same as the order in which you define the dimension references. The physical ordering is tightly coupled with the sparsity definition. Define the dimension references in the order of most dense to least dense. Time is usually a dense dimension, and listing it first expedites data loading and time-based analysis. For more information on defining dimension references, see "Dimensions Page" or "Dimensions Tab" in the *Warehouse*



*Builder Online Help*. For more information on sparsity, see "Advanced Dialog Box" in the *Warehouse Builder Online Help*.

### Default Aggregation Method

You can define aggregations that should be performed on the cube. For ROLAP cubes, you can only define a single aggregation method for the cube. For MOLAP cubes, you can define a different aggregation method for each dimension of each measure. Warehouse Builder enables you to use the same aggregation function for all the cube measures or specify different aggregate functions for each measure.

Warehouse Builder supports the following default aggregation methods: SUM, SSUM (scaled SUM), AVERAGE, HAVERAGE (hierarchical average), MAX, MIN, FIRST, LAST, AND, OR, HIERARCHICAL\_FIRST and HIERARCHICAL\_LAST. If you do not want to perform aggregation, select NOAGG. The methods AND and OR are not applicable for cubes that use a multidimensional implementation.

---

**Note:** You cannot define aggregation for pure relational cubes.

---

### 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 6–9 describes the dimensionality of the Sales cube. It lists the name of the dimension and the dimension level that the cube references.

**Table 6–9 Dimensionality of the Sales Cube**

Dimension Name	Level Name
Products	Product
Customers	Customer
Times	Day

## Implementing a Cube

When you implement a cube, you specify the physical storage details for the cube. You can implement a cube in a relational form or a multidimensional form in the database.

The types of implementation you can use for cubes are:

- Relational implementation
- ROLAP implementation
- MOLAP implementation

To set the type of implementation for a cube, use the **Deployment Option** configuration property. For more details on setting this option, see "Configuring Cubes" in the *Warehouse Builder Online Help*.

### Relational and ROLAP Implementation of a Cube

The database object used to store the cube data is called a fact table. A cube must be implemented using only one fact table. The fact table contains columns for the cube measures and dimension references. For more information on setting the

implementation option for a cube, see ["Implementing Dimensional Objects"](#) on page 6-11.

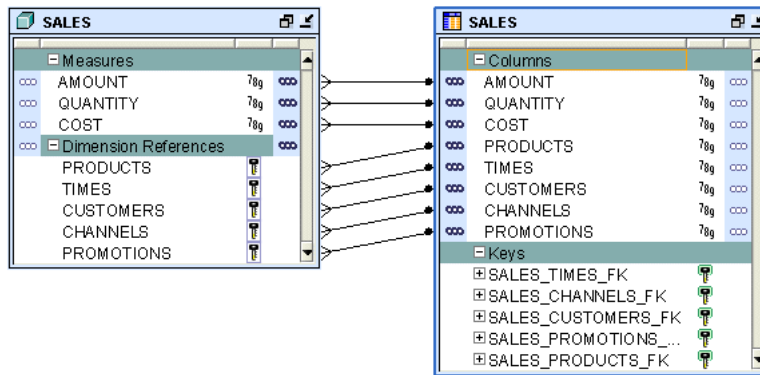
To implement a cube:

- Select a table or materialized view that will store the cube data.
- For each measure, select a column that will store the measure data.
- For each dimension reference, select a column that will store the dimension reference.

Each dimension reference corresponds to a column on the fact table and optionally a foreign key from the fact table to dimension table. The 1:n relationships from the fact tables to the dimension tables must be enforced.

[Figure 6-6](#) displays the bindings for the relational implementation of the SALES cube. The data for the SALES cube is stored in a table called SALES.

**Figure 6-6 Implementation of the Sales Cube**



### Binding

When you perform binding, you specify the database columns that will store the data of each measure and dimension reference of the cube. You can perform auto binding or manual binding for a cube. For more information on binding, see ["Binding"](#) on page 6-12.

**Auto Binding** When you perform auto binding, Warehouse Builder creates the table that stores the cube data and then binds the cube measures and references to the database columns. For detailed steps on performing auto binding, see ["Auto Binding"](#) on page 6-12.

When you perform auto binding for a cube, ensure that you auto bind the dimensions that a cube references before you auto bind the cube. You will not be able to deploy the cube if any dimension that the cube references has been auto bound after the cube was last auto bound.

For example, you create the SALES cube that references the TIMES and PRODUCTS dimensions and perform auto binding for the cube. You later modify the definition of the PRODUCTS dimension. If you now attempt to auto bind the SALES cube again, Warehouse Builder generates an error. You must first auto bind the PRODUCTS dimensions and then auto bind the cube.

**Manual Binding** In manual binding, you must first create the table or view that stores the cube data and then map the cube references and measures to the database columns

that store their data. Alternatively, you can use an existing database table or view to store the cube data.

For information about how to perform manual binding, see ["Manual Binding"](#) on page 6-13.

### **MOLAP Implementation of a Cube**

Storing the cube and its data in an analytic workspace is called a MOLAP implementation. You can store multiple cubes in the same analytic workspace. For more information on OLAP implementation, see ["MOLAP Implementation of Dimensional Objects"](#) on page 6-14.

## **Solve Dependency Order of Cube**

Certain business scenarios may require the dimensions in a cube to be evaluated in a particular order. The order in which the dimensions are evaluated is called the solve dependency order of the cube. For example, in the Sales cube, the Time dimension may need to be evaluated before the Products dimension. For each dimension of the cube, you can specify a dependency on another dimension of the cube.

The advantage of specifying the dependency order is that it enables Warehouse Builder to optimize the query speed of calculating the joins of the dimension and cubes. For example, retrieving results from the sales cube based on Time criteria may be more selective than retrieving result based on Products criteria. In this case, you can specify that for the Sales cube, the Products dimension depends on the Time dimension.

Specifying the solve dependency order is optional. If you do not specify a dependency order, the optimizer determines the solve-order with additional flexibility.

## **Designing the Target Schema**

To create a target schema, you create any of the dimensional or relational objects listed in [Table 6-1](#) on page 6-2. 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.

### **Designing a Relational Target Schema**

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, create an Oracle module that will contain the objects for your target schema. Ensure that the location associated with this module refers to the target schema.
2. Create the relational data objects.

You may have already imported some existing target objects. To create additional data objects, refer to ["Creating Relational Data Objects"](#) on page 6-41.

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 properties. You can choose to modify these default values.

See ["Configuring Data Objects"](#) on page 6-45.

4. Validate the data objects.

Validation verifies the metadata definitions and configuration properties of data objects. Correct any errors that are encountered during the validation.

See ["Validating Data Objects"](#) on page 6-45.

5. Generate code that will create these data objects in the target schema.

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 6-47.

## Designing a Dimensional Target Schema

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-focussed environment. This helps you obtain answers to complex analytical queries quickly and more efficiently.

### To design a dimensional target schema:

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

2. Create the dimensions required in your target schema.

See ["Creating Dimensions"](#) on page 6-41. 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. Create time dimensions.

Data warehouses use time dimensions extensively to store temporal data. See ["Creating Time Dimensions"](#) on page 6-43.

4. Create the cubes required for the target schema.

See ["Creating Cubes"](#) on page 6-44.

5. Configure the dimensions and cubes.

Configure the dimensional objects 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 ["Configuring Data Objects"](#) on page 6-45.

6. Validate the dimensions and cubes.

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

See ["Validating Data Objects"](#) on page 6-45.

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

See "Generating Data Objects" on page 6-47.

## Creating Oracle Data Objects

To create data objects, you can either start the appropriate wizard or use the Data Object Editor. Some objects, such as dimensions and cubes, can be created using a wizard or the Data Object Editor. Some objects, such as tables, can be created using the Data Object Editor only.

For objects that can be created using a wizard or the Data Object Editor, you right-click the node for the object, select **New**, and then **Using Wizard** or **Using Editor**.

After using a wizard, you may want to modify the object in the editor. In that case, right-click the object and select **Open Editor**.

## Creating Relational Data Objects

Relational data objects include tables, views, materialized views, and sequences. To create tables, views, and materialized views, use the Data Object Editor. Use the Create Sequence dialog box to create sequences.

You can create additional structures pertaining to relational objects such as constraints, indexes, and partitions. For more information about how to create these structures, see "Reference for Using Oracle Data Objects" in the *Warehouse Builder Online Help*.

### To create relational data objects:

1. In the Project Explorer, expand the Oracle node that corresponds to the target schema.
2. Right-click the node that represents the type of data object you want to create and select **New**.

For example, to create a table, right-click the Tables node and select **New**. The Data Object Editor is displayed.

3. Navigate to the Details panel of the Data Object Editor.
4. Use the tabs in the Details panel to define the data object.

For more information about the details to be entered on each tab, click the arrow at the top of the Details panel and select **Help**.

## Creating Dimensions

You can create dimensions using the Create Dimension Wizard or the Data Object Editor. Use the wizard to create a fully functional dimension object quickly. If you choose a relational implementation for the dimension, the wizard creates the implementation tables in the target schema using auto binding.

The Data Object Editor provides maximum flexibility to create a dimension. You can perform certain advanced tasks only by using the Data Object Editor.

### To create a dimension using the Create Dimension Wizard:

1. In the Project Explorer, expand the Oracle node that corresponds to the target schema.
2. Right-click the Dimensions node, select **New**, then **Using Wizard**.

The Welcome Page of the Create Dimension Wizard is displayed.

3. Click **Next**.

The Name and Description page is displayed.

4. Enter a name and an optional description for the dimension.

Dimension names should follow the rules specified in ["Naming Conventions for Data Objects"](#) on page 6-6.

5. Enter details on the following wizard pages.

- Storage Type page  
See ["Implementing a Dimension"](#) on page 6-22
- Dimension Attributes page  
See ["Defining Dimension Attributes"](#) on page 6-18
- Levels page  
See ["Defining Levels"](#) on page 6-18
- Level Attributes page  
See ["Defining Level Attributes"](#) on page 6-19
- Slowly Changing Dimension page  
See ["About Slowly Changing Dimensions"](#) on page 6-25

For additional information about the information to be provided on each wizard page, click **Help** on the page.

6. Click **Next**.

The Pre Create Settings page is displayed. This page lists the objects created to implement the dimension. Review the entries on this page.

7. Click **Next**.

The Dimension Creation Progress page is displayed. The progress bar displays the progress of the dimension creation. Wait till the progress bar reaches 100%.

8. Click **Next**.

The Summary page is displayed. This page lists the details of the dimension created in the previous step.

9. Click **Finish**.

The definition of the dimension and its implementation objects, if any, are created. For a relational or ROLAP dimension, the implementation tables and the sequence used to load the surrogate identifier of the dimension are created. For MOLAP dimensions, the analytic workspace used to store the dimension is created.

**To create a dimension using the Data Object Editor:**

1. In the Project Explorer, right-click the Dimensions node in the target module, select **New**, then **Using Editor**.

The Data Object Editor is displayed.

2. Use the following tabs on the Dimension Details panel to define the dimension.

- Name  
Dimension names should conform to the rules specified in ["Naming Conventions for Data Objects"](#) on page 6-6.

- Storage  
See ["Implementing a Dimension"](#) on page 6-22
- Attributes  
See ["Defining Dimension Attributes"](#) on page 6-18
- Levels  
See ["Defining Levels"](#) on page 6-18 and ["Defining Level Attributes"](#) on page 6-19
- Hierarchies  
See ["Defining Hierarchies"](#) on page 6-19
- SCD  
See ["About Slowly Changing Dimensions"](#) on page 6-25

For more information about the details to be entered on each tab, click the arrow at the top of the Dimension Details panel and select **Help**.

When you use the Data Object Editor to create dimensions, the implementation objects are not automatically created. For relational and ROLAP dimensions, you can create the implementation tables that store the dimension data by performing [Auto Binding](#).

## Creating Time Dimensions

You can create a fully functional time dimension using the Create Time Dimension Wizard. If you need more flexibility in defining your time dimension, use the Data Object Editor to create a dimension that stores temporal data. For information about using the Data Object Editor to create time dimensions, see ["Creating Dimensions"](#) on page 6-41.

### To create a time dimension using the Create Time Dimension Wizard:

1. In the Project Explorer, expand the Oracle node that corresponds to the target schema.
2. Right-click the Dimensions node, select **New**, then **Using Time Wizard**.  
The Welcome Page of the Create Time Dimension Wizard is displayed.
3. Click **Next**.  
The Name and Description page is displayed.
4. Enter a name and an optional description for the time dimension.  
Time dimension names should follow the rules specified in ["Naming Conventions for Data Objects"](#) on page 6-6.
5. Enter details on the following wizard pages. For information about the options on each wizard page, click **Help**.
  - Storage Type page  
See ["Implementing a Time Dimension"](#) on page 6-33
  - Data Generation page  
Specify the range of data to be stored in the time dimension. Also indicate the type of data stored in the time dimension, fiscal or calendar.
  - Levels page

Select the levels in the time dimension. The levels displayed on this page depend on the option you chose on the Data Generation page.

**6. Click Next.**

The Pre Create Settings page is displayed. This page lists the objects created to implement the dimension. Review the entries on this page.

**7. Click Next.**

The Dimension Creation Progress page is displayed. The progress bar displays the progress of the dimension creation. Wait till the progress bar reaches 100%.

**8. Click Next.**

The Summary page is displayed. This page lists the details of the dimension being created.

**9. Click Finish.**

The definition of the time dimension and its implementation objects, if any, are created. A mapping that loads the time dimension is also created.

For a relational or ROLAP time dimension, the implementation tables and the sequence used to load the surrogate identifier of the time dimension are created. For MOLAP dimensions, the analytic workspace used to store the time dimension is created.

## Creating Cubes

Use the Create Cube Wizard or the Data Object Editor to create cubes.

**To create a cube using the Create Cube Wizard:**

1. In the Project Explorer, expand the Oracle node that corresponds to the target schema.

2. Right-click the Cubes node, select **New**, then **Using Wizard**.

The Welcome Page of the Create Cube Wizard is displayed.

3. Click **Next**.

The Name and Description page is displayed.

4. Enter a name and an optional description for the cube.

Cube names should follow the rules specified in "[Naming Conventions for Data Objects](#)" on page 6-6.

5. Enter details on the following wizard pages. For information about the options on each wizard page, click **Help**.

- Storage Type page  
See "[Implementing a Cube](#)" on page 6-37
- Dimensions page  
See "[Cube Dimensionality](#)" on page 6-36
- Measures page  
See "[Cube Measures](#)" on page 6-36

6. Click **Next**.



The Summary page is displayed. This page lists the details of the cube being created.

7. Click **Finish**.

The definition of the cube and its implementation objects, if any, are created. For a relational or ROLAP cube, the implementation tables are created. For MOLAP cubes, the analytic workspace used to store the time dimension is created.

## 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. For more information, see "[Creating Additional Configurations](#)" on page 11-14.

All objects have a Deployable parameter, which is selected by default. To prevent an object from being deployed, clear this parameter.

You can configure objects using the Data Object Editor or the Project Explorer. To configure an object using the Data Object Editor, use the Configuration panel of the editor. This panel displays the configuration details for the object currently selected on the canvas. You can even drill down to, say and index in a table in the Selected Objects tab of the Explorer panel to see those configuration details.

### To configure an object using the Project Explorer:

1. In the Project Explorer, select the object and click the Configure icon.

*or*

Right-click the object and select **Configure**.

The Configuration Properties dialog box is displayed.

2. Select a parameter to display its description at the bottom of the right panel. Click **Help** for additional information.
3. Enter your changes and click **OK**.

## Validating Data Objects

Validation is the process of verifying metadata definitions and configuration parameters. These definitions must be valid before you proceed to generation and deployment of scripts.

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 display. Warehouse Builder enables you to open object editors and make corrections to any invalid objects before continuing. In addition to being a standalone operation, validation also takes place implicitly when you generate or deploy objects.

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

---

---

**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 configuration.

---

---

When you validate an object after it has been defined, the metadata definitions for the objects 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, validation messages display in the Validation Results window.

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

You can validate a single object or multiple objects at a 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 Project Explorer or the Data Object Editor to validate data objects.

When you validate objects, Warehouse Builder displays the Validation Results window that contains the results of the validation. For more information about this dialog box, click **Help** and then **Topic**.

### **Validating Data Objects Using the Project Explorer**

In the Project Explorer, select the data object and click the Validate icon. You can select multiple objects by holding down the **Ctrl** key while selecting objects.

*or*

In the Project Explorer, 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, ensure that the **Ctrl** key is pressed when you right-click.

### **Validating Data Objects Using the Data Object Editor**

Right-click the icon representing the data object on the Data Object Editor canvas and select **Validate**.

*or*

Select the object on the canvas and either click the Validate icon or select **Validate** from the Object menu.

## **Editing Invalid Objects**

The results of validating data objects are displayed in the Validation Results window. From this window, you can access the editors for objects and rectify errors in their definition, if any.

### **To edit invalid definitions:**

1. In the Validation Results window, double-click an invalid object from the tree or from the validation messages grid.

An editor for the selected object is displayed.

2. Edit the object to correct problems.
3. Close the editor when you are finished and re-validate.

## Generating Data Objects

When you generate data objects, Warehouse Builder produces the code required to create the data objects in the target schema. Warehouse Builder generates the following types of scripts:

- **DDL scripts:** Creates or drops database objects.
- **SQL\*Loader control files:** Extracts and transports data from file sources.
- **ABAP Scripts:** Extracts and loads data from SAP systems.

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.

Use the Project Explorer or the Data Object Editor to generate code for data objects. When you generate objects, Warehouse Builder displays the Generation Results window that contains the results of the generation. For more information about this window, click **Help** and then **Topic**.

### Generating Data Objects Using the Project Explorer

To generate a single data object, select the data object and click the Generate icon. Or right-click the data object and select **Generate**.

To generate code for multiple objects, select the objects by holding down the Ctrl key and click the Generate icon. Or select the data objects and, while continuing to hold down the Ctrl key, right-click and select **Generate**.

### Generating Objects Using the Data Object Editor

Open the Data Object Editor for the data object by right-clicking the object and selecting **Open Editor**. The canvas displays a node that represents the data object.

Right-click the data object node on the canvas and select **Generate**.

*or*

Select the data object node on the canvas. Click the Generate icon or select **Generate** from the Object menu.

## Viewing Generated Scripts

**To view the generated scripts:**

1. From the Generation Results window, select an object in the navigation tree on the left of the Generation Results window.
2. Select the Scripts tab on the right of this window.

The Scripts tab contains a list of the generated scripts for the object you selected.

3. Select a specific script and click **View Code**.

The selected script displays in a code viewer, which is read-only.

## Saving Generated Scripts to a File

### To save generated scripts:

1. From the Generation Results window, select an object from the navigation tree on the left.
2. Select the Scripts tab from the bottom section of the window.  
The Scripts tab contains a list of the generated scripts for the object you selected.
3. Select a specific script and click **Save As**.  
The Save dialog box opens and you can select a location where you want to save the script file.

## Deriving Business Intelligence Metadata

Warehouse Builder enables you to derive business intelligence objects from your existing relational and dimensional data objects. When you derive intelligence objects, Warehouse Builder tailors existing definitions to match the definitions used by the Oracle Discoverer End User Layer.

You can deploy intelligence objects derived from data warehouse design definitions directly to Oracle Discoverer.

### To derive intelligence objects:

1. If you have not already done so, create a business definition module that will contain the derived business intelligence objects.  
  
To create a business definition module, expand the Business Intelligence node in the Project Explorer, right-click Business Definitions and select **New**. The Create Business Definition Module Wizard displays. Specify a name and an optional description for the business definition module. Ensure that the location associated with this module refers to the Discoverer EUL to which the derived business definitions will be deployed.
2. To derive all the objects in an Oracle module, right-click the Oracle module in the Project Explorer and select **Derive**.  
  
To derive a particular object, right-click that object in the Project Explorer and select **Derive**.  
  
The Welcome page of the Perform Derivation Wizard is displayed. Click **Next** to proceed with the derivation.
3. On the Source Objects page, the Selected section displays the objects you selected in step 2. To derive additional objects, select the objects and move them from the Available list to the Selected list.
4. On the Target page, select the business definition module or business area that will contain the derived objects.
5. On the Rules page, specify the rules and parameters for the derivation.  
  
For more information about the rules and parameters, click **Help** on this page.
6. On the Pre Derivation page, review the selections you made. Click **Back** to modify selected values. Click **Next** to proceed.
7. The Derivation page displays a progress bar that indicates the progress of the derivation. Wait until the progress reaches 100% and click **Finish** to complete the derivation.

Once you derive business definitions, you can directly deploy them to Oracle Discoverer. For information about deploying to Discoverer, see "[Deploying Business Definitions to Oracle Discoverer](#)" on page 11-8.



---

---

## Creating Mappings

After you create and import data object definitions in Warehouse Builder, you can design extraction, transformation, and loading (ETL) operations that move data from sources to targets. In Warehouse Builder, you design these operations in a mapping.

This chapter contains the following topics that describe how to create, edit, and use mappings:

- [About Mappings and Operators](#)
- [Instructions for Defining Mappings](#)
- [Creating a Mapping](#)
- [Adding Operators](#)
- [Editing Operators](#)
- [Connecting Operators](#)
- [Using Pluggable Mappings](#)
- [Setting Mapping Properties](#)
- [Setting Operator, Group, and Attribute Properties](#)
- [Synchronizing Operators and Workspace Objects](#)
- [Using DML Error Logging](#)
- [Debugging a Mapping](#)

### About Mappings and Operators

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 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 the 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:** PL/SQL stands for Procedural Language/Standard Query Language. It extends SQL by adding constructs found in procedural languages, resulting in a structural language that is more powerful than SQL.

- **SQL\*Loader:** SQL\*Loader is an Oracle tool for loading data from files into Oracle Database tables. It is the most efficient way to load large amounts of data from flat files.
- **ABAP:** ABAP is a programming language for developing applications for the SAP R/3 system, a business application subsystem.

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 you select as sources have an impact on how you design the mapping. Based on the operators you select, Warehouse Builder assigns the mapping to one of the following Mapping Generation Languages:

- PL/SQL
- SQL\*Loader
- ABAP

Each of these code languages require you to adhere to certain rules when designing a mapping.

- **PL/SQL Mappings:** 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 "[Understanding Performance and Advanced ETL Concepts](#)" on page 9-1.
- **SQL\*Loader Mappings:** 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" in the *Warehouse Builder Online Help*.
- **ABAP Mappings:** When you define a SAP/R3 source, Warehouse Builder generates ABAP code. For mapping design considerations for SAP sources, see "[Defining the ETL Process for SAP Objects](#)" on page 4-27.

## Instructions for Defining Mappings

### Before You Begin

First verify that your project contains a warehouse target module with a defined location.

Also import any existing data you intend to use as sources or targets in the mapping.

**To define a mapping, refer to the following sections:**

1. [Creating a Mapping](#) on page 7-4
2. [Adding Operators](#) on page 7-11  
To design a mapping to extract from or load to a flat file, refer to "[Instructions for Using Flat File Sources or Targets in a Mapping](#)" on page 7-3.
3. [Editing Operators](#) on page 7-12
4. [Connecting Operators](#) on page 7-17
5. [Using Pluggable Mappings](#) on page 7-20
6. [Setting Mapping Properties](#) on page 7-23
7. [Setting Operator, Group, and Attribute Properties](#) on page 7-24



8. Configuring Mappings Reference in the *Warehouse Builder Online Help*
9. For PL/SQL mappings, you can also refer to "[Best Practices for Designing PL/SQL Mappings](#)" on page 9-1.
10. [Debugging a Mapping](#) on page 7-33
11. When you are satisfied with the mapping design, generate the code by selecting the Generate icon in the toolbar.

### Subsequent Steps

After you design a mapping and generate its code, you can next 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 "[Designing Process Flows](#)" on page 8-1.

Deploy the mapping, and any associated process flows you created, and then execute the mapping as described in "[Deploying to Target Schemas and Executing ETL Logic](#)" on page 11-1.

## Instructions for Using Flat File Sources or Targets in a Mapping

In a mapping you can use flat file operators as either sources or targets but not a mix of both. You can import file definitions from existing flat files and use that data as a source or target in the mapping. Or you can create your own flat file definition in the Mapping Editor to load data into a new flat file target.

### Creating a New Flat File Target

To create a new flat file definition for a target, complete the following steps:

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. Create the mapping definition as described in "[Creating a Mapping](#)" on page 7-4.
3. Drag and drop a flat file operator onto the canvas.
4. On the Add Flat File Operator dialog box, select the option [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 7-12.  
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 Project Explorer Files node under the module you specified.
7. Continue to define your mapping as described in "[Instructions for Defining Mappings](#)" on page 7-2.

### Creating a Source or Target Based on an Existing Flat File

To use an existing flat file as a source or target, complete the following steps:

1. In the Project Explorer, right-click the File node and select **New** to create a module for the flat files as described in "Creating Flat File Modules" in the *Warehouse Builder Online Help*.
2. Right-click the flat file module and select **Import** to import file definitions as described in "[Importing Definitions from Flat Files](#)" on page 4-7.
3. Decide to use the file as either a source or a target.

If you import a file for use as a target, Warehouse Builder generates PL/SQL code for the mapping. Review the details in "Flat File Target Operators" in the *Warehouse Builder Online Help* and then skip to step 7.

If you import a file for use as a source, you must decide whether to maintain the flat structure of the file using SQL\* Loader or to represent the data in PL/SQL format through an external table. Continue to the next step.

4. Refer to "External Table Operators versus Flat File Operators" in the *Warehouse Builder Online Help* to determine what type of operator to use in your mapping. If you select external table operator, continue to the next step.

If you select flat file operator, skip to step 7.

5. Create the external table as described in "Creating a New External Table Definition" in the *Warehouse Builder Online Help*.
6. In the Project Explorer, right-click the external table and select **Configure**. On the Data Files node, right-click and select **Create**.  
Enter the name of the flat file from which the external table inherits data. Enter the file name and the file extension such as *myflatfile.dat*.
7. Drag and drop the flat file operator or external table operator onto the canvas.
8. On the Add Operator dialog box, select the option [Select from Existing Repository Object and Bind](#).

You can now continue designing your mapping.

## Creating a Mapping

### To create a mapping:

1. Navigate to the **Mappings** node in the Project Explorer. This node is located under a warehouse target module, under the Databases folder, under the Oracle folder.
2. Right-click Mappings and then select **New**.  
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 "[Mapping Naming Conventions](#)" on page 7-14.
4. Click **OK**.

Warehouse Builder stores the definition for the mapping and inserts its name in the Project Explorer. Warehouse Builder opens a mapping editor for the mapping and displays the name of the mapping in the title bar.

### To open a previously created mapping:

1. From the Project Explorer, locate a warehouse target module under the Databases folder and then under the Oracle folder.

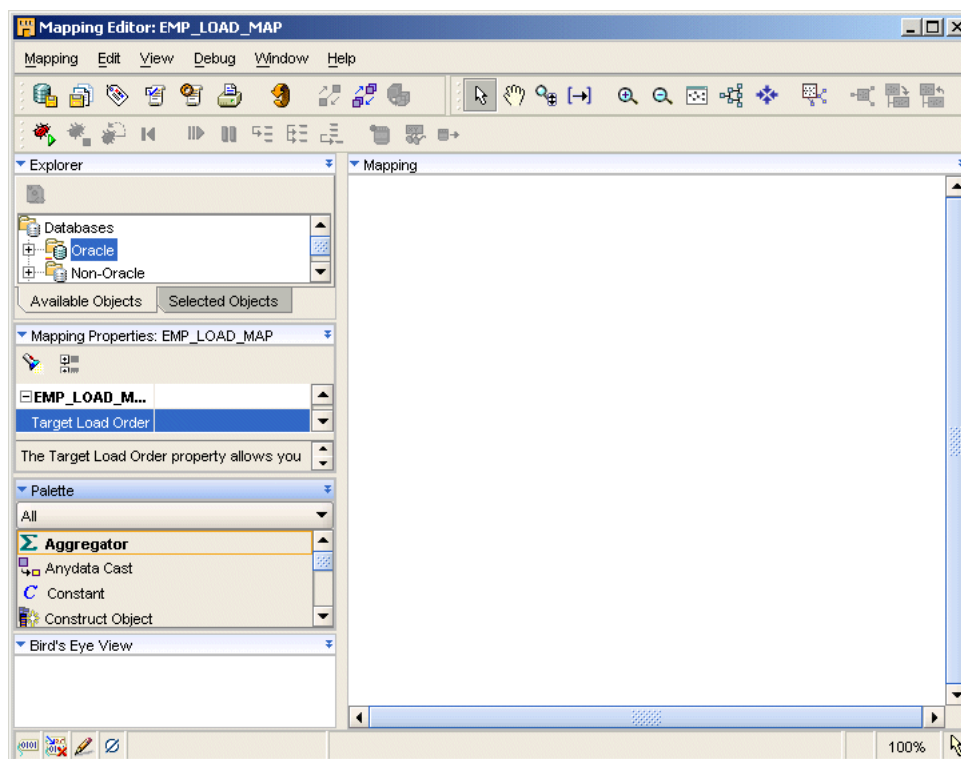
2. Expand the Mappings node.
  3. Open the Mapping Editor in one of the following ways:
    - Double-click a mapping.
    - Select a mapping and then from the **Edit** menu, select **Open Editor**.
    - Select a mapping and press **Ctrl + O**.
    - Right-click a mapping, and select **Open Editor**.
- Warehouse Builder displays the Mapping Editor.

## About the Mapping Editor

The first time you open the Mapping Editor, it displays with a menu bar, multiple toolbars, multiple windows along the left side, and a canvas on the right.

Figure 7-1 displays the Mapping Editor canvas.

**Figure 7-1 Mapping Editor Canvas**



### Standard Editor Components

The Mapping Editor has the following standard components common to most editors in Warehouse Builder:

- **Title Bar:** At the top of the editor, the title bar displays the name of the mapping and the access privileges you have on the mapping.
- **Menu Bar:** Below the title bar, the menu bar provides access to the editor commands. You can access the menu bar by clicking on one of its options or by using hot keys. For example, to access the Mapping menu, press **Alt +M**.

- **Toolbar:** Below the menu bar, the toolbar provides icons for commonly used commands.
- **Canvas:** The canvas provides the work space where you design and modify mappings.
- **Indicator Bar:** Along the lower edge of the editor you can see mode icons, indicators, and descriptions.

Figure 7–2 displays the Indicator Bar of the mapping Editor.

**Figure 7–2 Indicator Bar on the Mapping Editor**



In the left corner are Naming Mode, Rename Mode, Read/Write, and Validation Mode.

In the right corner are the percent zoom indicator and the navigation mode. In the preceding figure, the zoom level is at 100% and the navigation mode is set to Select Mode.

## Mapping Editor Windows

You can resize a window by placing your mouse on the border of the window, pressing the mouse button when the double sided arrow appears, and dragging your mouse to indicate the desired size.

You can move a window by placing the mouse on the Title Bar, and dragging the mouse to the desired location.

To show or hide windows, select **Window** from the menu bar and either activate or deactivate the check mark corresponding to the window.

### Explorer

When you first start the editor, Warehouse Builder displays the explorer in the upper left corner. The explorer provides a tree listing of all the activities on the canvas and their parameters. When you select an activity on the canvas, Warehouse Builder navigates to the activity on the explorer.

### Properties Inspector

When you first start the editor, Warehouse Builder displays the properties inspector in the lower left corner. The properties inspector displays the properties for the mapping, its operators, and attributes in the operators. Select an object either from the canvas or the explorer and Warehouse Builder displays the properties in the properties inspector.

### Palette

When you first start an editor, Warehouse Builder displays the palette along the left side and it contains operators that you can drag and drop onto the canvas. You can relocate the palette anywhere on the editor. You can choose to hide or display the palette by clicking on Operator Palette listed under **View** in the menu bar.

### Bird's Eye View

The Bird's Eye View 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 colored box that represents the portion of the canvas that is currently in focus. In the case of mappings that span more than the canvas size, you can click the blue box and drag it to the portion of the canvas that you want to focus on.

### Data Viewer

The Data Viewer enables you to view the data stored in the data object. See "[Data Viewer](#)" on page 6-7 for more information about the Data Viewer.

### Generation

The Generation panel displays the generation and validation results for a data object. This panel is hidden when you first open the editor window. It is displayed the first time you generate or validate a data object. You can to show or hide the Generation panel by selecting **Window** and then **Generation Results** from the editor menu.

The Generation window contains two tabs: Script and Message. The Script tab displays the generated scripts to implement the data object selected in the canvas. The Message tab displays the validation messages for the selected data object. Double-click a message to view the complete message text.

## Mapping Editor Toolbars

The Mapping Editor provides the following task oriented toolbars: general, graphic, generation, and palette. With the exception of the palette, the editor by default displays the toolbars below the menu bar. You can move, resize, or hide each of the toolbars.

- **General Toolbar:** Use this toolbar to call common operations such as save all, exporting diagram, validating, generating, and printing.
- **Diagram Toolbar:** Use this toolbar to navigate the canvas and change the magnification of objects on the canvas.
- **Debug Toolbar:** Use this toolbar to call commands for debugging the mapping.
- **Palette Toolbar:** The palette contains operator icons. To include an operator, drag an operator icon onto the Mapping Editor canvas. As Warehouse Builder includes over 50 operators, you may want to sort and display the operators based on type.

## Mapping Editor Display Options

You can control how the editor displays the mappings on the canvas by selecting **View** from the menu bar and 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.

## 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.










- **Oracle Source/Target Operators:** These operators represent Oracle Database objects in the mapping. It also contains Flat File Source and Target operators.
- **Remote and Non-Oracle Source and Target Operators:** The use of these operator have special requirements discussed in "Using Remote and non-Oracle Source and Target Operators" in the *Warehouse Builder Online Help*.
- **Data Flow Operators:** Data flow operators transform data.
- **Pre/Post Processing Operators:** Calls a function or procedure before or after executing a mapping
- **Pluggable Mapping Operators:** These are mappings that function as operators in other mappings.

### Oracle Source/Target Operators





Use source and target operators to represent relational database objects and flat file objects.

Table 7-1 lists each source and target operator alphabetically, gives a brief description.

**Table 7-1 Source and Target Operators**

Icon	Operator	Description
	Constant operator	Produces a single output group that can contain one or more constant attributes.
	Construct Object operator	Produces object types and collection types.
	Cube operator	Represents a cube that you previously defined.
	Data Generator operator	Provides information such as record number, system date, and sequence values.
	Dimension operator	Represents a dimension that you previously defined.
	Expand Object operator	Expands an object type to obtain the individual attributes that comprise the object type.
	External Table operator	Represents an external table that you previously defined or imported.
	Flat File operator	Represents a flat file that you previously defined or imported.
	Materialized View operator	Represents a materialized view that you previously defined.

**Table 7-1 (Cont.) Source and Target Operators**













Icon	Operator	Description
	Sequence operator	Generates sequential numbers that increment for each row.
	Table operator	Represents a table that you previously defined or imported.
	Varray Iterator operator	Iterates through the values in the table type.
	View operator	Represents a view that you previously defined or imported.

## Data Flow Operators





Use data flow operators to transform data in a mapping.

Table 7-2 lists each data flow operator alphabetically, gives a brief description. For more information on these transformation operators, see "Data Flow Operators" in the *Warehouse Builder Online Help*.

**Table 7-2 Data Flow Operators**

Icon	Operator	Description
	Aggregator operator	Performs data aggregations, such as SUM and AVG, and provides an output row set with aggregated data.
	Anydata Cast operator	Converts an object of type Sys.AnyData to either a primary type or to a user-defined type.
	Deduplicator operator	Removes duplicate data in a source by placing a DISTINCT clause in the select code represented by the mapping.
	Expression operator	Enables you 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.
	Filter operator	Conditionally filters out rows from a row set.
	Joiner operator	Joins multiple row sets from different sources with different cardinalities and produces a single output row set.
	Key Lookup operator	Performs a lookup of data from a lookup object such as a table, view, cube, or dimension.
	Match Merge operator	Data quality operator that identifies matching records and merges them into a single record.
	Name and Address operator	Identifies and corrects errors and inconsistencies in name and address source data.
	Pivot operator	Transforms a single row of attributes into multiple rows. Use this operator to transform data that contained across attributes instead of rows.
	Set Operation operator	Performs union, union all, intersect, and minus operations in a mapping.
	Sorter operator	Sorts attributes in ascending or descending order.

**Table 7–2 (Cont.) Data Flow Operators**





Icon	Operator	Description
	Splitter operator	Splits a single input row set into several output row sets using a boolean split condition.
	Table Function operator	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. You can use a table function operator as a target.
	Transformation operator	Transforms the attribute value data of rows within a row set using a PL/SQL function or procedure.
	Unpivot operator	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 7–3 lists the Pre/Post Process operators and the Mapping Parameter operators.

**Table 7–3 Pre/Post Processing Operators**




Icon	Operator	Description
	Mapping Input Parameter operator	Passes parameter values into a mapping.
	Mapping Output Parameter operator	Sends values out of a mapping.
	Post-Mapping Process operator	Calls a function or procedure after executing a mapping.
	Pre-Mapping Process operator	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 7–4 lists the Pluggable Mappings operators.

**Table 7–4 Pluggable Mapping Operators**

Icon	Operator	Description
	Pluggable Mapping operator	Represents a reusable mapping.
	Pluggable Mapping Input Signature operator	A combination of input attributes that flow into the pluggable mapping.
	Pluggable Mapping Output Signature operator	A combination of output attributes that flow out of the pluggable mapping.



## Adding Operators

The steps you take to add an operator to a mapping depend on the type of operator you select. This is because some operators are bound to workspace objects while others are not. As a general rule, when you add a data source or target operator, Warehouse Builder creates and maintains a version of that object in the Warehouse Builder workspace and a separate version for the Mapping Editor. For example, when you add a table operator to a mapping, Warehouse Builder maintains a separate copy of the table in the workspace. The separate versions are said to be *bound* together. That is, the version in the mapping is bound to the version in the workspace.

To distinguish between the two versions, this chapter refers to objects in the workspace either generically as *workspace objects* or specifically as *workspace tables*, *workspace views*, and so on. And 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 so that you can synchronize changing definitions of these objects. 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. In Warehouse Builder, you can synchronize automatically as described in "Managing Metadata Dependencies" in the *Warehouse Builder Online Help*. Alternatively, synchronize manually from within the Mapping Editor as described in "[Synchronizing Operators and Workspace Objects](#)" on page 7-25.

### To add an operator to a mapping:

1. Open the Mapping Editor.
2. From the **Mapping** menu, select **Add** and select an operator. Alternatively, you can drag an operator icon from the Palette and drop it onto the Mapping Editor canvas.

If you select an operator that you can bind to a workspace object, the Mapping Editor displays the **Add Mapping <operator name>** dialog box. For details on how to use this dialog box, see "[Add Operator Dialog Box](#)" on page 7-12.

If you select an operator that you cannot bind to a workspace object, Warehouse Builder may display a wizard or dialog box to assist you in creating the operator.

3. Follow any prompts Warehouse Builder displays 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.

## Adding Operators that Bind to Workspace Objects

You can bind the following operators to associated objects in the workspace using the [Add Operator Dialog Box](#):

- Cube operators
- Dimension operators
- External Table operators

- Flat File operators
- Materialized View operators
- Pre Mapping Process operators
- Post Mapping Process operators
- Sequence operators
- Table operators
- Transformation operators
- View operators

### **Add Operator Dialog Box**

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:

- [Create Unbound Operator with No Attributes](#)
- [Select from Existing Repository Object and Bind](#)

### **Create Unbound Operator with No Attributes**

Use this option when you want to use the Mapping Editor to define a new workspace object such as a new staging area table or a new target table.

After you select **Create unbound operator with no attributes**, type a name for the new object. 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 7-12. Next, to create the new workspace object in a target module, right-click the operator and select **Create and Bind**.

For an example on how to use this option in a mapping design, see "[Example: Using the Mapping Editor to Create Staging Area Tables](#)" on page 7-18.

### **Select from Existing Repository Object and Bind**

Use this option when you want to add an operator based on an object you 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 Control 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, 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.

## **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.

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 "[Setting Operator, Group, and Attribute Properties](#)" on page 7-24.

**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 "[Mapping Naming Conventions](#)" on page 7-14.

## 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.

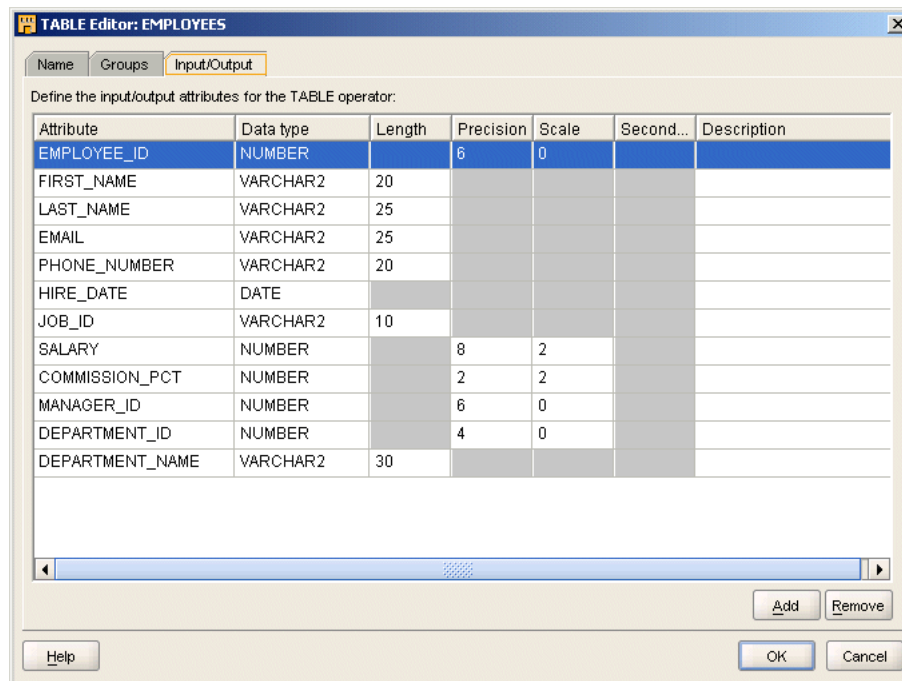
Depending on the operator, you can add and remove groups from the Groups tab. For example, you add input groups to Joiners and output groups to Splitters.

## 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.

Edit attribute information on the each of the remaining tabs.

[Figure 7-3](#) shows an Input/Output tab on the Operator Editor. In this example, the operator is a table and therefore has only the Input/Output tab. Other operators can have an Input tab and an Output tab.

**Figure 7-3 Input/Output Tab on the Operator Editor**

You can 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.

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.

## Mapping Naming Conventions

The rules for naming objects in the Mapping Editor depend on the naming mode you select in "Naming Preferences" on page 3-6. 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 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 mapping a name 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 conventions.

### 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 2000 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 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 on the mapping properties sheet. Right-click the mapping from the Design Center, select **Properties**, and view the names on the General tab.

Edit the description of the mapping as necessary. The description can be between 2 and 2,000 alphanumeric character and can contain blank spaces.

### **Naming Conventions for 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. This protects any expression or use of an attribute from corruption if it is manipulated within the operator.

### **Naming Conventions for Operators**

Business names for operator must meet the following requirements:

- The length of the operator name can be any string of 200 characters.
- The operator name must be unique on its attribute group, attribute and display set level with respect to its parent.

Physical names must meet the following requirements:

- All objects other than operators can contain a maximum of 30 characters. However, the limit is 28 for operators since Warehouse Builder reserves two characters for use when navigating through the OMB Scripting Language.
- The operator name must be unique on its group, attribute and display set level with respect to its parent.
- 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.
- Typically, you do not change bound names directly but indirectly by synchronizing from an operator to the workspace.
- When you rename the business name for an operator or attribute, Warehouse Builder propagates the new business name as the bound name when you synchronize. However, business names can be up to 200 character while bound names are limited to 30 characters. Therefore, Warehouse Builder uses the first 30 characters of the business name for the bound name.

## 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 7-5](#) describes the default display sets.

**Table 7-5 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

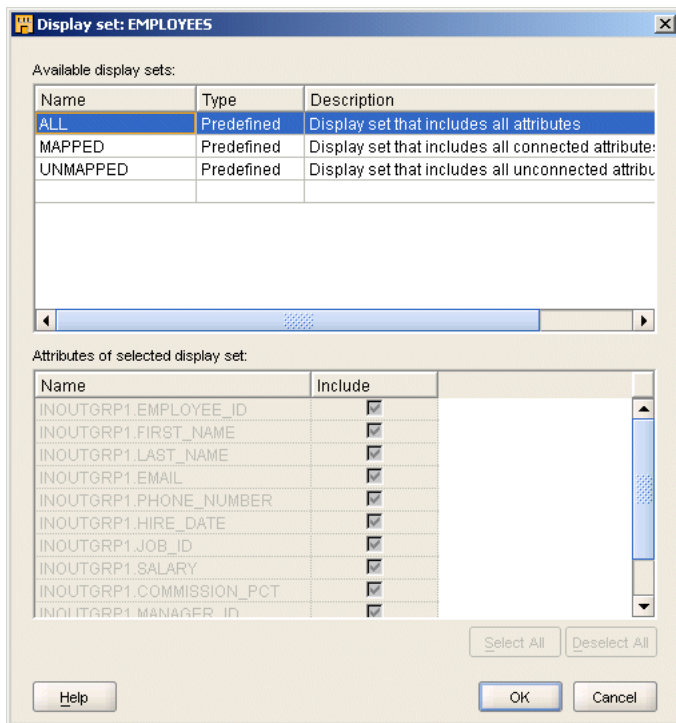
You can define display sets for any operator in a mapping.

**To define a display set:**

1. Right-click an operator, and select **Display Set**.

The Display Set dialog box is displayed as shown in [Figure 7-4](#).

**Figure 7-4 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 **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 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 View 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.

## Connecting Operators

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.

You can connect operators by one of the following methods:

- **Connecting Attributes:** Connect individual operator attributes to each other one at a time.
- **Connecting Groups:** Define criteria for connecting all the attributes between two groups.
- **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.

## Connecting Attributes

You can draw a line from a single output attribute of one operator to a single input attribute of another operator.

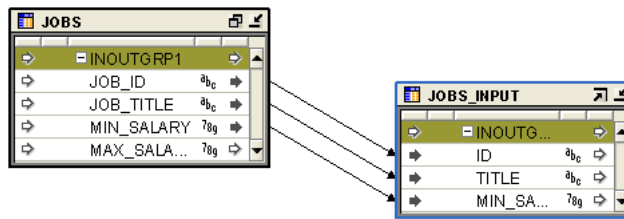
#### To connect attributes:

1. Click and hold down the 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 one through three until you create all the required data flow connections.

[Figure 7-5](#) displays a mapping with attributes connected.

**Figure 7-5 Connected Operators in a Mapping**

When connecting attributes, keep the following rules in mind:

- You cannot connect to the same input 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.

## 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 Connect Operators Dialog Box"](#) on page 7-19.

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 shown in ["Example: Using the Mapping Editor to Create Staging Area Tables"](#) on page 7-18.

### 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.

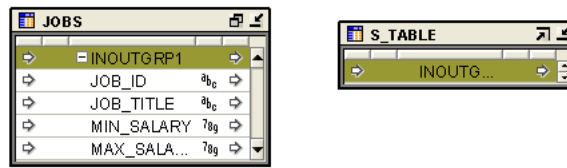
The following instructions describe how to create a staging table based on an existing source table. You can also use these instructions to create views, materialized views, flat files, and transformations.

#### To map a source table to a staging table:

1. In the Mapping Editor, add a source table.  
From the menu bar, select **Mapping**, select **Add**, then select **Data Sources/Targets**. In the **Data Sources/Targets** menu, select **Table Operator**.
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 7-6](#) with one source table and one staging area table without attributes.



**Figure 7-6 Unbound Staging Table without Attributes and Source Table**

3. With the mouse button 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 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 you specify.

### Using the Connect Operators Dialog Box

If you connect from one operator to a target operator with existing attributes, the Mapping Editor starts the Connect Operators dialog box.

Select one of the following criteria for copying and 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 one of the three options, select **Go**. The Connect Operators dialog box displays a list of connected attributes.

[Figure 7-7](#) displays the Connected attributes section.

**Figure 7-7 Connected Attributes**

Connected attributes:			
Map	Source attribute	Target attribute	Comments
<input checked="" type="checkbox"/>	COL1	COL1	
<input checked="" type="checkbox"/>	COL2	COL3	
<input type="checkbox"/>	COL3		Source will not be mapped

You can deselect attributes by clearing the **Map** check box. View the results of your selections under **Comments**.

When you select **OK**, Warehouse Builder copies the source attributes to the target group and connects the attributes.

### Copy Source Attributes to Target Group and Match

Use this option to copy source attributes to a target group that already contains attributes. Warehouse Builder connects from the source attributes to the new target

attributes based on the selections 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 for 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`.

After you set the matching criteria, click **Go**.

The **Displayed Mappings** field displays the possible connections between attributes which you can verify and deselect before implementing.

## Using Pluggable Mappings

You can reuse the data flow of a mapping by creating a pluggable mapping around the portion of the flow 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.

---

---

**Note:** The use of pluggable mappings requires the [Warehouse Builder Enterprise ETL Option](#).

---

---

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.

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 Global Explorer. 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 a Pluggable Mapping

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 two nodes, Standalone and Pluggable Mapping Folders. You can create pluggable mappings from either of these nodes.

### Standalone Pluggable Mapping

To create a standalone pluggable mapping:

1. Expand the Pluggable Mappings node in the Project Explorer.
2. Right-click Standalone, and select **New**.
3. This opens the Create Pluggable Mapping wizard, which guides you through the process of creating a new pluggable mapping. Click **Help** for information on the values to be entered on each page of the wizard.

Once you create a new pluggable mapping, Warehouse Builder opens the pluggable mapping editor and displays the name of the pluggable mapping on the title bar. The pluggable mapping editor is similar to the mapping editor, and you can add the desired operators from the palette to create a mapping.

A pluggable mapping is considered as an operator by the Warehouse Builder. You can insert it into any mapping. In the mapping editor, drag and drop Pluggable Mapping from the palette onto the canvas. This opens the Add Pluggable Mapping dialog box. You can select the desired pluggable mapping and add it to the mapping.

### Pluggable Mapping Folders

A folder is a grouping mechanism for pluggable mappings. You can keep your pluggable mappings private, or you can place them into folders (libraries) and then publish them so that others can access them for their design work. To create a new folder to store pluggable mappings:

1. Expand the Pluggable Mappings node in the Project Explorer.
2. Right-click Pluggable Mapping Folders, and select **New**. This opens the Create Pluggable Mapping Folder dialog box.
3. Enter a name for the folder and provide a description (optional).

4. Click **OK** to save the folder and exit the wizard.

The folder appears on the Project Explorer. The Pluggable Mapping Folders node gives you the option of creating a pluggable mapping either at the time of creating a folder or after creating the folder. You can also move a pluggable mapping to any folder on the tree.

At the time of creating the Pluggable Mapping folder, if you select the **Proceed to Pluggable Mapping Wizard** option, the Create Pluggable Mapping Wizard opens and you can create a new pluggable mapping.

If you do not select the option, only the pluggable mapping folder gets created. To create a pluggable mapping under this folder:

1. Under the Pluggable Mappings Folders node, right-click the folder and select **New**.
2. This opens the Create Pluggable Mapping wizard, which guides you through the process of creating a new pluggable 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 **Add**. To overwrite the default name assigned to the group, type over its name in the **Group** column. Enter its orientation as an input or output group in the **Direction** column. Enter an optional description of the group in the **Description** column.
- To remove a signature group, select the group you want to remove and click **Remove**.

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 you created.

If you defined multiple input signature groups, select the group to which you want to add attributes from the Group list box. Then click **Add** to add attributes. You can overwrite the default name given to each attribute by typing over the name in the Attribute column. You can change the data type of each attribute by clicking on its default data type and selecting a new data type from the resulting drop list. You can assign the length, precision, scale, and seconds precision by clicking the corresponding field and using the up and down arrows or typing in a number. Note that some of these fields are disabled depending on the data type you specify.

You can remove an attribute by selecting the attribute and clicking **Remove**.

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 you created.

If you defined multiple output signature groups, select the group to which you want to add attributes from the Group list box. Then click **Add** to add attributes. You can overwrite the default name given to each attribute by typing over the name in the Attribute column. You can change the data type of each attribute by clicking on its default data type and selecting a new data type from the resulting drop list. You can assign the length, precision, and scale by clicking the corresponding field and using the up and down arrows or typing in a number. Note that some of these fields are disabled depending on the data type you specify.

You can remove an attribute by selecting the attribute and clicking **Remove**.

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 Output Signature. Also, pluggable mapping Input and Output signatures can only be added within pluggable mappings. They cannot be added to normal mappings.

## Pluggable Mapping Editor

The pluggable mapping editor is similar to the mapping editor. Use the main panel to select and edit the operators that constitute your pluggable mapping. For more information on using this editor to design pluggable mappings, consult these topics:

- [Using Pluggable Mappings](#)
- [About the Mapping Editor](#)
- [Adding Operators](#)
- [Editing Operators](#)
- [Connecting Operators](#)
- [Setting Operator, Group, and Attribute Properties](#)
- [Synchronizing Operators and Workspace Objects](#)

## Setting Mapping Properties

When you select white space on the mapping canvas, the editor displays the mapping properties in the property inspector along the left side. You can set the following property for the mapping:

- [Target Load Order](#)

### Target Load Order

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, 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, Warehouse Builder assigns a random ordering as the default.

You can override the default load ordering by setting the Target Load Order property. If you make a mistake when reordering the targets, you can restore the default ordering by selecting the [Reset to Default](#) option.

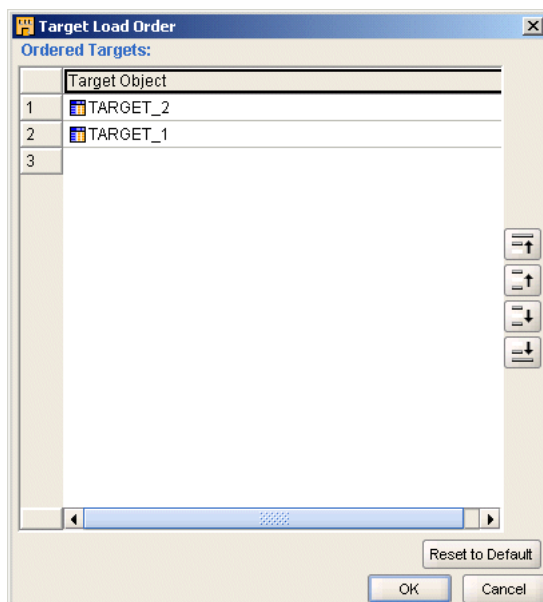
**To specify the loading order for multiple targets:**

1. Click whitespace in the mapping canvas to view the mapping properties in the Mapping Properties panel in the upper left corner.
2. Go to the Map Targets Load Order property and click the Ellipsis button on the right of this property.

Warehouse Builder displays the Targets Load Order dialog box which shows TARGET2 loading before TARGET1.

[Figure 7-8](#) displays the Target Load Order dialog box.

**Figure 7-8 Target Load Order Dialog Box**



3. To change the loading order, select a target and use the shuttle buttons on the right 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 change the mapping design such that the original order became invalid.

## Setting Operator, Group, and Attribute Properties

When you select an object on the canvas, the editor displays its associated properties in the Property panel along the left side.

You can view and set the following types of properties:

- **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.
- **Attribute Properties:** Properties that pertain to attributes in source and target operators. Examples of attribute properties are data type, precision, and scale.

## Synchronizing Operators and Workspace Objects

Many of the operators 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 you propagate changes:

**Synchronizing From a Workspace Object to an Operator:** After you begin using mappings in a production environment, there may be changes to the sources or targets that impact your ETL designs. Typically, the best way to manage these changes is through the Warehouse Builder Dependency Manager described in "Managing Metadata Dependencies" in the *Warehouse Builder Online Help*. Use the Dependency Manager to automatically evaluate the impact of changes and to synchronize all effected mappings at one time. Alternatively, in the Mapping Editor, you can manually synchronize objects as described in "Synchronizing From a Workspace Object to an Operator" on page 7-26.

**Synchronizing from an Operator to a Workspace Object:** When you make changes to an operator in a mapping, you may want to propagate those changes to its corresponding workspace definition. For example, the sources you imported and used in a mapping may have complex physical names for its attributes.

You can synchronize in the following method:

- **Synchronizing An Operator:** 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 An Operator

**To synchronize an operator, complete the following steps:**

1. Select an operator on the Mapping Editor canvas.
2. From the **Edit** menu, select **Synchronize** or right-click the header of the operator and select **Synchronize**.

The Synchronize Operator dialog box displays.

3. By default, Warehouse Builder selects the option for you to synchronize your selected operator with its associated object in the workspace. You can accept the default or select another workspace object from the list box.

In this step you also specify whether to perform inbound or outbound synchronization. Inbound synchronization synchronizes the data object with the mapping operator. Outbound synchronization synchronizes the mapping operator with the data object.

4. As an optional step, set the [Matching Strategies](#) and Synchronize strategy.
5. Click OK.

## Synchronizing From a Workspace Object to an Operator

In the Mapping Editor, you can synchronize from a workspace object for any of the following reasons:

- **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, attribute data type changes. To automatically propagate changes in a workspace object across multiple mappings, see "Managing Metadata Dependencies" in the *Warehouse Builder Online Help*.
- **Synchronize an operator with a new workspace object:** You can associate 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.
- **Prototype mappings using tables:** When working in the design environment, you could choose to design the ETL logic using tables. However, for production, you may want the mappings to source other workspace object types such as views, materialized views, or cubes.

## Synchronizing Operators based on Workspace Objects

[Table 7–6](#) lists operators and the types of workspace objects from which you can synchronize.

**Table 7–6 Operators Synchronized with Workspace Objects**

To: Operator	From: Workspace Object Type
Cube Operator	Tables, Views, Materialized Views, Flat Files, Dimensions, and Cubes
Dimension Operator	Tables, External Tables, Views, Materialized Views, Flat Files, Dimensions, and Cubes
External Table Operator	Tables, External Tables, Views, Materialized Views, Flat Files, Dimensions, and Cubes
Flat File Operator	Tables, External Tables, Views, Materialized Views, Flat Files, Dimensions, and Cubes
Key Lookup Operator	Tables only
Materialized View Operator	Tables, External Tables, Views, Materialized Views, Files, Dimensions, and Cubes
Post Mapping Process Operator	Transformations only
Pre Mapping Process Operator	Transformations only
Sequence Operator	Sequences only



**Table 7-6 (Cont.) Operators Synchronized with Workspace Objects**

To: Operator	From: Workspace Object Type
Table Operator	Tables, External Tables, Views, Materialized Views, Flat Files, Dimensions, and Cubes
Transformation Operator	Transformations only
View Operator	Tables, External Tables, Views, Materialized Views, Files, Dimensions, and Cubes

Note that when you synchronize from an external table operator, Warehouse Builder updates the operator based on the workspace external table only and not its associated flat file. To update an operator such as external table based on its associated flat file, see "Synchronizing an External Table Definition with a Record in a Flat File" in the *Warehouse Builder Online Help*.

## Synchronizing from an Operator to a Workspace Object

As you make changes to operators in a mapping, you may want to propagate those changes back to a workspace object. By synchronizing, you can propagate changes from the following operators: tables, views, materialized views, transformations, and flat file operators.

Synchronize from the operator to a workspace object for any of the following reasons:

- **Propagate changes:** Propagate changes 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.
- **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 7-7](#) lists the operators from which you can synchronize.

**Table 7-7 Outbound Synchronize Operators**

Mapping Objects	Create Workspace Objects	Propagate Changes	Replace Workspace Objects	Notes
External Tables	Yes	Yes	Yes	Updates the workspace external table only and not the flat file associated with the external table. See "Synchronizing an External Table Definition with a Record in a Flat File" in the <i>Warehouse Builder Online Help</i> .
Flat Files	Yes	Yes	No	Creates a new, comma-delimited flat file for single record type flat files only. Cannot replace an existing file.
Mapping Input Parameters	Yes	Yes	Yes	Copies input attributes and data types as input parameters.
Mapping Output Parameters	Yes	Yes	Yes	Copies output attribute and data types as return specification for the function.

**Table 7-7 (Cont.) Outbound Synchronize Operators**

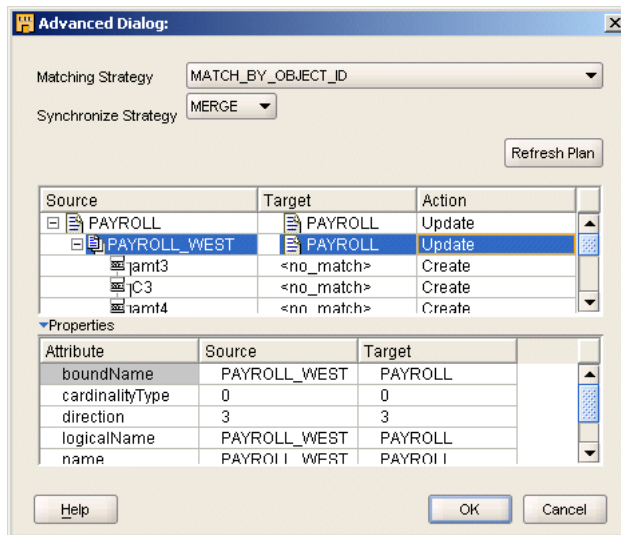
Mapping Objects	Create Workspace Objects	Propagate Changes	Replace Workspace Objects	Notes
Materialized Views	Yes	Yes	Yes	Copies attributes and data types as columns.
Tables	Yes	Yes	Yes	Copies attributes and data types as columns. Constraint properties are not copied.
Transformations	Yes	Yes	Yes	
Views	Yes	Yes	Yes	Copies attributes and data types as columns.

## 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 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 7-9](#), 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 7-9 Advanced Synchronizing Options**

## Matching Strategies

Set the matching strategies to determine how Warehouse Builder compares an operator to a workspace object. If synchronization introduces changes such as adding or deleting attributes in an operator, Warehouse Builder refreshes the Mapping Editor. If synchronizing removes an operator attribute, data flow connections to or from the attribute are also removed. If synchronizing adds an operator attribute, 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, it is interpreted as if the attribute were deleted and a new attribute added.

You can specify the following strategies for reconciling 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 identifiers of an operator attributes with those of a workspace object. 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 source object that do not correspond to attributes in the target object. This can occur when an attribute is added to the source or removed from the workspace object without properly synchronizing the change.

#### **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 you want to preserve the business names in the operator attributes. 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 target object, Warehouse Builder adds as new attributes.

## **Using DML Error Logging**

Error logging enables the processing of DML statements to continue despite errors being encountered 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 multi-table insert. It is useful in long-running, bulk DML statements.

Warehouse Builder provides error logging for the tables, views, and materialized views used in set-based PL/SQL mappings. To enable error logging, you set the Shadow table name property of the table, view, or materialized view. DML error logging is supported only for target schemas created on Oracle Database 10g Release 2 or later versions.

## About Error Tables

Error tables store error details. You can define error tables for tables, views, and materialized views only.

Error tables are used for the following purposes:

- DML error logging (including physical errors)
- Capturing logical errors when data rules are applied to tables, views, or materialized views

An error table is generated and deployed along with the base table, view, or materialized view. When you drop a data object, the shadow table associated with it is automatically dropped.

### Error Tables and DML Error Logging

When DML error logging is enabled for a data object by setting the Shadow table name property for the object, the error table contains the following:

- DML error columns, as described in [Table 7-8](#)
- all columns from the data object with which the shadow table is associated

**Table 7-8 DML Error Columns in Error Tables**

Column Name	Description
<code>ORA_ERR_NUMBER\$</code>	Oracle error number
<code>ORA_ERR_MSG\$</code>	Oracle error message text
<code>ORA_ERR_ROWID\$</code>	Rowid of the row in error (for update and delete)
<code>ORA_ERR_OPTYPE\$</code>	Type of operation: insert (I), update (U), delete (D)
<code>ORA_ERR_TAG\$</code>	Step or detail audit ID from the runtime audit data. This is the <code>STEP_ID</code> column in the runtime view <code>ALL_RT_AUDIT_STEP_RUNS</code> .

For scalar data types in the source data object, if no data rules are applied to the data object, 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, the columns in the error table are of the same data type as the source columns.

For example, the table `TEST` has the 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`, the data type for both `C1` and `C2` will be `VARCHAR2 (4000)`. If data rules are applied to `TEST`, `C1` will be `NUMBER` and `C2` will be of data type `VARCHAR2 (10)`.

## Error Tables and Data Rules

When one or more data rules are defined for a data object, the error table for this data object contains the following:

- Columns from the data object  
These columns are of the same data type and precision as the ones in the data object.
- DML error columns, as described in [Table 7–8](#)
- Data rule columns  
The data rule columns store details such as the operator that caused the data rule violation, the cause of the error, severity, and the audit run details.

## Using Error Tables for DML Error Logging and Data Rules

When you define data rules on a data object for which DML error logging is enabled, the error table generated by Warehouse Builder contains the columns from the data object, the data rules columns, and the DML error columns. The data type and precision of the columns from the data object are the same as the ones in the base data object. This could result in the failed inserts into the error table when errors occur during DML operations. For example, some errors, such as value too small, may cause error table insert failure.

Thus, if you want to perform DML error logging for a data object that has data rules applied, it is recommended that you create your own error tables. Ensure that the error table that you create contains the columns required for data rules and the DML error logging columns.

## Enabling DML Error Logging

DML error logging is generated for set-based PL/SQL mappings if the following conditions are satisfied:

- the Error table name property is set for the Table, View, or Materialized View operator.
- the PL/SQL Generated Mode configuration property of the module that contains the mapping is set to Oracle 10gR2, Oracle 11gR1, or Default.

If the value is set to Default, ensure that location associated with this module has the Version property set to 10.2 or 11.1.

### To enable error logging for a data object:

1. In the Project Explorer, right-click the data object for which DML error logging should be enabled, and select **Open Editor**.

The Data Object Editor for the data object is displayed.

2. On the canvas, select the data object.
3. In the Properties panel, specify a value for the Shadow table name property.

If you do not specify a shadow table name for a data object, DML error logging is not enabled for that object. However, when a data object has data rules associated with it, if you do not specify a 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`, the error table is called `EMP_ERR`.

When you use a data object in a mapping, the Error Table Name property for this data object is derived from the Shadow table name property of the data object.

---

---

**Note:** If you modify the error table name for a data object (using the Shadow table name property), you must synchronize all the operators bound to this data object.

---

---

## 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 property of the mapping. In the Project Explorer, right-click the mapping and select **Configure**. In the Maximum number of errors property, specify the maximum 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 7-32.

You can truncate the 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.

The Error table name and Truncate error table properties of Table, View, or Materialized View operators are not used for row-based code.

## DML Error Logging Limitations

DML error logging has certain limitations. DML error logging is not supported for non-scalar datatypes. In addition, each DML statement has specific limitations, which are listed in documentation related to that statement.

**See Also:** *Oracle Database SQL Language Reference* for limitations on DML error logging for each DML statement

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.

If you have an error table defined for a data object, you cannot upgrade the data object using the Upgrade option in the Control Center Manager. If you modify the Shadow table name property after the data object is deployed, you must first drop the data object and then redeploy it. If this data object was used in a mapping, ensure that you synchronize the mapping operator with the data object, drop the data object, redeploy the data object and the mapping.

## Debugging a Mapping

You can use the Mapping Editor to debug complex data flows you design in mappings. Once you begin a debug session and connect to a valid target schema, the debugging functions appear on the toolbar and under Debug on the Mapping Editor main menu. 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 behaves as expected. If you find problems, you can correct them and restart the debug session to ensure that the problems have been fixed before proceeding to deployment.

### Before you Begin

Ensure that you are connected to a Control Center and that the Control Center is running.

## Starting a Debug Session

To start a debug session, from the Mapping Editor, select **Debug** and then **Start**, or you can click Debug Start on the toolbar. The Mapping Editor switches to debug mode with the debug panels appearing in the bottom and the side of the editor, 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 displays to indicate 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 and test data must be defined for the database object. By default, the debugger uses the same source and target data that is currently defined for the non-debug deployment of the map.

## The Debug Panels of the Mapping Editor

When the Mapping Editor is opened in Debug mode it has new panels [Debug Info Panel](#) and [Debug Data Panel](#).

### Debug Info Panel

When the Mapping Editor is in Debug mode, the left middle panel is the Debug Info panel which 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 de-activate breakpoints. For more information, see "[Setting Breakpoints](#)" on page 7-35.

- **Test Data:** Displays a list of all data objects used in the mapping. The list also indicates which data objects have test data defined.

### Debug Data Panel

When the Mapping Editor is in Debug mode, the Debug Data panel is the right bottom panel. The Debug 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 you set. These watch tabs allow you to keep track 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 left bottom panel. It also contains the object type, the source, and a check mark that indicates that the database object has already been bound to the source or target operator.

The object type listed on the tab is determined by whether or not the column names in the data source 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 then 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 would then be deployed as a view when you run the mapping and the type will be 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, the name of the table is the name of the data operator prefixed by `DBG_`. (Note that, if this name already exists in the target, then Warehouse Builder adds a sequence number as a suffix to guarantee a unique object name.) Warehouse Builder creates the table in the target schema that you specified when you started the debug run. The debugger does not automatically drop that table, consequently you can always reuse it for other sessions. Constraints are not carried over for the new table.

When you create a new table, Oracle 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 are copied into the newly created table.

## Editing the Test Data

You can edit test data at anytime using the Define Test Data dialog box. If you change the binding of the operator to another database object, you must re-initialize 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**.

---



---

## Setting Breakpoints

If you are interested in how a specific operator is processing data, you can set a breakpoint on that operator which will 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 it is functioning as expected. Breakpoint settings are not stored when you close the mapping.

### To set or remove a breakpoint:

1. From the Mapping Editor, click an operator and then select **Debug** and then **Set Breakpoint**. You can also click the Breakpoint button 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 left bottom panel. If you are removing the breakpoint the name is removed. Use the **Clear** button on the Breakpoint tab to remove breakpoints.

2. Uncheck or check the breakpoints on the Breakpoint tab to disable or enable them.

## Setting Watches

The Step Data tab on the right bottom panel always shows the data for the current operator. If you want 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 in the case of sources and targets, the data that currently resides in the bound database objects. You can also

set watch points 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, click an operator and then select **Debug** and then **Set Watch**. You can also click the Set Watch button on the toolbar to toggle the watch on and off. The name of the operator will appear as an additional tab on the right bottom panel bottom containing the input and or output groups for the operator.

**To remove a watch:**

To remove a watch, again select the operator and use the watch button on the toolbar, use set watch from the debug menu or use toggle debug watch from the right mouse button menu.

## 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 **Re-initialize** from the Debug menu, or by clicking Re-initialize on the toolbar. Warehouse Builder generates the debug code and deploys the package to the target schema you specified.

You can choose to 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 use 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 then Warehouse Builder prompt you to designate the source with which to begin. For example, when two tables are mapped to a joiner, you will have to select the first source table 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 not stored and must be re-set each time you open a mapping.

As the debugger runs it generates debug messages whenever applicable. You can follow the data flow through the operators. The active operator is indicated by a red dashed box surrounding the operator.

## Debugging Mappings with Correlated Commit

How a mapping is debugged varies depending 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 then choose to go back and 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, 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 and 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 gets 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** or choose the Set as Starting Point menu item.

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 if the map had three source tables that were all connected to a single joiner. Assuming that each source table contains a large number of rows, too many rows to efficiently step through in the debugger (say more than 50000 rows). In this case, it would be a good idea to set the joiner operator as a starting point, and limit the row count for the one of the source tables to a more manageable number of rows (say 500) by using the Test Data Editor. In this case 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

map 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 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 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 map implementation, and the current step operator is set to the first executable source operator inside the pluggable map. The first executable source operator for the pluggable is one of the operators connected from the input signature operator.

You can now step through the pluggable map just as you would any other type of map. When the pluggable operator contains targets, the debugger loads these 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 that was just executed. When the pluggable 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**.

## Re-Initializing 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 re-initialize the debug session to continue debugging the mapping with the new changes. To re-initialize, click the re-initialize button on the toolbar or select the re-initialize menu item in the debug menu. Re-initializing both regenerates and re-deploys the debug code. After re-initialization, 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 window 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.

---

---

## 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 and to describe constraints between the activities. Constraints can be conditional branches, loops, parallel flows or serial dependencies. Activities can be mappings, transforms or external commands such as email, 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 "[Process for Defining and Using Schedules](#)" on page 11-17.

This chapter contains the following topics:

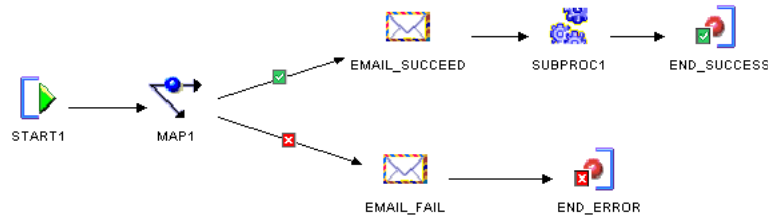
- [About Process Flows](#)
- [Instructions for Defining Process Flows](#)
- [About the Process Flow Editor](#)
- [Adding Activities to Process Flows](#)
- [Creating and Using Activity Templates](#)
- [About Transitions](#)
- [Expressions](#)
- [Defining Transition Conditions](#)

### About Process Flows

A process flow describes dependencies between Warehouse Builder mappings and external activities such as email, FTP, and operating system commands.

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 email 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 Scripting Language, as described in the *Oracle Warehouse Builder API and Scripting Reference*.

## About Process Flow Modules and Packages

Process flow modules allow you to group process flow packages. Process flow packages, in turn, allow you to group process flows. 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 packages but must be contained within the same module.

### 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.

## Instructions for Defining Process Flows

To define a process flow, refer to the following sections:

1. [Creating Process Flow Modules](#) on page 8-3
2. [Creating Process Flow Packages](#) on page 8-3
3. [Creating Process Flows](#) on page 8-4
4. [Creating and Using Activity Templates](#) on page 8-10
5. [Adding Activities](#) on page 8-7
6. [Connecting Activities](#) on page 8-14

7. Using Activities in Process Flows in the *Warehouse Builder Online Help*
8. [Using Parameters and Variables](#) on page 8-15
9. Configuring Process Flows in the *Warehouse Builder Online Help*
10. Validating and Generating Process Flows
11. 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 "[Process for Defining and Using Schedules](#)" on page 11-17.
12. Deploying Process Flows as described in "[The Deployment and Execution Process](#)" on page 11-7.

## Creating Process Flow Modules

Before working with process flows, create a process flow module. The module is a container by 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 Project Explorer and select **New**.

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 type in 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 Project Explorer, and prompts you to create a 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 Project Explorer and click **New**.

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, please 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 Project Explorer and click **New**.

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 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 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 "[Instructions for Defining Process Flows](#)" on page 8-2.

## About the Process Flow Editor

After you create a process flow module and package, use the Process Flow Editor to design and edit process flows. The Process Flow Editor includes a variety of activities that you can add and then connect with transitions to design a flow.

Activities represents units of work in a process flow. These units of work can involve components internal or external to Warehouse Builder. Transitions indicate the sequence and conditions to carry out the activities.

## Standard Editor Components

The Process Flow Editor has the following standard components common to most editors in Warehouse Builder:

- **Title Bar:** At the top of the Process Flow Editor, the title bar displays the name of the process flow.
- **Menu Bar:** Below the title bar, the menu bar provides access to the Process Flow Editor commands.
- **Toolbar:** Below the menu bar, the toolbar provides icons for commonly used commands.



- **Canvas:** The canvas provides the work space where you design and modify process flows. When you first create a new process, the Process Flow panel is displayed with a Start activity and an End activity.
- **Palette:** When you first start the Process Flow Editor, Warehouse Builder displays the palette along the left side. The Process Flow Editor contains activity icons that you can drag and drop on to the canvas. You can relocate the palette anywhere on the editor. You can choose to hide or display the palette by clicking the collapse icon on the palette.
- **Indicator Bar:** On the lower panel, under the Bird's Eye View panel and Canvas panel, you can see mode icons, indicators, and descriptions.

In the left corner are Naming Mode, Rename Mode, Read/Write, and Validation Mode.

In the right corner are the percent zoom indicator and the navigation mode. In the preceding figure, the zoom level is at 100% and the navigation mode is set to Select Mode.

## Process Flow Editor Windows

You can resize windows by placing your mouse on the border of the window, pressing the mouse button when the double sided arrow appears, and dragging your mouse to indicate the desired size.

You can move a window by placing the mouse pointer on the title bar of the window and then dragging the window to the required position.

To show or hide windows, select **Window** from the menu bar and either activate or deactivate the check mark corresponding to the window.

### Explorer

When you first start the editor, Warehouse Builder displays an Explorer panel for the editor in the upper left corner. The explorer provides a tree listing of all the activities on the canvas and their parameters. When you select an activity on the canvas, Warehouse Builder navigates to the activity on the explorer.

### Object Details

When you first start the editor, Warehouse Builder displays the Object Details panel on the left side. This panel displays the properties for all activities and their parameters. Select an activity either from the canvas or the explorer and Warehouse Builder displays its properties. If you select an activity parameter in the Explorer, then the object details window displays the properties for that parameter. You can edit properties displayed with a white background but not those with a gray background.

### Palette

When you first start an editor, Warehouse Builder displays the palette along the left side and it contains activity icons that you can drag and drop onto the canvas. You can relocate the palette anywhere on the editor. You can choose to hide or display the palette by clicking on Operator Palette listed under **View** in the menu bar.

### Bird's Eye View

Use the Bird's Eye View panel to navigate large and complex process flows.

## Opening the Process Flow Editor

### To open the Process Flow Editor:

1. From the Process Flows node in the Project Explorer, select a process flow module. If no process flow modules are listed, then create a process flow module as described in "[Creating Process Flow Modules](#)" on page 8-3.
2. Select a process flow package from a process flow module. If no process flow packages are listed, then create a process flow package as described in "[Creating Process Flow Packages](#)" on page 8-3.
3. Select a process flow from the Project Explorer. If no process flows are listed in the process flow package, then right-click the process flow package and select **Create Process Flow**.

Warehouse Builder prompts you to name the process flow and then starts the editor for you.

4. To open an existing process flow, double-click the process flow in the Project Explorer.

Alternatively, select a process flow and then from the Edit menu, select **Open Editor**. You can also, select a process flow and press **Ctrl+O**. You can also, right-click a process flow, and select **Open Editor**.

Warehouse Builder displays the Process Flow Editor in the [Select mode](#).

## Navigating the Process Flow Editor

The Process Flow Editor includes a variety of tools to assist you in navigating, selecting, and viewing objects on the canvas. Commands you will use frequently when designing Process Flows include the following:



### Select mode

Use the select mode to select objects on the canvas. When you select an activity, the editor displays a blue border around the activity. You can edit, move, or delete the activity.

You can edit the activity using the object details window in conjunction with the Available Objects tab in the editor explorer window. When you select a transition, the editor changes the arrow from black to blue. Edit the transition in the object details.

To activate the Select mode, click the icon in the toolbar or select **Edit** and **Select Mode** from the menu.



### Navigation Edge

Navigation Edge assists you in navigating complex designs on the canvas. Select the icon from the toolbar and then select an activity on the canvas. When you release the mouse button, Warehouse Builder navigates to the next activity in the flow and moves the focus to that activity. To navigate backward through a flow, select the navigation edge icon and then select the last activity in the flow.

For navigating and displaying complex designs in the editor, you may find the following tools useful:

- Pan
- Interactive Zoom
- Zoom In

- Zoom Out
- Fit in Window
- Auto Layout
- Center
- Expand Child Graph
- Visit Child Graph
- Return to Parent Graph

## Adding Activities to Process Flows

You can add activities in a process flow using the Explorer tree in the Warehouse Builder.

### 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 editor 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 emails 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 8-10. For email, for example, use an email 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" in the *Warehouse Builder Online Help*.

### Adding Activities

**To add an activity to a process flow:**

1. View the activities listed in the palette located along the left side of the editor.

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.

2. 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.

3. To accept the default name, press **Enter**. To change the name, type in the new name.

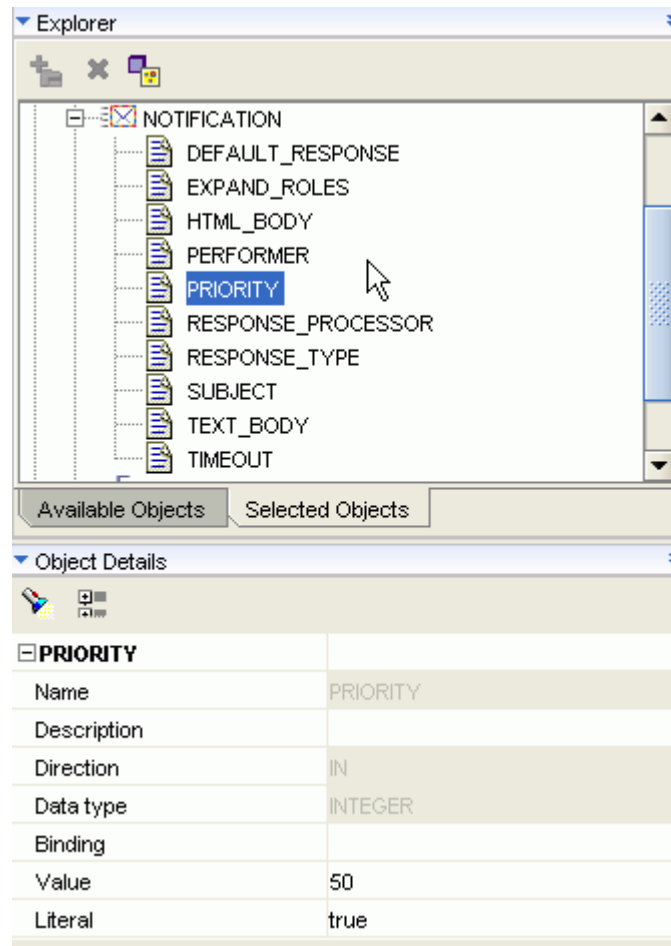
The editor lists the activity on the explorer pane located at the left side of the editor and in the object details pane along the left side.

4. In Object Details pane, enter the parameters for the activity.

These parameters vary according to the type of activity. For each parameter, Warehouse Builder defines a read-only **Name**, **Direction**, and **Data Type**. And for each parameter, you can specify values for **Binding**, **Literal**, **Value**, and **Description**.

For example, [Figure 8–2](#) shows the parameters for a notification activity which includes DEFAULT\_RESPONSE, EXPANDED\_ROLES, HTML\_BODY, PERFORMER, PRIORITY, RESPONSE\_PROCESSOR, RESPONSE\_TYPE, SUBJECT, TEXT\_BODY, and TIMEOUT.

**Figure 8–2 The 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 "Using Activities in Process Flows" in the *Warehouse Builder Online Help*.

**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 type in 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.

Table 8–1 provides the Literal Value Type, Format, and Example.

**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	[+ -]YYYYYYYYYY-MM	+000000001-01
DMINTERVAL	[+ -]DDDDDDDDDD 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 which you can override by typing in a new value or using the field **Binding**. In the absence of a list of possible values or a default value, you must type in a value.

**Description**

You can type 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 email 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 Project Explorer, 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**.
3. Assign a name for the folder.

Consider creating a folder for each type of template 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**.

---

---

Follow the prompts in the Create Activity Template Wizard to complete the [Name and Description Page](#), the [Parameters Page](#), and the wizard summary page.

5. See "[Using Activity Templates](#)" on page 8-11 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 you select in "[Naming Preferences](#)" on page 3-6. 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 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 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 2000 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 between 2 and 2000 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
- FTP
- File Exists
- Manual
- Notification
- Set Status
- Sqlplus
- User Defined
- Wait

## Parameters Page

The wizard displays parameters based on the type of activity you previously selected in the [Activity Templates](#).

Enter default values for the activity. When you use the activity template in a process flow, you can retain or edit the default values.

In [Figure 8-3](#), for example, you could edit the default values for the email subject and message body to contain the name of the mapping.

**Figure 8-3 Parameters Page for Email Activity Template**

	NAME	DATATYPE	DIR...	DEFAULT VALUE
1	SMTP_SERVER	STRING	IN	our_email_server
2	PORT	INTEGER	IN	1234
3	FROM_ADDRESS	STRING	IN	
4	REPLY_TO_ADDRESS	STRING	IN	
5	TO_ADDRESS	STRING	IN	admin@abccompany.com
6	CC_ADDRESS	STRING	IN	
7	BCC_ADDRESS	STRING	IN	
8	IMPORTANCE	STRING	IN	High
9	SUBJECT	STRING	IN	Failed Mapping: <mapping name>
10	MESSAGE_BODY	STRING	IN	<Mapping name> failed. Please correct the error...

Buttons: Help, < Back, Next >, Finish, Cancel

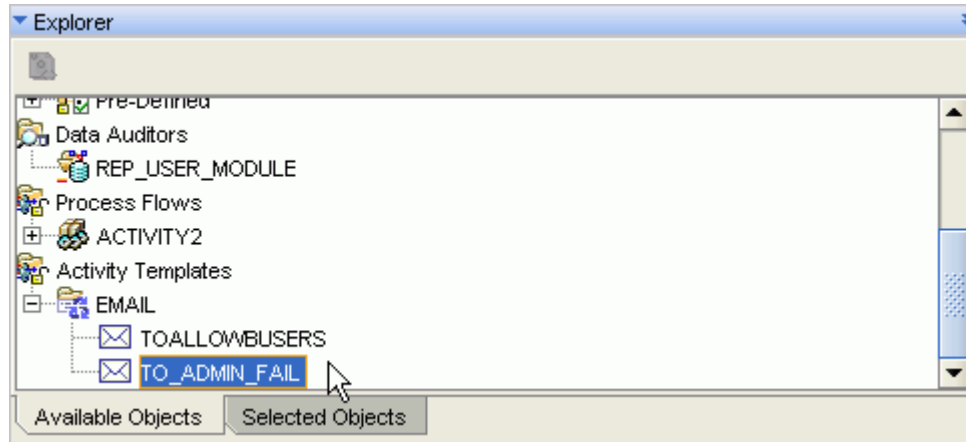
## Using Activity Templates

Complete the following steps to use an activity template:

1. In the Project Explorer, navigate to the process flow module under the Process Flows node.
2. To open the Process Flow Editor, right-click the Process Flow module and select **Open Editor**.
3. In the Process Flow Editor, click the **Available Objects** tab in the Explorer panel and expand **Activity Templates**.

Figure 8–4 displays the Explorer window with the activity template expanded.

**Figure 8–4 Explorer Panel with an Activity Template Selected**

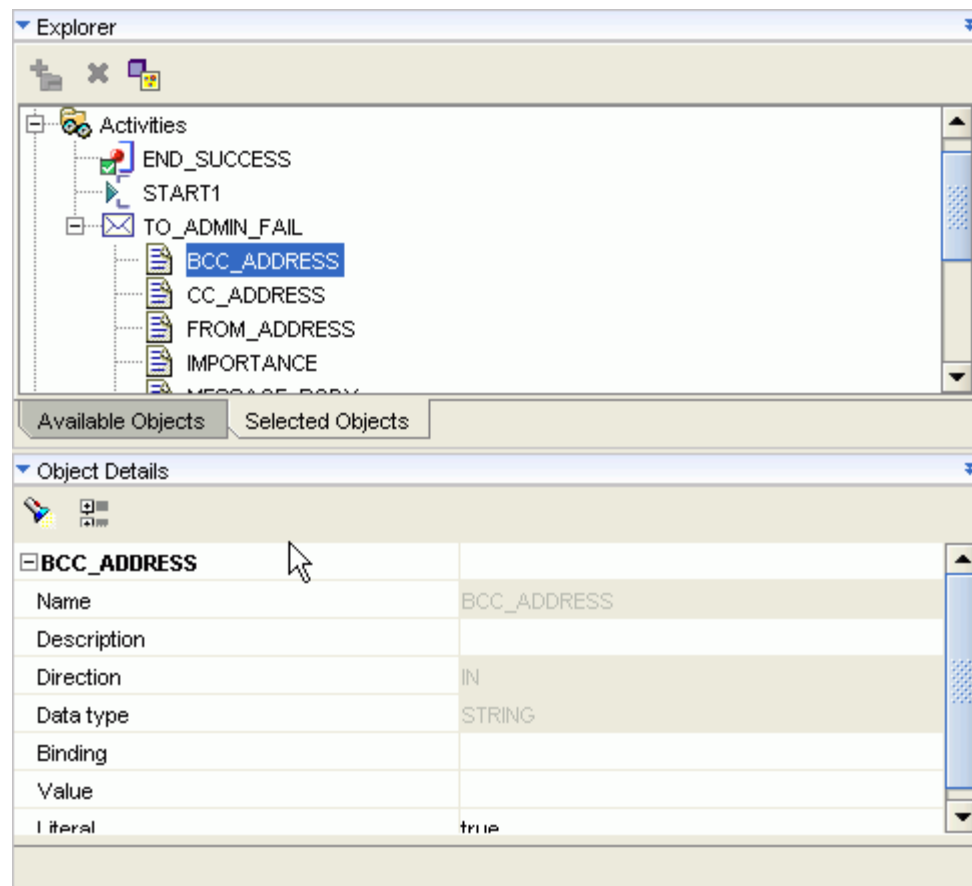


4. Drag and drop the activity template onto the canvas.  
Activity templates in a process flow acts like regular activities.
5. To edit the activity, be sure to click the Selected Objects tab in the Explorer window and then edit the activity in the Object Details panel.

Figure 8–5 displays the Explorer panel with the BCC\_ADDRESS parameter of the EMAIL activity selected.



Figure 8-5 Editing an Activity Template



## 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 and it 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, 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 towards 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 explorer and object selector windows.

3. In the object selector window, view or edit the following attributes:

**Name:** The editor assigns a default name which you can change.

**Description:** You can type 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 conditions, click the button in the Condition as described in "[Defining Transition Conditions](#)" on page 8-17. If 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 "Using Activities in Process Flows" in the *Warehouse Builder Online Help*.

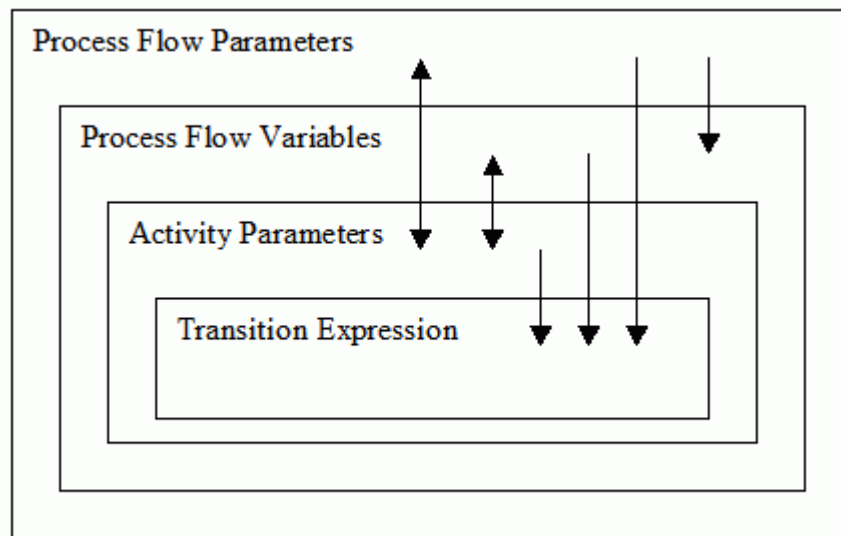
## Using Parameters and Variables

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. 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 8–6 shows the direction in which the data is passed.

**Figure 8–6** Relationship between the scope and the direction in which the data is passed



Process flows follow the following rules for allowing the data to be passed between data stores:

1. Process flow variables can be initialized from 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 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.

A variable may not pass data out to a Process Flow parameter. So, this is accomplished by the use of an Assign operator, which can be bound to the variable and the parameter.

## 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 will have access to the outer scope which encloses it. So, an expression for an activity parameter will be able to 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, in the case where 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.' So the expression may reference any PL/SQL function that is accessible to the Control Center user.

## Global Expression Values

Warehouse Builder also 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'

**Table 8–2 (Cont.) Global Expression Values**

Identifier	Type	Description
RETURN_RESULT_NUMBER	NUMBER	Enumeration of RESULT_RESULT1 = SUCCESS2 = WARNING3 = ERROR
RETURN_CODE	NUMBER	Integer 0-255, specific to 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.

Identifier	Type	Description
SUCCESS	NUMBER	SUCCESS enumerated value
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 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 8–3.

**Table 8–3 Types of Conditions for Transitions**

Icon	Transition	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.
	Warning	The process flow continues only if the preceding activity ends with warnings.
	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.

Extended transition is valid only for Notification activities because they are the only activity that return an extended result. The activity acquires this icon when set to an outcome of #MAIL, #NOMATCH, #TIE, or #TIMEOUT. Table 8–4 lists the output and the description of the Extended transition.

**Table 8-4 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 email notification and so it was canceled.
#TIMEOUT	The notification did not receive a response within the configured amount of time.

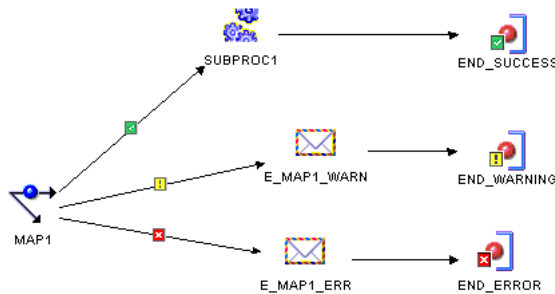
If the activity has only one outgoing activity, then you can specify any of the conditions listed in [Table 8-3](#) or leave the transition as unconditional.

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 8-7](#) 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, two transition conditions would be true and the process flow would be invalid.

**Figure 8-7 Outgoing Transition Conditions**



---

---

# Understanding Performance and Advanced ETL Concepts

Use this chapter as a guide for creating ETL logic that meets your performance expectations.

This chapter includes 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](#)

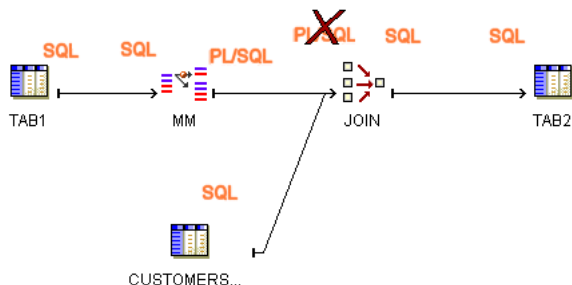
Warehouse Builder generates code for PL/SQL mappings that meet the following criteria:

- 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 dataflow 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 taking note of the input and output code types for each operator in the mapping.

For example, you can see that the mapping in [Figure 9-1](#) is invalid because the Match-Merge operator `MM` generates PL/SQL output but the subsequent Join operator accepts SQL input only.

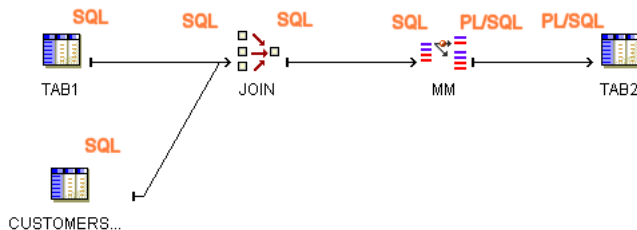
**Figure 9–1 Mapping Violates Input Requirement for Join Operator**



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 9–2 displays a mapping in which source tables are joined before the Match-Merge. Figure 9–3 displays a mapping in which the results from the Match-Merge are loaded into a staging table before performing the join.

**Figure 9–2 Valid Mapping Design with Sources Joined Before Match-Merge**



**Figure 9–3 Valid Mapping Design with Staging Table**

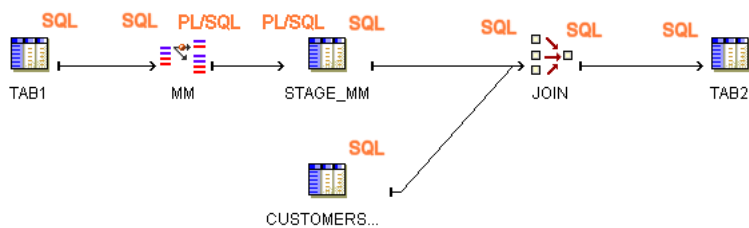


Table 9–1 and Table 9–2 list the implementation types for each Warehouse Builder operator. These tables also indicate whether or not PL/SQL code includes the operation associated with the operator in the cursor. This information is relevant in determining which operating modes are valid for a given mapping design. It also determines what auditing details are available during error handling.



**Table 9–1 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 PL/SQL	Yes, except when loading= UPDATE and database is not 10g 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 10g or higher.	Yes	Yes. Not part of cursor.
Sequence as source	SQL	Yes	Yes	Yes, part of cursor.

**Table 9–2 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 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 PL/SQL	Yes	Yes	Yes
Filter	SQL PL/SQL	Yes	Yes	Yes
Joiner	SQL	Yes	Yes, only if part of the cursor.	Yes, only if part of the cursor.
Key Lookup	SQL	Yes	Yes, only if part of the cursor.	Yes, only if part of the cursor.
Mapping Input Parameter	SQL PL/SQL	Yes	Yes	Yes
Mapping Output Parameter	SQL PL/SQL	Yes	Yes	Yes
Match-Merge	SQL input PL/SQL output (PL/SQL input from XREF group only)	No	Yes	Yes. Not part of cursor.

**Table 9–2 (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
Name and Address	PL/SQL	No	Yes	Yes. Not part of cursor.
Pivot	SQL PL/SQL	Yes	Yes	Yes
Post-Mapping Process	Irrelevant	Yes, independent of dataflow.	Yes	Yes
Pre-Mapping Process	Irrelevant	Yes, independent of dataflow.	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 PL/SQL	Yes	Yes	Yes
Table Function	SQL or PL/SQL input 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 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](#)
- [Row Based](#)
- [Row Based \(Target Only\)](#)
- Set based fail over to row based
- Set based fail over to row based (target only)

The default operating mode you select depends upon the performance you expect, the amount of auditing data 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 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 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, Joins, and Key Lookups. 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.

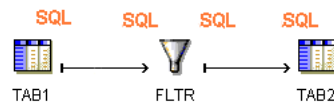
These general rules are explained in the following sections.

### Set Based

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 9-4 shows a simple mapping and the associated logic 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 9-4 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 dataflow that cannot be performed in SQL, Warehouse Builder does not generate set based code and displays an error when you execute the package in set based mode.

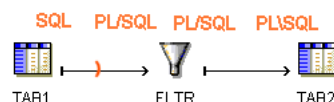
For target operators in a mapping, the loading types INSERT/UPDATE and UPDATE/INSERT are always valid for set based mode. Warehouse Builder supports UPDATE loading in set based mode only when the Oracle Database is 10g or higher. Warehouse Builder also supports the loading type DELETE in set based mode. For a complete listing of how Warehouse Builder handles operators in set based mappings, see Table 9-2 on page 9-3.

### Row Based

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 9-5 shows a simple mapping and the associated logic 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 9-5 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
- Join
- Key Lookup
- Sequence
- Set
- Sorter

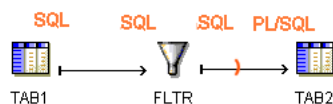
For the preceding operators to be included in the cursor, do not directly precede it by an operator that generates PL/SQL code. In other words, you cannot run the mapping in row-based mode if it contains a Transformation implemented as procedure, a Flat File used as a source, a Match-Merge, or Name and 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)

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.

Table 9–6 shows a simple mapping and the associated logic 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 9–6 Simple Mapping Run in Row Based (Target Only) Mode**



Row based (target only) 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 9-7.

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 9-10.

## 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, 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 impacts all affected targets uniformly. For more information about correlated commit, see ["Committing Data from a Single Source to Multiple Targets"](#) on page 9-7.

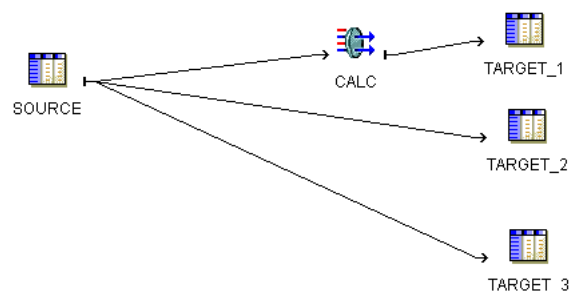
**Manual:** Select manual commit control for PL/SQL mappings when you want to interject complex business logic, perform validations, or run other mappings before committing data. For examples, see ["Embedding Commit Logic into the Mapping"](#) on page 9-9 and ["Committing Data Independently of Mapping Design"](#) on page 9-10.

### 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 impacts all affected targets uniformly.

[Figure 9-7](#) 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 9-7 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 impacts 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 include an Advanced 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 9–3](#) shows the valid combinations you can select.

**Table 9–3 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, 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 9–7](#) on page 9-7. 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, 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 or post mapping 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 or post mapping 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 or post mapping 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 Project Explorer, 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, refer to ["About Manual Commit Control"](#) on page 9-10.

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 9-4](#). Warehouse Builder uses the default setting for any optional parameters that you do not specify.



**Table 9–4 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 <i>status</i> .  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 property set to Manual.

In the Project Explorer, right-click the mapping and select **Configure**. Under the Code Generation Options, set the Commit Control property 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 9–4](#). 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 9–4](#) 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 9-12.

### 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 Sqlplus 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 property set to Manual.

In the Project Explorer, right-click the mapping and select **Configure**. Under Code Generation Options, set the Commit Control property to Manual.

4. Design a process flow using a sqlplus 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 sqlplus 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 sqlplus activity.

In the editor explorer, select **SCRIPT** under the sqlplus activity and then double-click **Value** in the object inspector.

[Figure 9-8](#) displays the Explorer panel and the Object Inspector panel with **SCRIPT** selected.

**Figure 9–8 Specifying a Script in the Sqlplus Activity**

7. Optionally apply a schedule to the process flow as described in ["Process for Defining and Using Schedules"](#) on page 11-17.
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, you must add a column and define 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 9-14.

3. In the mapping properties, view the Target Load Order property by clicking the Ellipses 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, use the Target Load Order dialog box to define the loading order.

For more information, see ["Target Load Order"](#) on page 7-23.

4. Ensure that the Use Target Load Ordering configuration property 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 Mapping 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 9–1](#), 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 9–1 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 9–1](#), 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 9–1](#) contains a field Employee ID in both Employee and Payroll records, 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 9–5](#) and [Table 9–6](#)) to maintain the relationship between the master and detail records. Use the Mapping Sequence operator to generate this additional value.

[Table 9–5](#) represents the target table containing the master records from the file in [Example 9–1](#) on page 9-14. 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 9–5 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 9–6 represents the target table containing the detail records from the file in Example 9–1 on page 9-14. 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 9–5.

**Table 9–6 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 9-19.

---

This procedure outlines general steps for building such a mapping. Additional detailed instructions are available at:

- [Using the Import Metadata Wizard](#) on page 4-5
- Flat File Operator in the *Warehouse Builder Online Help*
- [Adding Operators that Bind to Workspace Objects](#) on page 7-11
- Sequence Operator in the *Warehouse Builder Online Help*
- Configuring Mappings Reference in the *Warehouse Builder Online Help*

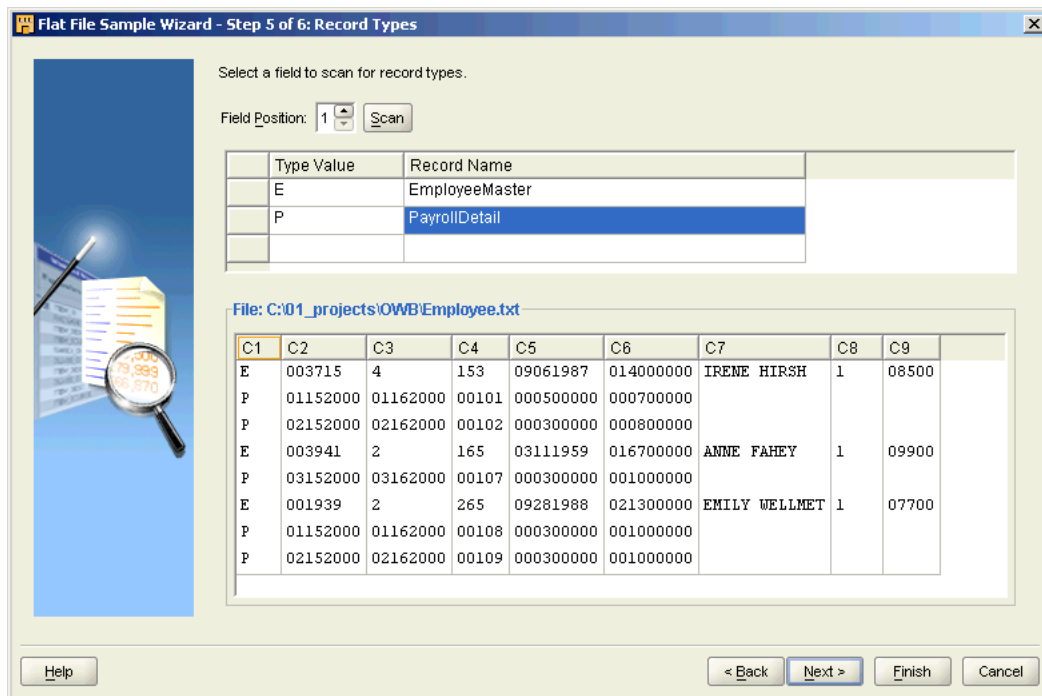
**To extract from a master-detail flat file and maintain master-detail relationships, use the following steps:**

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.

Figure 9–9 shows the Flat File Sample Wizard for a multiple-record-type flat file containing 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`.

**Figure 9–9 Naming Flat File Master and Detail Record Types**



- Drop a Flat File operator onto the Mapping Editor canvas and specify the master-detail file from which you want to extract data.
- Drop a Sequence operator onto the mapping canvas.
- 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 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.

- 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 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.

- Map all of the necessary flat file master fields to the master table and detail fields to the detail table.

Figure 9–10 displays the mapping of the fields.

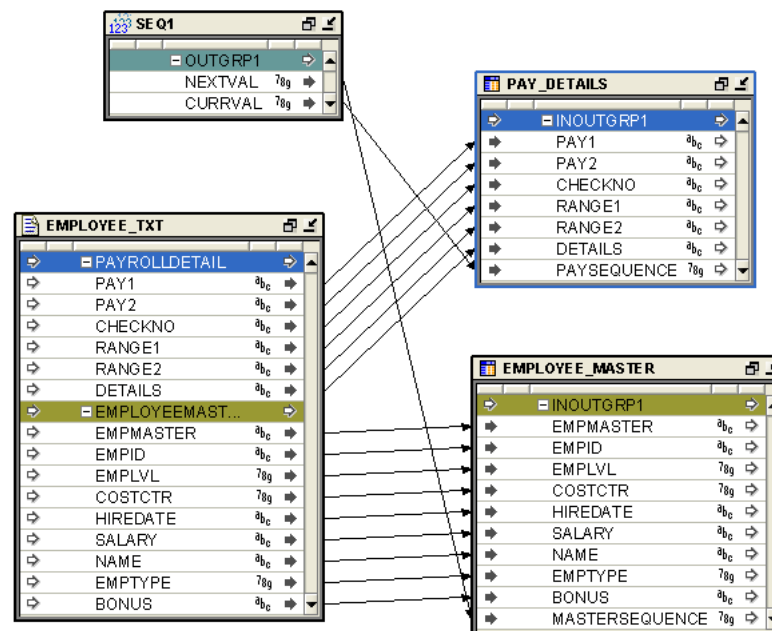
- Map the Sequence NEXTVAL attribute to the additional sequence column in the master table.

Figure 9–10 displays the mapping from the NEXTVAL attribute of the Sequence operator to the master table.

- Map the Sequence CURRVAL attribute to the additional sequence column in the detail table.

Figure 9–10 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 9–10 Completed Mapping from Master-Detail Flat File to Two Target Tables**



- 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 (see "Sequence Operator" in the *Warehouse Builder Online Help*).
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 (see "Sequence Operator" in the *Warehouse Builder Online Help*).

### 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 purpose of the initial load. Operators such as the Aggregator, Filter, or Match and 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 on importing flat file sources, see "[Using the Import Metadata Wizard](#)" on page 4-5.
- For additional information on using the Flat File as a source, see "Flat File Operator" in the *Warehouse Builder Online Help*.
- For additional information on using Table operators, see "[Adding Operators that Bind to Workspace Objects](#)" on page 7-11.
- For additional information on using the Data Generator operator, see "Data Generator Operator" in the *Warehouse Builder Online Help*.
- For additional information on using the Constant operator, see "Constant Operator" in the *Warehouse Builder Online Help*.
- For additional information on configuring mappings, see "Configuring Mappings Reference" in the *Warehouse Builder Online Help*.

### 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, as shown in [Figure 9-9](#) on page 9-16. This will make it easier to 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.

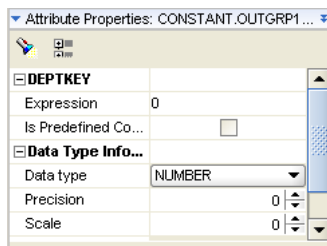
7. Map all of the necessary flat file master fields to the master table and detail fields to the detail table, as shown in [Figure 9–12](#) on page 9-21.
8. Map the Data Generator operator's RECNUM attribute to the RECNUM columns in the master and detail tables, as shown in [Figure 9–12](#) on page 9-21.
9. Add a constant attribute in the Constant operator.

If the master row unique identifier column is of a CHAR data type, make the constant attribute a CHAR type with the expression ' \* '.

If the master row unique identifier column is a number, make the constant attribute a NUMBER with the expression ' 0 '.

[Figure 9–11](#) shows the expression property of the constant attribute set to ' 0 '. This constant marks all data rows as "just loaded".

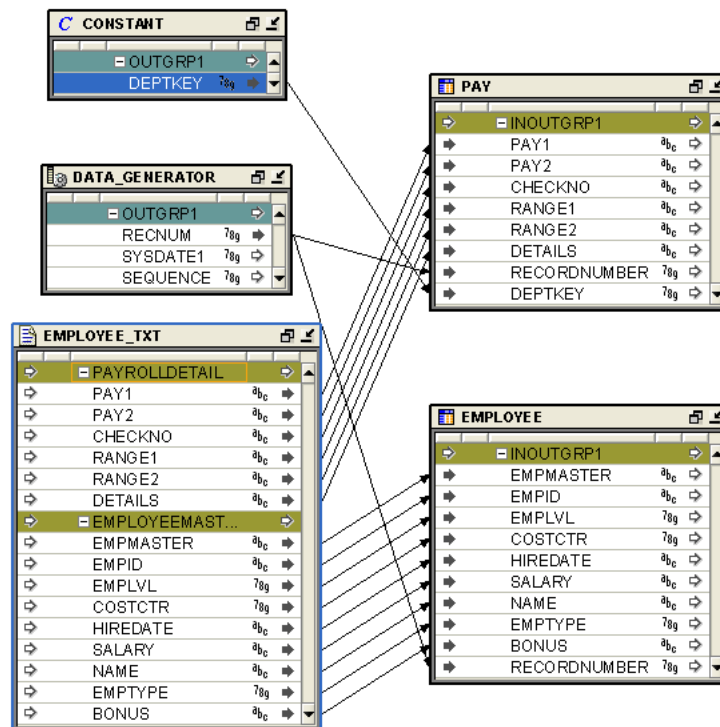
**Figure 9–11 Constant Operator Properties**



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 9–12](#) 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 9–12 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](#)

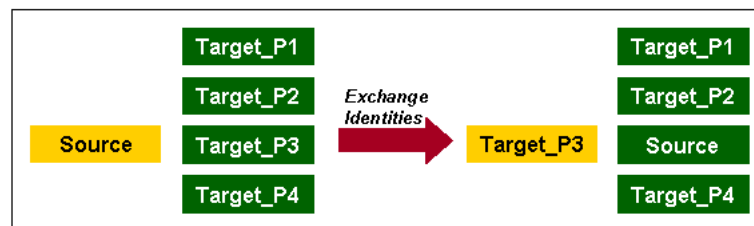
## About Partition Exchange Loading

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 9–13 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, 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 9i completes the loading process without data movement.

**Figure 9–13 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 Project Explorer, right-click a mapping and select **Configure**.  
Warehouse Builder displays the Configuration Properties window.
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, 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 9-24.
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, 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.

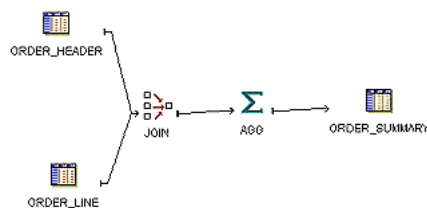
- **Indirect PEL:** By default, Warehouse Builder creates and maintains a temporary table that stages the source data before initiating the partition exchange process. For example, use Indirect PEL when the mapping includes a remote source or a join of multiple sources.
- **Direct PEL:** You design the 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 9-14](#) 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 9-14 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 9–14](#) 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.

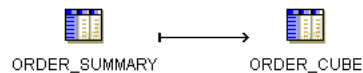
#### To instantly load data to a target using Direct PEL:

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 9–14](#) is ORDER\_SUMMARY and should be of the same structure as the final target, ORDER\_CUBE in [Figure 9–15](#).

2. Create a second mapping that loads data from the staging table to the final target such as shown in [Figure 9–15](#). Configure this mapping to use Direct PEL.

**Figure 9–15 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 "Using Partitions" in the *Warehouse Builder Online Help*.

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 Data Object Editor 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:

Ydddd

Ydddd\_Qd

Ydddd\_Qd\_Mdd

Ydddd\_Qd\_Mdd\_Ddd

Ydddd\_Qd\_Mdd\_Ddd\_Hdd

Ydddd\_Qd\_Mdd\_Ddd\_Hdd\_Mdd

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 Data Object 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 Project Explorer, right-click the table and select **Configure**. The Configuration Properties dialog box 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 9-26.

## 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" in the *Warehouse Builder Online Help* for instructions on using transportable modules



---

---

# Introducing Oracle Warehouse Builder 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 transformation using Warehouse Builder. It also describes how to import transformation definitions.

This chapter contains the following topics:

- [About Transforming Data Using Warehouse Builder](#)
- [About Transformations](#)
- [About Transformation Libraries](#)
- [Defining Custom Transformations](#)
- [Editing Custom Transformations](#)
- [Importing PL/SQL](#)

## 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 the 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 10-2.

- **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 also use custom transformations that you define in the Mapping Editor through the Transformation operator. For more information on these operators, see "Data Flow Operators" in the *Warehouse Builder Online Help*.

## Benefits of Using Warehouse Builder for Transforming Data

Warehouse Builder enables you to reuse PL/SQL as well as to write your own PL/SQL transformations. To enable faster development of warehousing solutions, Warehouse Builder provides custom transformations written in PL/SQL. These custom transformations can be used in Warehouse Builder mappings.

Because SQL and PL/SQL are versatile and proven languages widely used by many information professionals, the time and expense of developing an alternative transformation language is eliminated by using Warehouse Builder. With Warehouse Builder, you can create solutions using existing knowledge and a proven, open, and standard technology.

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.

## 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

Transformations, in Warehouse Builder, can be categorized as follows:

- [Predefined Transformations](#)
- [Custom Transformations](#)

The following sections provide more details about these types of 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 Oracle Library that consists of built-in and seeded functions and procedures. You can directly use these predefined transformations to transform your data. For more information on the Oracle Library, see "[Types of Transformation Libraries](#)" on page 10-4.

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 "Transformations" in the *Warehouse Builder Online Help*.

### Custom Transformations

A custom transformation is one this is created by the user. Custom transformations can use predefined transformations as part of their definition.

Custom transformations contains 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 Global Explorer. It is also created automatically under the Transformations node of every Oracle module in the Project Explorer.

Functions can be defined by the user or imported from a database. A function transformation takes 0-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 Global Explorer. It is also automatically created under the Transformations node of each Oracle module in the Global Explorer.

Procedures can be defined by the user or imported from a database. A procedure transformation takes 0-n input parameters and produces 0-n output parameters.

- **Packages:** The Packages category contains packages, which in turn contain functions, procedures, and PL/SQL types. This category is available under the Custom node of the Public Transformations node in the Global Explorer. It is also automatically created under the Transformations node of each Oracle module in the Global Explorer.

PL/SQL packages can be created or imported in Warehouse Builder. The package body may be modified. The package header, which is the signature for the function or procedure, cannot be modified.

- **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 using the Packages node in the Transformations node of the Project Explorer. It is also available under every package that you define in the following path of the Global Explorer: Public Transformations -> Custom -> Packages.

For more information about creating custom transformations, see ["Defining Custom Transformations"](#) on page 10-5.

In addition to the above categories, you can also import PL/SQL packages. Although you can modify the package body of an imported package, you cannot modify the package header, which is the signature for the function or procedure. For more information on importing PL/SQL packages, see ["Importing PL/SQL"](#) on page 10-14.

## 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 project. This library contains the standard Oracle Library and an additional library for each Oracle module defined within the project.

Transformation libraries are available under the Public Transformations node of the Global Explorer in the Design Center.

## Types of Transformation Libraries

Transformation libraries can be categorized as follows:

- **Oracle Library**

This is a collection of predefined functions from which you can define procedures for your Global Shared Library. The Oracle Library is contained in the Global Explorer. 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 Oracle Library.

- **Global Shared 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 Global Shared 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 on creating transformations in the Global Shared Library, see "[Defining Custom Transformations](#)" on page 10-5.

When you deploy a transformation defined in the Global Shared Library, the transformation is deployed to the location that is associated with the default control center.

## Accessing Transformation Libraries

Since 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 using the following:

- **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. Since 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 Global Shared Library and the predefined transformations from the Oracle Library.

- Add Transformation Operator Dialog Box

The Transformation operator in the Mapping Editor enables you to add transformations, both from the Oracle library and the Global Shared Library, to a mapping. You can use this operator to transform data as part of the mapping.

- Create Function Wizard, Create Procedure Wizard, Edit Function Dialog Box, or Edit Procedure Dialog Box

The Implementation page on these wizards or the Implementation tab of these editors enable 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.

## 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 Global Shared Library or to a module in a project.

### Custom Transformations in the Global Shared Library

Custom transformations that are part of the Global Shared 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 Global Shared 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 Global Explorer to define custom transformations that can be used across all projects in the workspace. [Figure 10-1](#) displays the Global Explorer used to create such transformations.

#### To create a custom transformation in the Global Shared Library:

1. From the Global Explorer, expand the Public Transformations node and then the Custom node.

Warehouse Builder displays 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.

2. Right-click the type of transformation you want to define and select **New**.

For example, to create a function, right-click **Functions** and select **New**. To create PL/SQL types, expand the package in which you want to create the PL/SQL type, right-click **PL/SQL Types** and select **New**.

3. For functions and procedures, Warehouse Builder displays the Welcome page of the Create Function Wizard or the Create Procedure wizard respectively. For PL/SQL types, Warehouse Builder displays the Welcome page of the Create PL/SQL Type Wizard.

Click **Next** to proceed. See "[Defining Functions and Procedures](#)" on page 10-7 for more information about the other pages in the wizard. For more information about creating PL/SQL types, see "[Defining PL/SQL Types](#)" on page 10-8.

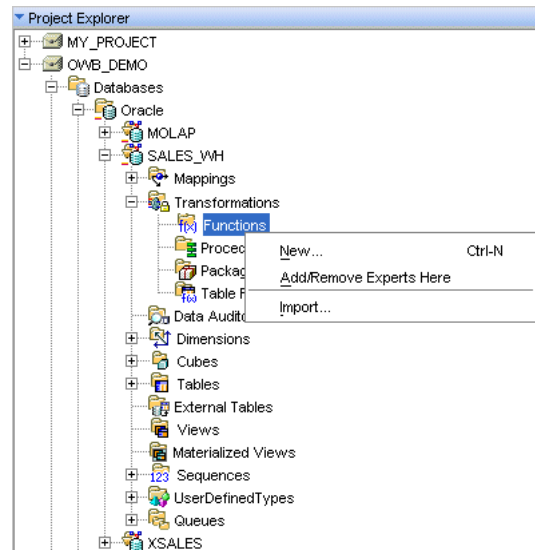
For packages, Warehouse Builder displays the Create Transformation Library dialog box. Provide a name and an optional description for the package and click **OK**. The new package is added to the Packages node. You can subsequently create procedures, functions, or PL/SQL types that belong to this package.

## Custom Transformations in a Project

Sometimes, you may need to define custom transformations that are required only in the current module or project. In this case, you can define custom transformations in an Oracle module of a project. Such custom transformations are accessible from all the projects in the current workspace. 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.

Figure 10–1 displays the Project Explorer from which you can create custom transformations that are accessible within the project in which they are defined.

**Figure 10–1 Creating Custom Transformations in an Oracle Module**



### To define a custom transformation in an Oracle module:

1. From the Project Explorer, expand the Oracle warehouse module node and then the Transformations node.
2. Right-click the type of transformation you want to create and select **New**.

For example, to create a package, right-click **Packages** and select **New**. To create PL/SQL types, expand the package node under which you want to create the type, right-click **PL/SQL Types** and select **New**.

For functions or procedures, Warehouse Builder displays the Welcome page of the Create Function Wizard or the Create Procedure Wizard respectively. For PL/SQL Types, the Welcome page of the Create PL/SQL Type Wizard is displayed. Click **Next** to proceed.

See "[Defining Functions and Procedures](#)" on page 10-7 for information about the remaining wizard pages. For more information about creating PL/SQL types, see "[Defining PL/SQL Types](#)" on page 10-8.

For packages, Warehouse Builder opens the Create Transformation Library dialog box. Provide a name and an optional description for the package and click **OK**. The package is added under the Packages node. You can subsequently create procedures, functions, or PL/SQL types that belong to this package.



## Defining Functions and Procedures

Use the following pages of the Create Function Wizard or Create Procedure Wizard to define a function or procedure.

- [Name and Description Page](#)
- [Parameters Page](#)
- [Implementation Page](#)
- [Summary Page](#)

### Name and Description Page

You use the Name and Description page 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 6-6.
- **Description:** Represents the description of the custom transformation. This is an optional field.
- **Return Type:** Represents the data type of the value returned by the function. You select a return type from the available options in the list. This field is applicable only for functions.

### Parameters Page

Use the Parameters page to define the parameters, both input and output, of the transformation. Specify the following details for each parameter:

- **Name:** Enter the name of the parameter.
- **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 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 at the time of executing the function or procedure.

### Implementation Page

Use the Implementation page to specify the implementation details, such as the code, of the transformation. To specify the code used to implement the function or procedure, click **Code Editor**. Warehouse Builder displays the Code Editor window. This editor contains two panels. The upper panel displays the code and the lower panel displays the function signature and messages.

When you create a function, the following additional options are displayed:

- **Function is deterministic:** This hint helps to avoid redundant function calls. If a stored function was called previously with the same arguments, 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.

- **Enable function for parallel execution:** This option declares that a stored function can be used safely in the child sessions of parallel DML evaluations. The state of a main (logon) session is never shared with child sessions. Each child session has its own state, which is initialized when the session begins. The function result should not depend on the state of session (static) variables. Otherwise, results might vary across sessions.

### Summary Page

The Summary page provides a summary of the options that you chose on the previous wizard pages. Click **Finish** to complete defining the function or procedure. Warehouse Builder creates the function or procedure and displays it under the corresponding folder under the Public Transformations and Custom nodes in the Global Explorer.

## 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.

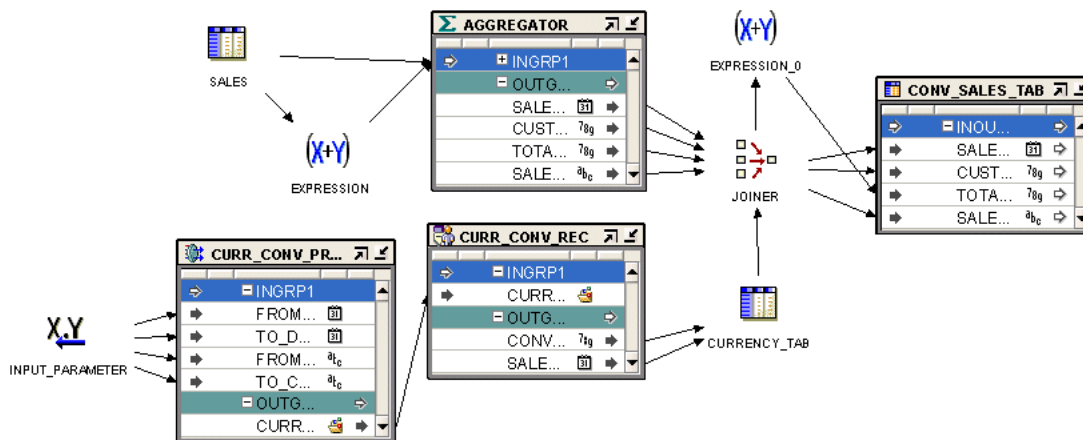
### Usage Scenario for PL/SQL Types

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 US Dollar. Thus all sales values must be converted to US Dollar. Since 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 10–2 displays the mapping that you use to obtain the sales amount in a specified currency using PL/SQL record types

Figure 10–2 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 Global Explorer, 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 Project Explorer and Global Explorer of the Design Center. For more details about creating PL/SQL types, see ["Defining PL/SQL Types"](#) on page 10-8.

Use the Create PL/SQL Types Wizard to create PL/SQL types. 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 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 10-8.

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.
- **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. The Seconds Precision is used only for `TIMESTAMP` 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 when you create 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, you can search for it by typing 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, 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.

## 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 Project Explorer, expand the Oracle module in which the transformation is created. Then expand the Transformations node.

To edit a transformation that is part of the Global Shared Library, from the Global Explorer, expand the Public Transformations node, and then the Custom node.

2. Right-click the name of the function, procedure, or package you want to edit and select **Open Editor**.

For functions or procedures, the Edit Function or Edit Procedure dialog box is displayed. Use the following tabs to edit the function or procedure definition:

- [Name Tab](#)
- [Parameters Tab](#)
- [Implementation Tab](#)

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.

### Name Tab

Use the Name tab to edit the name and description of the function or procedure. For functions, you can also edit the return data type.

### Parameters Tab

Use the Parameters tab to edit, add, or delete new parameters for a function or procedure. You can also edit and define the attributes of the parameters. The contents

of the Parameters tab are the same as that of the Parameters page of the Create Transformation Wizard. For more information about the contents of this page, see "[Parameters Page](#)" on page 10-7.

### Implementation Tab

Use the Implementation tab to review the PL/SQL code for the function or procedure. Click **Code Editor** to edit the code. The contents of the Implementation tab are the same as that of the Implementation page of the Create Transformation Wizard. For more information on the contents of the Implementation page, see "[Implementation Page](#)" on page 10-7.

## 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 Project Explorer, expand the Oracle module that contains the PL/SQL type. Then expand the Transformations node.  
To edit a PL/SQL type stored in the Global Shared Library, expand the Public Transformations node in the Global Explorer, 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 Editor**.

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.

## Importing PL/SQL

Use the Import Metadata Wizard to import PL/SQL functions, procedures, and packages into a Warehouse Builder project. You can edit, save, and deploy the imported PL/SQL functions and procedures. You can also view and modify imported packages.

The following steps describe how to import PL/SQL packages from other sources into Warehouse Builder.

### To import a PL/SQL function, procedure, or package:

1. From the Project Explorer, expand the project node and then Databases node.
2. Right-click an Oracle module node and select **Import**.

Warehouse Builder displays the Welcome page of the Import Metadata Wizard.

3. Click **Next**.
4. Select **PL/SQL Transformation** in the Object Type field of the Filter Information page.
5. Click **Next**.

The Import Metadata Wizard displays the Object Selection page.

6. 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.
7. Click **Next**.

The Import Metadata Wizard displays the Summary and Import page.

8. Verify the import information. Click **Back** to revise your selections.
9. Click **Finish** to import the selected PL/SQL transformations.

Warehouse Builder displays the Import Results page.

10. 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 Oracle module into which you imported the data.

## Restrictions on Using Imported PL/SQL

The following restrictions apply to the usage 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.



---

---

# 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 to this, Warehouse Builder also supports multiple physically different implementations of the same object definitions.

This chapter describes the implementation environment in Warehouse Builder. It also describes how to create and use schedules to automate the execution of ETL logic.

This chapter contains the following topics:

- [About Deployment and Execution in Warehouse Builder](#)
- [The Deployment and Execution Process](#)
- [Configuring the Physical Details of Deployment](#)
- [About Schedules](#)
- [Process for Defining and Using Schedules](#)

## About 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 from the logical objects in a Warehouse Builder workspace.

The data objects created when you designed the target schema are logical definitions. Warehouse Builder stores the metadata for these data objects in the workspace. To create these objects physically on 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. Use the Design Center or the Control Center Manager to deploy objects.

---

---

**Note:** Whenever you deploy an object, Warehouse Builder automatically saves all changes to all design objects to the workspace. You can choose to display a warning message by selecting **Prompt for commit** on the Preferences dialog box.

---

---

Deploying a mapping or a 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.

After deploying a mapping or a process flow, you must explicitly start the scripts, as described in "Starting the ETL Process" in the *Warehouse Builder Online Help*.

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.

You can deploy directly from the Design Center navigation tree or using the Control Center Manager.

---

---

**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.

---

---

## 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 can undo and redo an upgrade. This action is not available for some object types, such as schedules.
- **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.

## 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. Double-click the status to view details about the warning.
- **Failed:** Indicates that deployment of the object failed. Double-click the status to view detailed information about why the deployment failed.

## About Execution

For objects that contain ETL logic such as mappings, process flows, and transformations, there is an additional step of execution. Execution is the process of executing the ETL logic defined in the deployed objects.

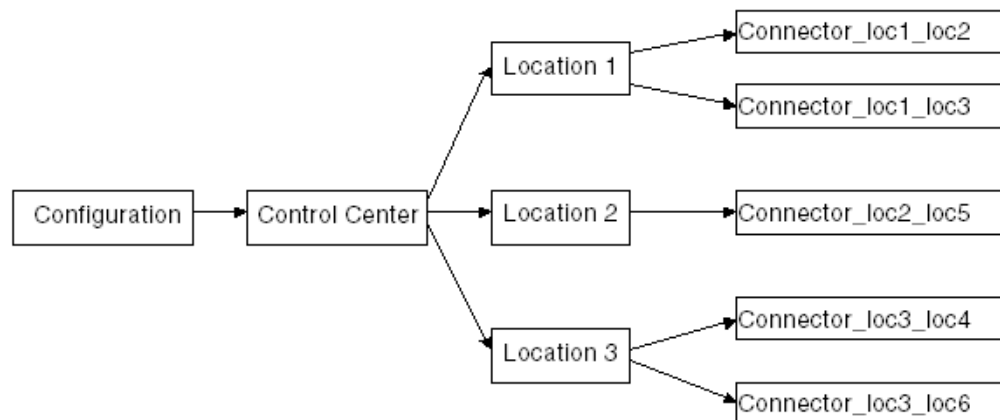
For example, you define a mapping that sources data from a table, performs transformations on the source data, and loads it into the target table. When you deploy this mapping, the PL/SQL code generated for this mapping is stored in the target schema. When you execute this mapping, the ETL logic is executed and the data is picked up from the source table, transformed, and loaded into the target table.

## About the Warehouse Builder Implementation Environment

To enable multiple physical implementations of a single design, Warehouse Builder uses a combination of the following: configurations, control centers, and locations.

Figure 11–1 describes the relationship between these components.

**Figure 11–1 Relationship Between Configurations, Control Centers, and Locations**



Configurations specify physical object properties that correspond to the environment to which the objects are deployed. A named configuration must be associated with a control center. You can change the control center that is associated with a

configuration. For more information about configurations, see "[About Configurations](#)" on page 11-12.

A control center refers to a repository on a target machine and it manages a set of source and target locations. A control center can be associated with only one named configuration at a time. For more information about control centers, see "[About Control Centers](#)" on page 11-4.

A location corresponds to the database, file, or application that Warehouse Builder sources data from or deploys data to. A location can be owned by only one control center. Each location can have one or more connectors that provide connections to other locations. For more information about locations and connectors, see "[About Locations](#)" on page 11-5 and "[About Connectors](#)" on page 11-6.

When you deploy objects, Warehouse Builder creates these objects in the target associated with the named configuration currently being used.

## About Control Centers

A Control Center stores detailed information about every deployment and execution, which you can access either by object or by job, including:

- The current deployment status of each object.
- A history of all deployment attempts for each object.
- A history of all ETL executions for each mapping and process flow.
- A complete log of messages from all deployment jobs and execution job details.

Any workspace can be used as a Control Center. Each workspace has a default Control Center which points to itself. For example, when the workspace called `REP_OWNER` is used to contain design metadata, then its default Control Center will also use the workspace called `REP_OWNER`.

You can use the default Control Center to deploy to the local system, or you can create additional Control Centers for deploying to different systems. Only one Control Center is active at any time and this is the one associated with the current [Active Configuration](#).

You can also access deployment and execution using the Repository Browser, as described in "Auditing Deployments and Executions" in the *Warehouse Builder Online Help*.

### Creating a Control Center

1. In the Connection Explorer, right-click Control Centers and select **New**.  
The Create Control Center dialog box is displayed.
2. Complete the dialog box. Click the **Help** button for additional details.

You can also create a Control Center using the Create Configuration Wizard.

### Activating a Control Center

The status bar in the Design Center displays the [Active Configuration](#). A named configuration is associated with only one Control Center. Objects are deployed to the control center associated with the active configuration.

#### To activate a Control Center:

1. In the Project Explorer, create or edit a configuration so that it uses the Control Center.

Refer to "[Creating New Configurations](#)" on page 11-13.

2. Activate that named configuration.

Refer to "[Activating Configurations](#)" on page 11-13.

## About Locations

Locations enable you to store the connection information to the various files, databases, and applications that Warehouse Builder accesses for extracting and loading data. Similarly, locations also store connection information to ETL management tools and Business Intelligence tools. For a detailed listing, see "[Supported Sources and Targets](#)" on page 4-2.

Oracle Database locations and file locations can be sources, targets, or both. For example, you can use a location as a target for storing temporary or staging tables. Later, you can re-use that location as a source to populate the final target schema.

In some cases, such as with flat file data, the data and metadata for a given source are stored separately. In such a case, create a location for the data and another for the metadata.

### Automatically Created Locations

During installation, Warehouse Builder creates an Oracle location named `OWB_REPOSITORY_LOCATION`. This location provides the connection details to the Warehouse Builder workspace. You cannot rename or delete the workspace location. Only a database administrator can change the password. To prevent unauthorized access to the database administrator password, all users are restricted from deploying to the workspace location.

### Creating Locations

In addition to the automatically created locations, you can create your own locations that correspond to target schemas that you want to use as sources or targets.

#### To create a location:

1. In the Connection Explorer, expand the Locations node and then the node that represents the type of location you want to create.

For example, to create an Oracle database location, expand the Locations node, the Databases node, and then the Oracle node.

2. Right-click the type of location and select **New**.

The Create <location\_type> Location dialog box is displayed.

3. Complete the dialog box. Click the **Help** button for additional details.

### Using SQL\*Net to Create Locations

When you create Oracle locations of type SQL\*Net, you must set up a TNS name entry for these locations. The TNS name must be accessible from the Oracle Database home. To do this, run the Net Configuration Assistant from the Oracle Database home.

While setting up a TNS name for use during deployment and execution of maps and process flows, the TNS name must be accessible from the Warehouse Builder home used to run the control center service. To make the TNS name accessible, run the Net Configuration Assistant from the Warehouse Builder home. Next, restart the control center service so that it can pick up the changes.

### **About Locations, Passwords, and Security**

Considering that all Warehouse Builder users can view connection information in a location, note that the passwords are always encrypted. Furthermore, Warehouse Builder administrators can determine whether or not to allow locations to be shared across users and persisted across design sessions. By default, locations are not shared or persisted.

**See Also:** *Oracle Warehouse Builder Installation and Administration Guide* for more information about managing passwords

### **Registering and Unregistering Locations**

All modules, including their source and target objects, must have locations associated with them before they can be deployed.

Registering a location establishes a link between the workspace and the locations of source data and deployed objects. You can change the definition of a location before it is registered, but not afterward. After the location is registered, you can only change the password. To further edit a location or one of its connectors, you must first unregister the location. Unregistering deletes the deployment history for the location.

Locations are registered automatically by deployment. Alternatively, you can explicitly register a location in the Control Center.

#### **To register a location:**

1. Open the Control Center Manager and select a location from the navigation tree.
2. From the File menu, select **Register**.  
The Location dialog box is displayed.
3. Check the location details carefully.  
Click **Help** for additional information.
4. Click **OK**.

#### **To unregister a location:**

1. Open the Control Center Manager and select a location from the navigation tree.
2. From the File menu, select **Unregister**.
3. Click **OK** to confirm the action.

### **Deleting Locations**

To delete a location, right-click the location in the Connection Explorer and select **Delete**. If the delete option is not available here, this indicates that the location has been registered in a control center and is likely being utilized. Verify that the location is not in use, unregister the location in the Control Center Manager, and then you can delete the location from the Connection Explorer.

## **About Connectors**

A connector is a logical link created by a mapping between a source location and a target location. The connector between schemas in two different Oracle Databases is implemented as a database link, and the connector between a schema and an operating system directory is implemented as a database directory.

You do not need to create connectors manually if your user ID has the credentials for creating these database objects. Warehouse Builder will create them automatically the first time you deploy the mapping. Otherwise, a privileged user must create the objects and grant you access to use them. You can then create the connectors manually and select the database object from a list.

**See Also:**

- *Oracle Database SQL Language Reference* for more information about the `CREATE DATABASE LINK` command
- *Oracle Database SQL Language Reference* for more information about the `CREATE DIRECTORY` command

**To create a database connector:**

1. In the Connection Explorer, expand the Locations folder and the subfolder for the *target* location.
2. Right-click **DB Connectors** and select **New**.  
The Create Connector wizard opens.
3. Follow the steps of the wizard. Click the **Help** button for specific information.

**To create a directory connector:**

1. In the Connection Explorer, expand the Locations folder and the subfolder for the *target* location.
2. Right-click **Directories** and select **New**.  
The Create Connector dialog box opens.
3. Click the **Help** button for specific information about completing this dialog box.

## 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 with the object settings and the Control Center that you want to use.
2. Deploy 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 11-8.

3. Review the results of the deployment. If an object fails to deploy, then fix the problem and try again.
4. Start the ETL process.  
For information about starting the ETL process, see "[Starting ETL Jobs](#)" on page 11-11.
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.
7. Repeat these steps.

---

---

**Note:** Warehouse Builder automatically saves all changes to the workspace before deployment.

---

---

## Deploying Objects

Deployment is the process of creating physical objects in a target location from the metadata using your generated code. 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 Project Explorer or using the Control Center Manager.

Deployment from the Project Explorer is restricted to the default action, which may be set to Create, Replace, Drop, or Update. To override the default action, use the Control Center Manager, which provides full control over the deployment process.

### To deploy from the Project Explorer:

Select the object and click the Deploy icon on the toolbar. You can also select the object, and then choose **Deploy** from the Design menu.

Status messages appear at the bottom of the Design Center window. For notification that deployment is complete, select **Show Deployment Completion Messages** in your preferences before deploying.

### To deploy from the Control Center Manager:

1. Open a project.
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. Refer to "[Creating Additional Configurations](#)" on page 11-14.

3. In the Control Center Manager navigation tree, expand the location node containing the object to be deployed. Select the objects to be deployed.

You can select multiple objects by holding down the Ctrl key while selecting the objects.

4. Set the deployment action for the selected objects in the Object Details panel.
5. Click the Deploy icon.

## Deploying Business Definitions to Oracle Discoverer

After you create your business definitions, you can deploy them to Oracle Discoverer. The method used to deploy business definitions depends on the version of Oracle Discoverer to which business definitions are deployed and the licensing option you use. For more information about the various licensing options, see "[Product Options and Licensing](#)" on page 1-2.

[Table 11-1](#) summarizes the combinations possible when you deploy business definitions to Oracle Discoverer using the different licensing options.



**Table 11-1 Different Methods of Deploying Business Definitions**

	<b>Warehouse Builder Core Functionality</b>	<b>Warehouse Builder Enterprise ETL Option</b>
Versions Lower than Oracle Discoverer 10g Release 2	Generate scripts for the business definitions, copy these scripts to an .eex file, and import the .eex file into Oracle Discoverer.  See <a href="#">"Deploying Business Definitions Using the Core Functionality"</a> on page 11-10.	Use the Control Center to create an .eex file and then import the .eex file into Oracle Discoverer.  See <a href="#">"Deploying Business Definitions to Earlier Versions of Oracle Discoverer"</a> on page 11-9.
Oracle Discoverer 10g Release 2	Generate scripts for the business definitions, copy these scripts to an .eex file, and import the .eex file into Oracle Discoverer.  See <a href="#">"Deploying Business Definitions Using the Core Functionality"</a> on page 11-10.	Use the Control Center to directly deploy to Oracle Discoverer.  See <a href="#">"Deploying Business Definitions Directly to Oracle Discoverer"</a> on page 11-9.

### Deploying Business Definitions Directly to Oracle Discoverer

You can directly deploy business definitions to Oracle Discoverer, just like you deploy other data objects, using the Control Center or Project Explorer. The business definitions are deployed to the Discoverer location associated with the Business Definition module that contains these business definitions.

Before you deploy business definitions, ensure that a valid Discoverer location is associated with the Business Definition module. For information about how to associate a Discoverer location with a Business Definition module, see ["Setting the Connection Information"](#) in the *Warehouse Builder Online Help*.

When you deploy business definitions directly to Oracle Discoverer 10g Release 2, the following steps are performed:

1. Creates an .eex file that contains the definitions of the business definitions.
2. Connects to the EUL specified in the Discoverer location associated with the Business Definition module containing the business definitions.

---

**Note:** If the EUL is in a different database from your object definitions, a connector is created. This connector is deployed when you deploy the business definitions.

---

3. Imports the .eex file into Oracle Discoverer.

During the import, any new business definitions are appended on top of the existing definitions. You must validate the EUL and remove redundant definitions. For example, you deploy an item folder that contains four items. Subsequently, you delete one item from the item folder. When you redeploy the item folder, it still contains four items. This is because only new definitions are being appended, but old definitions are not removed.

### Deploying Business Definitions to Earlier Versions of Oracle Discoverer

You cannot directly deploy business definitions to versions of Oracle Discoverer earlier than 10g Release 2. However, you can still transfer your business definitions to Discoverer using the following work around.

When you deploy business definitions to a location that is associated with a version of Discoverer lower than 10g Release 2, the deployment will fail. But an .eex file that contains the business definitions is created. This .eex file is assigned a default name, for example, *2022.eex*, and is stored in the `OWB_ORACLE_HOME\deployed_scripts` directory. You can connect to the EUL using Oracle Discoverer and import this .eex file.

### Deploying Business Definitions Using the Core Functionality

When you use the core functionality of Warehouse Builder, you cannot directly deploy business definitions to Oracle Discoverer. You also cannot use the Control Center to create an .eex file as described in "[Deploying Business Definitions to Earlier Versions of Oracle Discoverer](#)" on page 11-9. However, you can save your business definitions to Discoverer using the steps described in the following section. For more information about the core functionality, see "[Product Options and Licensing](#)" on page 1-2.

Use the following steps to save business definitions to Discoverer:

1. Associate a valid location with the Business Definition Module that contains the business definitions.

Although you cannot use this location to deploy business definitions, defining the location ensures that the credentials of the EUL user are included in the generated code. When you define the location, a check is performed to determine if the relational schema that the intelligence objects reference is in the same database as the objects. If they are in different databases:

- A connector is created to the Discoverer location.
- The name of the database link used by the connector is included in the generated code.

The connector is created under the Discoverer location node associated with the Business Definition module. Ensure that you deploy this connector to create a database link.

2. Right-click the business definition module that contains the business definitions that you want to deploy to Discoverer and select **Generate**.

The Generation Results window is displayed.

3. Navigate to the Scripts tab of the Generation Results window.

This tab lists all the business definitions with the names of the corresponding files that store the scripts generated for these definitions.

4. Select all the objects that you want to save to Oracle Discoverer.

You can select multiple files by pressing down the Ctrl key.

5. Click the **Save As** button.

The Save As dialog box is displayed.

6. Select the directory in which you want to save the generated scripts. Ensure that you save all the files in a single directory.

For example, you save all the generated scripts in the directory `c:\sales\generated_scripts`.

7. Copy all the generated scripts to a single .eex file.

Use the operating system commands to concatenate the generated scripts into a single file. For example, in Windows, you open a Command Prompt window and execute the following steps:

```
c:> CD c:\sales\generated_scripts
c:\sales\generated_scripts> COPY *.xml sales_scripts.eex
```

This copies all the generated .xml files to an .eex file called sales\_scripts.eex.

8. Edit the .eex file created in the previous step using any text editor and perform the following steps:
  - a. Add the following lines at the beginning of the file:
 

```
<?xml version="1.0" encoding="UTF-8"?>
<EndUserLayerExport SourceEULId="20030730144738"
SourceEULVersion="4.1.9.0.0" MinimumCodeVersion="4.1.0"
ExportFileVersion="4.1.0">
```
  - b. Add the following lines at the end of the file:
 

```
</EndUserLayerExport>
```
9. Open Oracle Discoverer Administrator and connect to the EUL into which you want to import the business definitions.
10. From the File menu, select **Import**.  
The Import Wizard is displayed.
11. Import the .eex file you created into Discoverer Administrator.

## Reviewing the Deployment Results

You can monitor the progress of a job by watching the status messages at the bottom of the window and the Status column of 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 deployment details:

Double-click the job in the Job Details panel.

The Deployment Results window will appear. For a description of this window, select **Topic** from the Help menu.

### To view deployed scripts:

1. Open the Deployment Results window, as described in the previous steps.
2. 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.

## 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 ETL, 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 "[Process for Defining and Using Schedules](#)" on page 11-17.

Like deployment, you can start ETL from the Project Explorer or using the Control Center Manager. You can also start ETL using tools outside of Warehouse Builder that execute SQL scripts.

Starting a mapping or a process flow involves the following steps:

1. Generating the PL/SQL, SQL\*Loader, or ABAP script, as needed.
2. Deploying the script, as needed.
3. Executing the script.

**To start ETL from the Project Explorer:**

Select a mapping or a process flow, then select **Start** from the Design menu.

**To start ETL from the Control Center Manager:**

Select the mapping or process flow, then click the Start icon in the toolbar.

Alternatively, you can select the mapping or process flow, then select **Start** from the File menu.

**Viewing the Data**

After completing ETL, you can easily check any data object in Warehouse Builder to verify that the results are as you expected.

**To view the data:**

In the Project Explorer, 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 "[Process for Defining and Using Schedules](#)" on page 11-17.
- Use a third-party scheduling tool.

## Configuring the Physical Details of Deployment

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. You can create a different named configuration for each deployment location, with different settings for the object parameters in each one.

Before deployment, be sure to check the configuration of the target objects, the mappings, and the modules.

For an object to be deployable:

- Its target location must be fully defined, valid, and selected for the object's module.
- Its Deployable parameter must be selected, which it is by default.
- It must validate and generate without errors.

## About Configurations

When you create a repository, Warehouse Builder creates a named configuration and a control center. This configuration is referred to as the *default configuration* and is named `DEFAULT_CONFIGURATION`. The control center is named `DEFAULT_CONTROL_CENTER`. The `DEFAULT_CONFIGURATION` is associated with the `DEFAULT_CONTROL_CENTER`.

A named configuration contains the physical details for every object in a given project that are required to deploy a data system. You can create additional named configurations as described in "[Creating Additional Configurations](#)" on page 11-14.

Named configurations are used as a means to implement different physical parameters for the same design on different systems (for example, development, production, testing). Named configurations enable 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.

Each named configuration is associated with only one control center. The default control center always points to the local unified repository and allows for direct deployment and execution on that local system.

### Active Configuration

Only one named configuration is active at a time. This configuration is called the *active* configuration. All configuration settings that you set are stored against this named configuration.

When you first install Warehouse Builder, the default configuration is set as the active configuration. By default, all objects that you deploy, are deployed using the default control center as it links to the default configuration.

Changing the active configuration changes the storage of physical configuration settings, the locations used, and the control center through which the deployment is done.

### Creating New Configurations

To create a new named configuration:

1. In the Project Explorer, select a project and expand the navigation tree.
2. Select **Configurations**.
3. From the Design menu, select **New**.

The Create Configuration wizard opens.

4. On the Name page, provide a name and an optional description.

Click **Set and save as my active configuration for this project** to set the new named configuration the active configuration. Any configuration parameters that you set for design objects are saved in this named configuration. Also, any objects that you deploy are deployed to the control center associated with this new named configuration.

5. On the Details page, select the control center that should be associated with this configuration.

If you have not already created the control center, click **New** to create one and associate it with the configuration.

6. Click **Finish** to close the wizard and create the named configuration.

The new named configuration appears in the Configurations folder.

### Activating Configurations

There can be only one active configuration at a time. Any objects that you deploy are deployed to the control center associated with the active configuration. To implement

your design to a different target system, you must deploy objects to the control center associated with that target system. You can do this by activating the named configuration associated with the control center.

**To activate a named configuration:**

1. Right-click the named configuration you want to activate and select **Open Editor**.
2. On the Name tab, select **Set and save as my Active Configuration for this project**.

**To activate a named configuration for the current session only:**

1. In the Project Explorer, select a project and expand the navigation tree.
2. Expand the Configurations folder.
3. Select a named configuration.
4. From the Design menu, select **Set As Active Configuration**.

The selected named configuration is set as the active configuration for the current session. If you exit Warehouse Builder and log in subsequently, the changes are not saved.

Any changes that you make to the configuration parameters of objects are saved in the active configuration. If you switch to the previous named configuration, then these parameters will have their previous settings. The Control Center associated with this named configuration is now active and stores all new information about validation, generation, and deployment of objects in the project.

## Creating Additional Configurations

You can create additional named configurations that are associated with different control centers for a single project. To implement your logical design on different systems, you deploy your project separately for each named configuration. You must first activate a named configuration before you deploy objects using that configuration.

---



---

**Note:** To create additional named configurations, you must licence the [Warehouse Builder Enterprise ETL Option](#).

---



---

### Scenario Requiring Multiple Configurations

An enterprise data warehouse typically has multiple environments such as development, testing, and production. The same design objects need to be created in all these environments. But the objects need different physical configuration settings in each environment

[Table 11–2](#) lists the configuration settings for a particular table in the development, testing, and production environments.

**Table 11–2 Configuration Properties for a Table in Different Environments**

Configuration Property	Development Environment	Test Environment	Production Environment
Tablespace	DEV	TEST	PROD
Parallel Access Mode	PARALLEL	NOPARALLEL	PARALLEL
Logging Mode	NOLOGGING	NOLOGGING	LOGGING

You can implement this scenario by creating different configurations for each environment. Specify different values for the configuration properties of the object in each named configuration. By switching among different named configurations, you can change the physical design without making any changes to the logical design. You can easily deploy a single data system to several different host systems or to various environments.

### Setting Configuration Properties for a Named Configuration

**To set configuration properties for a particular named configuration:**

1. Set the named configuration as the active configuration.  
For information about activating a configuration, see "[Activating Configurations](#)" on page 11-13.
2. In the Project Explorer, right-click the object that you want to configure and select **Configure**.
3. Specify the configuration properties for the selected object.
4. Repeat steps 2 and 3 for all objects in the project for which you want to set configuration properties.

You can now deploy the design objects. The Control Center that is presented in the Deployment Manager is the Control Center that is associated with the active configuration.

### Deploying a Design to Multiple Target Systems

Creating multiple configurations enables you to deploy the same design to multiple target systems. To deploy a design to a particular target system, you activate the named configuration associated with the target system and deploy your design.

**To deploy a set of design objects to multiple target systems:**

1. If you have not already done so, create a named configuration for each target system to which design objects must be deployed.

See "[Creating New Configurations](#)" on page 11-13.

For each named configuration, ensure that you create a separate control center that points to the target system. Also set the configuration properties for design objects in each named configuration, as described in "[Setting Configuration Properties for a Named Configuration](#)" on page 11-15.

2. Activate the named configuration associated with the target system to which design objects must be deployed.

See "[Activating Configurations](#)" on page 11-13.

3. Resolve errors related to deployment or execution.

You may encounter some errors even if you validated data objects before deploying them. These errors could be caused by configuration properties you set. Configuration property values represent physical property information that is not semantically checked before deployment. For example, the value you specify for the Tablespace Name property is not checked against the database at validation time. Such errors are encountered during deployment.

4. Deploy the design objects.

See "[Deploying Objects](#)" on page 11-8.

5. Repeat steps 2 to 5 for each target system to which design objects must be deployed.

### **Benefit of Creating Additional Configurations**

A named configuration stores the physical details for every object in a project. You can create multiple named configurations for a project, each containing different physical settings for objects. The physical settings for design objects in each named configuration are preserved.

Consider a scenario that requires the same design objects to be implemented in different target systems, that is, different control centers. For each target system, you can create a named configuration and then specify different configuration properties for design objects in that named configuration. To implement your design to a particular environment, you simply activate the named configuration associated with that environment and deploy objects.

Configuration property values belong to the named configuration object and are preserved. You do not have to reset configuration values when you switch between named configurations. The configuration properties that you see in the Design Center are the settings associated with the active configuration. The status bar at the bottom of the Design Center displays the name of the active configuration.

For example, the named configuration associated with the development environment is currently active. So any changes you make to configuration property values are made to the development environment. For the table `MY_TABLE`, you set the Tablespace configuration parameter to `DEV`. Next activate the named configuration associated with the production environment. The configuration values displayed will be the same values that you set the last time the production configuration was active. The Tablespace configuration parameter for `MY_TABLE` is null. You set it to `PROD`. This change affects only the production configuration. Switching back to the development configuration will show the Tablespace configuration parameter for `MY_TABLE` remains as `DEV`. Every object instance in a project has unique configuration values. So, in this example, setting tablespace value for `MY_TABLE` has no effect on any other table. Each table instance must be individually configured.

Another advantage of multiple configurations is the ease with which it enables you to make changes to your existing environment. For example, you design objects, implement your development environment, deploy objects, and then move to the testing environment. You then need to change some objects in the development environment. To do this, you just activate the named configuration associated with the development environment, make the changes to objects, regenerate scripts, and deploy objects. To return to the testing environment, you activate the testing configuration. There is no need to make changes to the design objects.

## **About Schedules**

Use schedules to plan when and how often to execute operations that you designed within **Warehouse Builder**. You can apply schedules to mappings and process flows that you want to execute in an Oracle Database, version 10g or higher.

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 11-8.

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**

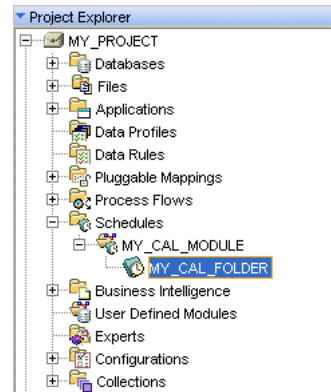


generates codes that follows the iCal calendaring standards, which can be deployed to a scheduler such as Oracle 10g Scheduler or Applications Concurrent Manager.

Schedules are defined in the context of projects and contained in schedule modules under the **Schedules** node on the Project Explorer.

Figure 11–2 displays schedules on the Project Explorer.

**Figure 11–2 Schedules on the Project Explorer**



For every new project you create, Warehouse Builder creates a default schedule module, MY\_CAL\_MODULE. Create schedules under the default module or create a new module by right-clicking the **Schedules** node and selecting **New**.

### Deploying Warehouse Builder Schedules to Oracle Workflow

To successfully deploy Warehouse Builder schedules to Oracle Workflow, ensure access to the correct version of Oracle Workflow as described in the *Oracle Warehouse Builder Installation and Administration Guide*. Scheduled jobs should be deployed to a standard database location, not to a Workflow Location. Only Process Flow packages should be deployed to Oracle Workflow.

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.

For remote Oracle Workflow locations and remote Warehouse Builder 10g 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 Control Center schema, then the deployment location must have the CREATE DATABASE LINK system privilege.

## Process for Defining and Using Schedules

1. To create a module to contain schedules, right-click the Schedules node and select **New**.
2. To create a schedule, right-click a schedule module and select **New**.  
Warehouse Builder displays the Schedule Wizard.
3. On the Name and Description page, type a name for the schedule that is 24 characters or less.

The rules for most Warehouse Builder objects is that physical names can be 1 to 30 alphanumeric characters and business names can be 1 to 2000 alphanumeric characters.

4. Follow the instructions in the Schedule Wizard.

Use the wizard to specify values for Start and End Dates and Times, Frequency Unit, and Repeat Every. When you complete the wizard, Warehouse Builder saves the schedule under the schedule module you selected.

**See Also:** "Example Schedules" on page 11-18 for examples of schedules

5. On the Project Explorer, right-click the schedule you created with the wizard and select **Open Editor**.

Warehouse Builder displays the schedule editor. Review your selections and view the list of calculated execution times. For complex schedules, you can now enter values for the By Clauses.

6. To apply a schedule to a mapping or process flow, right-click the object in the Project Explorer and select **Configure**. In the Referred Calendar field, click the Ellipsis button to view a list of existing schedules.

For any mapping or process flow you want to schedule, the physical name must be 25 characters or less and the business name must be 1995 characters or less. This restriction enables Warehouse Builder to append to the mapping name the suffix `_job` and other internal characters required for deployment and execution.

7. Deploy the schedule.

Recall that when you deploy a mapping, for example, you also need to deploy any associated mappings and process flows and any new target data objects. Likewise, you should also deploy any associated schedules.

When properly deployed with its associated objects, the target schema executes the mapping or process flow based on the schedule you created.

## Example Schedules

Use [Table 11-3](#) as a guide for defining schedules.

**Table 11-3 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

**Table 11-3 (Cont.) Example Repeat Expressions for Schedules**

<b>Schedule Description</b>	<b>Frequency Units</b>	<b>Repeat Every</b>	<b>By Clause</b>
Last workday of every month.	monthly	1 month	By Day = MON,TUE,WED,THU,FRI; By Set Pos = -1
On March 10th.	yearly	1 year	By Month = MAR By Month Day = 10
Every 12 days.	daily	12 days	n/a
Every day at 8 am and 5 pm.	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 three days of every month.	hourly	1 hour	By Month Day = 1,2,3



# Part II

---

## Example Cases

This part contains the following chapters:

- Chapter 12, "Loading Data Stored in a Microsoft Excel File"
- Chapter 13, "Connecting to SQL Server and Importing Metadata"
- Chapter 14, "Loading Transaction Data"
- Chapter 15, "The Fastest Way to Load Data from Flat Files"
- Chapter 16, "Importing from CA ERwin and Other Third-Party Design Tools"
- Chapter 17, "Reusing Existing PL/SQL Code"
- Chapter 18, "Sourcing from Flat Files with Variable Names"
- Chapter 19, "Transferring Remote Files"
- Chapter 20, "Inspecting Error Logs in Warehouse Builder"
- Chapter 21, "Updating the Target Schema"
- Chapter 22, "Managing Multiple Versions of a BI Implementation"



## Loading Data Stored in a Microsoft Excel File

### Scenario

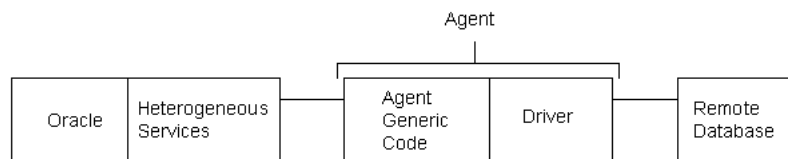
A company stores its employee data in an Excel file called `employees.xls`. This file contains two worksheets: `employee_details` and `job_history`. You need to load the data from the `employee_details` worksheet into a target table in Warehouse Builder.

### Solution

To load data stored in an Excel file into a target table, you must first use the Excel file as a source. Warehouse Builder enables you to source data stored in a non-Oracle source, such as Microsoft Excel, using the Heterogeneous Services component of the Oracle Database.

Figure 12-1 describes how the Oracle Database uses Heterogeneous services to access a remote non-Oracle source.

**Figure 12-1 Heterogeneous Services Architecture**



The Heterogeneous Services component in the database communicates with the Heterogeneous Services agent process. The agent process, in turn, communicates with the remote database.

The agent process consists of agent-generic code and a system-specific driver. All agents contain the same agent-generic code. But each agent has a different driver depending on the type of data being sourced.

## Case Study

This case study shows you how to use an Excel file called `employees.xls` as a source in Warehouse Builder.

### Step 1: Install ODBC Driver for Excel

To read data from Microsoft Excel, you must have the ODBC driver for Excel installed.

**Step 2: Delimit the Data in the Excel File (Optional)**

To source data from an Excel file, define a name for the range of data being sourced:

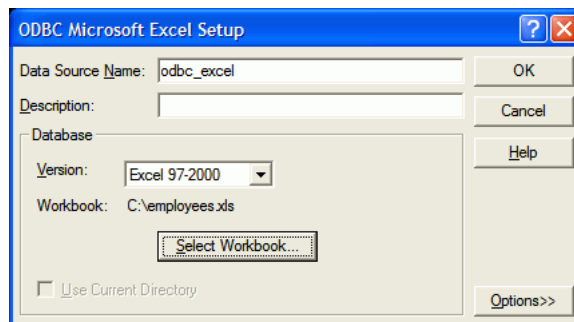
1. In the `employee_details` worksheet, highlight the range that you want to query from Oracle.  
The range should include the column names and the data. Ensure that the column names confirm to the rules for naming columns in the Oracle Database.
2. From the Insert menu, select **Name** and then **Define**. The Define Name dialog box is displayed. Specify a name for the range.

**Step 3: Create a System DSN**

Set up a System Data Source Name (DSN) using the Microsoft ODBC Administrator.

1. Select **Start, Settings, Control Panel, Administrative Tools, Data Sources (ODBC)**.  
This opens the ODBC Data Source Administrator dialog box.
2. Navigate to the System DSN tab and click **Add** to open the Create New Data Source dialog box.
3. Select **Microsoft Excel Driver** as the driver for which you want to set up the data source.  
Click **Finish** to open the ODBC Microsoft Excel Setup dialog box as shown in [Figure 12-2](#).

**Figure 12-2 ODBC Microsoft Excel Setup Dialog Box**



4. Specify a name for the data source. For example, `odbc_excel`.
5. Click **Select Workbook** to select the Excel file from which you want to import data.
6. Verify that the Version field lists the version of the source Excel file accurately.

**Step 4: Create the Heterogeneous Services Initialization File**

To configure the agent, you must set the initialization parameters in the heterogeneous services initialization file. Each agent has its own heterogeneous services initialization file. The name of the Heterogeneous Services initialization file is `initSID.ora`, where `SID` is the Oracle system identifier used for the agent. This file is located in the `ORACLE_HOME/hs/admin` directory.

Create the `initexcelsid.ora` file in the `ORACLE_HOME/hs/admin` directory as follows:

```
HS_FDS_CONNECT_INFO = odbc_excel
```



```
HS_AUTOREGISTER = TRUE
HS_DB_NAME = hsodbc
```

Here, `odbc_excel` is the name of the system DSN you created in Step 3. `excelsid` is the name of the Oracle system identifier used for the agent.

### Step 5: Modify the listener.ora file

Set up the listener on the agent to listen for incoming requests from the Oracle Database. When a request is received, the agent spawns a Heterogeneous Services agent. To set up the listener, modify the entries in the `listener.ora` file located in the `DATABASE_ORACLE_HOME/network/admin` directory as follows:

```
SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (SID_NAME = excelsid)
      (ORACLE_HOME = c:\oracle\db92)
      (PROGRAM = hsodbc)
    )
    (SID_DESC =
      (SID_NAME = PLSExtProc)
      (ORACLE_HOME = c:\oracle\db92)
      (PROGRAM = extproc)
    )
  )
```

1. For the `SID_NAME` parameter, use the SID that you specified when creating the initialization parameter file for the Heterogeneous Services, which, in this case, is `excelsid`.
2. Ensure that the `ORACLE_HOME` parameter value is the path to your Oracle Database home directory.
3. The value associated with the `PROGRAM` keyword defines the name of the agent executable.

Remember to restart the listener after making these modifications.

---



---

**Note:** Ensure that the initialization parameter `GLOBAL_NAMES` is set to `FALSE` in the database's initialization parameter file. `FALSE` is the default setting for this parameter.

---



---

### Step 6: Create an ODBC Source Module

Use the following steps to create an ODBC source module:

1. From the Project Explorer, create an ODBC source module. On the navigation tree, ODBC is listed within the Non-Oracle node under the Databases node.
2. You can provide the connection information for the source location either at the time of creating the module, or while importing data into this module.
3. To provide connection information while creating the module, on the Connection Information page, click **Edit** and provide the following details:

Ensure that the service name you provide is the same as the `SID_NAME` you specified in the `listener.ora` file.

Provide the host name and the port number using the Host Name and Port number fields respectively.

Because you are not connecting to an Oracle database, you can provide dummy values for user name and password. The fields cannot be empty.

The Schema field can be left empty because you will not be importing data from a schema.

**Step 7: Import Metadata from Excel Using the Metadata Import Wizard**

Use the Metadata Import Wizard to import metadata from the Excel file into Warehouse Builder. Select Tables as the Filter condition. The wizard displays all the worksheets in the source Excel file under the Tables node in the list of available objects.

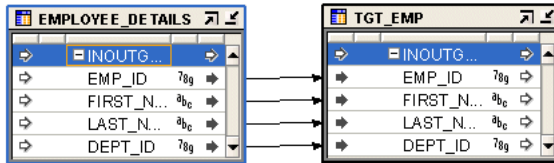
1. Select **employee\_details** and use the right arrow to move it to the list of selected objects.
2. Click **Finish** to import the data.

The data from the employee\_details worksheet is now stored in a table called `employee_details` in the ODBC source module.

**Step 8: Create a Mapping to Load Data Into the Target Table**

In the Warehouse Builder console, expand the module that contains the target table. Use the table called `employee_details` in the ODBC source module as a source to load data into the target table. [Figure 12-3](#) displays the mapping used to load data into the target table.

**Figure 12-3 Mapping to Load Data Into the Target Table**



**Step 9: Deploy the Mapping**

Use the Control Center Manager or Design Center to deploy the mapping you created in step 8. Ensure that you first deploy the source module before you deploy the mapping.

**Troubleshooting**

This section lists some of the errors that you may encounter while providing the connection information.

**Error**

ORA-28546: connection initialization failed, porbable Net8 admin error

ORA-28511: lost RPC connection to heterogeneous remote agent using  
 SID=(DESCRIPTION=(ADDRESS\_LIST=(ADDRESS=(PROTOCOL=TCP)(Host=localhost)(PORT=1521))  
 )(CONNECT\_DATA=(SID=oracledb)))

ORA-02063: preceeding 2 lines from OWB###

**Probable Cause**

Providing the same SID name as that of your database.

**Action**

Provide an SID name different from the SID name of your database.

**Error**

ORA-28500: connection from ORACLE to a non-Oracle system returned this message:  
[Generic Connectivity Using ODBC] [H006] The init parameter <HS\_FDS\_CONNECT\_INFO>  
is not set. Please set it in init<orasid>.ora file.

**Probable Cause**

Name mismatch between SID name provided in the `listener.ora` file and the name of the `initSID.ora` file in `ORACLE_HOME/hs/admin`.

**Action**

Ensure that the name of the `initSID.ora` file and the value provided for the `SID_NAME` parameter in `listener.ora` file is the same.

**Tip:** Ensure that you restart the listener service whenever you make changes to the `listener.ora` file.



---

---

## Connecting to SQL Server and Importing Metadata

### Scenario

Your company has data that is stored in SQL Server and you would like to import this into Warehouse Builder. Once you import the data, you can perform data profiling to correct anomalies, and then transform the data according to your requirements by using mappings.

### Solution

In Warehouse Builder, you can connect to non-Oracle data sources. Once connected, you can import metadata just as from any Oracle data source.

### Case Study

To connect to SQL Server and import metadata, refer to the following sections:

1. ["Creating an ODBC Data Source"](#) on page 13-1
2. ["Configuring the Oracle Database Server"](#) on page 13-2
3. ["Adding the SQL Server as a Source in Warehouse Builder"](#) on page 13-3
4. [What's Next](#) on page 13-3

If you encounter problems implementing this solution, see ["Troubleshooting"](#) on page 13-3.

## Creating an ODBC Data Source

You must create an ODBC data source to connect to the SQL Server. To do this, you must set up a System Data Source Name (DSN):

1. Select **Start, Control Panel, Administrative Tools, Data Sources (ODBC)**.  
This opens the ODBC Data Source Administrator dialog box.
2. Navigate to the **System DSN** tab and click **Add** to open the Create New Data Source dialog box.
3. Select **SQL Server** as the driver for which you want to set up the data source.
4. Click **Finish** to open the Create A New Data Source to SQL Server Wizard.
5. In the **Name** field, specify a name for the data source. For example, `sqlsource`.
6. In the **Server** field, select the server to which you want to connect and click **Next**.

7. Specify whether the authentication should be done at the Operating System level or at the server level. Click **Next**.
8. Select the database file and click **Next**.
9. Accept the default values in the next screen and click **Finish**.
10. Test the data source to verify the connection.

## Configuring the Oracle Database Server

Next, you must configure Oracle Database to connect to SQL Server. Warehouse Builder can then use this configuration to extract metadata from the SQL Server. There are two steps involved in this:

- [Creating a Heterogeneous Service Configuration File](#)
- [Editing the listener.ora file](#)

### Creating a Heterogeneous Service Configuration File

You must create the heterogeneous file in the `ORACLE_HOME\hs\admin` directory. The naming convention for this file should be as follows:

- Must begin with `init`
- Must end with the extension `.ora`
- Must not contain space or special characters

For example, you can name the file `initsqlserver.ora`.

Enter the following in the file:

```
HS_FDS_CONNECT_INFO = sqlsource
HS_FDS_TRACE_LEVEL = 0
```

Here, `sqlsource` is the name of the data source that you specified while creating the ODBC data source.

### Editing the listener.ora file

You must add a new SID description in the `listener.ora` file. This file is stored in the `ORACLE_HOME/network/admin` directory.

Modify the file as shown:

```
SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (SID_NAME = sqlserver)
      (ORACLE_HOME = c:\oracle10g\oracle_home)
      (PROGRAM = hsodbc)
    )
    (SID_DESC =
      (SID_NAME = PLSExtProc)
      (ORACLE_HOME = c:\oracle10g\oracle_home)
      (PROGRAM = extproc)
    )
  )
```

The `SID_NAME` parameter must contain the name of the configuration file you created in the previous step. However, it must not contain the `init` prefix. For example, if the

configuration file you created in the previous step was `initsqlserver.ora`, then the value of the `SID_NAME` parameter should be `sqlserver`.

`ORACLE_HOME` must point to the Oracle home location of your database installation.

The value associated with the `PROGRAM` keyword defines the name of the executable agent, which, in this case, is `hsodbc`.

Restart the listener service after making these modifications.

## Adding the SQL Server as a Source in Warehouse Builder

The final step involves adding an ODBC module in Warehouse Builder, and importing the data from the SQL server into this module.

### To add an ODBC source module in Warehouse Builder:

1. Within a project in the Project Explorer, navigate to the Databases, Non-Oracle node.
2. Right-click **ODBC** and select **New**.
3. Create a new ODBC module using the Create Module Wizard.
4. You can provide the connection information for the source location either at the time of creating the module, or while importing data into this module.
5. In the Edit Location dialog box, make sure that you enter User Name and Password within double quotation marks (""). For example, if the user name is mark, enter "mark".
6. For Service Name, enter the SID name you provided in the `listener.ora` file. Also select the schema from which you wish to import the metadata.

### To import metadata into the ODBC module:

1. Right-click the module and select **Import**.
2. Import the metadata using the Import Metadata Wizard.

The tables and views available for import depend on the schema you selected when providing the connection information.

## What's Next

After you successfully import metadata into Warehouse Builder, you can use the data profiling functionality to check the quality of the data. Or you can skip data profiling and proceed with designing a mapping to extract, transform, and load the data.

## Troubleshooting

Some of the errors that you may encounter while providing the connection information are listed here:

### Error

**ORA-28500:** connection from ORACLE to a non-Oracle system returned this message: [Generic Connectivity Using ODBC][Microsoft][ODBC Driver Manager] Data source name not found and no default driver specified (SQL State: IM002; SQL Code: 0)

**ORA-02063:** preceding 2 lines from OWB\_###

**Probable Cause**

Creating the DSN from the User DSN tab.

**Action**

Create the DSN from the System DSN tab.

**Error**

**ORA-28500:** connection from ORACLE to a non-Oracle system returned this message:  
[Generic Connectivity Using ODBC][Microsoft][ODBC SQL Server Driver][SQL  
Server]Login failed for user 'SA'. (SQL State: 28000; SQL Code: 18456)

**ORA-02063:** preceding 2 lines from OWB\_###

**Probable Cause**

The user name and password in the Edit Location dialog box are not enclosed within double quotation marks.

**Action**

Enter the user name and password within double quotation marks.

**Tip:** Make sure that you restart the listener service whenever you make changes to the `listener.ora` file.



---



---

## Loading 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 re-entered, 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
```

### 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 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.

### Case Study

This case study shows you how to create ETL logic to load transaction data in a particular order using Warehouse Builder.

#### 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 Warehouse Builder Import Metadata Wizard. Proceed to 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.

---

#### Step 2: Create an External Table

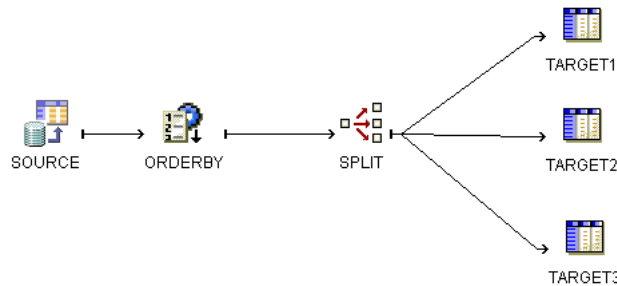
To simplify the use of sampled flat file object in a mapping, create an external table (`JOURNAL_EXT`) using the Create External Table Wizard, based on the flat file imported and sampled in Step 1.

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 14–1](#) shows you how the source is ordered and split.

**Figure 14–1 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 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 TARGET 1, TARGET 2, and TARGET 3 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.

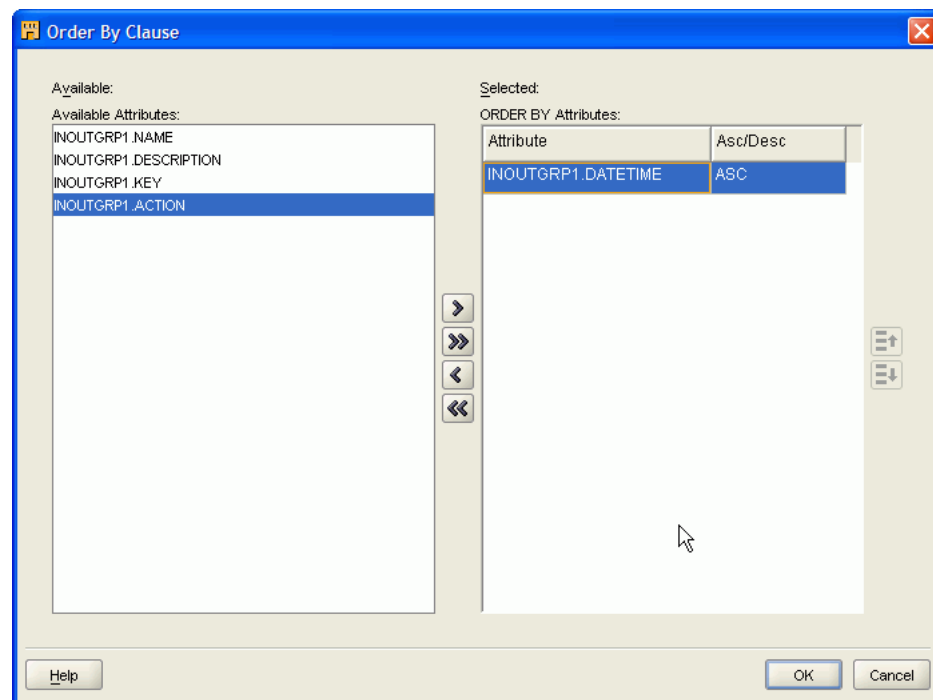
#### Step 5: Add an External Table Operator

Drag and drop a mapping external table operator onto the mapping editor 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. [Figure 14-2](#) shows you how to order the table based on the timestamp of the transaction data in ascending order.

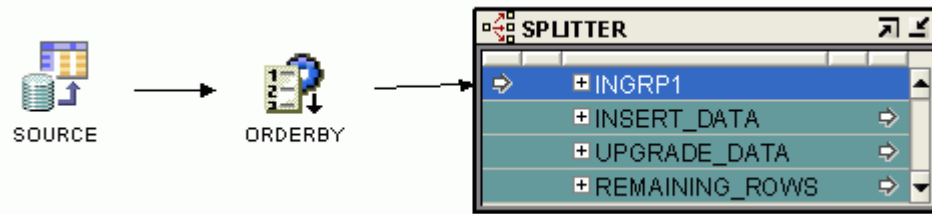
**Figure 14-2** Order By Clause Dialog Box



#### 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 14-3](#) shows how to join the SOURCE operator with the ORDERBY operator which is linked to the SPLITTER operator.

**Figure 14–3 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 `UPDATE_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: one for `INSERT_DATA`, one for `UPDATE_DATA`, and one 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 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 take place in the exact order in which the transaction was recorded at 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 operations. 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 Project Explorer, 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 a mapping, Warehouse Builder displays the results in the Generation Results 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, as shown in [Figure 14–1](#) on page 14-2. The resulting target table thus maintains the sequential integrity of the transactions recorded at source.

---



---

## The Fastest Way to Load Data from Flat Files

### Scenario

The weekly sales data of a company is stored in a flat file called `weeklysales.txt`. This data needs to be loaded into a table in the Warehouse Builder workspace.

An example data set of the source file is defined as:

```
SALESREP, MONTH, PRODUCT_ID, W1_QTY, W2_QTY, W3_QTY, W4_QTY
100, JAN02, 3247, 4, 36, 21, 42
101, JUL02, 3248, 24, 26, 4, 13
```

Each record in the file contains details of the quantity sold by each sales representative in each week of a month. This data needs to be loaded into the Warehouse Builder workspace.

### Solution

Warehouse Builder provides two methods of loading data stored in flat files. The methods are:

- [Solution 1: Using SQL\\*Loader](#)
- [Solution 2: Using External Tables](#)

[Table 15–1](#) lists the differences between using SQL\*Loader and external tables in the procedure used to load data from flat files.

**Table 15–1 Differences Between SQL\*Loader and External Tables**

SQL*Loader	External tables
Requires multiple steps to enable data transformation.	The transformation and loading of data is combined into a single SQL DML statement.
You must load the data into a staging area and then transform the data in a separate step.	There is no need to stage the data temporarily before inserting it into the target table.

## SQL \*Loader

SQL\*Loader is an Oracle tool that enables you to load data from flat files into tables in an Oracle Database. In Warehouse Builder, use the Flat File operator to load data using SQL\*Loader.

SQL\*Loader is the only method you can use to load data from a flat file into a database whose version is Oracle 8i Release 3 (8.1.7) or earlier.

### When To Use SQL\*Loader

Use SQL\*Loader to load data from a flat file if:

- The database version is Oracle 8i Release 3 (8.1.7) or earlier.
- No complex transformations are required on the input data.

## External Tables

An external table is a database object that enables you to access data stored in external sources. External tables allow flat files to have the same properties as database tables (read-only) and extend the power of SQL to reading flat files. You can also query, join, transform, and constrain the flat file data before loading it into the database.

---

---

**Note:** External tables are supported only from Oracle9i onwards.

---

---

In Warehouse Builder, use the external table object and the Mapping External Table operator to load data from a flat file into the workspace. The design benefit of using external tables is that it extends additional database features to a flat file. By using external tables instead of flat files, you can apply complex transformations to the data in flat files that were previously only used for relational files.

## Benefits of Using External Tables

- Provides faster access to flat files because the external data can be accessed in parallel during a load.
- Can perform heterogeneous joins with database tables or other external tables.

## When To Use External Tables

- To transform flat file data before loading into the database
- To perform complex transformations, such as joins and aggregations, on the flat file data before loading it into the Warehouse Builder workspace

External tables can be faster when the following conditions are met:

- The hardware has multiple processors.
- The flat file is large (has many records).

When these conditions are met, the benefits of parallel processing will outperform SQL\*Loader processing.

## Solution 1: Using SQL\*Loader

Use SQL\*Loader to load data from the flat file into the target table. Warehouse Builder provides the Flat File operator that enables you to load data into a target table using SQL\*Loader.

However, the transformations that you can perform on data loaded using a flat file operator are limited to SQL\*Loader transformations only. You can use only the following mapping operators when you use a Flat File operator as a source:

- Filter operator
- Constant operator
- Data Generator operator
- Mapping Sequence operator

- Expression operator
- Transformation operator

To load data using SQL\*Loader, create a mapping that uses the mapping flat file operator to represent the source data. Map the output of this operator directly to the target table.

## Solution 2: Using External Tables

Use external tables to load data from the flat file `weeklysales.txt` into the workspace table `SALES_DATA`. Create a mapping that contains the External Table operator as the source. This External Table operator must be bound to the external table object that you create referring to the flat file. Map the output of the external table operator directly to the target table.

### Mapping to Load Data Using External Tables

In the mapping that loads the data from the flat file, use the External Table operator to represent the source data. Map the output of the External Table operator to the target table `SALES_DATA`.





---

---

## Importing from CA ERwin and Other Third-Party Design Tools

### Scenario

A movie rental company uses tools from different vendors for data modelling, extraction, transformation and loading (ETL), and reporting purposes. Using a variety of tools has led to several metadata integration issues for this company. Often, the design work done using one tool cannot be completely integrated or reused in another. This company wants to find a method to streamline and integrate all its metadata designs and ETL processes using a single tool.

### Solution

Warehouse Builder enables the company to import and integrate metadata designs from different tools and use them for data modelling and ETL purposes using only one tool. Warehouse Builder uses the seamlessly integrated technology from Meta Integration Technology Inc. (MITI) to import the metadata and reuse the data models designed by other third-party tools.

This case study shows you how to easily import design files developed using CA ERwin into Warehouse Builder. You can then reuse the metadata for ETL design and reporting using a single tool. You can follow this model to import files from other tools such as Sybase PowerDesigner and Business Objects Designer.

### Case Study

This case study shows you how the movie rental company can migrate their ERwin data model designs into Warehouse Builder. They can also use this model to import designs from other third party tools and consolidate their design metadata in a central workspace. Follow these steps:

1. [Download Metadata from CA ERwin](#)
2. [Install the Meta Integration Model Bridge](#)
3. [Create an MDL File from the CA ERwin Data](#)
4. [Import the MDL file into Warehouse Builder](#)

Use Warehouse Builder Transfer Wizard to import the ERwin metadata into Warehouse Builder.

### Download Metadata from CA ERwin

Download the design metadata from CA ERwin to your local system.

---

## Install the Meta Integration Model Bridge

Warehouse Builder enables you to integrate with Meta Integration Model Bridges (MIMB). These bridges translate metadata from a proprietary metadata file or repository to the standard CWM format that can be imported into Warehouse Builder using the Warehouse Builder Transfer Wizard. To import files from different design tools into Warehouse Builder, you must first obtain an MIMB license and install the bridges on your system. Follow these steps to complete the installation.

### To download MIMB:

1. Download the Model Bridge product from the following Web site:  
<http://www.metaintegration.net/Products/Downloads/>
2. Install the MIMB by running the setup on your system.
3. During installation, select **Typical with Java Extensions** as the installation type from the Setup Type page.

If the set up program is not able to find a JDK on your computer, you must provide the JNI library directory path name. Your path environment variable must contain the metaintegration directory. If not, you must add it to the path:

```
c:\program files\metaintegration\win32
```

## Create an MDL File from the CA ERwin Data

Create an MDL file from CA ERwin using Warehouse Builder.

After you install the MIMB product, follow these steps to create an MDL file from ERwin and other third party design tools:

1. From the Project Explorer, select and expand the Project node to which you want to import the metadata. In this example, the ERwin files are imported into MY\_PROJECT.
2. From the **Design** menu, select **Import, Bridges** to start the Warehouse Builder Transfer Wizard.  
  
This wizard seamlessly integrates with the MITI technology to translate the third-party metadata into a standard CWM format that is imported into Warehouse Builder. Follow the wizard to complete the import.
3. In the Metadata Source and Target Identification page, select **CA ERwin 4.0 SP1 to 4.1** in the From field.
4. In the Transfer Parameter Identification page, provide the path where the ERwin files are located in the Erwin4 Input File field. In this example, the company wants to import the `Emovies.xml` file from ERwin.
5. Accept the default options for all other fields.  
  
In the OWB Project field, enter the Warehouse Builder project where you want to import the ERwin file. In the Warehouse Builder MDL field, enter a name and select the location to store the .mdl file that will be generated.
6. Complete the remaining wizard steps and finish the import process.

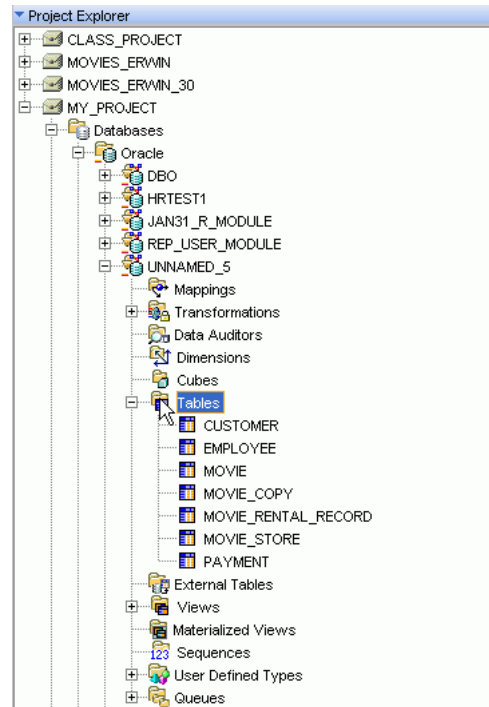
## Import the MDL file into Warehouse Builder

Import the MDL file to import metadata from the CA ERwin file into Warehouse Builder. To import the MDL file:

1. Select MY\_PROJECT and from the **Design** menu, select **Import, Warehouse Builder Metadata** to open the Metadata Import dialog box.

2. In the File Name field, specify the name of the .mdl file you generated in "Create an MDL File from the CA ERwin Data" on page 16-2.
3. Click **Import** to import the metadata into Warehouse Builder.  
If the metadata file version and the workspace version are not compatible, then the Metadata Upgrade window pops up. Click **Upgrade** to upgrade the .mdl file.
4. After you finish importing the ERwin files into Warehouse Builder, expand the MY\_PROJECT folder, then the Databases node, and then the Oracle node. You can see the imported source metadata objects, as shown in [Figure 16-1](#).

**Figure 16-1 Metadata Objects Imported from CA Erwin**



5. Double-click the table names to see the properties for each of these tables. Warehouse Builder imports all the metadata including descriptions and detailed information on table columns and constraints, as shown in [Figure 16-2](#).

**Figure 16–2 Table Properties Imported from CA Erwin**

Table Details: UNNAMED_5_EMPLOYEE								
Name	Columns	Constraints	Indexes	Partitions	Attribute Sets	Data Rules	Data Viewer	
	Name	Data Type	Length	Precision	Scale	Seconds Precision	Not Null	Defa
1	EMPLOYEE_NUMBER	NUMBER		0	0		<input type="checkbox"/>	
2	STORE_NUMBER	NUMBER		0	0		<input type="checkbox"/>	
3	EMPLOYEE_FIRST_NAME	NUMBER		0	0		<input type="checkbox"/>	
4	EMPLOYEE_LAST_NAME	VARCHAR2	15				<input type="checkbox"/>	
5	EMPLOYEE_ADDRESS_1	VARCHAR	20				<input checked="" type="checkbox"/>	
6	EMPLOYEE_ADDRESS_2	CHAR	20				<input checked="" type="checkbox"/>	
7	EMPLOYEE_CITY	VARCHAR	20				<input checked="" type="checkbox"/>	
8	EMPLOYEE_STATE	CHAR	2				<input type="checkbox"/>	
9	EMPLOYEE_ZIP	NUMBER		0	0		<input checked="" type="checkbox"/>	
10	EMPLOYEE_PHONE	VARCHAR2	0				<input checked="" type="checkbox"/>	
11	EMPLOYEE_SSN	NUMBER		0	0		<input checked="" type="checkbox"/>	
12	HIRE_DATE	DATE					<input checked="" type="checkbox"/>	
13	SALARY	VARCHAR	0				<input type="checkbox"/>	
14	SUPERVISOR	NUMBER		0	0		<input type="checkbox"/>	
							<input type="checkbox"/>	

- The designers at the movie rental company can use these sources tables to model ETL designs in Warehouse Builder, generate ETL code, and run reports on them. Furthermore, Warehouse Builder enables them to easily import all the scattered third-party design metadata and consolidate all their design and development efforts.

---



---

## 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 8i 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 the customer has a PL/SQL package for Oracle 8i, then by importing it into Warehouse Builder they can generate code for both Oracle 8i and Oracle 9i.

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 by taking these 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: Migrate Custom Code into a Mapping](#) by migrating the legacy PL/SQL code functionality into a new Warehouse Builder mapping and phase out the custom package.
- [Step 4: Generate Code for Oracle 9i](#)

Follow these steps to handle a custom PL/SQL package in Warehouse Builder.

### Step 1: Import the Custom PL/SQL Package

In the Project Explorer, 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 and selecting **Import**. 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, they 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.

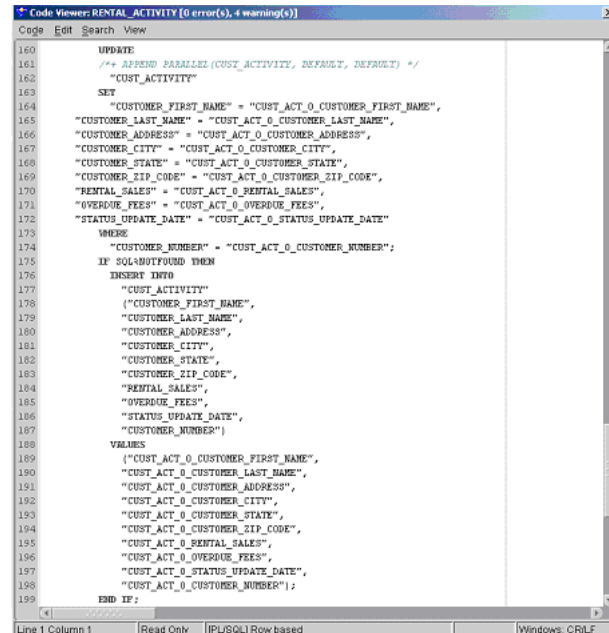
Follow the next steps to take advantage of these features in Warehouse Builder and to automatically maintain, monitor, and generate your PL/SQL code.

### Step 3: Migrate Custom Code into a Mapping

To take advantage of the code generation, maintenance, and auditing features, you can gradually migrate the legacy PL/SQL code functionality into 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, 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 17-1](#) 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 17-1** Sample Code



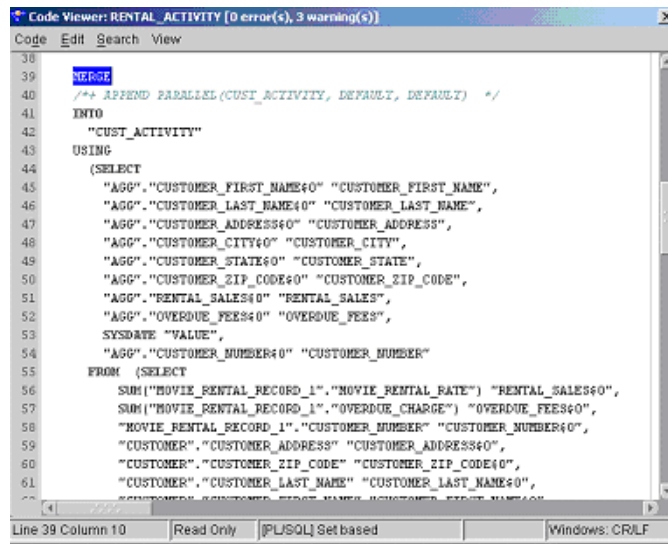
```
Code Viewer: RENTAL_ACTIVITY [0 error(s), 4 warning(s)]
Code Edit Search View
160      UPDATE
161      /*+ APPEND PARALLEL(CUST_ACTIVITY, DEFAULT, DEFAULT) */
162      "CUST_ACTIVITY"
163      SET
164      "CUSTOMER_FIRST_NAME" = "CUST_ACT_0_CUSTOMER_FIRST_NAME",
165      "CUSTOMER_LAST_NAME" = "CUST_ACT_0_CUSTOMER_LAST_NAME",
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;
```

### Step 4: Generate Code for Oracle 9i

If you upgrade to Oracle 9i version of the database, you only need to re-deploy the `Rental_Activity` mapping created in Step 3. Warehouse Builder generates code optimized for the new database version. [Figure 17-2](#) shows the MERGE statement from a sample of code generated for the same mapping for Oracle 9i.



Figure 17-2 Sample Code for Oracle 9i



```
Code Viewer: RENTAL_ACTIVITY [0 error(s), 3 warning(s)]
Code Edit Search View
38
39
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",
62 "CUSTOMER"."CUSTOMER_FIRST_NAME" "CUSTOMER_FIRST_NAME&0")
63
```

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.



---

---

## Sourcing from 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 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 an external process 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 `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 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 that copies the generated file, saves it as `s_data.dat`, and deletes 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:**

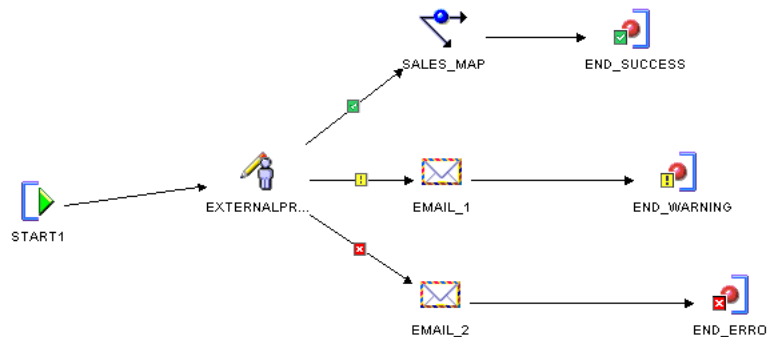
1. ["Creating the Process Flow"](#) on page 18-2
2. ["Setting Parameters for the External Process Activity"](#) on page 18-2
3. ["Configuring the External Process Activity"](#) on page 18-6

4. "Designing the Mapping" on page 18-7
5. "Deploying and Executing" on page 18-7

## Creating the Process Flow

Create a process flow that starts a mapping on the condition that the external process activity completes successfully. Your process flow should resemble [Figure 18–1](#). For more information on creating the process flow, refer to "Instructions for Defining Process Flows" on page 8-2.

**Figure 18–1 Process Flow with External Process Transitioning to a Mapping**



## Setting Parameters for the External Process Activity

This section describes how to specify the DOS commands for renaming the generated file. The DOS commands you issue from the external process 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:

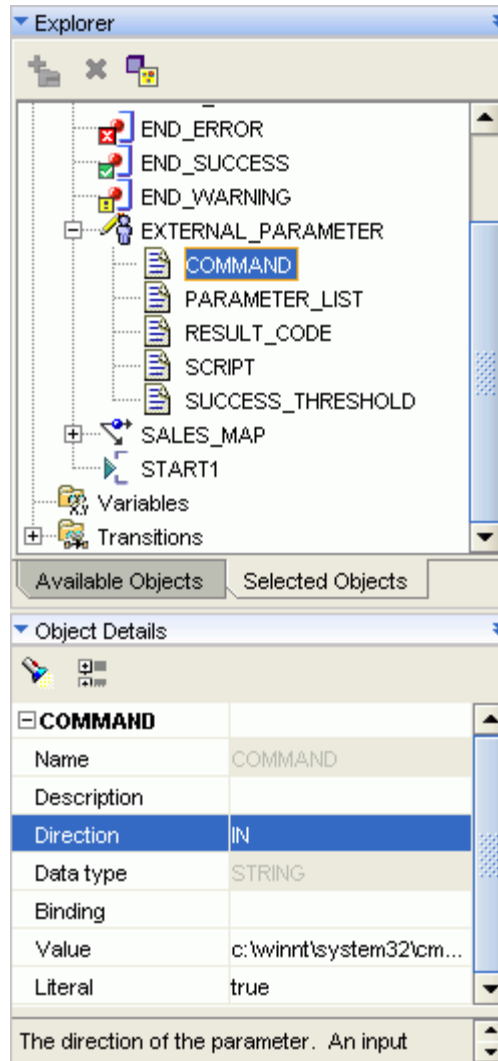
- [Method 1: Write a script within Warehouse Builder](#)
- [Method 2: Call a script maintained outside of Warehouse Builder](#)

### 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, type 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. Your Activity View should resemble [Figure 18–2](#).

**Figure 18–2 External Process Parameters When Script Maintained in this Product**



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 18–1 lists the substitute variables you can type for the external process 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 local when you configure the activity as described in "Configuring the External Process Activity" on page 18-6. These values are set when you register the locations at deployment.

**Table 18–1 Substitute Variables for the External Process 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.

**Table 18–1 (Cont.) Substitute Variables for the External Process Activity**

Variable	Value
\${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 external process activity:

1. Write the script and save it on the file directory. For example, you can write the following script and save it as `c:\staging_files\rename_file.bat`:

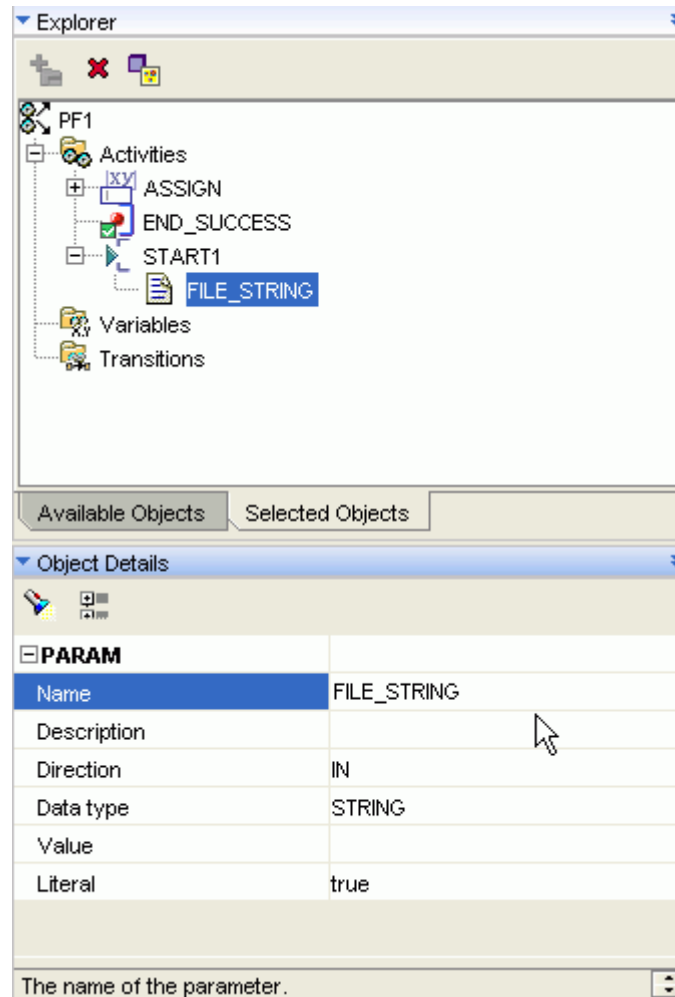
```
copy c:\staging_files\%1*.dat c:\staging_files\s_data.dat
del c:\staging_files\%1*.dat
```

In this sample script, we pass 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 Available Objects tab of the Explorer panel displayed in the Process Flow Editor.

To add a start parameter, click **Add** on the upper left corner of the Explorer pane in the Available Objects tab. Create a start parameter named `FILE_STRING` as shown in [Figure 18–3](#). 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.

Figure 18–3 Start Activity in the Activity View

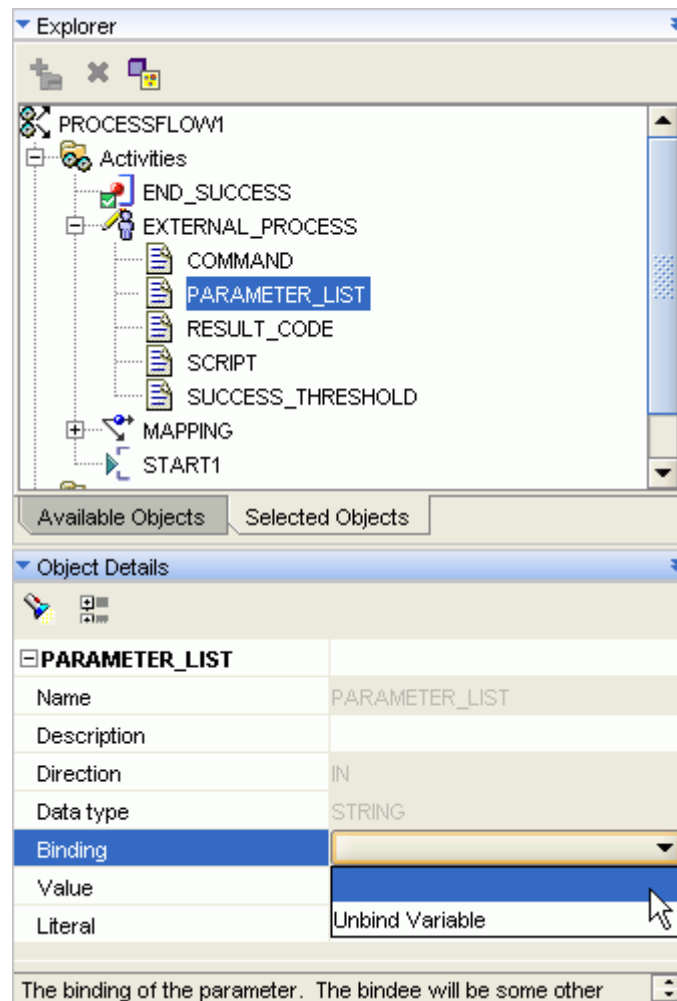


3. Select the external process activity on the canvas and edit its parameters as shown in Figure 18–4.

For the COMMAND parameter, type 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, type `c:\staging_files\rename_file.bat`.

For PARAMETER\_LIST, click the row labeled Binding and select the parameter you defined for the start activity, FILE\_STRING

Accept the defaults for all other parameters for the external process. Your Activity View for the external process activity should resemble Figure 18–4.

**Figure 18–4 External Process Parameters When Calling an Outside Script**

## Configuring the External Process Activity

When you apply conditions to the outgoing transitions of an external process, you must define the meaning of those conditions when you configure the external process activity.

### To configure the external process activity:

1. Right-click the process flow on the navigation tree and select **Configure**.
2. Expand the external process activity and the Path Settings. 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 as listed in [Table 18–1](#) on page 18-3. Use the list to select the values.

Because this case study does not use substitution variables, accept the defaults values.

4. Set the Deployed Location to the computer where you deploy the process flow.
5. Select **Use Return as Status**.



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, shown in [Figure 18-1](#) on page 18-2, if the external process returns a success value, the process flow continues down the success transition and executes the downstream mapping.

## Designing the Mapping

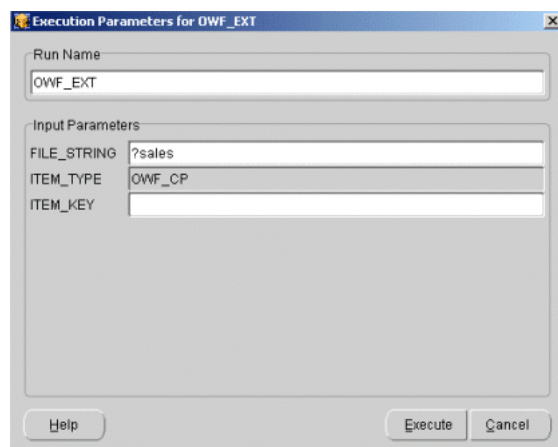
Now you can design a mapping with `s_data.dat` as the source. You can create a PL/SQL mapping or a SQL\*Loader mapping. For a PL/SQL, map the flat file source to an external table and design the rest of the mapping with all the operators available for a PL/SQL mapping. For SQL\*Loader, map the flat file 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 type values for the parameter you created to pass into the script, `FILE_STRING`. For this case study, type `?sales` where the question mark is the separator, as shown in [Figure 18-5](#). The external activity then executes the command `rename_file.bat sales`.

**Figure 18-5 External Process Activity in the Activity View**



## 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.

## Creating a Schedule

Use schedules to plan when and how often to execute operations such as mappings and process flows that you deploy through Warehouse Builder.

**To create a scheduler:**

1. Right-click the **Schedules** node in the Project Explorer and select **New**.

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. Or, select an existing location from the location list. Click **Edit** to type in the connection information and test the connection.

**4. Click Next.**

The wizard displays the Summary page. Verify the name and status of the new Scheduler module.

When you click **Finish**, Warehouse Builder stores the definition for the module and inserts its name in the Project Explorer, and prompts you to create a schedule.

---

---

## Transferring 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 host 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 `salesrv1`. 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 assumes there is 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 `salesrv1` is copied to the `local` computer and then trigger the execution of the `salesresults` mapping.

**To transfer files and start a dependent mapping, refer to the following sections:**

1. ["Defining Locations"](#) on page 19-6.

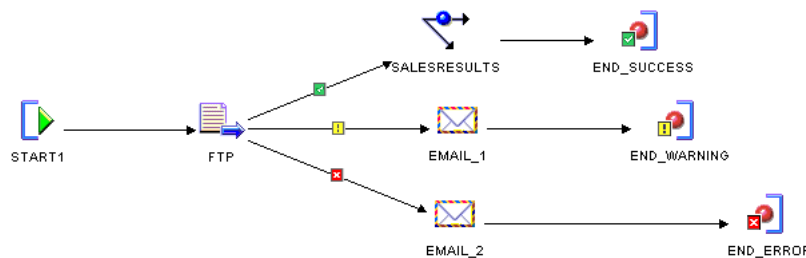
2. ["Creating the Process Flow"](#) on page 19-2
3. ["Setting Parameters for the FTP Activity"](#) on page 19-2
4. ["Configuring the FTP Activity"](#) on page 19-5
5. ["Registering the Process Flow for Deployment"](#) on page 19-5

After you complete the instructions in the above sections, you can run the process flow.

## 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 19-1](#).

**Figure 19-1 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 typing values for the FTP activity parameters on the Activity View as displayed in [Figure 19-2](#).

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 and/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, type the path to the FTP executable such as `c:\winnt\system32\ftp.exe`. Also, type 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, type the appropriate command in `PARAMETER_LIST` to direct Warehouse Builder to the file. For a Windows operating system, type 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, type 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 described above. 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 Available Objects tab of the Explorer panel in the Process Flow Editor.
2. For the COMMAND parameter, type 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`.

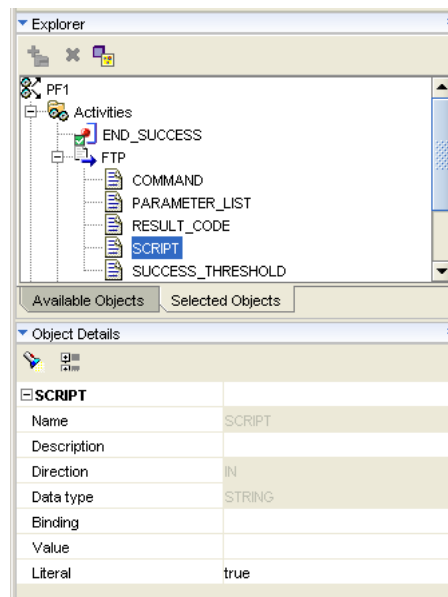
3. For the PARAMETER\_LIST parameter, type the `Task.Input` variable.

When defining a script in Warehouse Builder and using Windows FTP, you must type `"-s: ${Task.Input}"` into PARAMETER\_LIST.

For UNIX, type `"${Task.Input}"`.

4. Navigate and highlight the SCRIPT parameter. Your Available Objects tab should display similar to [Figure 19–2](#).

**Figure 19–2 Activity View for FTP Activity Using a Script**



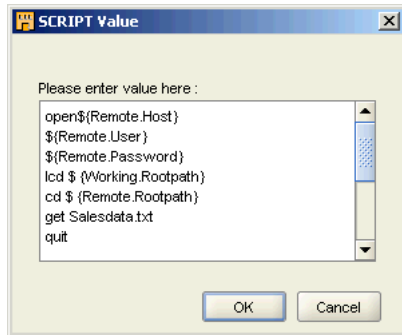
5. Click the Ellipses displayed to the right of the Value field displayed in the Object Details panel.

Warehouse Builder displays the SCRIPT Value editor. Write or copy and paste FTP commands into the editor.

Figure 19–2 shows a script that opens a connection to the remote host, changes the directory to the local computer, changes the directory to the remote host, transfers the file, and closes the connection.

Notice that the script in Figure 19–3 includes `${Remote.User}` and `${Remote.Password}`. These are substitution variables. Refer to "Using Substitution Variables" on page 19-4 for more details.

**Figure 19–3** 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, consider that you create 10 process flows that utilize FTP activities to access a file on `salesrv1` under a specific directory. If the file is moved, 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 19-6.

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 you provided for the associated location.

Table 19–1 lists the substitute variables 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 local when you configure the FTP activity. For more information, see "Configuring the FTP Activity" on page 19-5.

**Table 19–1** Substitute Variables for the FTP Activity

Variable	Value
<code>\${Working.RootPath}</code>	The root path value for the location of the Runtime Service host.
<code>\${Remote.Host}</code>	The host value for the location involved in transferring data to or from the Runtime Service host.
<code>\${Remote.User}</code>	The user value for the location involved in transferring data to or from the Runtime Service host.
<code>\${Remote.Password}</code>	The password value for the location involved in transferring data to or from the Runtime Service host.

**Table 19–1 (Cont.) Substitute Variables for the FTP Activity**

Variable	Value
\${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, shown in [Figure 19–1](#) on page 19-2, 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.

[Figure 19–4](#) shows the registration information for the REMOTE\_LOCATION. For the LOCAL\_FILES, only the root path is required.

**Figure 19–4 Example Location Registration Information**

This location has not been registered. Please complete the location parameters.

Name: PFM1\_LOCATION

Description:

Password:

Type: HOST:PORT:SERVICE

Host: localhost

Port: 1521

Service Name: oracledb

Schema: REP\_OWNER

Version: 11i

Test Connection

Test Results:

Help OK Cancel

Now 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 Connection Explorer and select **New**. For `salesrv1`, 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`.



---

---

## Inspecting Error Logs in Warehouse Builder

### Scenario

While working with Warehouse Builder, the designers need to access log files and check on different types of errors. This case study outlines all the different types of error messages that are logged by Warehouse Builder and how to access them.

### Solution

Warehouse Builder logs the following types of errors when you perform different operations:

- [Installation Errors](#) on page 20-1
- [Metadata Import and Export Errors](#) on page 20-1
- [Validation Errors](#) on page 20-2
- [Generation Errors](#) on page 20-3
- [Deployment and Execution Errors](#) on page 20-4
- [Name and Address Server Errors](#) on page 20-4

### Case Study

This case study shows you how to retrieve error logs after performing different operations in Warehouse Builder.

#### Installation Errors

When you run the Oracle Universal Installer to install Warehouse Builder, the installation error logs are automatically stored in:

```
C:\ProgramFiles\Oracle\Inventory\logs\installActions<timestamp>.log
```

When you run the Warehouse Builder Repository Assistant, the workspace installation error logs are stored in:

```
OWB_ORACLE_HOME\owb\reposasst\log.txt.0
```

When you run the Warehouse Builder Runtime Assistant, the runtime installation error logs are stored in:

```
OWB_ORACLE_HOME\owb\rtasst\log.txt.0
```

#### Metadata Import and Export Errors

**Metadata Import:** When you import a project or specific objects into your workspace using the Metadata Import Utility, Warehouse Builder records details of the import process in a log file. You can specify the name and location of this log file from the Metadata Import dialog box.

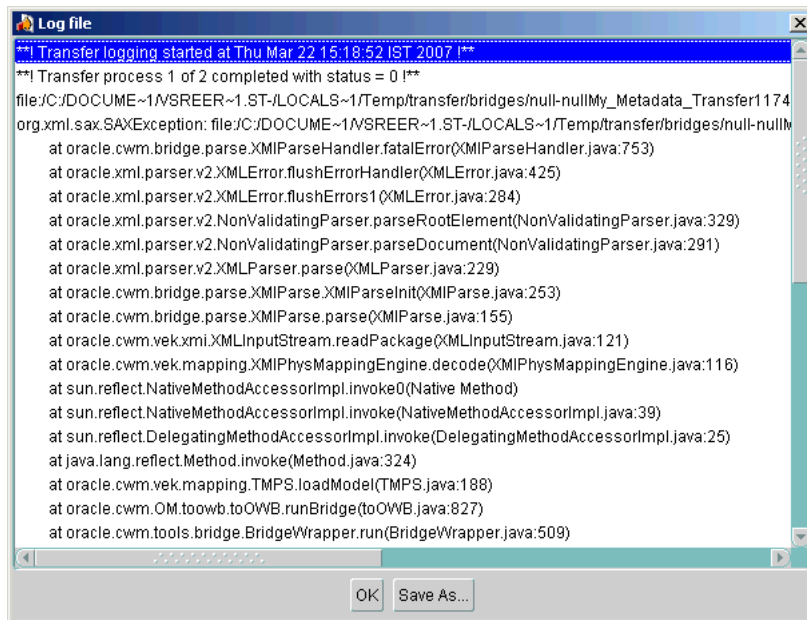
**Metadata Export:** When you export a Warehouse Builder project or specific objects using the Metadata Export Utility, Warehouse Builder records the details of the export in a log file. You can specify the name and location of this log file from the Metadata Export dialog box.

### Metadata Import Using the Transfer Wizard

If you are importing design metadata using the Warehouse Builder Transfer Wizard, then you can view the log file after the import is complete. Warehouse Builder displays the My Metadata Transfer dialog box.

Click **View Log File** to view the log file, as shown in [Figure 20–1](#). Click **Save As** to save the log file to your local system.

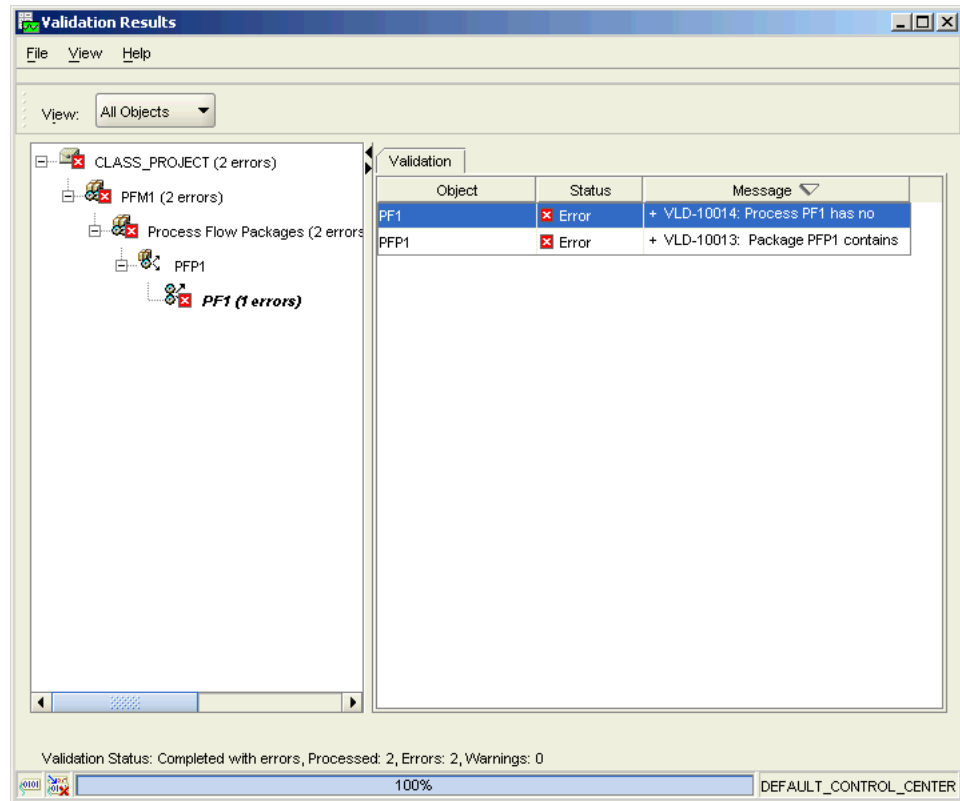
**Figure 20–1 Metadata Import Log File**



### Validation Errors

In Warehouse Builder, you can validate all objects by selecting the objects from the console tree and then selecting **Validate** from the Object menu. After the validation is complete, the validation messages are displayed in the Validation Results window, as shown in [Figure 20–2](#).

Figure 20–2 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.

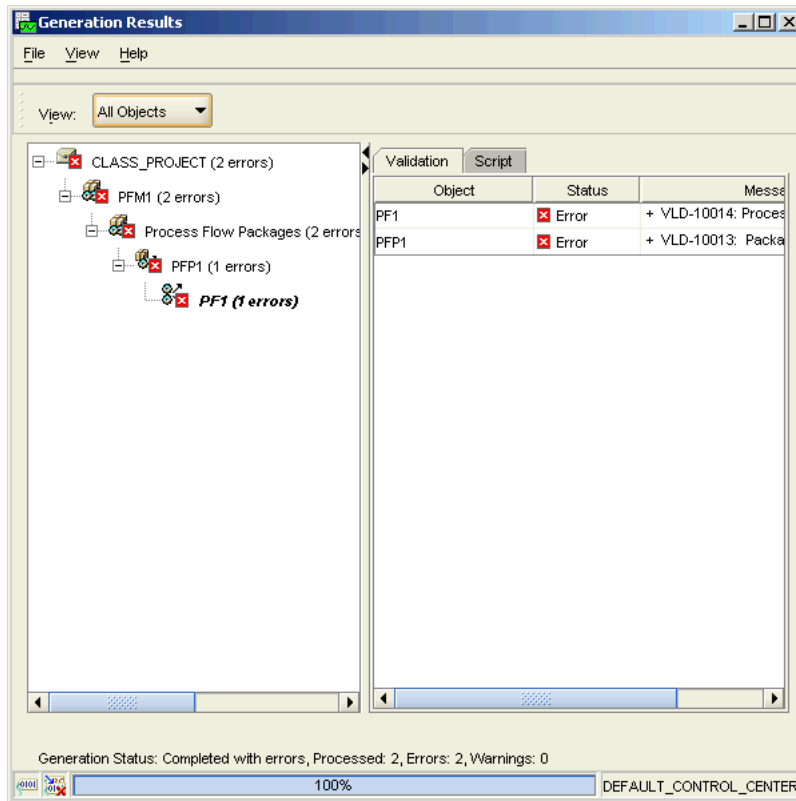
On the Validation tab of the Validation Results window, double-click an object name in the Object column to display the editor for that object. You can fix errors in the editor. Double-click a message in the Message column to display the detailed error message in a message editor window. To save the message to your local system, select **Code** in the menu bar, then select **Save as File**.

Warehouse Builder saves the last validation messages for each previously validated objects. You can access these messages at any time by selecting the object from the console tree in the Project Explorer, select **View** from the menu bar, and then click **Validation Messages**. The messages are displayed in the Validation Results window.

### Generation Errors

After you generate scripts for Warehouse Builder objects, the Generation Results window displays the generation results and errors, as shown in Figure 20–3. Double-click an error under the Messages column on the Validation tab to display a message editor that enables you to save the errors to your local system.

**Figure 20–3 Generation Results Window**



### 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 Project Explorer, select the project. Then from the Tools menu, select **Preferences**. In the Preferences dialog box, click the Logging option in the object tree to the left. In the list box on the right, you can set the log file path, file name and maximum file size. You can also select the types of logs 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 **Messages Log**. This Message Log dialog box 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 (RAB). The RAB 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.

### Name and Address Server Errors

If you are using the Name and Address cleansing service provided by Warehouse Builder, you can encounter related errors.

Name and address server start up and execution errors can be located at:

```
OWB_ORACLE_HOME\owb\bin\admin\NASver.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`.



---

---

## Updating the 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 using two source schemas, Human Resources (HR) and Order Entry (OE) and was loaded into the Warehouse (WH) target schema. Recently you were made aware 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 as show in [Figure 21-1](#). The length of the REGION\_NAME column in the REGIONS table was extended to 100 characters.

**Figure 21-1** Changed REGIONS Table

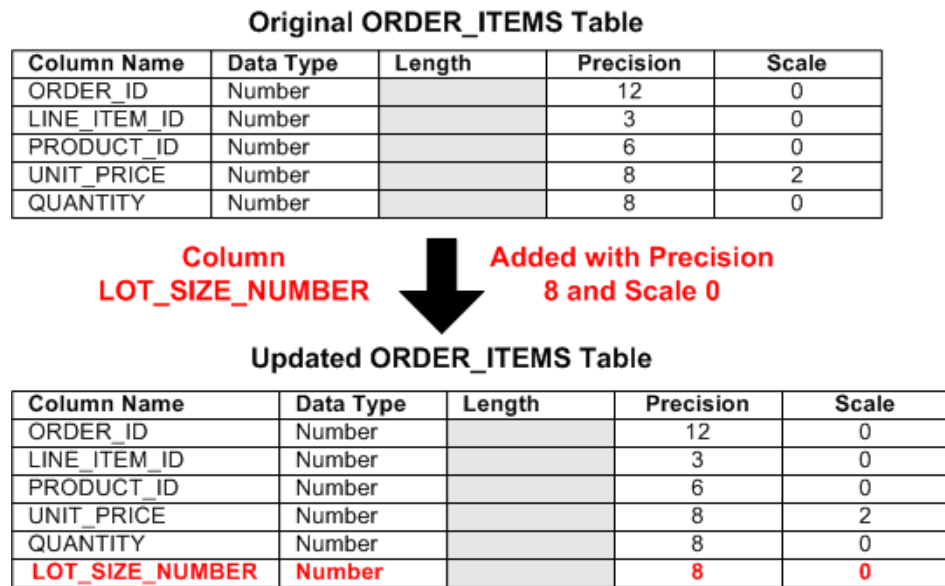
Original REGIONS Table				
Column Name	Data Type	Length	Precision	Scale
REGION_ID	Number		0	0
REGION_NAME	Varchar2	25		

**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 as shown in [Figure 21-2](#). A row called LOT\_SIZE\_NUMBER was added to the ORDER\_ITEMS table with a precision of 8 and scale of 0.

Figure 21–2 Changed ORDER\_ITEMS Table



### Solution

In order 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 need to 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: Re-Deploy 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. In order to do this, you must have a procedure or system in place that can notify you when changes are made to source objects.

In our scenario, you were made aware by the group managing the HR and OE schemas that some objects had been changed. There were two changes, the first was made to the HR schema. The REGION\_NAME column was extended from 25 to 100 characters to accommodate longer data. The second change was made to the OE schema. The LOT\_SIZE\_NUMBER column was added and needs to 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 need to 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 need to 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 need to find a way to integrate the new column into the WH schema. Since 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 need to connect this new column to the SALES cube.

## Step 3: Reimport Changed Objects

Since two source objects have changed, you must start by reimporting 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 displays at the end of the import process displays 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, 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 need to be reconciled.

In our scenario, we need to 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**.

The Metadata Dependency Manager opens and the Impact Analysis diagram reveals that the CUSTOMER dimension in the WH schema is the only object impacted by the REGIONS table.

This step requires that you have already set up the Repository Browser. For more information on setting this up, see *Oracle Warehouse Builder Installation and Administration Guide*.

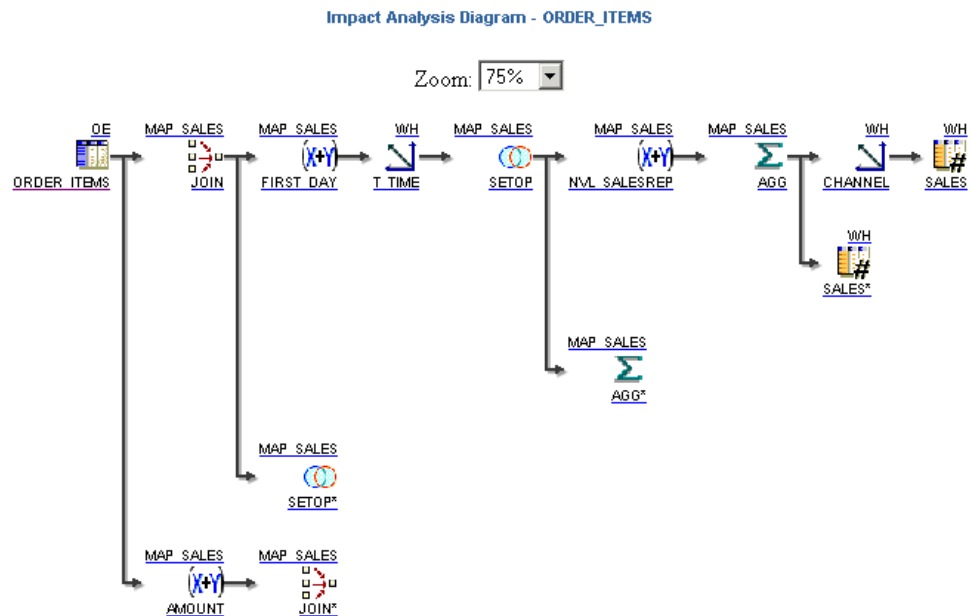
2. Open the CUSTOMER dimension in the Data Object Editor and update the Region Name level attribute to 100 character length.
3. Open the MAP\_CUSTOMER mapping that connects the source to the target. For both the REGIONS table operator and the CUSTOMER dimension operator, perform an inbound synchronization from data object to mapping operator.

The mapping operators must be synchronized with the mapping objects they represent in order to generate code based on the updated objects.

You have now completed updating the metadata associated with Change #1.

For Change #2, since it introduced a new column, you do not need to update the data flow the same way you did for Change #1. Make sure 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 shown in Figure 21-3, we can see that this is only the mapping MAP\_SALES.

**Figure 21-3 Impact Analysis Diagram for ORDER\_ITEMS**



### Step 5: Redesign your Target Schema

Since Change #2 introduced the new LOT\_SIZE\_NUMBER column to the ORDER\_ITEMS table, you need to 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 JOIN, and then to the SETOP, and then add it to the AGG operators. Ensure that you are doing a sum operation in the AGG operator.
5. Finally, connect the LOT\_SIZE\_NUMBER output attribute of the AGG operator to the NUMBER\_OF\_IND\_UNITS input attribute of the SALES cube.

---

### **Step 6: Re-Deploy Scripts**

After the mappings have been debugged, use the Design Center to regenerate and re-deploy 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 generated from the mappings. Use the Mapping Debugger to complete this task. If you find any errors, resolve them and re-deploy the scripts.

### **Step 8: Update Your Discoverer EUL**

If you are using Discoverer as your reporting tool, proceed by updating your EUL.

#### **To update your Discoverer EUL:**

1. Identify the objects that need to be updated in the 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 Project Explorer, 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 Discover EUL.

### **Step 9: Execute the ETL Logic**

After the mappings have been deployed, execute and load data to the target.



---

---

## Managing Multiple Versions of a BI Implementation

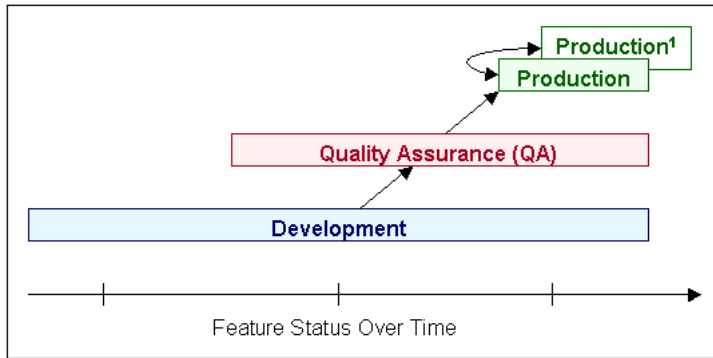
This case study focuses on the lifecycle of a business intelligence system. This case study covers two approaches for managing individually changing versions of your BI system once you have implemented in production. The approach you select depends on the phase of your BI system development lifecycle.

### Scenario

After a period of development and testing, one company implements its BI system in production. The Production version of the system typically changes as new features are incrementally implemented from Development, and as Production bugs are discovered and fixed. At the same time, the Development version of the system continues to evolve with new functionality. This company now has several individually changing versions of the system and faces a challenge familiar to all companies, regardless of how many BI environments they maintain: how to best manage changes in different versions of the system.

One version of this common scenario is depicted in [Figure 22-1](#), where the Development environment is consistently more advanced than the functionality in Production, and QA is somewhere between the two extremes. Development changes are incrementally propagated to QA and subsequently to Production. At the same time, Production has its own cycle of changes, denoted in [Figure 22-1](#) as the shadow environment labeled 'Production<sup>1</sup>', and used for controlled problem solving. 'Production' and 'Production<sup>1</sup>' are at the same stage of development, and serve to illustrate the errors that occur in Production, which are fixed and implemented directly in Production, but that must somehow be merged with Development. Other companies may have fewer or more differing environments for their BI systems, but the same maintenance challenges still apply.

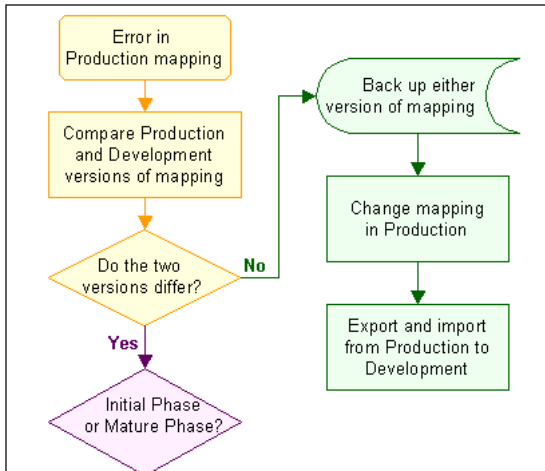
**Figure 22–1 Typical Lifecycle of a Business Intelligence System**



Companies may need multiple environments for their BI systems, as illustrated in [Figure 22–1](#), because they typically implement incremental changes to the system. However, some companies implement only whole projects in Production. [Figure 22–1](#) does not apply to these companies.

In this case study, a company finds a problem with a mapping in Production. The first step is to compare the Production version of the mapping with the Development version of the mapping, as illustrated in [Figure 22–2](#). If the mapping is identical in both environments, the solution is simple: make the changes in either environment and copy the mapping to override the older version. If the mapping in Production differs from its Development version, then the approach depends on whether the BI system is in its initial or mature phase.

**Figure 22–2 Comparing the Production Mapping to Development**



## Approach

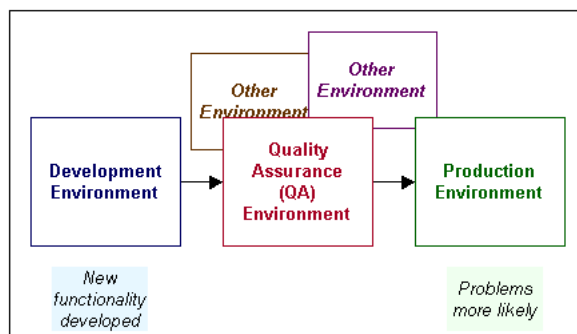
Typically, there are two phases that mark the BI system lifecycle: [Initial Phase](#) and [Mature Phase](#). The two phases present different needs and call for two different version management methodologies, each of which has benefits and drawbacks.

### Initial Phase

After implementation of a business intelligence system in Production, the system is generally in its initial phase, depicted in [Figure 22–3](#). The initial phase is marked by

aggressive changes in the Development environment, coupled with errors sometimes found in Production. Because Production bugs are more likely in this mode, consider a management methodology that facilitates quick updates to each environment.

**Figure 22–3 Initial Phase: Changes in Production More Likely**



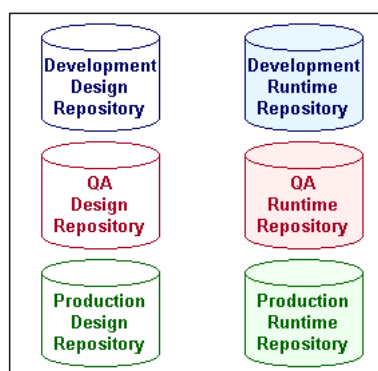
Companies often have two to five different environments. For the initial phase, this company keeps a separate definition of the metadata in each different environment (in this case, Development, QA, and Production). To propagate a change from Production, the company exports only those portions of the system that have changed and imports them into the Development definition.

### Case Study

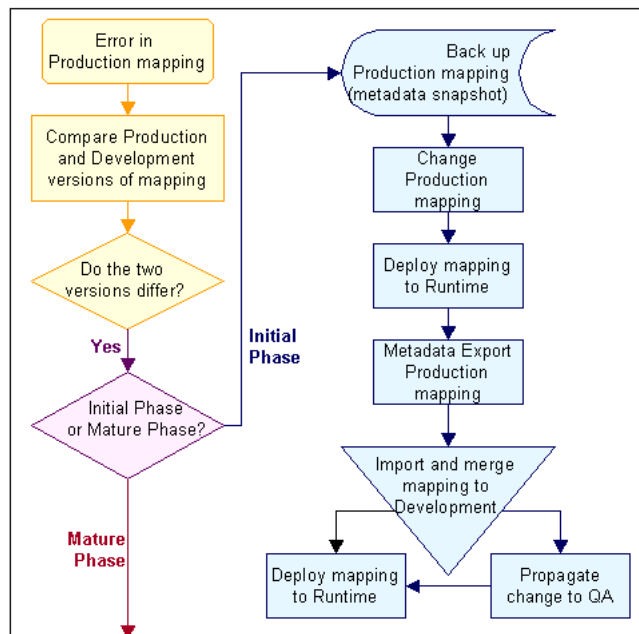
The company has recently implemented its BI system in production, and the system is still in its initial phase, where many additional features are yet to be tested and rolled out. The production system is fairly new, and therefore the occurrence of problems is higher in this phase.

The company decides to keep a separate design repository—or definition of the system design—for each environment, as depicted in [Figure 22–4](#). In addition, they implement their processes into a separate runtime repository for each environment.

**Figure 22–4 Initial Phase: Separate Design Repositories**



In this example, an error occurs in a Production mapping. The company changes the mapping in Production, then exports its definition, and merges it into Development, as illustrated in [Figure 22–5](#).

**Figure 22–5 Initial Phase: Propagate Changes from Production to Development****To correct an error found in a Production mapping during the initial phase:**

1. For backup, capture the definition of any mapping before modifying it.
 

Create a full metadata snapshot of the mapping in the Production Design Repository. Do the same with the Development and QA versions of the mapping. Because you can restore objects from full snapshots only, a full snapshot is essential when you create a backup.
2. Correct the mapping in the Production design repository and deploy it to the Production target schema.
 

This results in a changed version of the mapping that must be propagated to other environments.
3. Use Metadata Export utility to export only the changed mapping from Production.
 

From the Design menu, select **Export** and then **Warehouse Builder Metadata**. This displays the Metadata Export dialog box.
4. Use Metadata Import to import and merge the change to Development and QA.
  - From the Metadata Import dialog box Import Options, select **Merge metadata**.
  - From the Metadata Import dialog box Match By options, select the **Universal Identifier** option.

Matching objects by Universal Identifier is important when maintaining multiple individually changing environments.

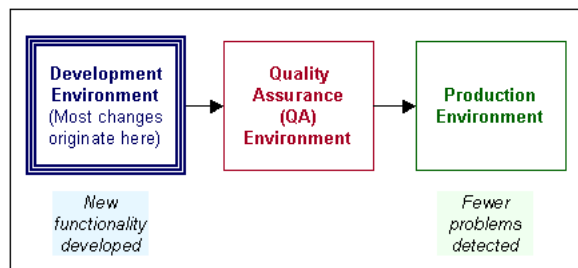
Merging the change into Development and QA can vary in complexity depending on the changed object. If the change in the mapping in this example consists of increasing the column width of a table, the merge is simple. A merge can be more complicated and time-consuming if, for example, join criteria are changed, and other dependencies exist.



## Mature Phase

The second is the mature phase, depicted in [Figure 22–6](#). The mature phase is marked by continued changes in the Development environment, but a decrease in changes required in Production.

**Figure 22–6 Mature Phase: Fewer Changes in Production**



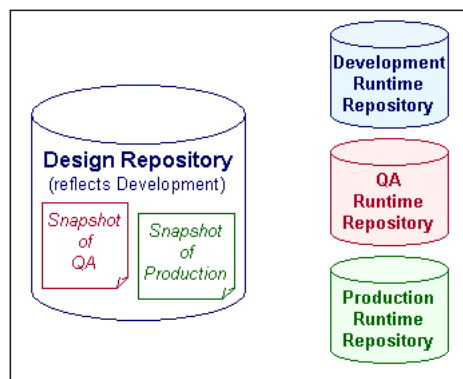
For this mode, the company chooses a methodology that saves space and administration costs: it maintains only one active definition of the BI system design, and this definition reflects the development state of the system. The company stores the design definitions of the QA and Production environments in backup, and extracts and restores changed portions of these systems when required.

### Case Study

At this stage, the company's BI system has stabilized and is now in its mature phase. Some additional functionality is still being developed in the Development environment, but fixes originating in Production are rare.

Although they continue to implement their processes into a separate runtime repository for each environment, the company decides to keep only one design repository, as depicted in [Figure 22–7](#).

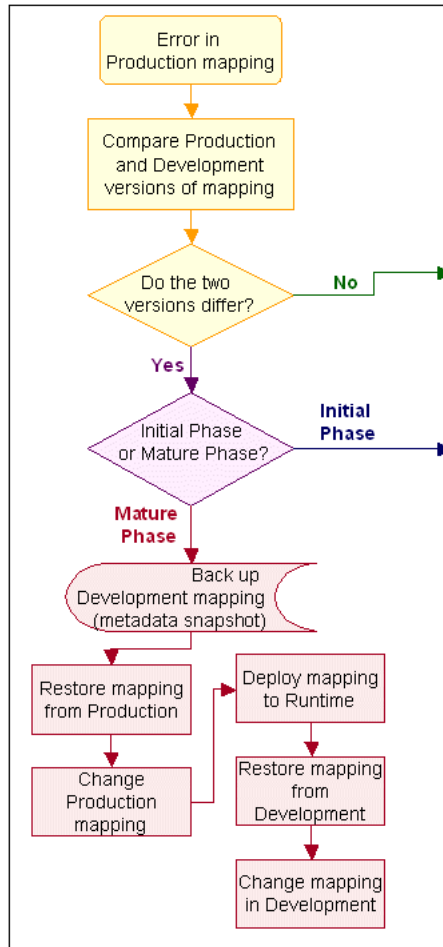
**Figure 22–7 Mature Phase: One Design Repository Reflecting Development**



The one design repository reflects the Development environment, because it is the one active environment that regularly originates design changes. The design repositories from the QA and Production environments are stored as metadata snapshots inside the Development Design Repository. Snapshots are a backup mechanism that consumes minimal space, and still provides access to any objects that you need to restore. Because design changes rarely originate in Production or QA, storing those definitions in snapshots makes sense.

Although it is more rare during the mature phase, errors still occur in the Production environment. In this example, an error occurs in a Production mapping. The company changes the mapping in Production, then restores its definition from a snapshot in Development and makes the same change there, as illustrated in Figure 22–8.

**Figure 22–8 Mature Phase: Propagate Changes from Production to Development**



**To correct an error found in a Production mapping during the mature phase:**

1. Compare the Production version of the mapping in your Production snapshot to the Development version of the same mapping in your Design Repository.
  - If the two differ, the company follows the rest of the steps in this procedure.
  - If the two are identical, correct the mapping as in Step 8, then deploy it to their Design and Production Runtime Repositories, and then update their Production snapshot with the changed mapping.

Consult the online help for instructions on comparing snapshots to objects, deploying, and on updating snapshots.

2. Back up the Development version of the mapping by creating a full metadata snapshot of it.

The Development version of the mapping may differ from the Production version if developers have been working on a new iteration of that mapping. This step

---

preserves their work. Creating a full snapshot is essential, because you can only restore from a full snapshot.

3. Restore the mapping in question from the Production snapshot.  
This mapping should be identical to the one running in Production.  
Consult the online help for instructions on restoring objects from metadata snapshots.
4. Correct the mapping that you have restored from the Production snapshot.
5. Deploy the corrected mapping to the Production Runtime Repository.
6. Remove the existing definition of the mapping from the snapshot of the Production Design Repository and update the snapshot with the new version of the mapping.
7. Restore the mapping from the full snapshot you took as a backup in Step 2.  
This is the mapping from the Development Design Repository. Typically, this mapping has had other work done to it as part of development of new features.  
Optionally repeat this same step for QA.
8. Make the same correction to this Development version of the mapping that you made in Step 4 to the Production version of the mapping.

The cost of this methodology is that every change has to be made at least twice, in the Production and Development versions of the object. The company uses this methodology only because the mature phase does not require frequent changes originating in Production. The benefits of this approach are the minimal administration costs and reduced space requirements on the database.



---

---

# Index

SCDs  
  *see* slowly changing dimensions

## A

---

ABAP  
  scripts, generating, 4-29  
accessing  
  transformation libraries, 10-4  
active configuration, 11-13  
activities  
  in process flows, 8-7  
activity templates, 8-10  
adding  
  mapping operators, 7-11  
addresses, cleansing, 5-26  
analytic workspace, 6-14  
applying  
  data rules, 5-43  
attribute analysis, 5-5  
attribute properties, setting, 7-24  
attributes  
  connecting, 7-17  
  dimension attributes, 6-18  
  level attributes, 6-19  
  setting properties, 7-24  
auto binding  
  rules, 6-24

## B

---

BI lifecycle, 22-1  
BINARY\_DOUBLE data type, 6-3  
BINARY\_FLOAT data type, 6-3  
binding  
  about, 6-12  
  auto binding, 6-12  
  auto binding, rules, 6-24  
  manual binding, 6-13  
BLOB data type, 6-3  
business definitions  
  about, 4-33  
  deploying, 11-8  
  deriving, 6-48  
Business Domain, 4-12, 4-15, 4-19, 4-24  
business identifier, 6-18

business intelligence objects  
  about, 4-32  
  deploying, 11-8  
  deriving, 6-48  
business names  
  business name mode, 3-7  
  maximum length, 3-7  
  requirements, 3-7  
  syntax for business object names, 3-7

## C

---

CA ERwin  
  importing designs from, 16-1  
CASS reporting, 5-30  
changes  
  propagating through multiple BI environments, 22-1  
  rolling out to the target schema, 21-1  
CHAR data type, 6-3  
cleansing  
  addresses, 5-26  
  names, 5-26  
CLOB data type, 6-4  
coding conventions, xviii  
collections  
  Create Collection Wizard, 3-9  
  creating, 3-9  
  deleting shortcuts in, 3-9  
  editing, 3-10  
  renaming, 3-10  
commit strategies  
  committing multiple mappings, 9-10  
  committing to multiple targets, 9-7  
configurations  
  about, 11-12  
  activating, 11-13  
  active configuration, 11-13  
  creating, 11-13  
  named configuration, 11-13  
  setting properties, 11-15  
configuring  
  data objects, 6-45  
  data profiles, 5-34  
  master-detail mappings, 9-17  
  master-detail mappings, direct path load, 9-21

- PL/SQL mappings, 9-1
- runtime parameters, SAP files, 4-29
- SAP file physical properties, 4-28
- SAP, loading type parameter, 4-28
- connecting
  - attributes, 7-17
  - groups, 7-18
  - operators, 7-17
- connectivity
  - ODBC, 4-3
  - OLE DB drivers for heterogeneous data sources, 4-3
- connectivity agent, 4-3
- connectors
  - about, 11-6
  - creating, database connector, 11-7
  - creating, directory connector, 11-7
- containers
  - SAP application, 4-22
- control centers
  - about, 11-4
  - activating, 11-4
  - creating, 11-4
- conventional path loading
  - for master-detail relationships, 9-15
  - master-detail flat files, 9-17
- conventions
  - coding, xviii
  - example, xviii
- correlated commit, design considerations, 9-8
- creating
  - collections, 3-9
  - control centers, 11-4
  - cubes, 6-44
  - data auditors, 5-44
  - data profiles, 5-33
  - data rules, 5-42
  - dimensional objects, 6-11
  - dimensions, 6-41
  - display sets, 7-16
  - locations, 11-5
  - mappings, 7-1
  - PL/SQL types, 10-8, 10-10
  - pluggable mappings, 7-21
  - process flows, 8-4
  - relational data objects, 6-41
  - time dimensions, 6-43
- cubes, 6-35
  - creating, 6-44
  - default aggregation method, 6-37
  - dimensionality, 6-36
  - example, 6-37
  - measures, 6-36
  - MOLAP implementation, 6-39
  - relational implementation, 6-37
  - ROLAP implementation, 6-37
  - solve dependency order, 6-39
- custom transformations
  - about, 10-3
  - defining, 10-5

- editing, 10-12

## D

---

- data
  - cleansing, 5-26
  - test, 7-34
  - viewing, 6-7
- data auditors
  - creating, 5-44
  - using, 5-45
- data flow operators, 7-9
- data model designs
  - importing from third-party tools, 16-1
- Data Object Editor
  - components, 6-6
  - creating data objects, 6-8
  - starting, 6-7
  - using, 6-8
- data objects
  - about, 6-1
  - data type for columns, 6-3
  - dimensional objects, creating, 6-11
  - dimensional objects, implementing, 6-11
  - editing, 6-8
  - monitoring data quality, 5-43
  - naming conventions, 6-6
  - validating, 6-45
  - viewing, 6-7
- data profiles
  - configuring, 5-34
  - creating, 5-33
- data profiling
  - about, 5-3
  - generating corrections, 5-37
  - performance tuning, 5-40
  - performing, 5-32
  - steps, 5-32
  - types, 5-4
  - types, attribute analysis, 5-5
  - types, functional dependency, 5-6
  - types, referential analysis, 5-6
  - viewing corrections, 5-39
  - viewing results, 5-35
- data quality
  - ensuring, 5-1 to 5-9
  - Match-Merge operator, 5-9
- data quality operators, 5-9
- data rules
  - about, 5-31
  - applying, 5-43
  - creating, 5-42
  - deriving, 5-36
  - using, 5-41
- data sources
  - defining SAP objects, 4-22
  - E-Business Suite, 4-11
  - flat files, 4-7
  - Microsoft Excel, 12-1
  - Oracle Heterogeneous Services, 4-3

- Oracle Transparent Gateways, 4-3
  - PeopleSoft, 4-15
  - Siebel, 4-18
  - SQL Server, 13-1
  - data types
    - list of supported, 6-3
  - Data Viewer, 6-7
  - databases
    - importing definitions from, 4-6
    - reimporting definitions, 4-9
  - DATE data type, 6-4
  - debugging
    - mappings, 7-33
    - starting point, 7-37
  - default naming mode, 3-7
  - defining
    - ETL process for SAP objects, 4-27
    - mappings, 7-1
    - mappings containing SAP objects, 4-28
    - process flows, 8-2
    - schedules, 11-17
    - test data, 7-34
    - updating source definitions, 4-11
  - definitions
    - importing definitions from a database, 4-6
    - reimporting database definitions, 4-9
  - deploying
    - about, 11-1
    - business definitions, 11-8
    - data objects, 11-8
    - deployment actions, 11-2
    - deployment errors, 20-1
    - deployment results, 11-11
    - deployment status, 11-3
    - PL/SQL scripts for transparent tables, 4-32
    - process flows, 8-2
    - single design to multiple target systems, 11-15
  - deployment actions, 11-2
  - deployment and execution
    - steps, 11-7
  - deriving
    - business definitions, 6-48
    - data rules, 5-36
  - designing
    - process flows, 8-1
    - target schema, 6-39
    - target schema, dimensional, 6-40
    - target schema, relational, 6-39
  - dimensional objects, 6-10
    - binding, 6-12
    - creating, 6-11
    - creating, about, 6-11
    - deployment options, 6-15
    - implementing, 6-11
    - unbinding, 6-14
  - dimensions
    - about, 6-16
    - binding, 6-24
    - business identifier, 6-18
    - control rows, 6-21
    - creating, 6-41
    - dimension attributes, 6-18
    - dimension roles, 6-19
    - example, 6-20
    - hierarchies, 6-19
    - implementing, 6-22
    - level attributes, 6-19
    - level relationships, 6-20
    - levels, 6-18
    - MOLAP implementation, 6-25
    - parent identifier, 6-19
    - relational implementation, 6-22
    - ROLAP dimension limitations, 6-17
    - ROLAP implementation, 6-22
    - rules, 6-16
    - snowflake schema implementation, 6-23
    - star schema implementation, 6-22
    - surrogate identifier, 6-18
    - time dimensions, about, 6-30
    - value-based hierarchies, 6-21
  - direct path loading
    - for master-detail relationships, 9-19
    - master-detail flat files, 9-21
  - display sets
    - creating, 7-16
    - defined, 7-16
  - displaying
    - welcome pages for wizards, 3-5
  - DML error logging
    - about, 7-29
    - enabling, 7-31
    - in ETL, 7-32
    - limitations, 7-32
- ## E
- 
- E-Business Suite
    - importing metadata, 4-11
  - editing
    - collections, 3-10
    - invalid objects, 6-46
    - PL/SQL types, 10-13
    - process flows, 8-4
    - transformation properties, 10-12
  - enabling
    - DML error logging, 7-31
  - ensuring data quality, 5-1 to 5-9
  - error logs
    - interpreting error logs, 20-1
  - error tables
    - about, 7-30
  - ERwin
    - importing designs from, 16-1
  - ETL
    - improving runtime performance, 9-1
  - example conventions, xviii
  - Excel files
    - loading data from, 12-1
  - executing
    - mappings from SQL\*Plus, 9-10

- execution
  - about, 11-3
  - errors, 20-1
- external tables
  - loading data from flat files, 15-1
- extracting from master-detail flat files, 9-14, 9-15

## F

---

- file transfer
  - in process flows, 19-1
- filters, with a transform, 10-14
- flat files
  - configuring master-detail mappings, 9-21
  - extracting master and detail records, 9-15
  - importing master-detail flat files, 9-15
  - loading data from, 15-1
  - mapping, 7-3
  - master-detail mappings, post-update scripts for
    - direct path loads, 9-21
  - master-detail, example, 9-14
  - master-detail, extracting from, 9-14
  - master-detail, operations after initial load, 9-19
  - variable names, in process flows, 18-1
- FLOAT data type, 6-4
- foreign keys, ensuring referential integrity, 9-13
- FTP
  - using in process flows, 19-1
- functional dependency, 5-6
- functions
  - as transformations, 10-3
  - defining, 10-5
  - editing, 10-12

## G

---

- generating
  - about, 6-47
  - configuring SAP file physical properties, 4-28
  - data objects, 6-47
- generating corrections, 5-37
- generation
  - errors, 20-1
- global shared library, 10-4
- group properties, 7-24
- groups
  - connecting, 7-18
  - setting properties, 7-24

## H

---

- heterogeneous data sources, 4-3
- hiding
  - welcome pages for wizards, 3-5
- hierarchies
  - about, 6-19
  - value-based hierarchies, 6-21
- householding, 5-9

## I

---

- impact analysis
  - rolling out changes to target schema, 21-1
- implementation
  - multiple versions of BI implementation, 22-1
- implementing
  - dimensional objects, 6-11
  - MOLAP cubes, 6-39
  - MOLAP dimensions, 6-25
  - relational cubes, 6-37
  - relational dimensions, 6-22
  - ROLAP cubes, 6-37
  - ROLAP dimensions, 6-22
  - snowflake schema, 6-23
  - star schema, 6-22
- importing
  - definitions, database systems, 4-6
  - designs from third-party tools, 16-1
  - flat files, 4-7
  - from E-Business Suite, 4-11
  - from flat files, 4-7
  - from Microsoft Excel, 12-1
  - from PeopleSoft, 4-15
  - from SAP R/3, 4-20
  - from Siebel, 4-18
  - from SQL Server, 13-1
  - Import Metadata Wizard, 4-6
  - master-detail flat files, 9-15
  - Oracle database metadata, 4-6
  - PL/SQL functions, 10-14
  - reimporting database definitions, 4-9
- improving runtime performance, 9-1
- INI file for SAP Connector, 4-22
- input signature, 7-22
- installation
  - errors, 20-1
- INTEGER data type, 6-4
- INTERVAL DAY TO SECOND data type, 6-4
- INTERVAL YEAR TO MONTH data type, 6-4

## L

---

- language, SAP, 4-29
- languages, setting locale preferences, 3-2
- levels
  - dimension, 6-18
- loading
  - conventional path for master-detail targets, 9-17
  - data from Excel files, 12-1
  - data from flat files, 15-1
  - direct path for master-detail targets, 9-21
  - master and detail records, 9-15
  - master-detail relationships, 9-15, 9-19
  - master-detail relationships, direct path, 9-19
  - SAP data into the workspace, 4-30
  - transaction data, 14-1
- loading types
  - for SAP, 4-28
- locales, setting, 3-2
- locations



- creating, 11-5
- deleting, 11-6
- registering, 11-6
- unregistering, 11-6
- logical name mode *See* business name mode
- logical names *See* business names, 3-7
- logs
  - interpreting error logs, 20-1
  - message log preferences, 3-6
- LONG data type, 6-4

## M

---

- main procedure, 9-10
- Mapping Editor
  - about, 7-5
  - components, 7-5
  - toolbars, 7-7
  - windows, 7-6
- mapping operators
  - about, 7-1
  - adding, 7-11
  - connecting, 7-17
  - editing, 7-12
  - Match-Merge operator, 5-9
  - Name and Address operator, 5-26
  - setting, 7-24
  - synchronizing with workspace objects, 7-25
  - that bind to repository objects, 7-11
  - types of, 7-8
- mappings
  - about, 7-1
  - configuring, 9-1
  - configuring master-detail, 9-17
  - creating, 7-1
  - debugging, 7-33
  - defining, 7-1
  - defining mappings with SAP objects, 4-28
  - executing from SQL\*Plus, 9-10
  - for flat files, 7-3
  - for PEL, 9-23
  - master-detail mappings, 9-13
  - naming conventions, 7-14, 8-10
  - PL/SQL mappings, 9-1
  - setting properties, 7-23
- master-detail flat files
  - as sources, about, 9-14
  - configuring mappings, 9-17
  - configuring mappings, direct path load, 9-21
  - example of a master-detail flat file, 9-14
  - extracting from, 9-15
  - extracting from, using conventional path load, 9-15
  - extracting from, using direct path load, 9-19
  - importing and sampling, 9-15
  - operations after initial load, 9-19
  - performance, 9-15, 9-19
  - post-update scripts for direct path loads, 9-21
  - RECNUM, 9-20
  - sample mapping, conventional path loading, 9-17

- sample mapping, direct path loading, 9-21
- match rules
  - address match rules, 5-20
  - conditional match rules, 5-13
  - custom match rules, 5-22
  - firm match rules, 5-18
  - person match rules, 5-16
  - weight match rules, 5-15
- Match-Merge operator, 5-9
  - example, 5-10
  - match rules, 5-12
  - merge rules, 5-23
  - restrictions, 5-10
  - using, 5-24
- MDSYS.SDO\_DIM\_ARRAY data type, 6-4
- MDSYS.SDO\_DIM\_ELEMENT data type, 6-4
- MDSYS.SDO\_ELEM\_INFO\_ARRAY data type, 6-4
- MDSYS.SDO\_GEOMETRY data type, 6-4
- MDSYS.SDO\_ORDINATE\_ARRAY data type, 6-4
- MDSYS.SDO\_POINT\_TYPE data type, 6-4
- MDSYS.SDOAGGRTYPE data type, 6-4
- message log preferences, 3-6
- metadata
  - import and export errors, 20-1
  - Import Metadata Wizard, 4-6
  - importing from databases, 4-6
  - importing from flat files, 4-7
- Microsoft Excel
  - loading data from, 12-1
- modes
  - business name mode, 3-7
  - logical name mode *See* business name mode
  - naming mode, default, 3-7
  - physical name mode, 3-7
- modules
  - defining SAP objects, 4-22
  - process flows, 8-2, 8-3
  - SAP application, 4-22
- MOLAP implementation, 6-10
- monitoring
  - data objects, using auditors, 5-45
  - data quality, 5-43
- multiple configurations
  - scenario, 11-14
- multiple-record-type flat files
  - master-detail structure, 9-14
  - master-detail structure, example of, 9-14

## N

---

- Name and Address
  - country postal certifications, Australia, 5-31
  - country postal certifications, Canada, 5-30
  - country postal certifications, United States, 5-30
  - operator, 5-26
  - purchasing license, 5-26
- Name and Address operator, 5-26
  - best practices, 5-29
  - CASS reporting, 5-30
  - enabling, 5-26

- Name and Address server
  - errors, 20-1
- names
  - business name mode, 3-7
  - business object names, syntax for, 3-7
  - cleansing, 5-26
  - default naming mode, 3-7
  - flat files with variable names in process
    - flows, 18-1
  - logical name mode *See* business name mode
  - physical name mode, 3-7
  - physical object names, syntax for, 3-7
- naming
  - modes, default naming mode, 3-7
  - objects, business name mode, 3-7
  - objects, logical name mode *See* business name mode
  - objects, physical name mode, 3-7
  - setting naming preferences, 3-6
  - transformation names, 3-7
- naming conventions, for data objects, 6-6
- NCHAR data type, 6-4
- NCLOB data type, 6-4
- non-Oracle database systems
  - as data sources, 4-3
- NUMBER data type, 6-5
- NVARCHAR2 data type, 6-5

## O

---

- objects
  - defining mappings with SAP objects, 4-28
  - invalid objects, editing, 6-46
  - syntax for business names, 3-7
  - syntax for logical names, 3-7
  - syntax for physical names, 3-7
- ODBC for heterogeneous data sources, 4-3
- OLE DB drivers for heterogeneous data sources, 4-3
- operating modes
  - row based, 9-5
  - row based (target only), 9-6
  - selecting a default mode, 9-4
  - set based, 9-5
- operator attributes, 7-17
- operator properties
  - setting, 7-24
- operators
  - connecting, 7-17
  - data flow, 7-9
  - data quality operators, 5-9
  - editing, 7-12
  - mapping
    - binding to repository objects, 7-11
  - Match-Merge operator, 5-9
  - Name and Address, 5-26
  - Name and Address operator, 5-26
  - synchronizing with workspace objects, 7-25
- operators, mapping, 7-1
  - adding, 7-11
  - connecting, 7-17

- editing, 7-12
  - types of, 7-8
- optimizing the repository, 3-5
- Oracle Heterogeneous Services, 4-3
- Oracle library, 10-4
- Oracle Transparent Gateways, 4-3
- ordering
  - multiple targets, 9-13
- output signature, 7-22

## P

---

- packages
  - as transformations, 10-3
  - defining, 10-5
  - editing, 10-12
  - process flows, 8-2, 8-3
- parent identifier, 6-19
- Partition Exchange Loading (PEL), 9-22
  - about, 9-23
  - configuring targets for, 9-26
  - mappings for, 9-23
  - performance considerations, 9-25
  - restrictions on, 9-26, 9-27
- PeopleSoft
  - importing metadata, 4-15
- physical names
  - physical name mode, 3-7
  - syntax for physical object names, 3-7
- PL/SQL
  - deploying PL/SQL scripts for transparent tables, 4-32
- PL/SQL code
  - handling existing PL/SQL code, 17-1
- PL/SQL mappings, 9-1
- PL/SQL types
  - about, 10-8
  - as transformations, 10-3
  - creating, 10-8, 10-10
  - editing, 10-13
- pluggable mappings
  - about, 7-20
  - creating, 7-21
  - embedded, 7-21
  - reusable, 7-20
- pooled tables, 4-21
- predefined transformations, 10-2
- preferences
  - displaying welcome pages for wizards, 3-5
  - locale, 3-2
  - message log preferences, 3-6
  - naming preferences, 3-6
- procedures
  - as transformations, 10-3
  - defining, 10-5
  - editing, 10-12
- Process Flow Editor, 8-4
- process flows
  - about, 8-1
  - activities in, 8-7

- complex conditions in, 8-17
- creating, 8-4
- defining, 8-2
- deploying, 8-2
- designing, 8-1
- editing, 8-4
- handling flat files with variable names, 18-1
- modules, 8-2, 8-3
- packages, 8-2, 8-3
- transferring remote files with FTP, 19-1
- transitions, 8-13
- See also* process runs
- production
  - changes in, 22-1
- projects
  - importing PL/SQL into, 10-14
- properties
  - mapping, 7-23
  - setting, 7-24

**R**

---

- RAW data type, 6-5
- RECNUM attribute, 9-20
- RECNUM columns, 9-20
- records
  - extracting and loading master and detail records, 9-15
  - relationships between masters and details in flat files, 9-14
- referential analysis, 5-6
- referential integrity, ensuring in mappings, 9-13
- reimporting
  - database definitions, 4-9
- relating master and detail records, 9-14
- relational data objects
  - creating, 6-41
- relational implementation, 6-10
- remote files
  - transferring, 19-1
- remote function call (RFC), 4-22
  - RFC connection, 4-23
  - SAP RFC connection, 4-23
- renaming
  - collections, 3-10
- repository
  - optimizing, 3-5
- requirements
  - for business names, 3-7
- reverse engineering, 4-5
- ROLAP implementation, 6-10
- roles
  - dimension roles, 6-19
- row based, 9-5
- row based (target only), 9-6
- row based versus set based
  - loading transaction data, 14-1
- runtime performance, improving, 9-1
- runtime, SAP, 4-29

**S**

---

- sampling
  - master-detail flat files, 9-15
- SAP
  - Business Domain, 4-12, 4-15, 4-19, 4-24
  - configuring SAP file physical properties, 4-28
  - defining ETL process for SAP objects, 4-27
  - defining mappings with SAP objects, 4-28
  - defining SAP objects, 4-22
  - loading SAP data into the workspace, 4-30
- SAP ABAP Editor, 4-31
- SAP application source module, 4-22
- SAP Business Areas, 4-21
- SAP Connector
  - about, 4-20
  - creating definitions, 4-22
  - required files, 4-22
- SAP file physical properties
  - Data File Name, 4-29
  - File Delimiter for Staging File, 4-29
  - Nested Loop, 4-29
  - SAP System Version, 4-29
  - SQL Join Collapsing, 4-29
  - Staging File Directory, 4-29
  - Use Single Select, 4-29
- SAP parameters
  - language, 4-29
  - loading type, 4-28
  - runtime, 4-29
- SAP R/3
  - executing a SAP mapping, 4-31
  - importing metadata, 4-20
  - updating source modules, 4-27
- SAP remote function call (SAPRFC.INI), 4-22
- SAP table types
  - cluster, 4-21
  - importing, 4-21
  - pooled, 4-21
  - transparent, 4-21
- SAPRFC.INI file, 4-22, 4-23
- schedules
  - creating, 11-17
  - defining, 11-17
  - example, 11-18
  - using, 11-17
- scheduling
  - about, 11-16
  - ETL jobs, 11-12
- set based mode, 9-5
- set based update, 9-5
- set based versus row based
  - loading transaction data, 14-1
- set based versus row based modes, 9-4
- setting
  - a starting point, 7-37
  - locale preferences, 3-2
  - mapping properties, 7-23
  - message log preferences, 3-6
  - naming preferences, 3-6
  - wizard preferences, 3-5

- Setting the Language Parameter, 4-29
- Setting the Runtime Parameter, 4-29
- Siebel
  - importing metadata, 4-18
- signatures
  - input, 7-22
  - output, 7-22
- Six Sigma
  - metrics, 5-8
- slowly changing dimensions
  - about, 6-25
  - hierarchy versioning, 6-27
  - type 1, 6-26
  - type 2, about, 6-26
  - type 2, updating, 6-28
  - type 3, 6-30
  - types, 6-25
- source modules
  - importing definitions, 4-6
  - SAP application, 4-22
- sources
  - master-detail flat file sources, 9-14
  - master-detail flat files, 9-14
  - master-detail flat files, example, 9-14
  - updating source definitions, 4-11
- SQL\*Loader
  - loading data from flat files, 15-1
- starting
  - Data Object Editor, 6-7
- starting point, setting, 7-37
- surrogate identifier, 6-18
- synchronizing
  - operators and workspace objects, 7-25
- syntax
  - for business object names, 3-7
  - for logical object names, 3-7
  - for physical object names, 3-7
- SYS.ANYDATA data type, 6-5
- SYS.LCR\$\_ROW\_RECORD data type, 6-5

## T

---

- tables
  - deploying PL/SQL scripts for transparent tables, 4-32
- target load ordering, 9-13
- target schema
  - designing, 6-39
  - rolling out changes, 21-1
- targets
  - defining load orders, 9-13
  - multiple targets in a mapping, 9-7
- templates
  - activity, 8-10
- test data, defining, 7-34
- Text String Matching, 4-12, 4-15, 4-19, 4-25
- time dimensions
  - about, 6-30
  - creating, 6-43
  - creation best practices, 6-31

- dimension attributes, 6-32
- hierarchies, about, 6-33
- implementing, 6-33
- level attributes, about, 6-32
- levels, about, 6-31
- populating, 6-34
- TIMESTAMP data type, 6-5
- TIMESTAMP WITH LOCAL TIMEZONE data type, 6-5
- TIMESTAMP WITH TIMEZONE data type, 6-5
- transaction data
  - loading, 14-1
- transferring
  - remote files, 19-1
- transformation filter data, 10-14
- transformation libraries
  - about, 10-4
  - accessing, 10-4
  - global shared library, 10-4
  - Oracle library, 10-4
  - types, 10-4
- transformation properties, 10-12
- transformations
  - about, 10-2
  - custom, 10-3
  - custom example, 10-14
  - importing, 10-14
  - introduction to, 10-1 to 10-14
  - names, unique, 3-7
  - predefined, 10-2
  - types of, 10-2
- transition conditions, 8-17
- Transition Editor, 8-17
- transitions
  - conditions of, 8-17
  - in process flows, 8-13
- transparent tables, 4-21
- tuning
  - data profiling performance, 5-40
- type 2 SCDs
  - about, 6-26
  - updating, 6-28
- type 3 SCDs
  - about, 6-30

## U

---

- unique
  - transformation names, 3-7
- updating
  - SAP object definitions, 4-27
  - source definitions, 4-11
  - target schema, 21-1

## V

---

- validating
  - about, 6-45
  - data objects, 6-46
  - editing invalid objects, 6-46

- validation
  - about, 6-45
  - editing invalid objects, 6-46
  - errors, 20-1
- VARCHAR data type, 6-5
- VARCHAR2 data type, 6-5
- viewing
  - data, 6-7
  - data objects, 6-7

## **W**

---

- wizards
  - Create Collection Wizard, 3-9
  - Create Data Auditor Wizard, 5-44
  - Import Metadata Wizard, 4-6
  - welcome pages, displaying, 3-5

## **X**

---

- XMLFORMAT data type, 6-5
- XMLTYPE data type, 6-6

