

**Oracle Data Integrator 12c:
Integration and Administration**
Student Guide – Volume II

D82167GC10
Edition 1.0
May 2014
D86565

ORACLE®

Author

Steve Friedberg

**Technical Contributors
and Reviewers**

Phil Scott
Gerry Jurrens
Brent Dayley
Surendra Babu
Rick Green
Viktor Tchemodanov
Julien Testut
Alex Kotopoulos
Alessandro Leite

Editor

Daniel Milne

Graphic Designer

Seema Bopaiah

Publishers

Michael Sebastian
Veena Narasimhan

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Disclaimer

This document contains proprietary information and is protected by copyright and other intellectual property laws. You may copy and print this document solely for your own use in an Oracle training course. The document may not be modified or altered in any way. Except where your use constitutes "fair use" under copyright law, you may not use, share, download, upload, copy, print, display, perform, reproduce, publish, license, post, transmit, or distribute this document in whole or in part without the express authorization of Oracle.

The information contained in this document is subject to change without notice. If you find any problems in the document, please report them in writing to: Oracle University, 500 Oracle Parkway, Redwood Shores, California 94065 USA. This document is not warranted to be error-free.

Restricted Rights Notice

If this documentation is delivered to the United States Government or anyone using the documentation on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS

The U.S. Government's rights to use, modify, reproduce, release, perform, display, or disclose these training materials are restricted by the terms of the applicable Oracle license agreement and/or the applicable U.S. Government contract.

Trademark Notice

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Contents

1 Introduction to Integration and Administration

Course Objectives	1-2
Lesson Objectives	1-3
Agenda of Lessons	1-4
Agenda	1-7
Why Oracle Data Integrator?	1-8
Conventional Integration Process: ETL	1-10
Extract Load Transform (E-LT)	1-11
ODI Architecture and Components	1-13
ODI Architecture	1-14
ODI Components: Overview	1-15
Using ODI Studio	1-16
Designer Navigator (Work Repository)	1-17
Operator Navigator (Work Repository)	1-18
Topology Navigator (Master Repository)	1-19
Security Navigator (Master Repository)	1-20
What Is an Agent?	1-21
ODI Agents	1-22
Three Types of Agents: Java EE, Standalone, Collocated Standalone	1-23
Using the Three Types of Agents	1-24
Standalone Agent: Example	1-25
ODI Console	1-26
Enterprise Manager FMW Console	1-27
Management Pack for ODI for Enterprise Manager Cloud Control	1-28
Management Pack for ODI for EM CC ODI Home Page	1-29
Agenda	1-30
ODI Repositories	1-31
Master and Work Repositories	1-32
Repository Setup: Example	1-34
Repository Setup: Multiple Master Repositories	1-35
Components: Global View	1-36
Possible ODI Methodology	1-37
Checklist of Practice Activities	1-38
Starting Oracle Data Integrator	1-39
Using Online Help	1-40

Quiz 1-41
Summary 1-43
Practice 1-1: Logging In and Help Overview 1-44

2 Administering ODI Repositories

Objectives 2-2
Agenda 2-3
Initial Repository Administration Tasks 2-4
Steps to Set Up ODI Repositories 2-5
1. Run Repository Creation Utility 2-6
1a. Create Schemas 2-7
1b. Create Passwords and Tablespaces 2-8
2. Connect to the Master/Work Repository 3. Create a Wallet 2-9
Connecting to the Master/Work Repository 2-10
Exporting the Master Repository 2-11
Importing the Master Repository 2-12
Creating a Work Repository 2-13
Changing the Work Repository Password 2-15
Quiz 2-16
Summary 2-17
Checklist of Practice Activities 2-18
Practice 2-1: Creating and Connecting to ODI Master and Work Repositories 2-19

3 ODI Topology Concepts

Objectives 3-2
Agenda 3-3
What Is Topology? 3-4
What Is in the Topology? 3-5
Agenda 3-6
What Is a Data Server? 3-7
Data Servers: Examples 3-8
Important Guideline 1 3-9
What Is a Physical Schema? 3-10
Physical Schemas: Properties 3-11
Technology Terminology Among Vendors 3-12
Important Guideline 2 3-13
Agenda 3-14
Infrastructure for Two Production Sites: Example 3-15
ODI Design: Physical Architecture of the Two Production Sites 3-16
Logical Schemas and Contexts 3-17
What Is a Logical Schema? 3-18

Important Guideline 3	3-19
Logical Versus Physical Architecture	3-20
Design Time Versus Run Time	3-21
What Is a Context?	3-22
A Context Maps a Logical to a Physical Schema	3-23
Defining Contexts	3-24
Mapping Logical and Physical Resources	3-25
Agenda	3-27
ODI Physical Agents	3-28
Creating a Physical Agent	3-29
ODI Agent Parameters	3-30
Launching a Stand-Alone Agent: Examples	3-32
Stopping the ODI Agent	3-33
Deploying and Configuring a Java EE Agent	3-34
Load Balancing: Example	3-37
Important Guideline 5	3-39
Infrastructure with Agents: Example	3-40
Defining Agents: Example	3-41
Special Case: Fragmentation Problem	3-42
Special Case: Important Guideline 6	3-44
Special Case: Defining the Physical Architecture	3-45
Special Case: The Infrastructure	3-46
Special Case: Physical Architecture in ODI	3-47
Agenda	3-48
Planning the Topology	3-49
Matrix of Logical and Physical Mappings	3-50
Quiz	3-51
Summary	3-54
Checklist of Practice Activities	3-55
Practice 3-1: Configuring a Standalone Agent by Using the Common Administration Model	3-56

4 Describing the Physical and Logical Architecture

Objectives	4-2
Agenda	4-3
What Topology Navigator Contains	4-4
Topology Navigator: Overview	4-5
Review: Context Connects Logical to Physical	4-7
Objects You Create in the Practice	4-8
Defining a Context	4-9
Agenda	4-10

Physical Architecture View	4-11
Prerequisites for Connecting to a Server	4-12
Important Note	4-13
Creating a Data Server	4-14
Creating a Data Server: JDBC	4-15
JDBC Driver	4-16
JDBC URL	4-17
Creating a Data Server: JNDI	4-18
Testing a Data Server Connection	4-19
Creating a Physical Schema	4-20
Agenda	4-21
Logical Architecture and Context Views	4-22
Creating a Logical Schema	4-23
Creating a Logical Agent	4-24
Editing a Context to Link Logical and Physical Agents	4-25
Quiz	4-26
Summary	4-28
Checklist of Practice Activities	4-29
Practice 4-1: Working with Topology	4-30

5 Setting Up a New ODI Project

Objectives	5-2
Agenda	5-3
What Is a Project?	5-4
Oracle Data Integrator Projects: Overview	5-5
How to Use ODI Projects in Your Work	5-6
Creating a New Project	5-7
Agenda	5-8
What Is a Folder?	5-9
Creating a New Folder	5-10
Organizing Projects and Folders	5-11
Agenda	5-12
What Is a Knowledge Module?	5-13
Types of Knowledge Modules	5-14
Which Knowledge Modules Are Needed?	5-15
Knowledge Modules: Examples	5-16
Importing Knowledge Modules	5-17
Replacing Existing KMs	5-18
Knowledge Module Editor	5-20
Editing a Knowledge Module	5-21
Agenda	5-22

Exporting and Importing	5-23
Exporting an Object	5-24
Importing an Object	5-25
ID Numbers: Overview	5-26
Import Types	5-27
Choosing the Import Mode	5-28
Import Report	5-29
Agenda	5-30
What Is a Marker?	5-31
Tagging Objects with Markers	5-32
Removing Markers	5-33
Marker Groups	5-34
Project and Global Markers	5-35
Creating a Marker Group	5-36
Quiz	5-37
Summary	5-39
Checklist of Practice Activities	5-40
Practice 5-1: Setting Up a New ODI Project	5-41

6 Oracle Data Integrator Model Concepts

Objectives	6-2
What Is a Model?	6-3
Agenda	6-4
Relational Model	6-5
Relational Model: Tables and Columns	6-6
Relational Model: Keys	6-7
Relational Model: Foreign Keys	6-8
Relational Model: Constraints	6-9
Relational Model: Indexes	6-11
Relational Model Support in ODI	6-12
Additional Metadata in ODI	6-13
FlexFields	6-15
Agenda	6-16
What Is Reverse-Engineering?	6-17
Methods for DBMS Reverse-Engineering	6-18
Other Methods for Reverse-Engineering	6-19
Standard Versus Customized Reverse-Engineering	6-20
Reverse-Engineering Life Cycle	6-21
Agenda	6-22
Creating a Model by Reverse-Engineering	6-23
Step 1: Creating and Naming a New Model	6-24

Note: Creating and Naming a New Model	6-25
Step 2: Defining a Reverse-Engineering Strategy	6-26
Step 3: Starting the Reverse-Engineering Process	6-28
Using RKM for Customized Reverse-Engineering	6-29
Selective Reverse-Engineering	6-31
Step 4: Fleshing Out Models	6-32
Shortcuts	6-33
Smart Export and Import	6-34
Quiz	6-35
Summary	6-37
Checklist of Practice Activities	6-38
Practice 6-1 Overview: Creating Models by Reverse-Engineering	6-39

7 Organizing ODI Models and Creating ODI Datastores

Objectives	7-2
Agenda	7-3
What Is a Model Folder?	7-4
Creating a Model Folder	7-5
What Is a Submodel?	7-6
Creating a Submodel	7-7
Organizing Datastores into Submodels	7-8
Setting Up Automatic Distribution	7-9
Agenda	7-10
Creating Datastores	7-11
Creating a Datastore in a Model	7-12
Adding Columns to a Datastore	7-13
Agenda	7-14
What Is a Constraint in ODI?	7-15
Constraints in ODI	7-16
Creating a Mandatory Column	7-17
Agenda	7-18
Creating a Key	7-19
Checking a Key	7-20
Creating a Reference	7-21
Creating a Simple Reference	7-22
Creating a Complex Reference	7-23
Checking a Reference	7-24
Agenda	7-25
Creating a Condition	7-26
Checking a Condition	7-27
Agenda	7-28

Audit/Explore: When and Why	7-29
Audit/Explore Process: Overview	7-30
Agenda	7-31
Displaying the Contents of a Datastore	7-32
Viewing the Distribution of Values	7-33
Analyzing the Contents of a Datastore	7-34
Agenda	7-35
Defining Business Rules in ODI	7-36
From Business Rules to Constraints	7-37
Deducing Constraints from Data Analysis	7-38
Testing a Constraint	7-39
Auditing a Model or Datastore	7-40
Reviewing Erroneous Records	7-41
Quiz	7-42
Summary	7-44
Checklist of Practice Activities	7-45
Practice 7-1: Checking Data Quality in the Model	7-46

8 ODI Mapping Concepts

Objectives	8-2
Agenda	8-3
What Is a Mapping?	8-4
Business Rules for Mappings	8-5
Where Are the Rules Defined?	8-6
Agenda	8-7
What Is an Expression?	8-8
What Is a Join?	8-9
What Is a Filter?	8-10
What Is a Lookup?	8-11
What Is a Set?	8-12
What Are Some of the Others?	8-13
New with Patch: Pivot and Unpivot	8-14
Agenda	8-15
How Does ODI Implement Business Rules?	8-16
Business Problem	8-17
Implementing the Rules	8-18
Integration Process	8-19
Process Details	8-20
Process Implementation: Example 1	8-21
Process Implementation: Example 2	8-22
Process Implementation: Example 3	8-23

Agenda	8-24
What Is the Staging Area?	8-25
Execution Location	8-26
Agenda	8-27
From Business Rules to Processes	8-28
Knowledge Modules	8-29
What Is a Knowledge Module?	8-30
Code Generation	8-31
KM Types Used in Mappings	8-32
Agenda	8-33
Purpose of a Mapping	8-34
What Is an Expression?	8-35
Creating a One-to-One Mapping	8-36
Creating and Naming a Mapping	8-37
Defining the Target Datastore	8-38
Multiple Targets	8-39
Defining the Source Datastore	8-40
Connecting the Ports to Make the Map	8-41
Defining the Expressions	8-42
Valid Expression Types	8-43
Saving the Mapping	8-44
Running the Mapping	8-45
Quiz	8-46
Summary	8-48
Checklist of Practice Activities	8-49
Practice 8-1: Mapping: Simple Transformations	8-50

9 Designing Mappings

Objectives	9-2
Agenda	9-3
Multiple-Source Datastores	9-4
Creating a Join Manually	9-5
Advanced Joins	9-6
Types of Joins	9-7
Setting Up a Join	9-8
Creating Lookups	9-10
Using Lookups	9-11
Agenda	9-13
Filters in ODI	9-14
Defining a Filter Manually	9-15
Setting Up a Filter	9-16

Agenda	9-17
Physical Mapping Diagram	9-18
Flow in the Physical Diagram	9-20
What Defines the Flow?	9-21
Scenario	9-22
Basic Process	9-23
Agenda	9-24
Purpose of a Staging Area	9-25
Placing the Staging Area	9-26
Important Note	9-27
Specifying the Staging Area	9-28
Agenda	9-29
Options for Expressions	9-30
Setting Options for Expressions	9-31
Disabling an Expression	9-32
Enabling a Mapping for Inserts or Updates	9-33
Agenda	9-34
Execution Location and Syntax	9-35
Why Change the Execution Location?	9-36
Changing the Execution Location	9-37
ODI Mapping Execution Simulation	9-38
Agenda	9-39
Which KMs for Which Flow?	9-40
Knowledge Modules: Additional Information	9-42
Identifying IKMs and LKMs	9-43
IKMs and LKMs: Strategies and Methods	9-44
Specifying an LKM	9-45
Specifying an IKM	9-46
Common KM Options	9-47
Flow: Example 1	9-48
Flow: Example 2	9-49
Flow: Example 3	9-50
Quiz	9-51
Summary	9-52
Checklist of Practice Activities	9-53
Practice 9-1: Mapping: Complex Transformations	9-54
Practice 9-2: Mapping: Implementing Lookup	9-55

10 Mappings: Monitoring and Troubleshooting

Objectives 10-2

Agenda 10-3

Operator Navigator: Viewing the Log	10-4
Using Operator Navigator	10-5
Hierarchy: Sessions, Steps, Tasks	10-6
Viewing Details of Sessions, Steps, and Tasks	10-7
Monitoring Execution of an Mapping	10-8
Troubleshooting a Session	10-9
1. Identifying the Error	10-10
2. Reviewing the Code	10-11
3. Fixing the Code and Restarting the Session	10-12
4. Fixing the Mapping	10-13
Keys to Reviewing the Generated Code	10-14
Agenda	10-15
Common Errors and Symptoms	10-16
Important Note	10-18
Tips for Preventing Errors	10-19
Using Attribute Panel for Quick Edits	10-20
Quiz	10-21
Summary	10-23
Checklist of Practice Activities	10-24
Practice 10-1: Creating ODI Mapping: Exporting a Flat File to a Relational Table	10-25

11 Designing Mappings: Advanced Topics 1

Objectives	11-2
Agenda	11-3
Business Rules in Mappings	11-4
Business Rule Elements	11-5
More Elements	11-6
Expression Editor	11-7
Agenda	11-9
Using a Variable in Code	11-10
Binding Versus Substitution	11-12
Case Sensitivity	11-13
Agenda	11-14
Defining a Dataset	11-15
Using Set-Based Operators	11-16
Example of SET: UNION	11-17
Agenda	11-18
Types of Sequences	11-19
Support for Native Sequences	11-20
Creating a Native Sequence	11-21

Referring to Sequences	11-22
Note: Sequences Updated by Agent	11-23
Using Standard Sequences in Mappings Correctly	11-24
Using ODI Standard Sequences in Mappings	11-25
Populating Native Identity Attributes	11-26
Sequences: Best Practices	11-27
Automatic Temporary Index Management	11-28
Tracking Variables and Sequences	11-29
How Variable and Sequence Tracking Works	11-30
Variable Actions	11-31
Definition Tab of Session Step or Session Task	11-32
Quiz	11-33
Summary	11-34
Checklist of Practice Activities	11-35
Practice 11-1: Using Native Sequences with ODI Mapping	11-36
Practice 11-2: Using Temporary Indexes	11-37
Practice 11-3: Using Sets with ODI Mapping	11-38

12 Designing Mappings: Advanced Topics 2

Objectives	12-2
Agenda	12-3
Partitioning	12-4
Definition in Datastore After Reverse-Engineering	12-5
Using Partitioning in a Mapping	12-6
Agenda	12-7
Reusable Mappings	12-8
Using Reusable Mappings: Example	12-9
Derived Select (Subselect) for Reusable Mappings	12-10
Agenda	12-11
What Is a User Function?	12-12
Why Use User Functions?	12-13
Properties of User Functions	12-15
Using User Functions	12-16
Creating a User Function	12-17
Defining an Implementation	12-18
Syntax and Implementations	12-19
User Functions at Design Time	12-20
User Functions at Run Time	12-21
Note: Functions in Execution Log	12-22
Agenda	12-23
Using Substitution Methods	12-24

Substitution Methods: Examples	12-26
Agenda	12-27
Description of KM Steps	12-28
Details of the Steps	12-29
Setting KM Options	12-30
Developing Your Own KM: Guidelines	12-31
Complex File Technology	12-33
Quiz	12-34
Summary	12-35
Checklist of Practice Activities	12-36
Practice 12-1: Creating and Using Reusable Mappings	12-37
Practice 12-2: Developing a New Knowledge Module	12-38

13 Using ODI Procedures

Objectives	13-2
Agenda	13-3
What Is a Procedure?	13-4
Procedure: Examples	13-5
Creating Procedures: Overview	13-7
Agenda	13-8
Creating a New Procedure	13-9
Agenda	13-10
Creating a Command	13-11
Arranging Tasks in Order	13-13
Which Parameters Should Be Set?	13-14
Valid Types of Commands	13-15
More Elements	13-16
Why Use a Source Command?	13-17
Agenda	13-18
Types of Options	13-19
Creating a New Option	13-20
Making a Command Optional	13-21
Using an Option Value in a Command	13-22
Agenda	13-23
Procedure Execution	13-24
Using the Operator Navigator to View Results	13-25
Quiz	13-26
Summary	13-28
Checklist of Practice Activities	13-29
Practice 13-1: Creating an ODI Procedure	13-30

14 Using ODI Packages

- Objectives 14-2
- Agenda 14-3
- What Is a Package? 14-4
- Creating a Package 14-5
- Agenda 14-6
- Creating and Naming a Package 14-7
- Package Diagram 14-8
- Package Diagram Toolbar 14-9
- Agenda 14-11
- Package Steps 14-12
- Creating a Package Step 14-13
- What Is an ODI Tool? 14-14
- Creating an ODI Tool Step 14-15
- Tool Steps: Best Practices 14-16
- Agenda 14-17
- Sequencing Steps 14-18
- A Simple Package 14-19
- Sequencing Package Steps 14-20
- Agenda 14-21
- Executing a Package 14-22
- Agenda 14-23
- Basic Step Types 14-24
- Advanced Step Types 14-25
- Agenda 14-26
- Creating Model, Submodel, and Datastore Steps 14-27
- Models, Submodels, and Datastore Steps 14-28
- Agenda 14-30
- Creating a Variable Step 14-31
- Variable Steps 14-32
- Agenda 14-34
- Controlling Execution 14-35
- Error Handling 14-36
- Creating a Loop 14-37
- The Advanced Tab 14-38
- Quiz 14-39
- Summary 14-41
- Checklist of Practice Activities 14-42
- Practice 14-1: Creating an ODI Package 14-43
- Practice 14-2: Using ODI Packages with Variables and User Functions 14-44

15 Step-by-Step Debugger

Objectives 15-2
Agenda 15-3
Overview 15-4
Agenda 15-5
Process Overview 15-6
Starting a Session in Debug mode 15-7
Specifying Debug Properties 15-8
Control Execution Flow 15-9
Screen Step Numbering 15-10
Agenda 15-11
New Functionalities 15-12
Benefits for End Users 15-15
Agenda 15-16
Debug Toolbar 15-17
Toolbar: Current Cursor 15-18
Toolbar: Get Data 15-19
Toolbar: Step Into 15-20
Toolbar: Run to Task End 15-21
Toolbar: Run to Next Task 15-22
Toolbar: Run to Step End 15-23
Toolbar: Run to Next Step 15-24
Toolbar: Pause 15-25
Toolbar: Resume 15-26
Summary 15-27
Checklist of Practice Activities 15-28
Practice 15-1: Debugging Mappings 15-29

16 Managing ODI Scenarios

Objectives 16-2
Agenda 16-3
What Is a Scenario? 16-4
Properties of Scenarios 16-5
Agenda 16-6
Scenario-Related Tasks 16-7
Generating a Scenario 16-8
Regenerating a Scenario 16-9
Generation Versus Regeneration 16-10
Executing a Scenario from the GUI 16-11
Executing a Scenario from a Command Line 16-12
Executing a Scenario from a Package 16-13

Exporting a Scenario	16-14
Agenda	16-15
Preparing Scenarios for Deployment	16-16
Automating Scenario Management	16-17
Scheduling the ODI Scenario	16-18
Scheduling ODI Scenario with External Scheduler	16-21
Managing Schedules	16-22
Quiz	16-23
Summary	16-24
Checklist of Practice Activities	16-25
Practice 16-1: Creating and Scheduling Scenarios	16-26

17 Using Load Plans

Objectives	17-2
Should You Organize Executions with Load Plans?	17-3
What Are Load Plans?	17-4
Load Plan Editor	17-5
Load Plan Steps	17-6
Defining the Restart Behavior	17-7
Are Load Plans Substitutes for Packages or Scenarios?	17-9
Benefits of Using Load Plans	17-10
Handling Failed Load Plans	17-11
Quiz	17-12
Summary	17-13
Checklist of Practice Activities	17-14
Practice 17-1: Using Load Plans	17-15

18 Enforcing Data Quality with ODI

Objectives	18-2
Agenda	18-3
Why Data Quality?	18-4
When to Enforce Data Quality	18-5
Data Quality in Source Applications	18-6
Data Quality Control in the Integration Process	18-7
Data Quality in the Target Applications	18-8
Agenda	18-9
Data Quality Business Rules	18-10
From Business Rules to Constraints	18-11
Agenda	18-12
Data Quality System: Overview	18-13
Static and Flow Controls: Differences	18-14

Data Quality Control: Properties	18-15
Synchronous Control	18-16
What Is a Constraint?	18-17
What Can Be Checked?	18-18
Enforcing Data Quality in a Mapping	18-19
Agenda	18-20
Setting Up Static or Flow Control	18-21
Enabling Static or Flow Control	18-22
Agenda	18-23
Setting the Physical Options	18-24
Setting the Logical Options	18-25
Agenda	18-26
Selecting Which Constraints to Enforce	18-27
Selecting Which Constraints to Check	18-28
Differences Between Control Types	18-29
Agenda	18-30
Reviewing Erroneous Records	18-31
EnterpriseDataQuality Tool	18-32
Using the EDQ Tool	18-33
Quiz	18-34
Summary	18-36
Checklist of Practice Activities	18-37
Practice 18-1: Enforcing Data Quality with ODI Mappings	18-38

19 Working with Changed Data Capture

Objectives	19-2
Why Changed Data Capture?	19-3
CDC Techniques	19-4
Changed Data Capture in ODI	19-5
Journalizing Components	19-6
CDC Infrastructure in ODI	19-7
Simple Versus Consistent Set Journalizing	19-8
Limitations of Simple CDC Journalizing: Example	19-9
Consistent CDC Journalizing	19-10
Consistent CDC: Infrastructure	19-11
Setting Up Journalizing	19-12
Setting CDC Parameters: Example	19-13
Adding a Subscriber: Example	19-14
Starting Journal: Example	19-15
Journalizing Status	19-16
Viewing Data/Changed Data: Example	19-17

Using Changed Data	19-18
Oracle GoldenGate Integration	19-20
Oracle GoldenGate Integration in ODI 12c	19-21
Quiz	19-22
Summary	19-24
Checklist of Practice Activities	19-25
Practice 19-1: Implementing Changed Data Capture	19-26

20 Advanced ODI Administration

Objectives	20-2
Agenda	20-3
Introduction to ODI Security Navigator	20-4
Security Concepts: Overview	20-6
Defining Security Policies	20-8
Creating Profiles	20-9
Using Generic and Nongeneric Profiles	20-10
Built-in Profiles	20-11
Creating Users	20-12
Assigning a Profile to a User	20-13
Assigning an Authorization by Profile or User	20-14
Defining Password Policies	20-15
Setting User Preferences	20-17
ODI Security Integration: Overview	20-18
Implementing External Authentication (OPSS)	20-19
Implementing External Authentication (OPSS): Switching the Authentication Mode	20-21
Implementing External Password Storage	20-22
Agenda	20-24
Types of ODI Reports	20-25
Generating Topology Reports	20-26
Generated Topology Report: Example	20-27
Version Comparison Report: Example	20-28
Generating Object Reports	20-29
Agenda	20-30
Integration of ODI with Enterprise Manager	20-31
Java EE Agent and Enterprise Manager Configuration with WebLogic Domain: Overview	20-32
Using ODI Console: Example	20-33
Quiz	20-34
Summary	20-35
Checklist of Practice Activities	20-36

Practice 20-1: Setting Up ODI Security 20-37

Practice 20-2: Integration with Enterprise Manager and Using ODI Console 20-38

10

Mappings: Monitoring and Troubleshooting

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Describe what happens at run time
- Monitor the execution of mappings
- Troubleshoot runtime errors in mappings
- Prevent errors by following best practices when designing mappings



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Agenda

- **Monitoring Mappings**
 - Using the Operator Navigator
- Working with Errors

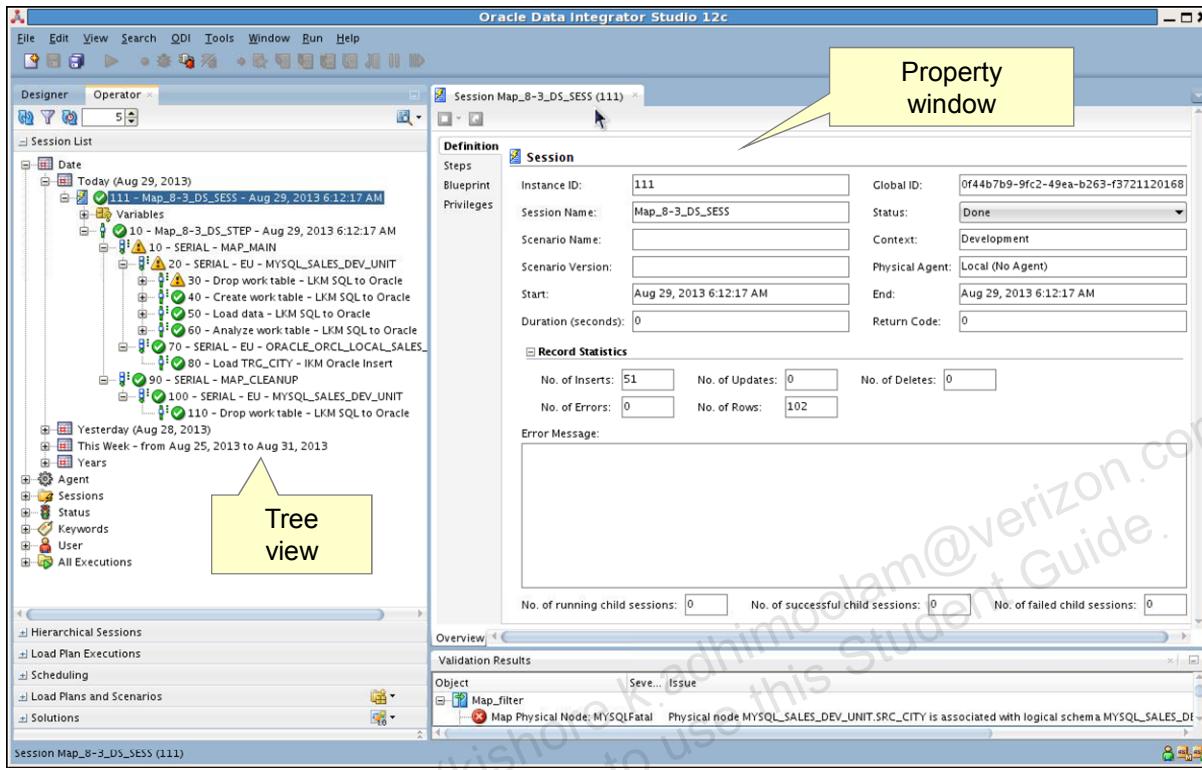


ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You now look at how to use the Operator Navigator to monitor the execution of a mapping. Through Operator Navigator, you can see the session that was created to run the mapping, and see how the mapping is broken down into smaller steps. The details of how ODI executes your mapping are now revealed.

Operator Navigator: Viewing the Log



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

ORACLE

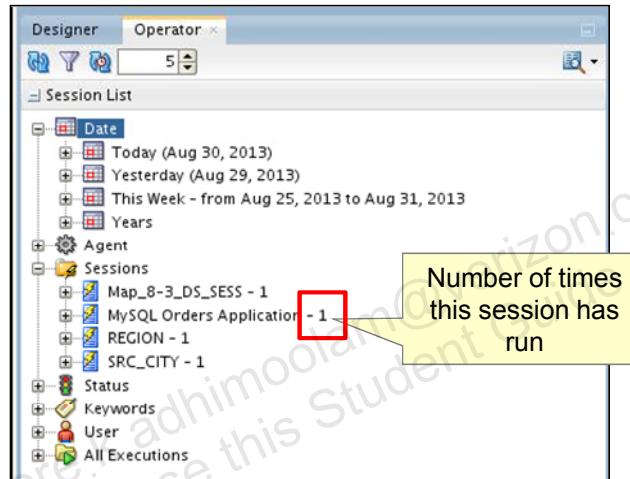
In the Operator Navigator, you may have different sessions available for examination. To help find the one you want, sessions are sorted in different ways in the tree view: by the date of execution, the physical agent used, the name of the session (sessions), the current state of execution, or by keywords. Alternatively, you can see all executions together. Different Operator views show the same information in slightly different formats.

On the toolbar, you can launch other ODI modules (Designer, Topology, and so on). You can manually refresh the session list or set it to automatic refresh. To remove all stored sessions, you can purge the log, and you can also see the schedule. Navigation buttons such as Back and Forward are helpful for quickly comparing two similar execution logs.

The property window shows you the properties of any object in the Session List. Double-clicking an item opens its property window.

Using Operator Navigator

- Operator Navigator shows the execution log.
- Each mapping launched is displayed as a session.
State shown:
 - Finished
 - Running
 - Waiting
- Operator Navigator enables you to:
 - Stop
 - Edit
 - Restart sessions.



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

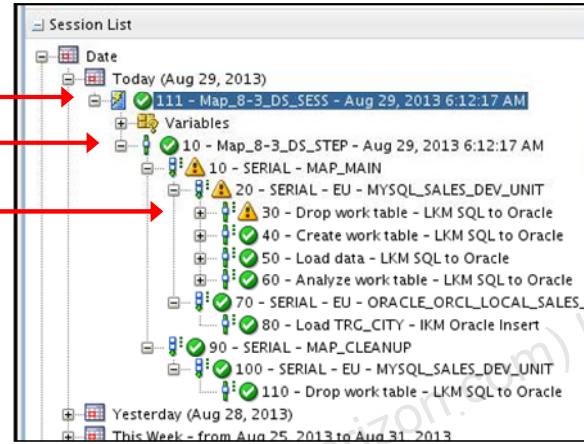
Operator Navigator serves primarily to show an execution log of the mappings that you launched. Other items, such as packages, are also visible, but those are not covered in this lesson.

Whenever you execute a mapping, you create a session. You must, therefore, find the session in Operator Navigator that corresponds to the time you launched your mapping.

In Operator Navigator, you can stop, edit, and restart sessions. This flexibility enables you to fix a problem in a session and continue without having to relaunch the mapping from Designer.

Hierarchy: Sessions, Steps, Tasks

- Sessions are made up of:
 - Steps: One mapping
 - Tasks: One command or statement
- A session, step, or task has one state at any given moment.
- Double-click a session, step, or task to display its details.
 - Done
 - Error
 - Running
 - Warning
 - Queued



ORACLE

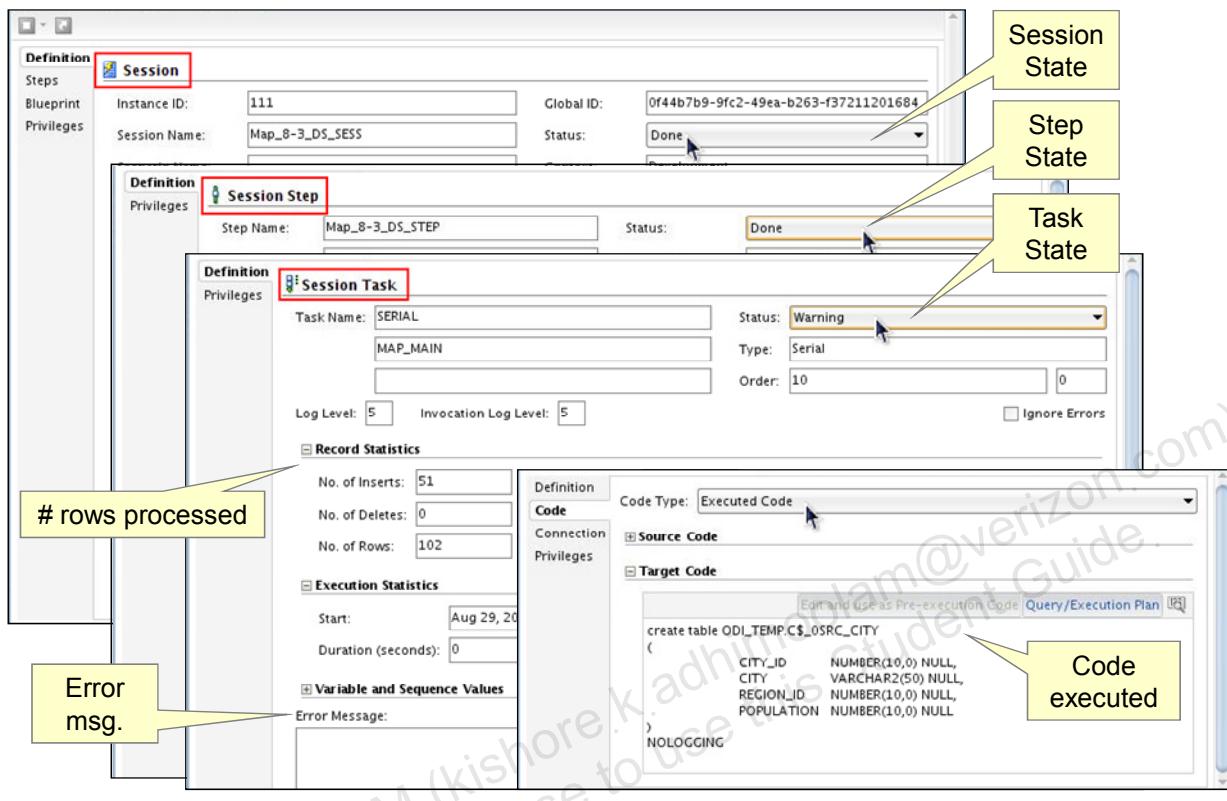
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Every time a mapping is executed, a session is created. However, a session can actually execute several mappings in sequence (known as a package). Thus in a session, a mapping corresponds to just one step. However, moving data around to execute a mapping is a complicated process. This process is made up of tasks. A task may consist of retrieving data from a table, writing to a file, creating a temporary table, or cleaning up. The series of tasks is determined entirely by the Knowledge Modules that you have chosen.

Each task has one state at any given moment: Running, Done, Error, or Warning. There can also be, at most, only one task in Running state in a given session at any given moment. The states of the tasks in a step then determine the state of the step: If all tasks are Done, the step is Done. A task can be completed with constraint errors. If a task has an error, that step is in the Error state, and so on. Similarly, the states of the steps determine the state of the session as a whole.

You can see the details of a session, step, or task by double-clicking it.

Viewing Details of Sessions, Steps, and Tasks



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Double-click a session step to see its details. You can see and change the session state on the Execution tab. You can put the session into the Waiting state to make it run when you restart the session. You can also see any relevant error messages and the number of rows processed so far.

When you double-click a session task, you see a similar window. On the Description tab, you can see the actual code executed. This is very useful for diagnosing advanced problems.

Note: To prevent the repository from getting too large, you should regularly purge the log. You can do this by clicking the Purge Log button in Operator Navigator. You can also purge the log automatically by using the `odiPurgeLog` tool.

Monitoring Execution of an Mapping

1. In Operator Navigator, locate the mapping session (by name or date).
2. Check the session state.
3. If there is no error, check the number of processed rows in the details.
4. If there is an error, start troubleshooting.

ORACLE

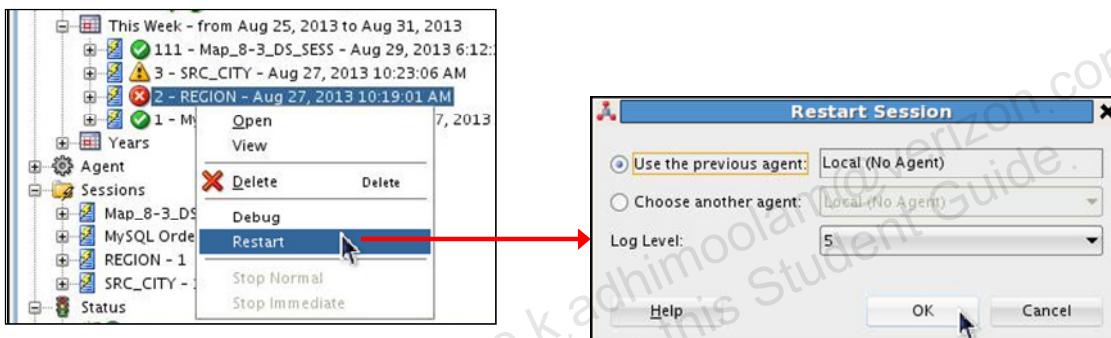
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You need to check the progress of a mapping frequently. To do so, perform the following:

1. Find the session in which the mapping was launched. As you know, you can sort the sessions in many ways to achieve this. If there are not many sessions, it is often easiest to use the All Sessions tree node. Then, the most recent session is at the top.
2. Then, double-click the session to view its properties.
3. If the state is not Error, look at the number of rows that have been processed. Consider how many valid and erroneous rows you expected.
4. If the state is Error, or the number of rows processed is not what you expected, start troubleshooting. This will be covered in the next few slides.

Troubleshooting a Session

1. Find the error.
2. Review the generated code.
3. Fix the generated code, then restart the session.
 - Repeat as needed until there is no error.
4. Fix the mapping.



ORACLE

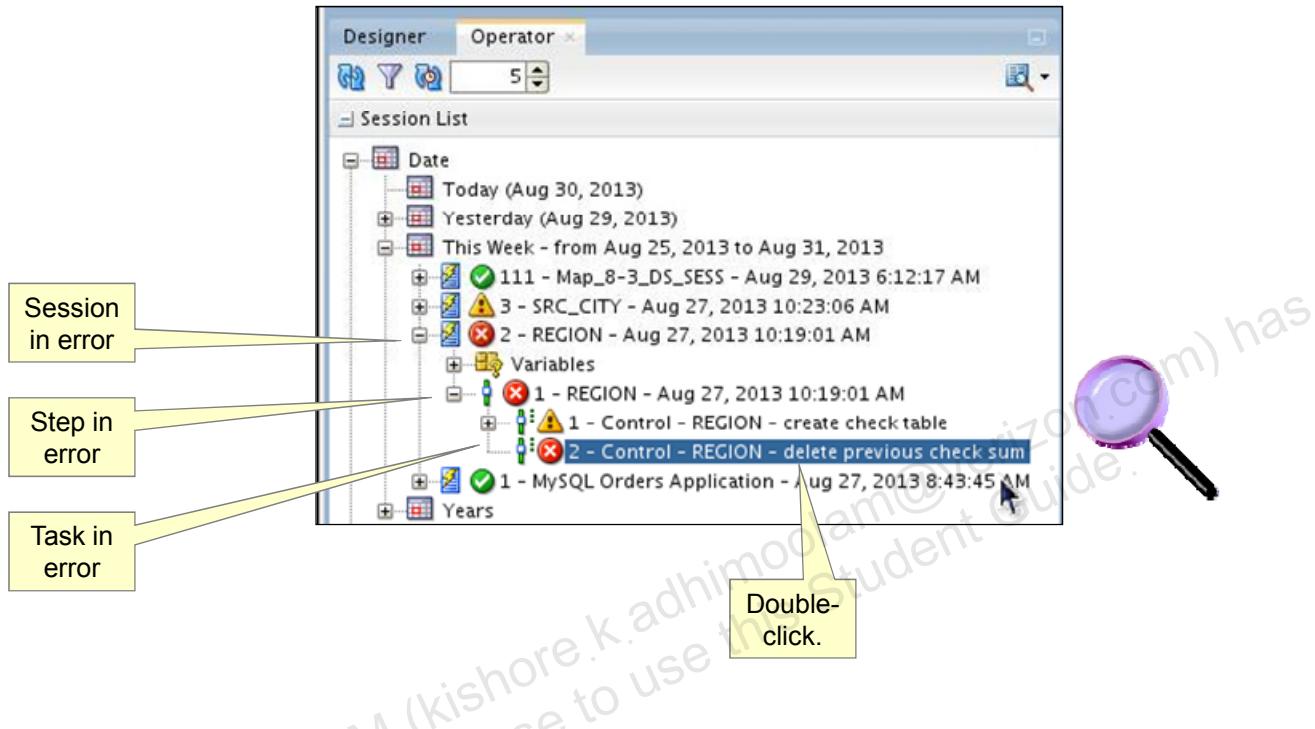
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The general process is as follows:

1. Finding the error is usually straightforward. You open Operator Navigator and find the session in error, the step in Error state, and finally the task in error.
2. Then you look at the code generated by ODI. To do this, double-click the task and click the Description tab. The code is usually in SQL and is dependent on the Knowledge Modules used by the mapping.
3. Fixing the generated code often takes the most time. You must have a solid understanding of how ODI works and how the relevant technologies work. You can directly edit the code in the code box to make the changes. Click the Apply or OK button when finished. Then, right-click the session and select Restart. Refresh the Session List to check whether your error has been resolved. If it has not been refreshed, you can keep making further changes and restarting the session.
4. When you have fixed the error in the generated code, you should update the mapping to take the change into account. Otherwise, the next time you execute the mapping, the old code with the error is generated again.

The next four slides examine this four-step process in detail.

1. Identifying the Error



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

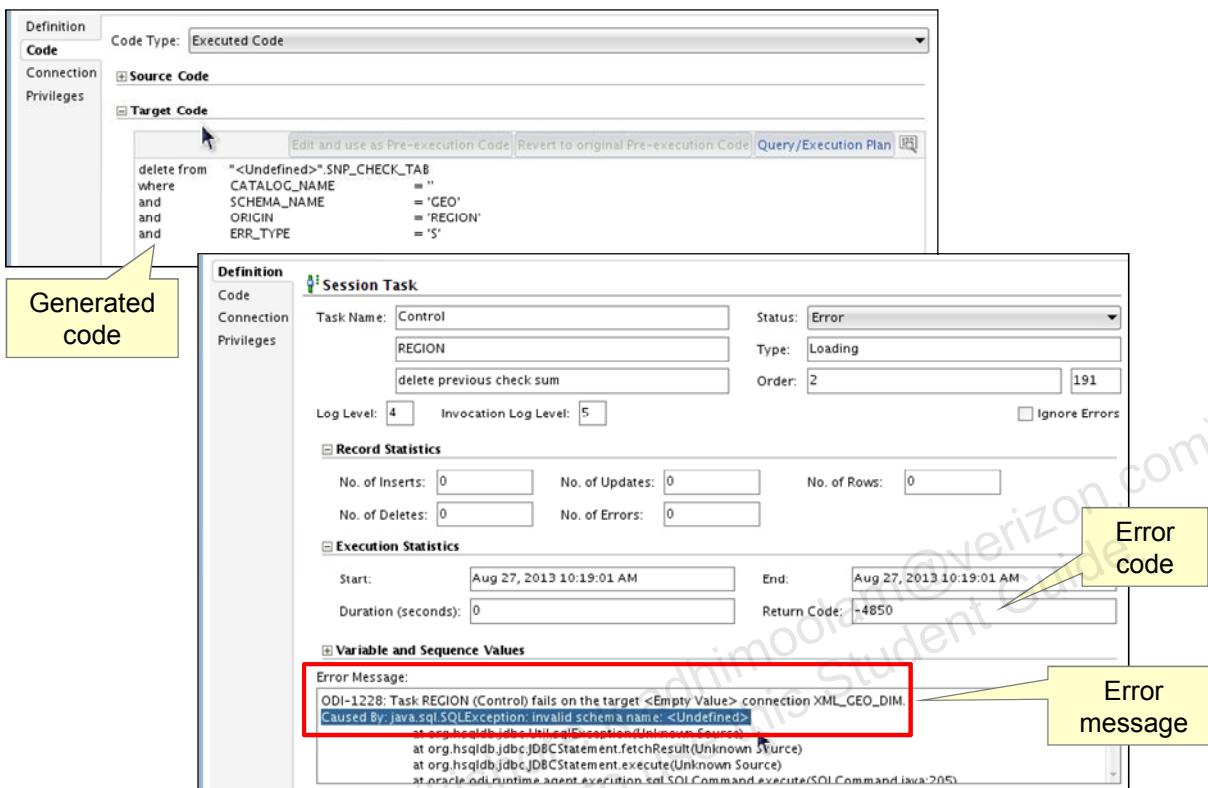
ORACLE

To find the error, you have to locate the task that has the error icon next to it. This task is found in a step with an error icon. This step is found in the session that is in error.

Double-click the task to see the generated code.

Alternatively, or in parallel, you can use the Debugger, covered in its own lesson titled, “Step-By-Step Debugger.”

2. Reviewing the Code



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In the Task window, click the Code tab to see the generated code.

To see the results of the error, click the Definition tab. Double-check Default Connection. In most simple cases, it is the target server. Similarly, double-check Loading Connection to ensure that it corresponds to the correct source server. Return Code shows the error code generated, usually by the DBMS server. Finally, the error message itself is displayed. The code and message are also visible in the step and session windows. You see what the terms "Loading" and "Default" refer to shortly.

3. Fixing the Code and Restarting the Session

1. Edit the code in the task.
 - You can copy and paste the code into an external SQL analysis tool, if required.
2. Select the session node.
3. Right-click and select Restart.
 - The session restarts.



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

To fix the code and restart the session, perform the following:

1. Make a change to the code in the task window. You may want to copy and paste the code into an external tool, such as SQL Plus or SQL Analyzer.
2. Select the node corresponding to the session that this task belongs to. You can also click the parent-level button on the toolbar twice to do this.
3. Right-click the session and select Restart. The session is now relaunched in the background. The task that had the error is the first task to be executed.

Refresh the window to see the results.

Note: Data already loaded in staging tables can be used as part of the debugging process.

4. Fixing the Mapping

- Manually fixing the code is not always the solution.
- You may need to go back to the mapping.
 - Fixing code in the Operator has **no** impact on the mapping.
 - Corrections **have** to be made to the mapping in Designer.



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Manually fixing the code can be useful for quickly getting your mapping to work. However, if you want to use this mapping again later, you must update it to fix the original problem. If you do not update the mapping, when you execute your mapping again, the same error is generated as before.

Also, some errors cannot be fixed by editing the generated code. For example, you may have the wrong choice of Knowledge Modules, execution location, or target or source datastores. In these cases, you must fix the problem at the mapping level.

Keys to Reviewing the Generated Code

- Code structure:
 - The code always has at least one command on the output (default) connection and sometimes on a loading connection, such as `SELECT... INSERT`.
- Executions appear in the column lists for `SELECT` or `INSERT` statements.
- Joins and filters appear in the `WHERE` clauses.
- SQL will be generated in the `SELECT` clause or `WHERE` clause depending on the lookup type.
- The rest of the code is generated from the selected Knowledge Modules and depends entirely on the mapping design.

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The generated code effectively has two halves. The part that is usually executed on the target is referred to as the default connection. The code to create worktables and populate the target tables appears here. Every command has code in this part. Some commands also take data from a loading connection. The code to `SELECT` data from source tables appears here.

This means that the target part of a mapping always appears in an `INSERT` statement. The source part appears in a `SELECT` statement.

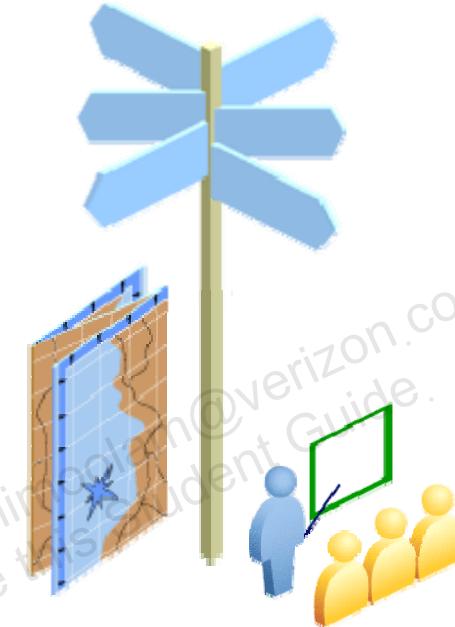
Join conditions, such as `SRC_ORDERS.CUST_ID=SRC_CUSTOMER.CUSTID`, appear in the `WHERE` clauses. Similarly, filters such as `CUSTOMER.AGE > 20` appear in the `WHERE` clauses.

The rest of the code, including immediate steps such as creating worktables, is generated by the Knowledge Modules used in the mapping. The design of your mapping determines how this happens.

The information on the Connection tab is useful in showing the source/target connections being used for that task.

Agenda

- Monitoring Mappings
- **Working with Errors**



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Troubleshooting a Session

If a problem occurs with a session, you want to troubleshoot it. Troubleshooting a session usually means finding the problem and modifying the generated code. You then check that your change has fixed the problem, and then update the original mapping. Though this is not the only method for troubleshooting, it is a good way of learning more about ODI. It is also more efficient than constantly re-executing the mapping.

Common Errors and Symptoms

- Syntax error in mapping, join, or filter:
 - Obvious error appears in a SELECT / INSERT or WHERE clause
 - “Column does not exist”
 - “Syntax error”
- Incorrect execution location for mapping, join, or filter:
 - A valid clause cannot be executed.
 - A clause is located at the wrong place (source/target), or needs to be performed in another command to work properly.
- Incorrect execution context or incorrect agent chosen:
 - Data servers unreachable (table/files/resources do not exist)
 - “Table not found”
- Incorrect KM chosen:
 - The statements are entirely irrelevant for the situation or have an inappropriate syntax for the technology.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

A variety of errors can occur, depending on the technologies, the Knowledge Modules, and even the operating systems involved. However, knowing the symptoms of some common errors helps you diagnose them. Example:

- A syntax error in a mapping, join, or filter means that the generated SQL code was not valid. You may see “Column does not exist,” “Syntax error,” or something similar as the error. The possible reasons are mistyped column names or invalid SQL entered as the expression. Use the Check Expression button wherever possible to minimize these errors. (Datastores refer to RDBMS table columns as attributes.)
- An incorrect execution location means making the wrong choice of source, target, or Staging area in the mapping window. The result may be a valid SQL clause that cannot be executed because it is on the wrong server. The result can be subtler: poor performance if code intended for a powerful server is executed on a desktop machine.
- Selecting an incorrect execution context means, for example, executing a mapping in the “production” environment instead of the “development” environment. The symptoms include errors about not being able to reach certain data servers, or tables, files, or resources on those servers.

- You may also have chosen an incorrect KM in the mapping window. This can cause statements irrelevant to your situation to be generated. Also, syntax inappropriate for the technology may be generated. For example, if you selected a Java Message Service (JMS) KM to populate a datastore on an Oracle server, you end up with bad syntax. ODI cannot always prevent this type of mistake. For best results, start with the “Generic SQL” KMs in any new environment because they are designed to work on any server. After your mapping works, you may want to use a more specialized KM for performance.

Important Note



Java exceptions are not always ODI errors.

Example:

```
-28 : S0022 : java.sql.SQLException:  
Column not found: C1 REGION ID in statement  
[select SRC_REGION.REGION_ID + C1_REGION_ID,  
SRC_REGION.COUNTRY_ID C2_COUNTRY_ID,  
SRC_REGION.REGION C3_REGION  
from SRC_REGION SRC_REGION where (1=1)]
```



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

When you see a “Java exception,” it is usually not an error in ODI. Read the text of the error message first to see whether a problem in your mapping caused the error. In many cases, the error actually comes from a database server. It may be as simple as a violation of an integrity constraint, caused by duplicate rows.

Tips for Preventing Errors

- Use the Expression Editor.
- Review your mappings before execution.
- Use the Error button on the mappings.
- Check the syntax of mappings:
 - Filters
 - Joins
 - Executions
- Do not change the execution location of mappings, filters, joins, or the selected KMs, unless you know the impact of the change.
- Use Static Control and Flow Control for data quality.

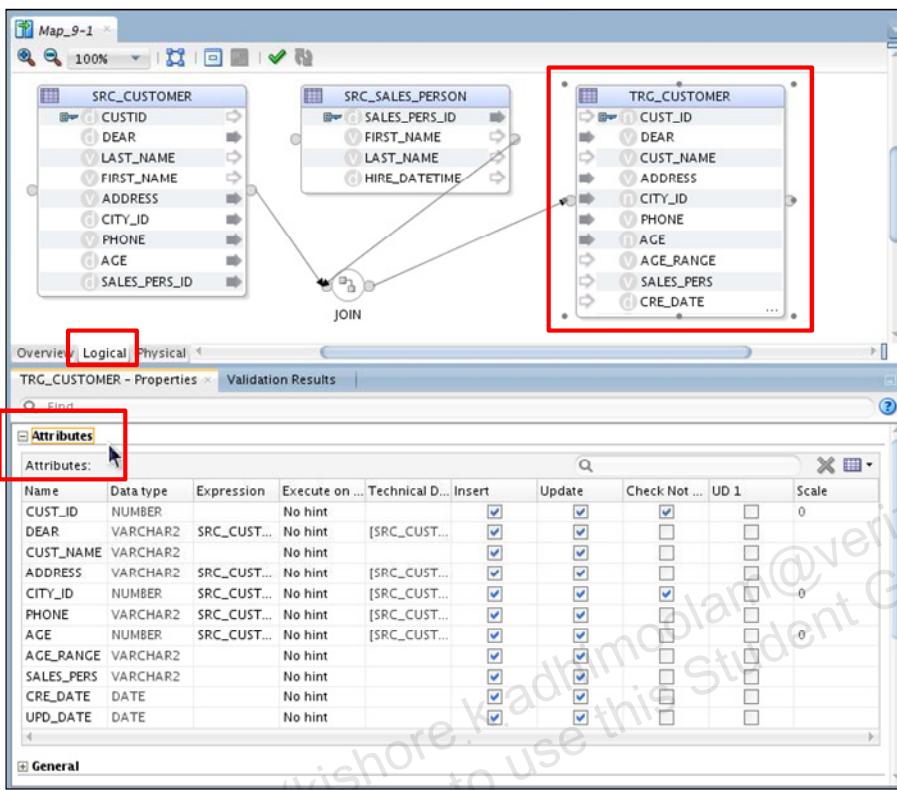
ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Some tips for reducing errors are as follows:

- For best results, use the Expression Editor to write expressions for joins, filters, and mappings. This ensures that the column names and database functions that you use exist in the technology.
- Before you execute a mapping, double-check your joins and executions. Ensure that your Knowledge Modules are correct. It is faster to detect the error *before* executing than checking for the result in Operator.
- Click the Error button at the top of the mapping window frequently. This informs you about any obvious errors in your diagram.
- When the configuration of your mapping enables it, use the Check Expression button to check that your SQL is valid.
- Changing the execution location of mappings, filters, or joins can have a big impact on the generated code. Be very cautious until you understand the consequences. The same applies to changing the Knowledge Modules used.
- The best practice is to use ODI's built-in data quality tools. For example, apply a static control to source data and flow control for the reverse.

Using Attribute Panel for Quick Edits



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You can use the Properties Attributes panel to perform the same actions as on the Logical tab of the Mapping Editor, in a nongraphical form:

- Adding and removing a component
- Editing a component
- Adding, removing, and configuring datasets
- Changing the target datastore

The properties of the following components are displayed in tabular form and can be edited in the Properties Attributes panel:

- Sources
- Components (Join, Lookup, Filter, Expression, Set, Split, Aggregate [and the new Pivot/Unpivot])
- Mappings

Note that components already defined on the Logical tab of the Mapping Editor are displayed in the Properties Attributes panel and that the components defined in the Properties Attributes panel are also reflected on the Logical tab. An example is to change a source Expression to be `SALES_PERS_ID > 300` to act like a filter.

Quiz

You can always fix an error by correcting the code manually.

- a. True
- b. False

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Answer: b

Explanation: Some errors cannot be fixed by editing the generated code. For example, you may have the wrong choice of Knowledge Modules, execution location, or target or source datastores. In these cases, you must fix the problem at the mapping level.

Quiz

Which is a symptom of a wrong choice of execution location?

- a. Errors about not being able to reach certain data servers, or tables, files, or resources on those servers
- b. “Column does not exist” or “Syntax error” errors
- c. A valid SQL clause that cannot be executed because it is on the wrong server, or poor performance if the code intended for a powerful server is executed on a desktop machine



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Answer: c

Explanation: An incorrect execution location means making the wrong choice of source, target, or Staging area in the mapping window. The result may be a valid SQL clause that cannot be executed because it is on the wrong server. The result can be subtle: poor performance if the code intended for a powerful server is executed on a desktop machine.

Summary

In this lesson, you should have learned how to:

- Describe what happens at run time
- Monitor the execution of mappings
- Troubleshoot runtime errors in mappings
- Prevent errors by following best practices when designing mappings



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Checklist of Practice Activities

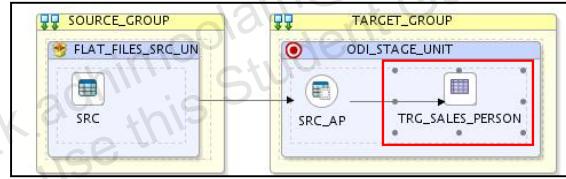
- ✓ 1-1: Logging In and Help Overview
- ✓ 2-1: Creating and Connecting to ODI Master and Work Repositories
- ✓ 3-1: Configuring Standalone Agents Using the Common Admin Model
- ✓ 4-1: Working with Topology
- ✓ 5-1: Setting Up a New ODI Project
- ✓ 6-1: Creating Models by Reverse-Engineering
- ✓ 7-1: Checking Data Quality in the Model
- ✓ 8-1: Creating ODI Mapping: Simple Transformations
- ✓ 9-1: Creating ODI Mapping: Complex Transformations
- ✓ 9-2: Creating ODI Mapping: Implementing Lookup
- 10-1: **Creating ODI Mapping: Exporting a Flat File to a Relational Table**
- 11-1: Using Native Sequences with ODI Mapping
- 11-2: Using Temporary Indexes
- 11-3: Using Sets with ODI Mapping
- 12-1: Creating and Using Reusable Mappings
- 12-2: Developing a New Knowledge Module
- 13-1: Creating an ODI Procedure
- 14-1: Creating an ODI Package
- 14-2: Using ODI Packages with Variables and User Functions
- 15-1: Debugging Mappings
- 16-1: Creating and Scheduling Scenarios
- 17-1: Using Load Plans
- 18-1: Enforcing Data Quality with ODI Mappings
- 19-1: Implementing Changed Data Capture
- 20-1: Setting Up ODI Security
- 20-2: Integration with Enterprise Manager and Using ODI Console

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Practice 10-1: Creating ODI Mapping: Exporting a Flat File to a Relational Table

1. In Topology Navigator, define FILE_GENERIC./home/oracle/labs/files/my_flat_files physical.
2. In Topology Navigator, define FLAT_FILES_SRC logical schema.
3. In Designer Navigator, create the Export-FF-RT project.
4. In Designer Navigator, create the Flat_File_1 source model.
 - a. Create SRC_SALES_PERSON datastore.
 - b. Point to resource: ~/labs/files/my_flat_files/SRC_SALES_PERSON.TXT.
 - c. Reverse-engineer and format the data (fixed length positions, data types).
5. In SQL Developer, create the RDBMS schema ODI_STAGE to host the ODI target datastore.
6. In SQL Developer, create the TRG_SALES_PERSON table to serve as ODI datastore for the target model.
7. In Topology Navigator, create the ODI target data server ODI_STAGE, physical schema ODI_STAGE, and logical schema ODI_STAGE.
8. In Designer Navigator, create the ODI target model Oracle_RDBMS1.
9. Reverse-engineer the model and check the populated TRG_SALES_PERSON datastore table.
10. Create a new ODI mapping to perform flat file to RDBMS table transformation.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Exporting a Flat File to a Relational Table

In this practice, you first use Topology Navigator to define physical and logical schemas for a flat file source.

You then use Designer Navigator to create a project within which you define a flat file source model. In the model, you create the SRC_SALES_PERSON datastore that points to a SRC_SALES_PERSON.TXT text file. You perform reverse-engineering to derive formatting information (fixed length positions and data types) from the text file.

Next you use SQL Developer to create the ODI_STAGE schema and TRG_SALES_PERSON table *in the RDBMS*. (This schema and table will host the ODI target datastore in a target model that you will define next in ODI.)

You then return to Topology Navigator to define the ODI_STAGE data server, physical schema, and logical schema for the relational target.

You then return to Designer Navigator to create the target model Oracle_RDMBS1. You reverse-engineer the model and check the populated TRG_SALES_PERSON table that you created earlier in the RDBMS.

Finally, you create a new ODI mapping to perform the source flat file to target RDBMS table data population.

Unauthorized reproduction or distribution prohibited. Copyright© 2015, Oracle and/or its affiliates.

KISHORE ADHIMOOLAM (kishore.k.adhimoolam@verizon.com) has
a non-transferable license to use this Student Guide.

11

Designing Mappings: Advanced Topics 1



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Use business rules, variables, and set-based operators with ODI mappings
- Use datasets and sequences, including native sequences, with ODI mappings
- Create temporary indexes to speed up retrieval



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Agenda

- **Working with Business Rules**
- Using Variables
- Datasets and Sets
- Using Sequences



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Overview

In this section, you get a quick overview of the various types of business rules that can be implemented in Oracle Data Integrator (ODI). These rules will be covered in greater detail subsequently.

Business Rules in Mappings

- The following business rules are defined in mappings:
 - Expressions
 - Filters
 - Joins
- ODI provides the following additional objects:
 - Variables and sequences
 - User-defined functions
 - Substitution methods



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Three main types of business rules can be defined in mappings:

- Expressions define the way in which one or more source attributes (usually table columns) are combined and transformed to produce a value for a target attribute.
- A filter reduces the amount of data incorporated into the mapping by specifying a rule that must be satisfied by each line.
- Joins link several source datastores by specifying how one datastore relates to one or more datastores.

Each of these business rules is expressed as SQL code that is dependent on the given technology. This SQL code can then use other types of ODI objects.

Variables and sequences are two ways of storing individual values that can be used by your mapping. The difference is that sequences must be numeric and are automatically incremented each time they are accessed.

User-defined functions enable you to encapsulate commonly used functionality in a macro.

Substitution methods are defined in the Sunopsis API. They provide the functionality, such as accessing names of tables or columns, session information, or even generating complete SQL statements.

Business Rule Elements

The following types of clauses are available to implement business rules:

Value	String values should be enclosed with quotation marks: 'USA', '1 Jan 2000'. Numbers do not need quotation marks: 10
Source Attribute	Drag an attribute or use the Expression Editor. It is prefixed with the datastore alias.
DBMS Function	Use the Expression Editor for the list of allowed functions and operators.
DBMS Aggregate	MAX(), MIN(), SUM(), AVG(), and so on. ODI automatically generates the GROUP BY clause.
Combination	Any combination of clauses is allowed: <code>SRC_SALES_PERSON.FIRST_NAME ' ' UPPER(SRC_SALES_PERSON.LAST_NAME)</code>



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Expressions may include any type of clause allowed in SQL languages:

- Constant string values should be enclosed within straight single quotation marks. This is generally not necessary for numbers.
- Source attributes should be prefixed with the datastore alias. For best results, use the Expression Editor to drag source attributes.
- All database functions provided by the DBMS running the mapping's generated code can be used in the mapping. This includes aggregate functions, such as MINimum, MAXimum, or Average. For these functions, ODI automatically manages the generation of GROUP BY clauses.
- Any combination of constants, attributes (columns), DBMS, or aggregate functions is allowed in mapping.

More Elements

Other ODI-specific elements include:

Variables	They may be specified either in substitution mode #<variable> or in binding mode :<variable>.
Sequences	They may be specified in either substitution mode #<sequence> or in binding mode :<sequence>.
User functions	They are used in the same way as DBMS functions, but replaced at code generation by their implementation.

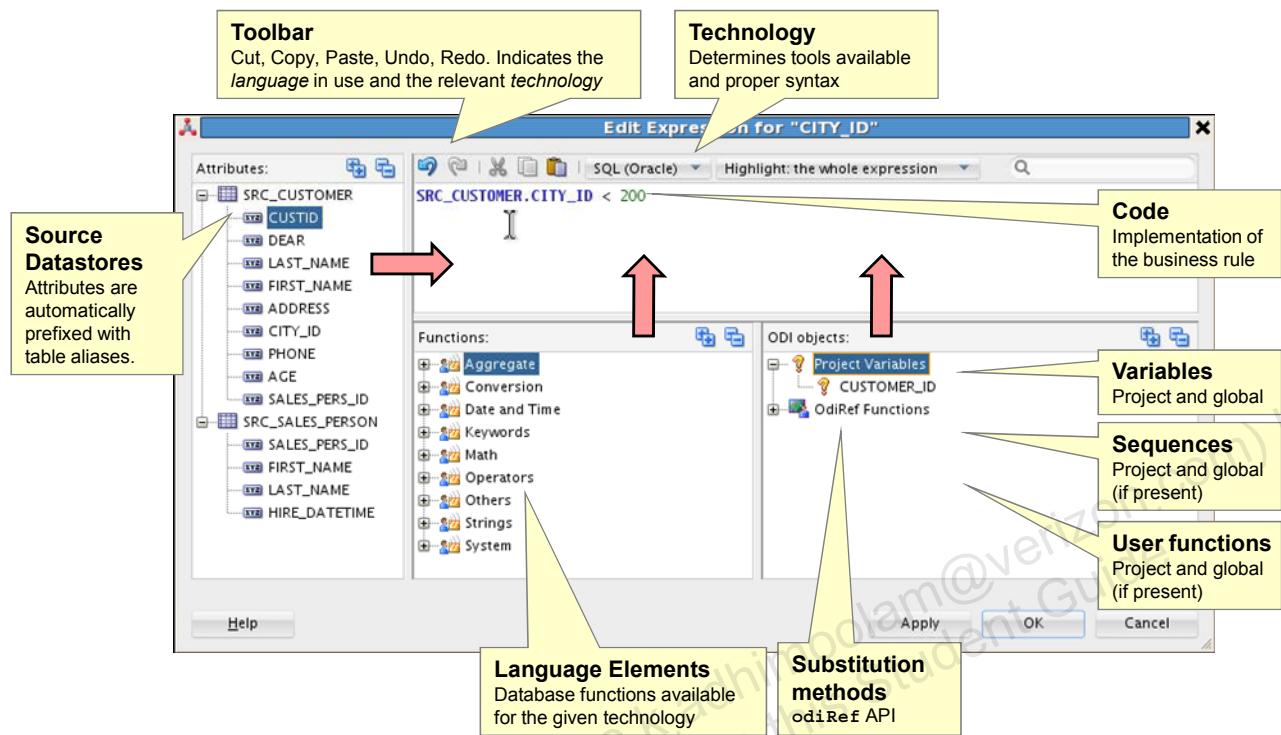


Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The elements are common to almost all database engines in one form or the other. However, a number of elements are specific to ODI. These are the focus of this lesson.

- You can refer to the current value of a variable by inserting its name preceded by the number sign. You will also see the use of binding mode.
- Sequences are used in a similar way, with the name preceded by the number sign or colon.
- The only difference between user functions and DBMS functions is that user functions are substituted by native database code at run time.

Expression Editor



Drag items into the Code box or double-click them.

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The area at the top right is where you can edit the code for your expression. Your code is colored according to the syntax of the technology. Recognized database functions, ODI objects, models, or attributes are highlighted specially. Thus, if you make a mistake, you know it immediately because of the absence of highlighting.

Above the code box is the toolbar. This contains the normal cut, copy, paste, undo, and redo tools. However, the toolbar also displays the relevant language and technology. This is very important because many of the elements shown depend on this technology. You must also ensure that the code you write is valid for the syntax of the language it is in.

The pane at the left contains all source datastores and attributes that are used in a mapping. When you drag an attribute name into the code box, it is automatically prefixed with the alias of the datastore.

The language elements pane at the bottom contains a list of all the database functions that are known to be supported for this technology. When you drag these functions into the code box, the correct number of arguments is automatically written.

At the bottom right, all the available ODI objects are displayed, including variables, sequences, and user functions. These objects are all grouped separately according to scope. The substitution methods available through the `odiRef` API are similarly visible.

Remember that you can drag all these items into the code box to quickly build an expression. You can also double-click items to add them to the current location in the edit box.

Note: Use the Expression Editor wherever possible to avoid making common mistakes in your code. It is particularly useful in complex mappings. You may forget to include the alias of a datastore, attempt to use a database function that does not exist on the chosen technology, or attempt to use a user function that has not been defined on this technology. The Expression Editor helps avoid these kinds of mistakes.

Agenda

- Working with Business Rules
- **Using Variables**
 - Binding
 - Substitution
- Datasets and Sets
- Using Sequences



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

A variable holds a specific value that can be inserted into your code at run time.

Note: This lesson does not have a hands-on practice using variables. A hands-on practice using variables will be performed in Lesson 14, in which you will use variables in packages.

Using a Variable in Code

- A variable is prefixed according to its scope:
 - Global variable: GLOBAL.<variable_name>
 - Project variable: <project_code>.<variable_name>
- Tip: Use the Expression Editor to avoid mistakes in the names of variables.
- Variables are used either by string substitution or by parameter binding.
 - Substitution: #<project_code or GLOBAL>.<variable_name>
 - Binding: :<project_code or GLOBAL>.<variable_name>



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Accessing the value of a variable in SQL expressions can be useful. To do this, you must prefix the name of the variable.

- For global variables, use the word GLOBAL, followed by a period, followed by the name of the variable.
- For project variables, use the project code, followed by a period, followed by the name of the variable. You see the project code for a given project by double-clicking the project in the Projects window.

As a best practice, use the Expression Editor by clicking the icon next to every expression box. The Expression Editor automatically adds the correct prefix for a given variable. Select the name of the variable from the Global Variables or Projects Variables trees in the Expression Editor.

There are two different ways in which the value of the variable can be incorporated into the code that is executed by the server.

- String substitution is the simplest, and usually, the preferable way. It works for all technologies. ODI replaces the text that refers to the variable with the actual value and sends it to the server.
- However, there is another method, called parameter binding. This method works only for SQL commands sent to databases. In this method, ODI sends only the statement, and then the parameters.

Wherever the name of the variable is used, you must prefix it with a symbol that specifies the method of use. For string substitution, prefix a number sign to the project code or the word GLOBAL. To use parameter binding, prefix a colon instead.

Binding Versus Substitution

- Substitution:
 - Is correct in most cases
 - The variable is replaced by its value before execution.
 - Use single straight quotation marks for nonnumeric values.
- Binding:
 - Works only for SQL clauses
 - The statement is prepared in the DBMS without the value, then the value is sent for processing.
 - It may be used for DBMSs that optimize repeated statements with different parameter values.

```
SELECT * FROM MYTABLE WHERE NAME LIKE  
'#GLOBAL.PATTERN'
```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The string substitution method works in every situation and is very often the correct choice. Because the variable name is replaced by its value, you must generally enclose nonnumeric values within single quotation marks. However, do not use quotation marks with numeric values. In general, you should always think about what makes sense for the language of the technology. For example, if your variable contains a single quotation mark and you enclose the variable name within single quotation marks, you probably break the syntax rules for the language.

The parameter binding method, however, is sometimes useful. It can be used only for clauses written in SQL. That is, it cannot be used in files, Lightweight Directory Access Protocol (LDAP) directories, and so on. However, it can avoid certain problems encountered in SQL when values contain single quotation marks, which can conflict with the syntax of strings. This method forces the database server to prepare an execution plan without the actual value. This is generally less efficient than sending the complete statement.

However, if the same statement is to be sent many times, the DBMS may be able to optimize the performance of the statement. In this case, it may be more efficient than text substitution. The best practice is to avoid using parameter binding until you are familiar with the two methods.

Case Sensitivity

- Variable names are case-sensitive.
- For example: *YEAR*, *Year*, and *year* are three different variables.

A graphic showing three pairs of letters in a stylized, 3D blue font. The first pair is 'Aa', the second is 'Bb', and the third is 'Cc'. This visual emphasizes the concept of case sensitivity in variable names.

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Note that the names of variables are case-sensitive. That is, uppercase and lowercase letters are distinguished. You may want to establish a convention, such as always using uppercase letters when declaring and using variable names.

Agenda

- Working with Business Rules
- Using Variables
- **Datasets and Sets**
 - **Datasets**
 - **Sets**
- Using Sequences

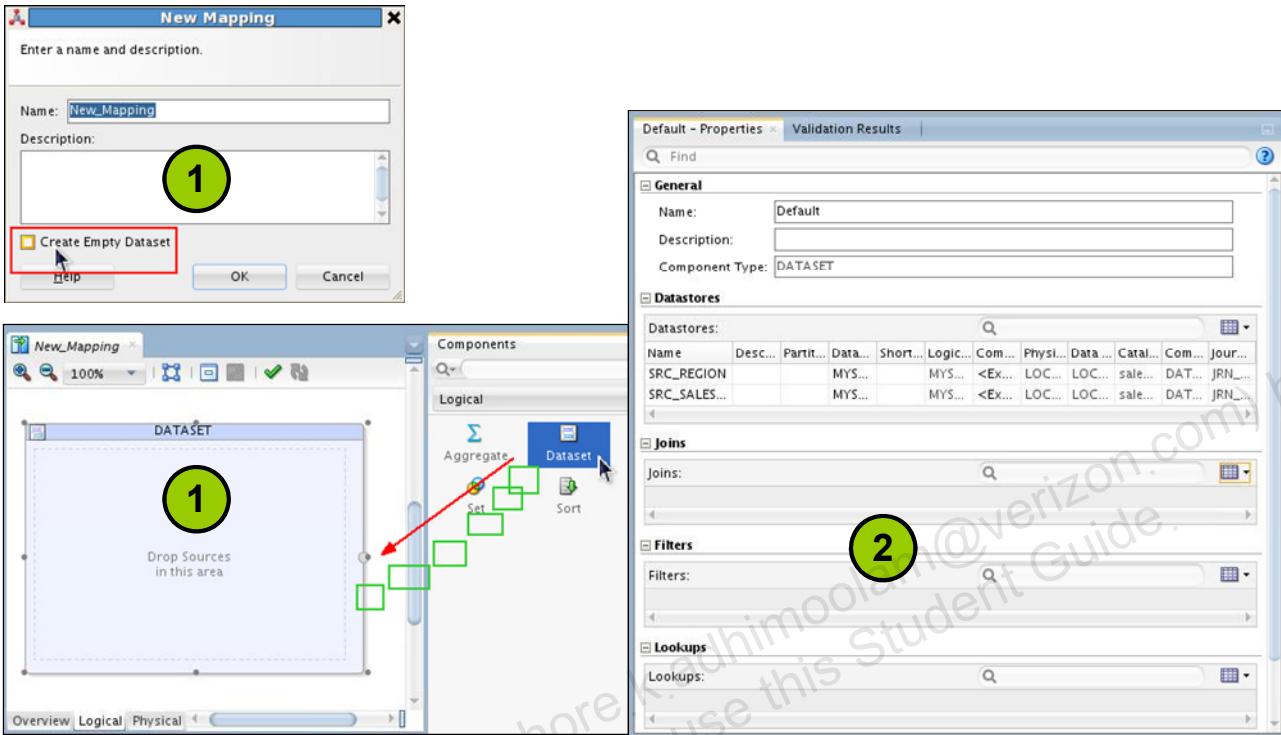


ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Do not confuse these two terms: sets and datasets. They are unrelated, though they may be used together in the same mapping.

Defining a Dataset



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Datasets exist primarily as a carryover from 11g. If you are used to using them from 11g, you can still use them. If you are new to 12c, there is nothing that requires a dataset, and they may just complicate things, and therefore can be avoided.

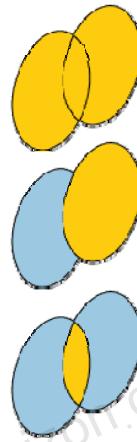
To create a new dataset:

1. When you create a new mapping, you have the option to also create an empty dataset. This was the default in ODI 11g.
Or you can drag a dataset component into the work area from the Components Palette.
2. This container is designed to hold two or more datastores (you get a warning if you try to save and/or validate and there is only one datastore in the dataset). You can select attributes/columns from the datastores.

After the datastores are in the dataset, you can connect them via a set component (union, intersect, minus), or a join (inner, outer, Cartesian, and so on), or a filter.

Using Set-Based Operators

- Make sure that your technology supports set-based operators.
 - Check **Support Set Operators**.
 - **List of Operators:**
INTERSECT, UNION, UNION ALL, MINUS
- In your mapping, add several **datasets**:
 - Save before adding datasets.
 - Share the same target structure.
 - Order counts for precedence.
- Select an IKM that supports datasets.
 - Datasets are merged from the Staging area.

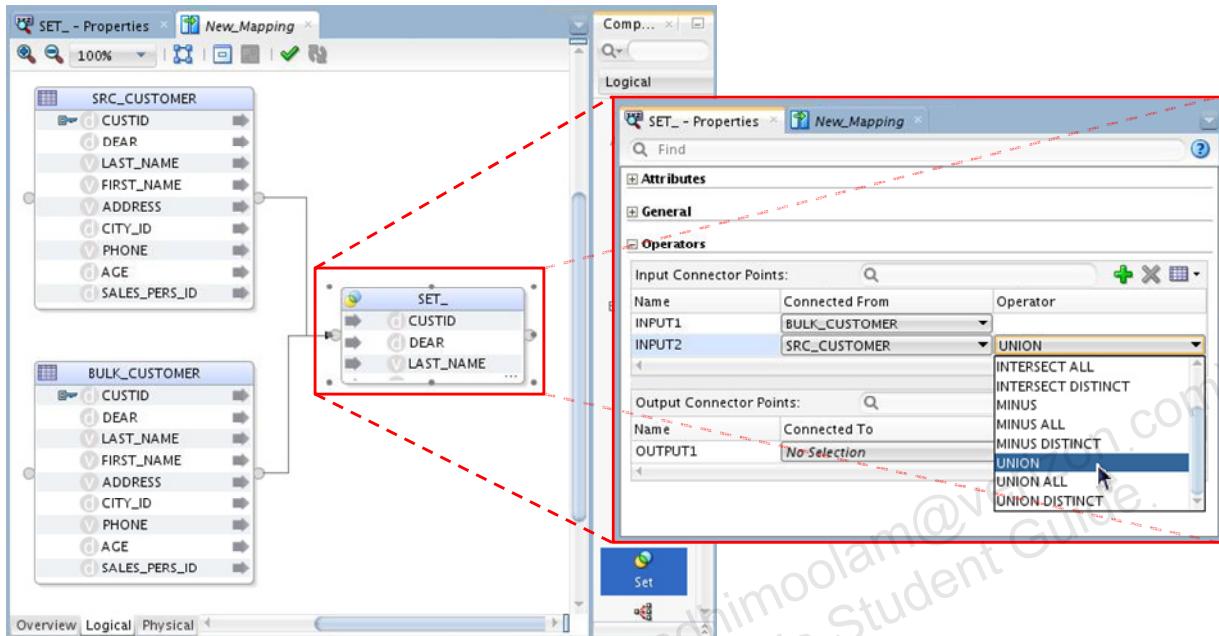


ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

This slide offers guidelines for using set-based operators successfully. Combining sets and datasets is optional—each can stand alone on their own.

Example of SET: UNION



(Not shown, but required, is an output target)

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

An example of a use is for taking a bulk datastore of a million customers and a new datastore of a thousand customers and comparing them. Both datastores have the same structure, but maybe different data. Assume the following example, where duplicates are possible in the datastore:

- Bulk data is rows A, B, C, D, E, F, I, J, K, L, M.
- New data is rows G, H, I, J, J.

Applying the operators:

- INTERSECT DISTINCT is rows I, J.
- INTERSECT ALL is rows I, J, J.
- MINUS is rows A, B, C, D, E, F, K, L, M.
- UNION is rows A, B, C, D, E, F, G, H, I, J, K, L, M.
- UNION ALL is rows A, B, C, D, E, F, G, H, I, I, J, J, J, K, L, M.

Agenda

- Working with Business Rules
- Using Variables
- Datasets and Sets
- **Using Sequences**
 - Sequences
 - Temporary Indexes
 - Tracking Variables



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

ODI sequences are primarily intended to replace the missing functionality in certain database engines. They function in the same way as a variable that is automatically incremented each time it is accessed. ODI sequences are mostly used to provide similar functionality as identity attributes.

This section explains how to configure your flow to force data to pass through the agent, and how to populate identity attributes.

Types of Sequences

- Native sequences
- Standard sequences
- Specific sequences



ORACLE®

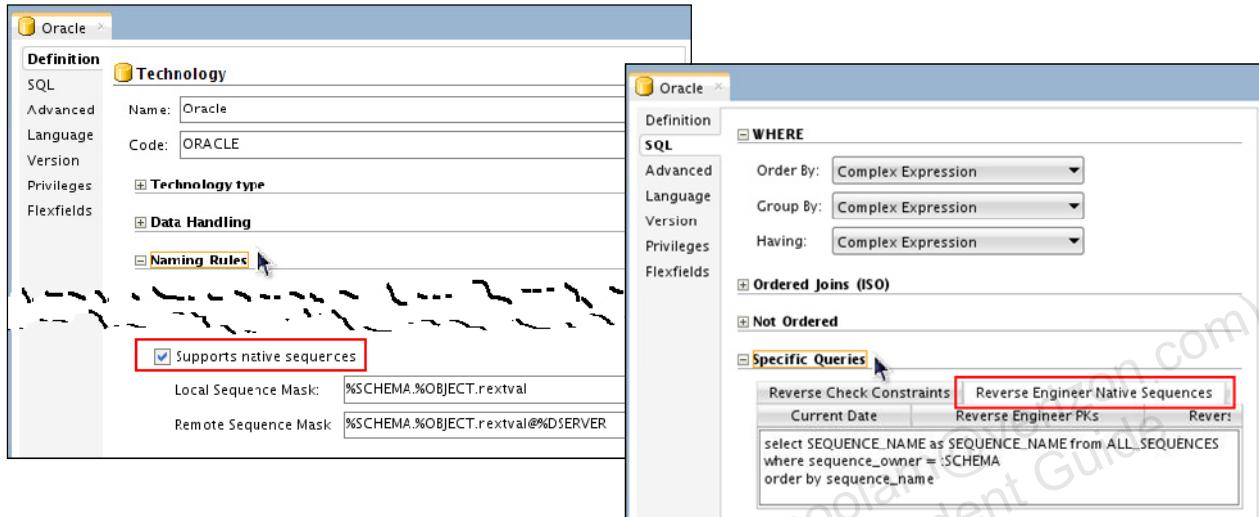
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Oracle Data Integrator sequences are intended to map native sequences from RDBMS engines, or to simulate sequences when they do not exist in the RDBMS engine. A sequence can be created as a global sequence or in a project. Global sequences are common to all projects, whereas project sequences are available only in the project where they are defined.

Oracle Data Integrator supports three types of sequences:

- Standard sequences, whose current values are stored in the repository
- Specific sequences, whose current values are stored in an RDBMS table cell. Oracle Data Integrator reads the value, locks the row (for concurrent updates) and updates the row after the last increment.
- Native sequence, that maps a RDBMS-managed sequence

Support for Native Sequences



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Oracle Data Integrator now provides support for a new type of sequence that directly maps to database-defined sequences. When created, these sequences can be picked from a list retrieved from the database. Native sequences are used as regular Oracle Data Integrator sequences, and the code generation automatically handles technology-specific syntax for sequences.

This feature simplifies the use of native sequences in all expressions, and enables cross-references when using such sequences.

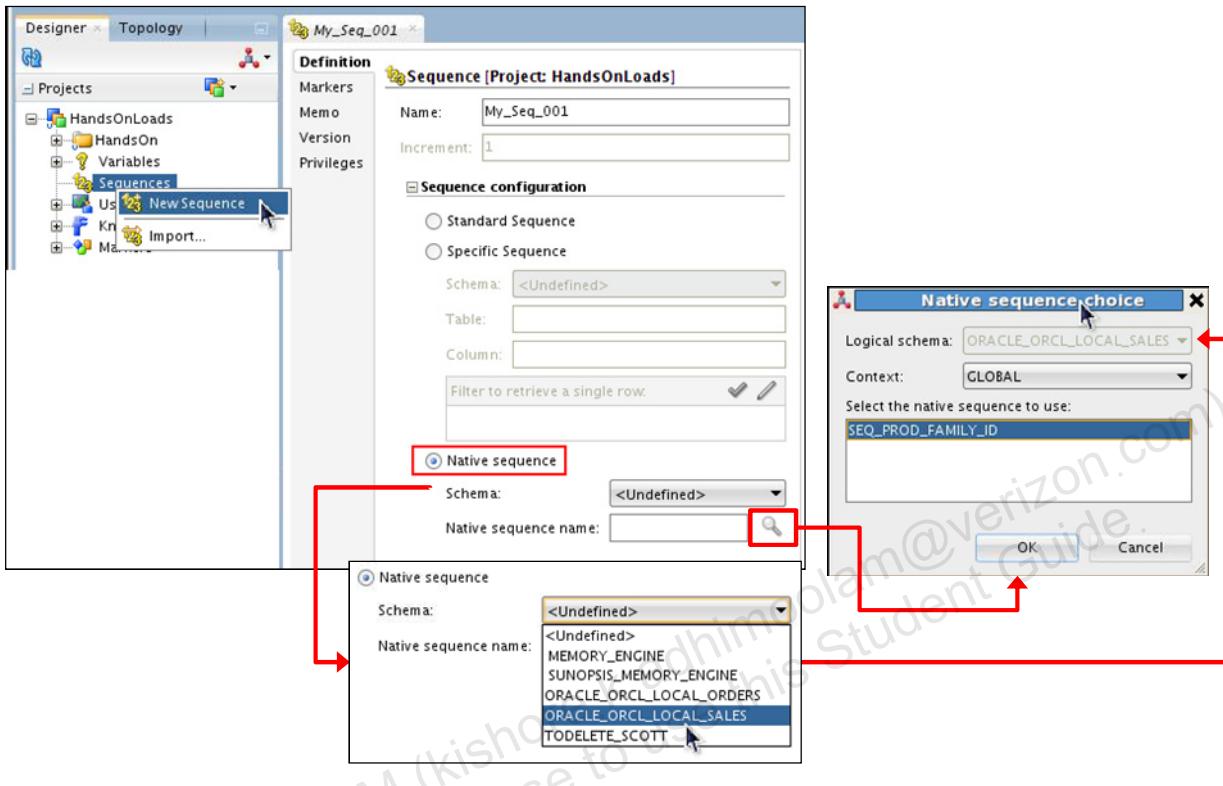
In Technology > Definition

- Select **Supports Native Sequence**.
- **Local Sequence Mask:** %SCHEMA.%OBJECT

In Technology > SQL, set **Reverse Engineer Native Sequence SQL Query**:

```
SELECT SEQUENCE_NAME FROM ALL_SEQUENCES WHERE
SEQUENCE_OWNER= :SCHEMA
```

Creating a Native Sequence



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

1. In Designer > Projects > *My_Project* > Sequences, right-click and select **New Sequence**.
2. In the Definition tab, select Native Sequence.
3. Select a schema.
4. Click the magnifying glass icon.
5. Select an existing native sequence in a given context.

Referring to Sequences

- Standard sequences:
 - In code, the sequence name is followed by `_NEXTVAL`.
 - This increments the sequence and then retrieves the new value.
 - You do not need to specify the scope.
 - Substitution versus binding:
 - Substitution: `#<sequence_name>_NEXTVAL`
 - Binding: `:<sequence_name>_NEXTVAL`
- Native sequences:
 - `<sequence_name>.NEXTVAL`

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Referring to a sequence in code, such as a SQL expression, is the same as for variables, except that the name of the sequence must be followed by the word `_NEXTVAL`. `_NEXTVAL` specifies that the sequence should be incremented, and then the new value should be returned.

Unlike variables, you do not need to specify the scope of a sequence when you use it. Thus, to refer to a sequence, use the hash or pound sign, the name of the sequence, an underscore, and the word `NEXTVAL`. Like variables, you can specify binding by using a colon instead of the pound or hash (#) sign.

You should note the difference between ODI sequences and native sequences. For example, to refer to a native Oracle sequence, you do not specify either substitution or binding, as Oracle manages the sequence. All you need to do is call it with the Oracle syntax, that is, the sequence name followed by a period and `NEXTVAL`.

Note: Sequences Updated by Agent



- An ODI Standard sequence is incremented by the agent each time it processes a row affected by this sequence.
- When a rule is entirely processed within a DBMS, Oracle Data Integrator sequences in this rule are *not* incremented.

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You must realize that ODI sequences are updated by the ODI Agent. This updating does not happen on the database itself. Thus, if a rule is performed entirely within a DBMS, any ODI sequences are not updated. If you want them to be updated, you should restructure your flow to pass through the agent.

Using Standard Sequences in Mappings Correctly

To generate a new sequence value for each row, your mapping should meet the following conditions:

- Mapping must be executed on the target.
- The attribute must be set for insert but not update.
- Unique keys must be checked in the flow not containing the attribute.
- The Not Null check box must be deselected for the attribute.
- The attribute must not be part of the update key.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The problem is that you want to use a native or ODI sequence in mapping. In particular, you want the sequence to generate a new value for each row that is processed. This is a rather complex situation, but can be handled in ODI. You have to ensure that the ODI Agent executes the rows individually. Also, you want to generate a sequence value only once for a row, so you avoid performing updates on the attribute. The general solution is as follows:

- First, set the mapping to be executed on the target. This means that the sequence value is generated only when the row is being inserted into the target table. This prevents the sequence value from being incremented unnecessarily.
- Next, set the attribute to be used only in inserts and not in updates. To do this, select the target attribute and select the Insert check box. Deselect the Update check box.
- You should also ensure that the attribute is not part of any unique key used in flow control. This is important because when any flow control is performed, the attribute shows no value.
- Similarly, deselect the Not Null check box for the attribute.
- Lastly, the attribute must not be part of an update key if you are using an Incremental Update IKM. So, click the name of the target table and select an update key that does not include this attribute. If necessary, set the update key to <Undefined> and manually select some other attributes.

Using ODI Standard Sequences in Mappings

In addition, if you use an ODI Standard sequence:

- Use the sequence in the mapping in binding mode
- Modify the flow to force the integration phase to pass all data through the agent
 - Use a multiple-technology “append” or “control append” IKM.
 - There should be a `SELECT / INSERT` command, rather than an `INSERT... SELECT`.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

If you are using an ODI sequence, you have a few more things to check.

- You must use the sequence in binding mode. This is the mode that is automatically chosen by the Expression Editor.
- The last step is the most difficult part. You must ensure that the flow configuration forces all data to pass through the agent in the integration phase. The simplest way to do this is to select a multiple-technology IKM. These IKMs typically use the agent to read data from one server, and then write it out on another. This is what you want.
- Avoid an IKM that uses an advanced mechanism for transferring data directly, such as linked servers or an export/import.
- To double-check that the IKM is appropriate, execute the mapping and open the session in Operator. Locate the step at the end that inserts rows into the target table. If it is split into a `SELECT` half and an `INSERT` half, the data is passing through the agent. If it is one continuous `INSERT...SELECT` command, the data is being transferred directly. This means that you will have only one new sequence value for the entire session.

Populating Native Identity Attributes

Your mapping should be as follows:

- No mapping for the identity attribute
 - Clear the Expression box.
 - Deselect the Active, Update, and Insert check boxes.
- Identity attribute's Not Null check box deselected
- No primary or alternate keys that reference the attribute
 - Applies only if using flow control
- Identity attribute not part of the update key
 - Applies only if using "Incremental Update" IKMs



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

A similar situation arises when you want to populate a table that has a native identity attribute defined on it in the database. You must be careful not to attempt to populate the attribute directly, because the database will most certainly prevent this and throw an error. You must also be careful not to attempt to use the attribute as an update key, because it does not usually contain any information. You proceed as follows:

- First, ensure that no mapping is defined on the attribute. Clear the mapping expression box, and deselect the Active Mapping, the Update, and the Insert check boxes.
- Deselect the Not Null check box. When you insert data, this attribute will certainly be empty. Therefore, selecting the Not Null check box creates spurious errors.
- Ensure that no primary or alternate keys reference the attribute. The keys prevent flow control from working properly. This is because flow control takes place on the Staging area where the identity attribute is blank. It is, therefore, unsuitable as a primary or alternate key.
- If you use an "Incremental Update" IKM, ensure that the attribute is not part of the update key. Check that the Key check box on the attribute is not selected. Also check that the update key defined for the target is not a primary or alternate key that references this attribute.

Sequences: Best Practices



- ODI sequences are not as fast as DBMS sequences.
- DBMS sequences should be used wherever possible.

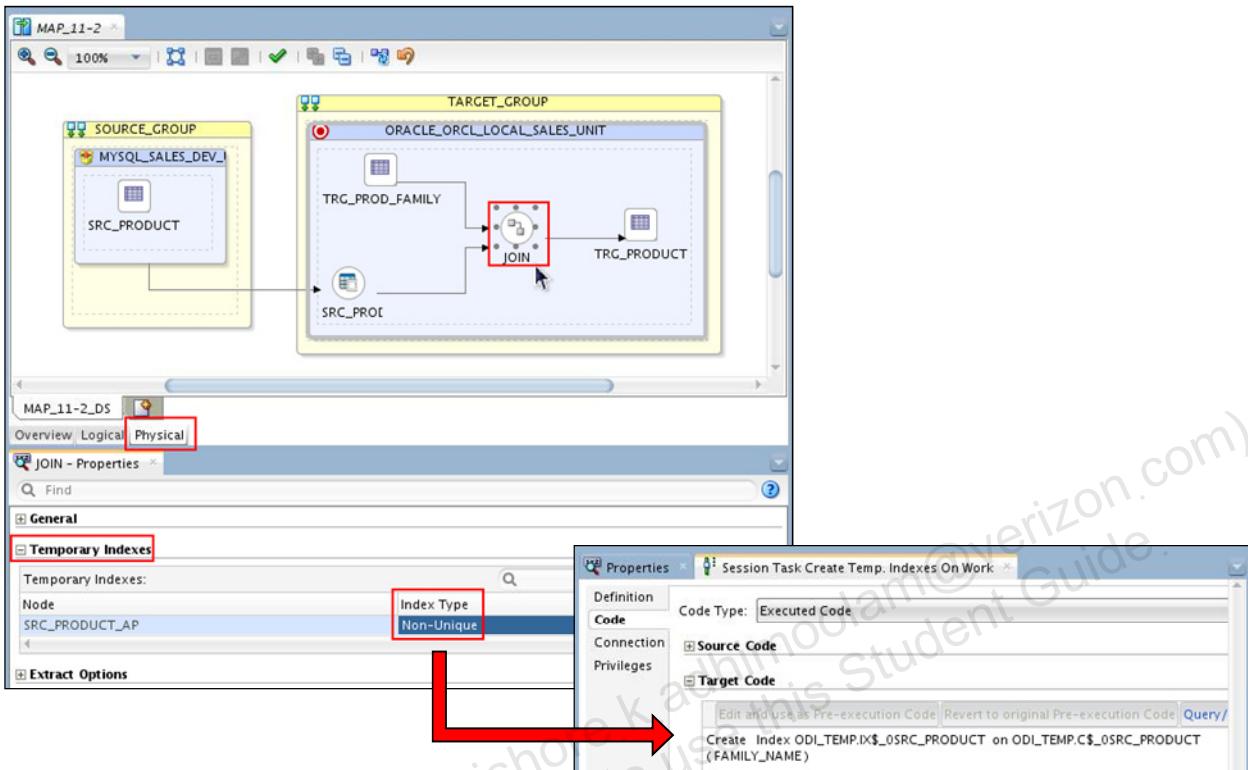


ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

By definition, ODI sequences are not as fast as DBMS sequences because each row must pass by the agent.

Automatic Temporary Index Management



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

If you are creating joins or filters on source tables, you can have Oracle Data Integrator automatically generate temporary indexes for optimizing the execution of these joins or filters. The user selects the type of index that needs to be created in the list of index types for the technology. Knowledge Modules automatically generate the code for handling index creation before executing the join and filters, as well as deletion after usage.

This feature provides automated optimization of join and filter execution, and enables better performances for integration mappings.

Tracking Variables and Sequences

- You can track the actual values of variables and sequences at run time.
- This feature allows you to see the actual values of variables and sequences at execution time, making it easier to troubleshoot ODI sessions.

```

Variable #ORACLE_DATA_INTEGRATOR_DEMO.PHONE_NUMBER PHONE
select
  C1_CUST_ID CUST_ID,
  C2_DEAR DEAR,
  C6_FIRST_NAME || '' || C5_LAST_NAME CUST_NAME,
  C3_ADDRESS ADDRESS,
  CIT_CITY CITY,
  C4_AGE AGE,
  C7_AGE_RANGE AGE_RANGE,
  SUBSTR(C9_FIRST_NAME || '' || C8_LAST_NAME ,1,100) SALES_PERS,
  'T_IND_UPDATE'
from
  SNPDEMO.TRG_CITY CIT, STAGING.C$_OTRG_CUSTOMER, STAGING.C$_ITRG_CUSTOMER, STAGING.C$_2TRG_CUSTOMER
where
  (1=1)

```



```

select
  C1_CUST_ID CUST_ID,
  C2_DEAR DEAR,
  C6_FIRST_NAME || '' || C5_LAST_NAME CUST_NAME,
  C3_ADDRESS ADDRESS,
  CIT_CITY CITY,
  6505066217 PHONE, Actual value of variable
  C4_AGE AGE,
  C7_AGE_RANGE AGE_RANGE,
  SUBSTR(C9_FIRST_NAME || '' || C8_LAST_NAME ,1,100) SALES_PERS,
  'T_IND_UPDATE'
from
  SNPDEMO.TRG_CITY CIT, STAGING.C$_OTRG_CUSTOMER, STAGING.C$_ITRG_CUSTOMER, STAGING.C$_2TRG_CUSTOMER
where
  (1=1)

```

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Customers use variables and sequences extensively in their ODI processes. Until the ODI 11.1.1.6 release, it was not possible to see the actual values of variables and sequences at run time.

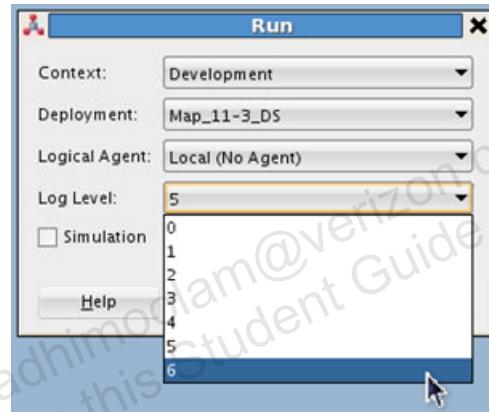
The new Tracking Variables and Sequences feature allows end users to see the actual values of variables and sequences at execution time, making it easier to troubleshoot ODI sessions.

You have to use log **level 6** to track variables. The Show Values button is a “+” in the summary screen at the Task level. At execution time, you can view or hide the actual values in the Step or Task code by using the Show/Hide Values button.

Bind variables are not resolved.

How Variable and Sequence Tracking Works

- A new log level has been introduced in ODI 11g: Level 6, which will have the same behavior as Level 5, but with the addition of variable tracking.
- Set Log Level to 6 in order to track variable and sequence values at run time.
- Note that tracking variable and sequence values will have a performance impact.



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

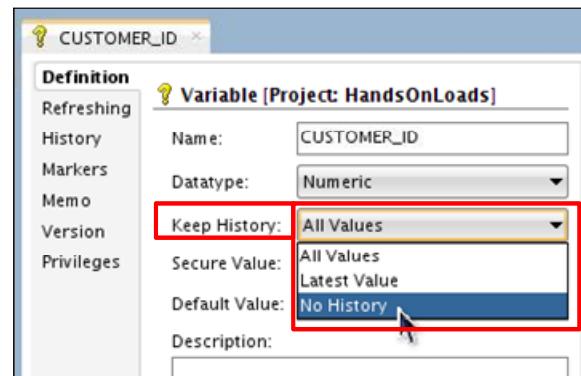
A new log level, Level 6, has been introduced. It has the same behavior as Level 5, but with the addition of variable tracking.

The new log level will be available for an execution invoked against any object type that can be executed from ODI Studio (for example, Mappings, Packages, Scenarios, Load Plans).

Log Level 6 is available only for starting or restarting a session. It is not available as a design setting in any of the editors because it may have a performance impact.

Variable Actions

The Action field, when displaying a variable, is **Keep History**.



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Variable values can be hidden by using the Secure Value check box. If Secure Value is selected:

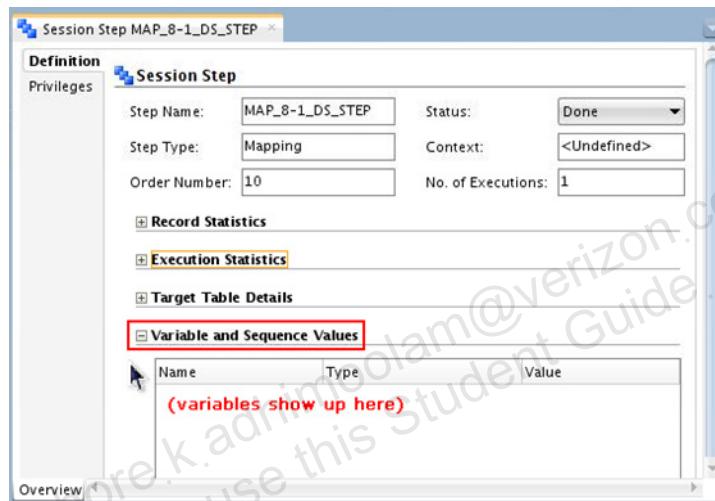
- The variable will never be tracked: it will be displayed unresolved in the source or target code, it will not be tracked in the repository, and it will not be “historized”
- The Keep History parameter is automatically set to `No History` and cannot be edited
- Variables storing passwords in Topology are not resolved

Prior to version 11g, the Keep History values used to have different names:

- All Values was called Historize.
- Latest value was still called Latest Value.
- No History was called Not Persistent.

Definition Tab of Session Step or Session Task

- Variable and Sequence values are displayed in the Definition panel of Session Step or Task Step in Operator.
- Expand the Variable and Sequence Values node to display them.
- Bind variables (:MY_VAR) are also tracked.



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The Start and End value of a Sequence are displayed, because a Sequence can have multiple values in the same session.

The functionality described previously is also available through the ODI Console:

- Setting Log Level to 6 during execution time
- Viewing Variable and Sequence values at the Session Step and Task level
- Show Values button at the Code panel level (note that "Show Values" is for the web console only, not the Studio user interface.)

Activating Variable and Sequence tracking can also be done from the command line, using `OdiStartScen` or the `OdiInvoke` web service.

Quiz

Which of the following statements is true?

- a. You should use native DBMS sequences wherever possible.
- b. If a rule is performed entirely within a DBMS, all ODI sequences are automatically updated.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Answer: a

Explanation:

a is true because native is usually faster.

b is false because ODI sequences are updated by the ODI agent. This updating does not happen on the database itself. Thus, if a rule is not performed entirely within a DBMS, all ODI sequences are *not* updated. If you want them to be updated, you should restructure your flow to pass through the agent.

Summary

In this lesson, you should have learned how to:

- Use business rules, variables, and set-based operators with ODI mappings
- Use data sets and sequences, including native sequences, with ODI mappings
- Create temporary indexes to speed up retrieval



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Checklist of Practice Activities

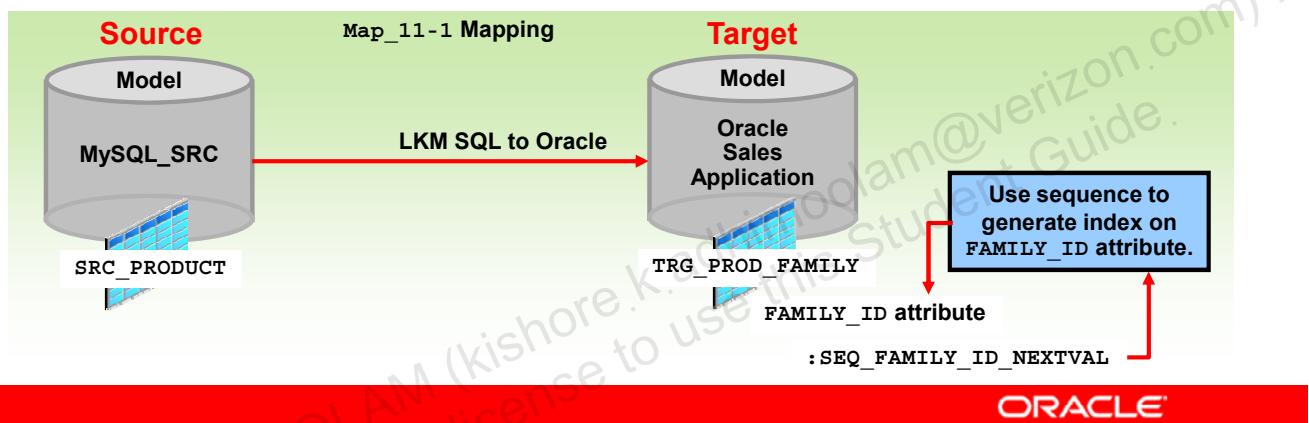
- ✓ 1-1: Logging In and Help Overview
- ✓ 2-1: Creating and Connecting to ODI Master and Work Repositories
- ✓ 3-1: Configuring Standalone Agents Using the Common Admin Model
- ✓ 4-1: Working with Topology
- ✓ 5-1: Setting Up a New ODI Project
- ✓ 6-1: Creating Models by Reverse-Engineering
- ✓ 7-1: Checking Data Quality in the Model
- ✓ 8-1: Creating ODI Mapping: Simple Transformations
- ✓ 9-1: Creating ODI Mapping: Complex Transformations
- ✓ 9-2: Creating ODI Mapping: Implementing Lookup
- ✓ 10-1: Creating ODI Mapping: Exporting a Flat File to a Relational Table
- 11-1: Using Native Sequences with ODI Mapping
- 11-2: Using Temporary Indexes
- 11-3: Using Sets (UNION) with ODI Mapping
- 12-1: Creating and Using Reusable Mappings
- 12-2: Developing a New Knowledge Module
- 13-1: Creating an ODI Procedure
- 14-1: Creating an ODI Package
- 14-2: Using ODI Packages with Variables and User Functions
- 15-1: Debugging Mappings
- 16-1: Creating and Scheduling Scenarios
- 17-1: Using Load Plans
- 18-1: Enforcing Data Quality with ODI Mappings
- 19-1: Implementing Changed Data Capture
- 20-1: Setting Up ODI Security
- 20-2: Integration with Enterprise Manager and Using ODI Console

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Practice 11-1: Using Native Sequences with ODI Mapping

1. In ODI, create the procedure `Create_ORCL_SEQ_FAM_ID` and execute it to create the sequence `SEQ_FAMILY_ID` in the RDBMS.
2. In ODI, create the native sequence `SEQ_FAMILY_ID`.
3. Create and execute the mapping `Map_11-1` to load the `TRG_PROD_FAMILY` target table by using the native sequence to generate ID numbers for that table.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In this practice, you first use ODI to define the procedure `Create_ORCL_SEQ_FAM_ID`. You then execute this procedure, which creates the sequence `SEQ_FAMILY_ID` in the RDBMS.

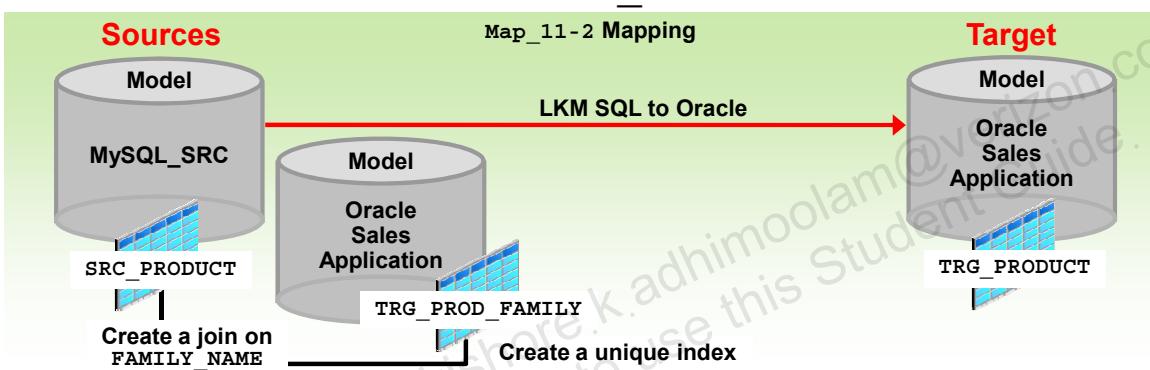
Next, you use ODI to create an *ODI definition* of the native sequence that you just created in the RDBMS, using the same name, `SEQ_FAMILY_ID`.

Finally, you create and execute the mapping `Map_11-1` to load the `TRG_PROD_FAMILY` target table by using the new native sequence to generate ID numbers for that table.

Note: This lesson does not have a hands-on practice using variables. A hands-on practice using variables will be performed in Lesson 14, in which you will use variables in packages.

Practice 11-2: Using Temporary Indexes

1. Create an ODI mapping, Map_11-2, to load data into the TRG_PRODUCT target table from two source tables.
2. Use source table TRG_PROD_FAMILY as a lookup table to obtain the ID number (populated by a sequence in the previous practice's mapping).
3. For the lookup, create a temporary index to join the two source tables on the FAMILY_NAME attribute.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In the previous practice, TRG_PROD_FAMILY was a *target* table. In this practice, TRG_PROD_FAMILY serves as one of the *source* tables.

First, you create a new ODI mapping, Map_11-2 to load data into the TRG_PRODUCT target datastore table in the Oracle Sales Application model. You specify the source tables as SRC_PRODUCT from the HSQL_SRC model, and TRG_PROD_FAMILY from the Oracle Sales Application model.

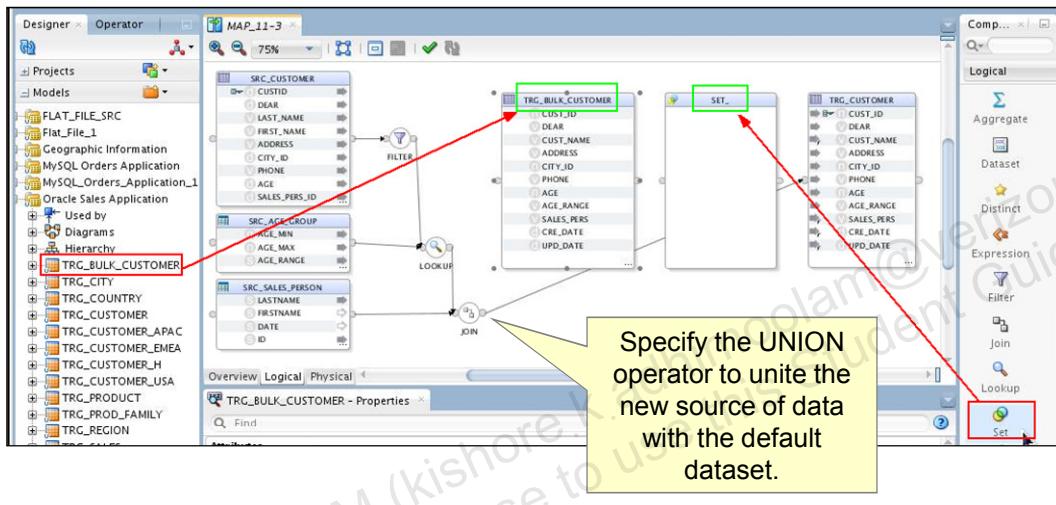
TRG_PROD_FAMILY will be used as a lookup table to obtain the ID number generated by the previous lesson's mapping, which used a sequence to populate the ID number.

For the lookup, you create a temporary index to join SRC_PRODUCT and TRG_PROD_FAMILY in their FAMILY_NAME attribute.

Finally, you execute the mapping, view the source code, and examine the inserted rows in the TRG_PRODUCT target datastore.

Practice 11-3: Using Sets with ODI Mapping

1. Create a duplicate of mapping MAP_9-2 as MAP_11-3.
2. Add a set component for the UNION.
3. For the source, specify the TRG_BULK_CUSTOMER datastore from the Oracle Sales Application model.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In this practice, you add a new set to a mapping. Building upon the MAP_9-2 mapping you created in Lesson 9, you will add a bulk feed of customer data from another system by adding a unioned datastore.

First, you make a copy of mapping MAP_9-2, naming the copy as MAP_11-3.

Now, you create a new set, which is the union of source and target bulk customers datastore from the Oracle Sales Application model.

You specify the UNION operator to unite this new source of data with the other join and lookup that you specified in MAP_9-2.

Next, you perform an execution *simulation* to preview the code that will be used for this new set with the UNION operator.

Finally, you execute the mapping and verify the rows inserted into the target table.

Note: Completing this practice is critical for all the following practice sessions.

12

Designing Mappings: Advanced Topics 2

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Use partitioning of datastores
- Use reusable mappings
- Use user functions
- Use substitution methods
- Modify and develop Knowledge Modules

Agenda

- **Partitioning**
- Configuring Reusable Mappings
- Using User Functions
- Substitution Methods
- Modifying Knowledge Modules



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Partitioning

- Support (sub)partitioning on datastores.
- Reverse-engineer partition information.
- Enable partition selection in mappings.
- Auto-generate SQL statements:
 - `SELECT * FROM MY_TABLE PARTITION (MY_PARTITION)`
- Defined by **technology**
 - **Partitioning Support:** No | Partitions | Sub-Partition
 - **Partition Mask:** `%OBJECT_NAME PARTITION (%PARTITION)`



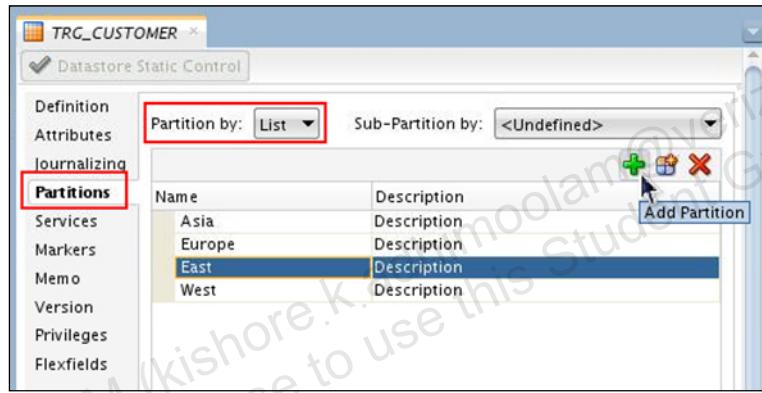
ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

ODI supports partitioning features of the data servers. Partitions can be reverse-engineered by using RKM's or manually created into models. When designing a mapping, you can define the partition to address on the sources and target datastores. ODI code generation handles the partition usage syntax for each technology that supports this feature.

Definition in Datastore After Reverse-Engineering

```
CREATE TABLE trg_customer
( customer_id      NUMBER(6)
, cust_first_name  VARCHAR2(20)
, cust_last_name   VARCHAR2(20)
, cust_address     VARCHAR2(20)
, nls_territory    VARCHAR2(30)
, cust_email       VARCHAR2(30) )
PARTITION BY LIST (nls_territory) (
PARTITION Asia     VALUES ('CHINA', 'THAILAND'),
PARTITION Europe   VALUES ('GERMANY', 'ITALY', 'SWITZERLAND'),
PARTITION West     VALUES ('AMERICA'),
PARTITION East     VALUES ('INDIA'));
```

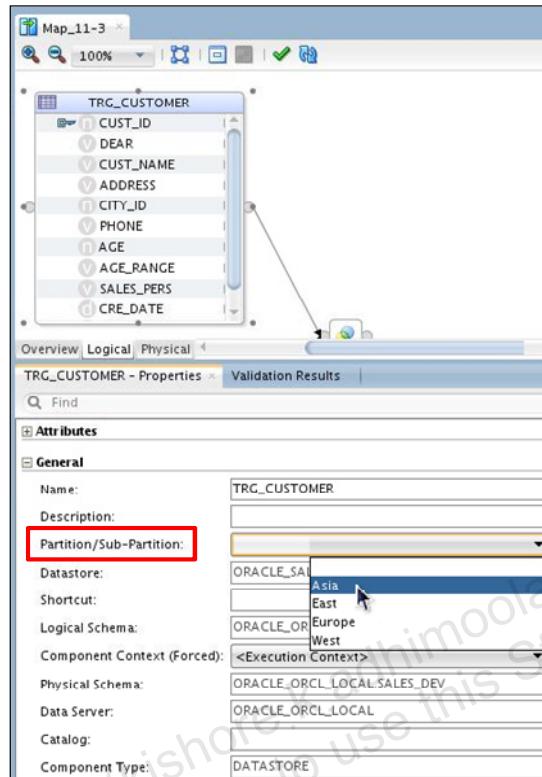


ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Partition information can be reverse-engineered by using reverse-engineering Knowledge Modules (RKM)s or can be manually created into models.

Using Partitioning in a Mapping



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In a mapping, you can specify the use of partitions on a target datastore.

Agenda

- Partitioning
- **Configuring Reusable Mappings**
- Using User Functions
- Substitution Methods
- Modifying Knowledge Modules



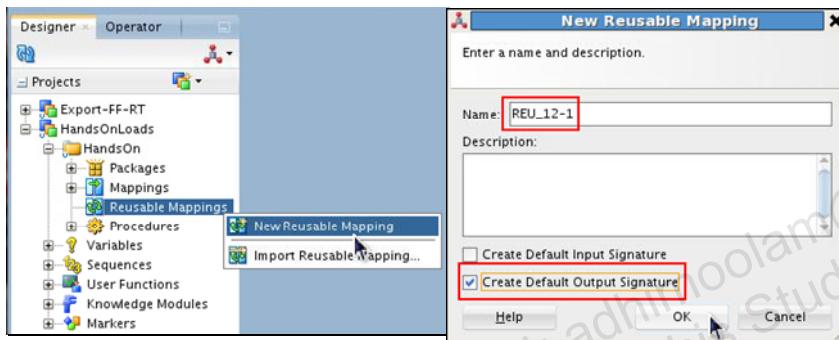
ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Reusable Mappings

ODI mappings:

- Conventional: Target datastore is *permanent* (predefined in the model). 
- Reusable: Target datastore is *temporary* (defined in the project). 



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Reusable mappings allow you to encapsulate a multistep integration (or portion of an integration) into a single component, which you can save and use just as any other components in your mappings. Reusable mappings are a convenient way to avoid creating a similar subroutine of data manipulation that you will use many times in your mappings.

For example, you could load data from two tables in a join component, pass it through a filter component, and then a distinct component, and then output to a target datastore. You could then save this procedure as a reusable mapping, and place it into future mappings that you create or modify.

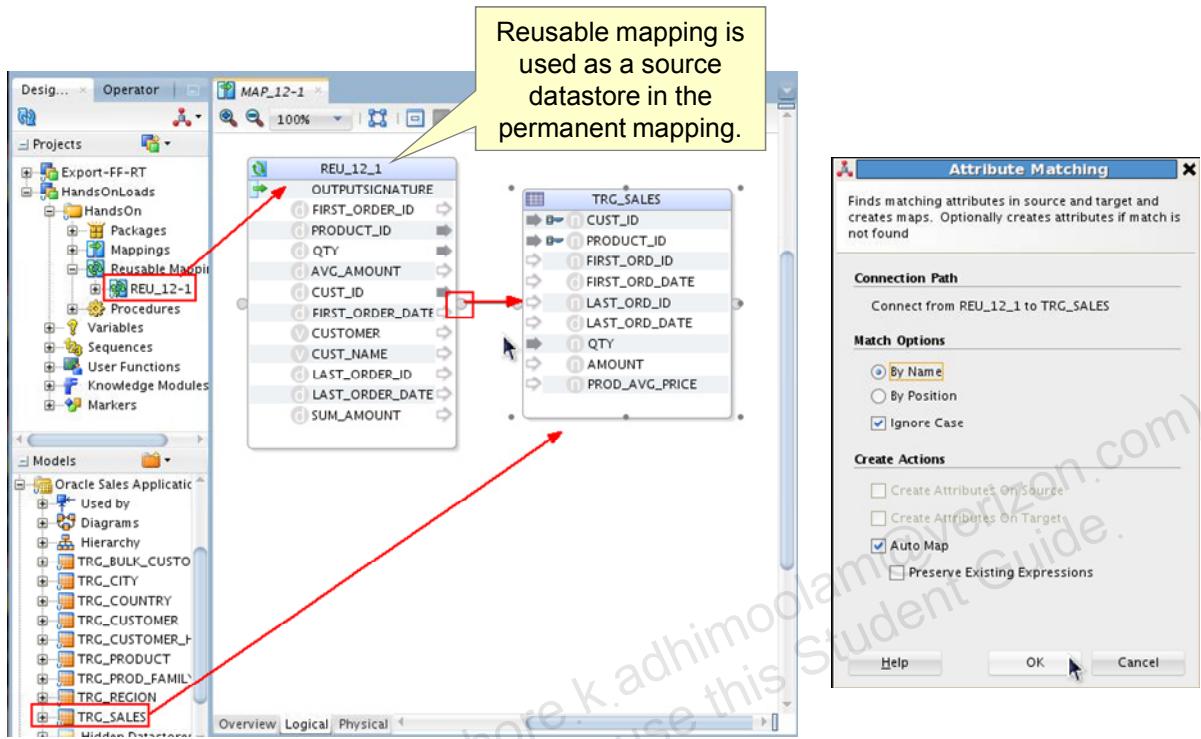
After you place a reusable mapping component in a mapping, you can select it and make modifications to it that only affect the current mapping.

Reusable mappings consist of the following:

- Input Signature and Output Signature components (zero, one, or many of each)
- Regular mapping components

By combining regular mapping components with signature components, you can create a reusable mapping intended to serve as a data source, as a data target, or as an intermediate step in a mapping flow. When you work on a regular mapping, you can use a reusable mapping as if it were a single component.

Using Reusable Mappings: Example



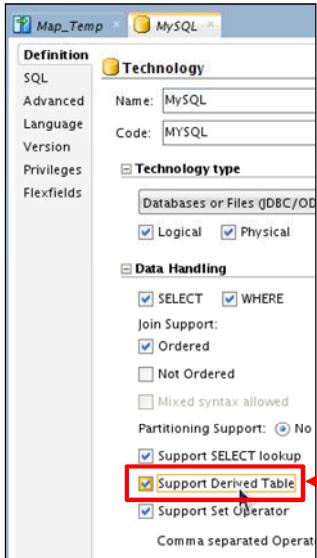
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

ORACLE

In this example, the reusable mapping REU_12-1, with its target datastore TRG_SALES, is used in place of a source datastore for the permanent mapping MAP_12-1.

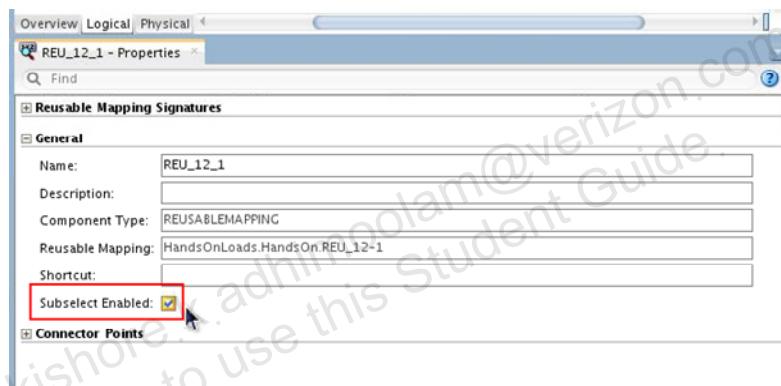
In the Mapping Properties, you can see that the rows of the temporary reusable mapping are mapped to the rows of the target datastore.

Derived Select (Subselect) for Reusable Mappings



In Technology > Definition, select Support Derived Table.

In a mapping, if a temporary source is used, a new check box appears:



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

If you are using a reusable mapping as a source or a lookup table in another mapping, you can generate a Derived Select (subselect) statement corresponding to the output signature. The code generated for the subselect is either a default generated code or a customized syntax defined in an IKM.

This feature eliminates the need for complex packages to handle reusable mappings and simplifies the execution of cascades of reusable mappings.

The Subselect Enabled check box is for backward compatibility only, and is enabled only if the Reusable Mapping has no Input Signatures and exactly one Output Signature, which is the equivalent of the 11g Temporary Interface. When this check box is deselected, the code generator will decide whether the reusable mapping expression is “flattened” into the generated SQL or whether a subselect is still used. In most cases, the reusable mapping can be flattened into the existing select of the enclosed mapping.

Agenda

- Partitioning
- Configuring Reusable Mappings
- **Using User Functions**
- Substitution Methods
- Modifying Knowledge Modules



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

ODI user functions are macros in ODI that are replaced at run time by code that is specific to the technology where they are used. They can be used in the same way as database functions, with the exception that the code for the function is substituted literally.

What Is a User Function?

- A user function is a cross-technology macro defined in a lightweight syntax.
- It is used to create an alias for a recurrent piece of code or to encapsulate a customized transformation.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

A user function is a cross-technology macro defined in a very simple language.

Why Use User Functions?

Example 1:

- A simple formula:
If <param1> is null then <param2> else <param1> end if
- Can be implemented differently in different technologies:

- Oracle

```
nvl(<param1>, <param2>)
```

- Other technologies:

```
case when <param1> is null  
      then <param2>  
      else <param1>  
      end
```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The question can be answered with these examples.

In the first, imagine a situation in which you need a transformation to work on different technologies. As you now know, the feature set of the transformation is dependent on the technology of the server on which it is executed. Now you want to use a function that behaves as follows: If the first parameter is null, you take the second parameter. However, if the first parameter is not null, you use it.

In Oracle technology, this implementation is easy. You use the built-in function `nvl`, which behaves similarly. But most technologies do not have this function, or have a different name for it. Therefore, you have to write it out in full. You have to use basic SQL syntax that works on every platform, that is, by using a `CASE` statement.

Why Use User Functions?

Example 2:

- A commonly used formula:

```
If <param1> = '1' then 'Mr'  
    else if <param1> = '2' then 'Ms'  
    else if <param1> = '3' then 'Mrs'  
    else null end if
```

- Could be aliased to:

```
NumberToTitle(<param1>)
```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In the second example, imagine a situation in which you regularly transfer data between two servers with different rules for titles. On the source server, you use “1” to mean “Mr,” “2” to mean “Ms,” and “3” to mean “Mrs.” On the target server, you store the title directly as a string. If you performed this transformation in only one place, there would be no problem. But in the imaginary scenario, you have to perform this transformation in dozens of places. Also, the specification on the source server might soon change.

Therefore, it is convenient to have a single function defined in ODI to handle this situation. You can call it NumberToTitle. You pass it a numeric title and it returns a string. You can then use this function everywhere. If the specification changes, you must change the function in only one place.

Properties of User Functions

- A user function always has:
 - One syntax: Defines how the function is called by other objects in ODI
 - Several implementations specific to different technologies
 - A scope:
 - Global functions can be used anywhere.
 - Project functions can be used within the project.
- User functions are organized into groups.

 ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Defining how other objects call the function is entirely independent of any underlying technology.

However, the implementations are each specific to a technology. They tell ODI how to translate the parameters into the language of the technology to obtain the right result.

User functions, such as variables and sequences, have a scope. That is, they either belong to a project or are global. Global functions can be used by any code in any project. Project functions can be accessed only by other objects in the same project.

Lastly, user functions are always organized into groups. Groups help to manage user functions of significant numbers.

Using User Functions

At design time, you can refer to user functions in the same way as any regular database function.

- You can use them in expressions, joins, filters, procedures, and so on.
- They are available in the Expression Editor.



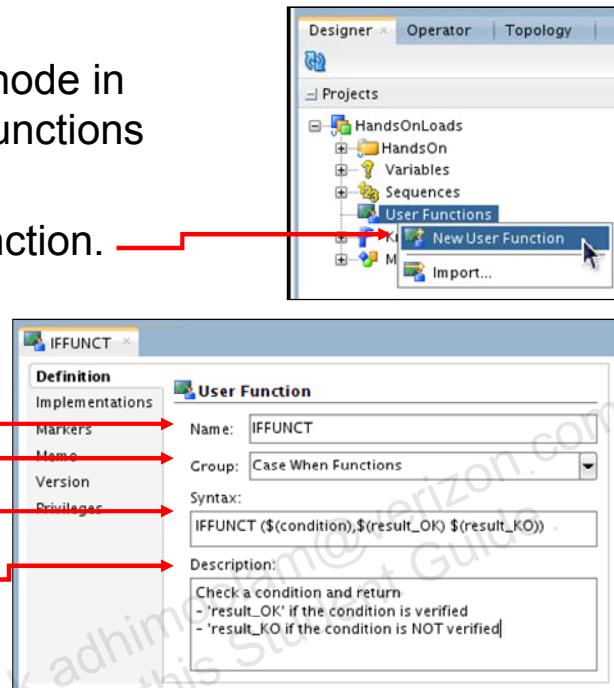
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

User functions behave in the same way as normal database functions at design time. This means that wherever you can refer to a database function, you can also refer to a user function. Of course, the user function must have an implementation for the given technology.

In particular, you can use them in expressions, joins, filters, and procedures. They also appear in the Expression Editor under the Global Functions or Project Functions nodes.

Creating a User Function

1. Select the User Functions node in a project, or Global User Functions in the Others view.
2. Right-click > New User Function.
3. Enter:



ORACLE

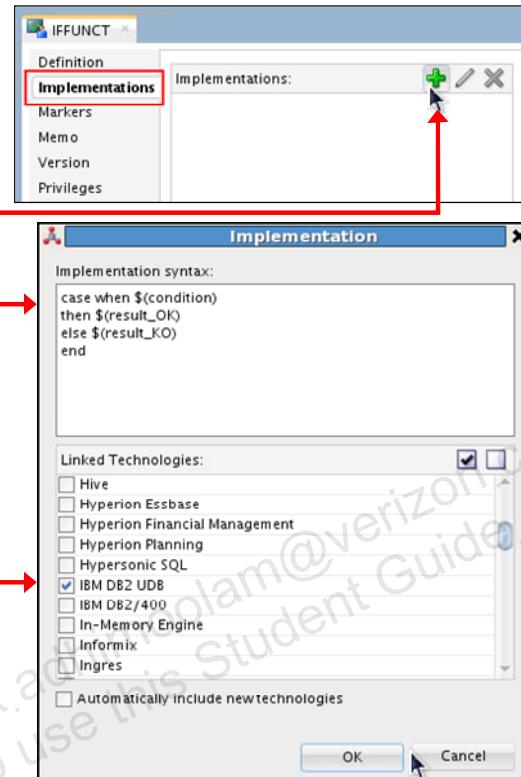
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The steps for creating a user function are as follows:

1. Select the Functions node where you want to create the function. This will either be under a project in the Projects view or in the Others view.
2. Right-click the node and select New User Function.
3. Enter the name, syntax, and description of your new function. In the description, it is a good idea to state what your function does in pseudocode or in natural language. You look at how the syntax works in the next few slides.
4. As stated, every function belongs to a group. If you have function groups defined, you can select one from the drop-down list. Otherwise, enter the name of a new group in the Group field.

Defining an Implementation

1. Click the Implementations tab.
2. Click the Add button.
3. Enter the code for the implementation.
4. Select the applicable technologies.
5. Click OK to save.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

After covering the syntax half of the function, you must create at least one implementation. To do this, perform the following:

1. Click the Implementations tab. You can now see the list of technologies for which the implementations exist, at the top of the window. The technologies without implementations are at the bottom.
2. Click the Add button next. This opens the Implementation window where you can enter the new implementation.
3. Enter the code that implements the user function on a given technology. You can use all the features and syntax available for that technology. The next few slides cover the specific syntax to use to refer to arguments of the function.
4. Select all technologies to which your implementation applies. One implementation might be used on several technologies.
5. Click OK to save your function and close the window.

Syntax and Implementations

You should use one format for arguments in syntax and implementation.

- Use one of the following formats:
 - `$(<arg_name>)`
 - `$(<arg_name>)<arg_type>`
 - If `<arg_type>` is supplied, it must be “s” (string), “n” (numeric), or “d” (date).
 - For example: `$title s`
- Argument names in the syntax and implementation should match exactly.
- Example:
 - Syntax:
`NullValue($myvariable), $mydefault)`
 - Implementation:
`case when $myvariable is null
 then $mydefault
 else $myvariable
end`



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Arguments are referred to in exactly the same way in the syntax and implementation for the function. Select a name for each argument and refer to it by prefixing it with a dollar sign and enclosing it within parentheses.

You can also specify the type by entering a letter immediately after the parentheses. This letter must be “s” for a string, “n” for a numeric value, or “d” for a date.

You must use the same argument names in the syntax and implementation. Type specification is not strictly required in the implementation. However, the best practice is to use it for consistency and readability.

You now see how to implement the “nullvalue” function. Note that the implementation consists of only the text that should be inserted directly into the SQL code. For example, you do not use the word “return.”

User Functions at Design Time

- Design time versus run time
- At design time, you can refer to user functions in the same way as any regular database function.
 - You can use them in mappings, joins, filters, procedures, and so on.
 - They are available in the Expression Editor.
- Example:

```
'Sale' || iffFunc(SALE.CLOSED, 'is', 'is not')  
      || ' closed.'
```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

At design time, you can treat user functions just as you would treat any regular database function.

The next slide offers considerations for user functions at run time.

User Functions at Run Time

When the code is generated:

- Project functions are identified first, followed by global functions.
- Is the function recognized?
 - Yes: The function name is replaced by the implementation corresponding to the technology of the generated code.
 - No: The generated code remains unchanged.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Several rules are followed when you cause the code to be generated. ODI first identifies the project functions that you have used, and then the global functions. This means that if you have a global function and a project function with the same name, it is the project function that will be used.

When a function name is recognized, it is replaced with the corresponding implementation. The implementation for the technology of the generated code is used.

If a function name is not recognized, it is not substituted. The function name itself appears in the generated code. This also occurs if you do not supply enough parameters for the function.

Note: Functions in Execution Log



- Functions are design-time objects.
- In the execution log, only their implementation in the language of the technology appears.

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Note that user functions exist only in ODI. They are not created in databases, and after a session is started, only their implementation exists. When you look in the execution log with Operator, you do not see the name of the function. Only the implementation in the relevant technology appears.

Agenda

- Partitioning
- Configuring Reusable Mappings
- Using User Functions
- **Substitution Methods**
- Modifying Knowledge Modules



Using Substitution Methods

- Methods are used to write generic code:
 - Table names in a context-dependent format
 - Information about the session
 - Source and target metadata information
- Use the method with the following syntax:

```
<%=odiRef.method_name(parameters)%>
```
- Refer to the *Oracle Data Integrator Substitution Methods Reference*.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Substitution methods are direct calls to ODI methods implemented in Java. These methods are used to generate text that corresponds to the metadata stored in the ODI repository or session information.

If you write an expression to express a business rule, you do not generally know in which context this expression is run. For instance, if you refer to a specific schema by its name in your test setup, it may not work in your production environment. But you want to be able to specify your table in a generic way. Substitution methods enable you to do this. You can specify the table name with a substitution method. At run time, ODI automatically adds the name of the appropriate physical schema for the context. You can also use substitution methods to retrieve information about the current session. For example, you can include the time when the session was launched, its name, or the code for the current context. This enables you to make fine-grained tweaks to expressions based on the executing environment.

Similarly, substitution methods give you access to the metadata about the source and target of your mapping. In some cases, it may be useful to get the actual name of the physical table being populated on a server. The general syntax to use a substitution method is as follows:

`<%=odiRef.method_name(parameters)%>`

Angle brackets with percentage signs enclose all substitution method calls. The equal sign tells ODI to replace the tags and everything inside with the result of the method call.

`odiRef` is a special ODI Java object, which contains all the methods for performing substitution.

You can obtain more information about these methods in the Substitution Methods reference guide. For ODI documentation, visit the Oracle Data Integrator website:

<http://www.oracle.com/technology/products/oracle-data-integrator/index.html>

Substitution Methods: Examples

- Mapping a target column to its default value defined in ODI:

```
' <%=odiRef.getColDefaultValue()%> '
```

- Mapping a column by using the system date:

```
'The year is:  
<%=odiRef.getSysDate("yyyy")%> '
```

- Writing a generic SELECT subquery in a filter:

```
ORDER_DATE >= (SELECT MAX(ORDER_DATE) - 7 FROM  
<%=odiRef.getObjectName("SRC_ORDERS")%>)
```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

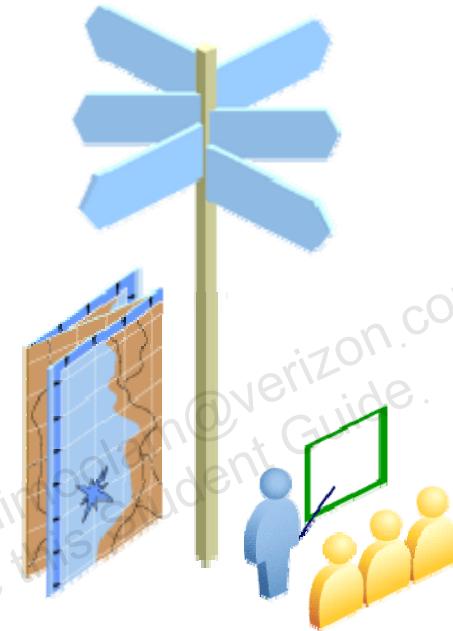
These examples show how to use substitution methods. First, you look at how to access the default value of a column. In ODI, you can specify a default value for each column. To access this default value, use the `getColDefaultValue` substitution method. If your column is a string, you must enclose the entire method call within single quotation marks. If the default value for this column is unknown, the result of this expression is the “unknown” string with single quotation marks.

You might want to record the system date into a “last updated” column. Normally, you use a function available in your database engine. What if your target is not a database, but a flat file? In this case, you can use the `getSysDate` substitution method. The system date on the machine where the agent is running is used. To call this function, pass one argument, which is the date format to be used. By passing “`yyyy`,” a four-digit date is returned. Thus, the final string is something like “The year is: 2006.”

The next example shows how to use a `SELECT` statement in a filter. For example, you want to import orders from the `ORDERS` table. You are retrieving only those orders processed in the last week. To do this, you create a subquery that retrieves the maximum order date minus seven days from the `ORDERS` table. However, the schema containing the `ORDERS` table may change on different servers. Therefore, use the `getObjectName` substitution method to refer to the table name. The generated code will contain the qualified name of the table. Thus, the final filter expression refers to something like `MSSQL_ORDERS_PROD.SRC_ORDERS`.

Agenda

- Partitioning
- Configuring Reusable Mappings
- Using User Functions
- Substitution Methods
- **Modifying Knowledge Modules**

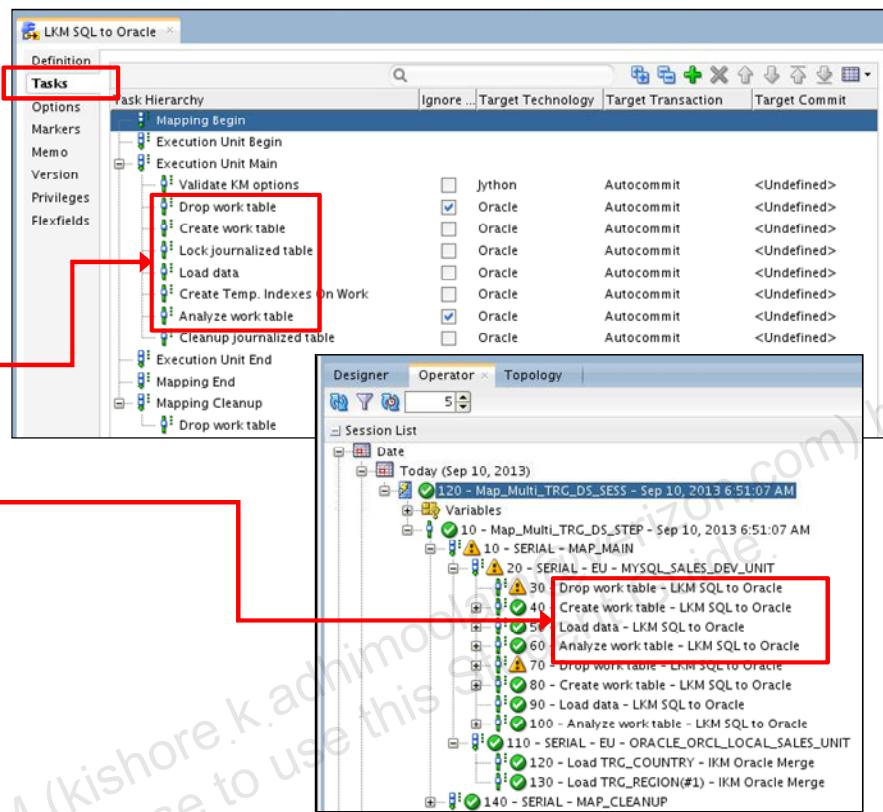


ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Description of KM Steps

- A KM is made of steps.
 - Each step has a name.
 - Steps are listed on the Details tab.
 - Same step names



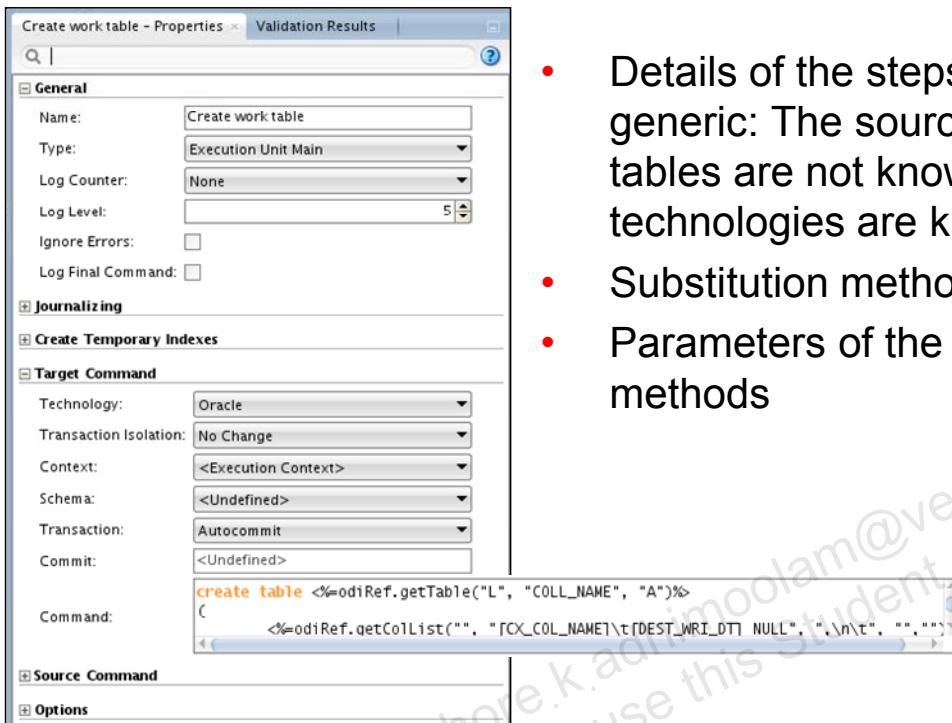
ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

A Knowledge Module is made of steps. After the execution, the step names will then be viewable in ODI Operator.

- Each step has a name and a template for code to be generated.
- The code to be generated by ODI lists the same step names.

Details of the Steps



- Details of the steps are generic: The source and target tables are not known, only the technologies are known.
- Substitution methods
- Parameters of the substitution methods

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Each step has a command on source and command on target. In the Command window, you can view and modify each command manually.

Substitution methods include placeholders for:

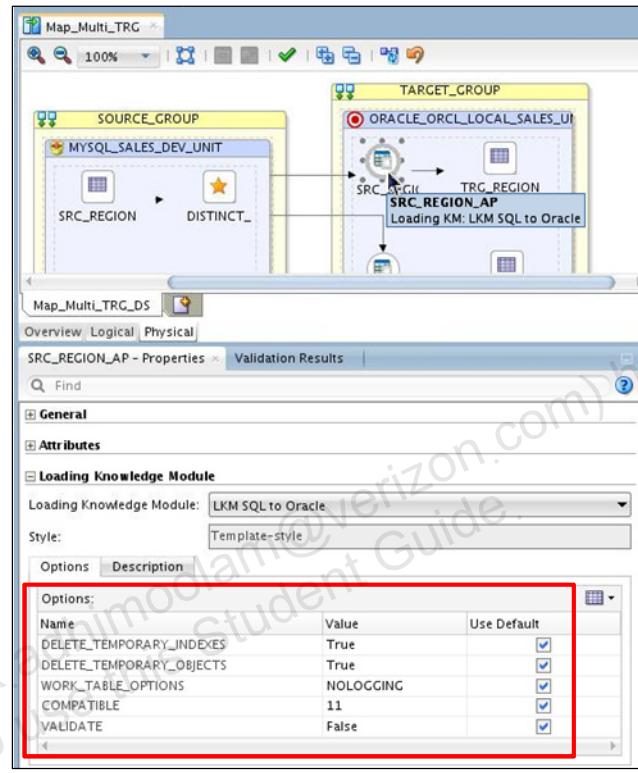
- Table names
- Column names

Parameters of the substitution methods:

- Tables
- Columns

Setting KM Options

- KMs have options that let users:
 - Turn options on or off
 - Specify or modify values used by the KM
- Options are defined in the projects tree, under the KM.
- Options are used in the KM code with the substitution method
`<%=odiRef.getOption("OptionName")%>`.
- KM options are defined on the Options tab of each KM step.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Each option can be On or Off and is defined on the Physical > Properties > Options tab of each step of the KM.

Developing Your Own KM: Guidelines

- Very few KMs are ever created. They are usually extensions or modifications of existing KMs.
- Duplicate existing steps and modify them. This prevents typos in the syntax of the `odiRef` methods.
- All mappings using the KM inherit the new behavior. Remember to make a copy of the KM if you do not want to alter existing mappings. Modify the copy, not the original.
- Modifying a KM that is already used is a very efficient way to implement modifications in the data flow and affect all the existing developments.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

One of the main guidelines when developing your own KM is to never start from the beginning. ODI provides more than 100 KMs. Take a look at these existing KMs, even if they are not written for your technology. The more examples you have, the faster you develop your own code. You can, for example, duplicate an existing KM and start enhancing it by changing its technology, or copying lines of code from another KM.

To speed up development, duplicate existing steps and modify them. This prevents typos in the syntax of the `odiRef` methods.

When developing your own KM, keep in mind that it is targeted to a particular stage of the integration process. As a reminder:

- LKMs are designed to load remote source data sets to the Staging area (into “C\$” tables).
- IKMs apply the source flow from the Staging area to the target. They start from the “C\$” tables, may transform and join them into a single “I\$” table, may call a CKM to perform data quality checks on this “I\$” table, and finally write the flow data to the target
- CKMs check data quality in a datastore or a flow table (“I\$”) against data quality rules expressed as constraints. The rejected records are stored in the error table (“E\$”).

- RKMs are in charge of extracting metadata from a metadata provider to the ODI repository by using the SNP_REV_xx temporary tables.
- JKMs are in charge of creating the Change Data Capture infrastructure.

Be aware of these common pitfalls:

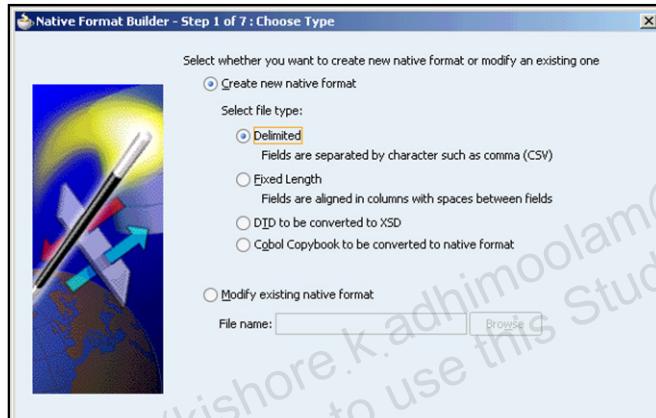
- Too many KMs: A typical project requires fewer than 5 KMs.
- Using hard-coded values including catalog or schema names in KMs: You should instead use the substitution methods `getTable()`, `getTargetTable()`, `getObjectName()`, or others as appropriate.
- Using variables in KMs: You should instead use options or FlexFields to gather information from the designer.
- Writing the KM completely in Jython or Java: You should do that if it is the only solution. SQL is often easier to read and maintain.
- Using `<%if%>` statements rather than a check box option to make code generation conditional.

Other common code writing recommendations that apply to KMs:

- The code should be correctly indented.
- The generated code should also be indented in order to be readable.
- SQL keywords such as “select,” “insert,” and so on should be in lowercase for better readability.

Complex File Technology

- Complex file formats (multiple record files) can now be integrated by using the new Complex File technology.
- This technology leverages a new built-in driver that transparently converts complex file formats into a relational structure by using a Native Schema (nXSD) description file.



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

With ODI, the Native Schema (nXSD) formats may be created manually, or with a graphical tool called the Native Format Builder Wizard (included with BPEL Process Manager). The Native Format Builder Wizard guides you through the creation of a native schema file from the formats shown in the screenshot.

A sample data file format for the selected type must already exist; you cannot create a native schema without one. You can also select to modify an existing native schema previously created with this wizard, except for those generated from a Document Type Definition (DTD) or COBOL copybook.

Quiz

Which of the following statements are true?

- a. RKM^s are used to perform customized reverse-engineering of data models for a specific technology.
- b. You can duplicate an existing KM and start enhancing it by changing its technology or copying lines of code from another KM.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Answer: a, b

Explanation: Both statements are true.

Summary

In this lesson, you should have learned how to:

- Use partitioning of datastores
- Use reusable mappings
- Use user functions
- Use substitution methods
- Modify and develop Knowledge Modules

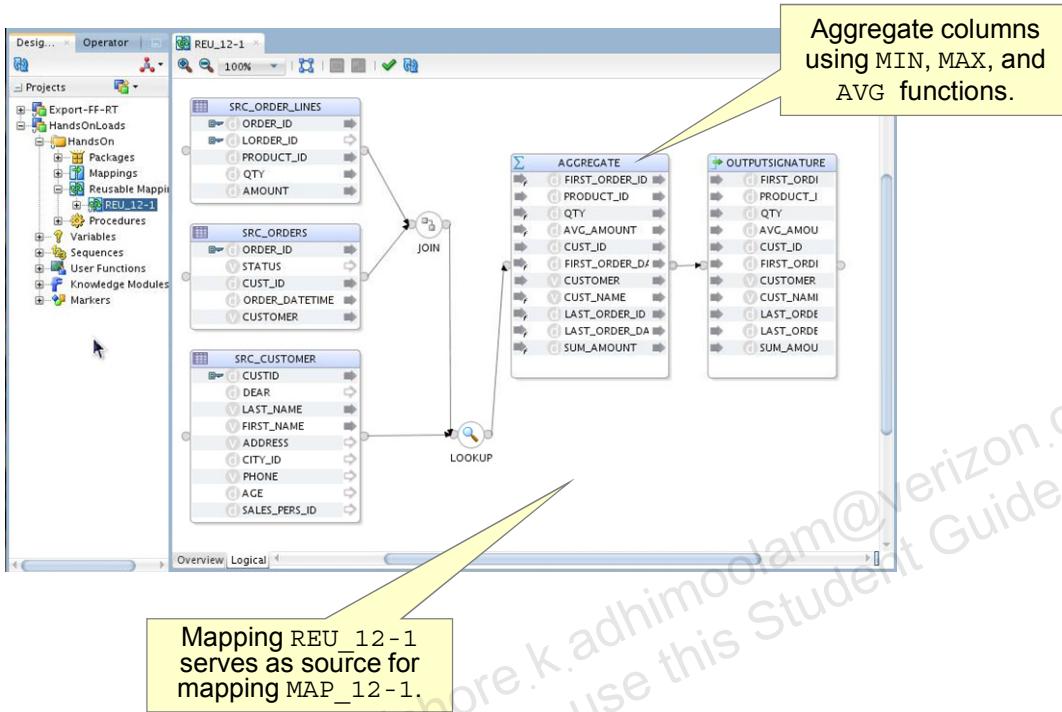
Checklist of Practice Activities

- ✓ 1-1: Logging In and Help Overview
- ✓ 2-1: Creating and Connecting to ODI Master and Work Repositories
- ✓ 3-1: Configuring Standalone Agents Using the Common Admin Model
- ✓ 4-1: Working with Topology
- ✓ 5-1: Setting Up a New ODI Project
- ✓ 6-1: Creating Models by Reverse-Engineering
- ✓ 7-1: Checking Data Quality in the Model
- ✓ 8-1: Creating ODI Mapping: Simple Transformations
- ✓ 9-1: Creating ODI Mapping: Complex Transformations
- ✓ 9-2: Creating ODI Mapping: Implementing Lookup
- ✓ 10-1: Creating ODI Mapping: Exporting a Flat File to a Relational Table
- ✓ 11-1: Using Native Sequences with ODI Mapping
- ✓ 11-2: Using Temporary Indexes
- ✓ 11-3: Using Sets (UNION) with ODI Mapping
- 12-1: **Creating and Using Reusable Mappings**
- 12-2: **Developing a New Knowledge Module**
- 13-1: Creating an ODI Procedure
- 14-1: Creating an ODI Package
- 14-2: Using ODI Packages with Variables and User Functions
- 15-1: Debugging Mappings
- 16-1: Creating and Scheduling Scenarios
- 17-1: Using Load Plans
- 18-1: Enforcing Data Quality with ODI Mappings
- 19-1: Implementing Changed Data Capture
- 20-1: Setting Up ODI Security
- 20-2: Integration with Enterprise Manager and Using ODI Console

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Practice 12-1: Creating and Using Reusable Mappings



ORACLE

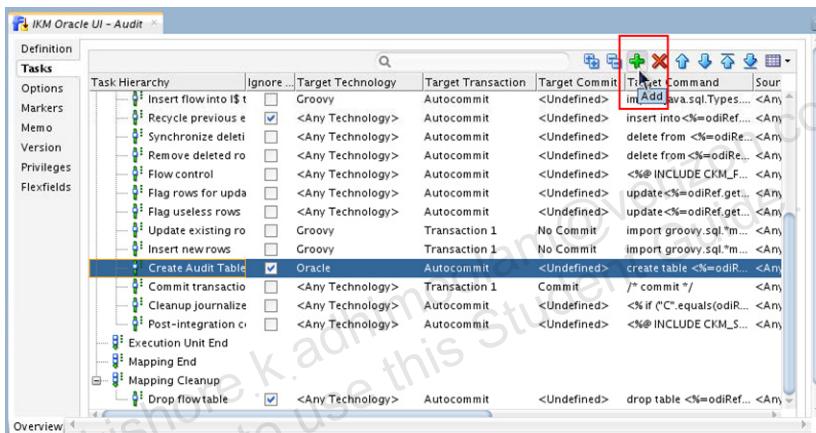
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In this practice, you perform the following steps:

1. Create a reusable mapping **REU_12-1**.
 - a. Use datastores **SRC_ORDERS** and **SRC_ORDER_LINES**, joined on **ORDER_ID**.
 - b. Use **SRC_CUSTOMER** as a lookup table.
2. Aggregate some of its columns by using the **MIN**, **MAX**, and **AVG** functions.
3. Create mapping **MAP_12-1**.
 - a. Use **REU_12-1** as the source.
 - b. Use datastore **TRG_SALES** as the target.
4. Execute **MAP_12-1** and examine the rows that are inserted into **TRG_SALES**.

Practice 12-2: Developing a New Knowledge Module

1. Duplicate the Knowledge Module, IKM SQL Incremental Update, naming the new KM IKM Oracle UI - Audit.
 - a. Insert two commands into KM: “Create Audit Table”, “Insert Audit Records”
2. Duplicate the mapping, MAP-Exp-FF-RT, naming the new mapping MAP-Exp-FF-RT-Audit.
 - b. Change the mapping’s IKM entry to use your new IKM Oracle UI - Audit.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In this practice, you duplicate an existing Knowledge Module, IKM SQL Incremental Update, naming the new Knowledge Module IKM Oracle UI - Audit. You add two commands to the Knowledge Module, “Create Audit Table” and “Insert Audit Records,” using command syntax provided in text files.

Next, you duplicate an existing mapping, MAP-Exp-FF-RT, naming the new mapping MAP-EXP-FF-RT. Change the new mapping’s IKM selection to use the new Knowledge Module you just created, IKM Oracle UI - Audit.

Finally, you execute the mapping MAP-EXP-FF-RT and examine the audit records inserted into the audit table created by your Knowledge Module.

13

Using ODI Procedures

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Describe the purpose of using an Oracle Data Integrator (ODI) procedure
- Create ODI procedures
- Add commands
- Provide options on your commands
- Execute and monitor ODI procedures



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Agenda

- **Procedures: Overview**
- Creating a Blank Procedure
- Adding Commands
- Adding Options
- Running a Procedure



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The first step is to define what procedures are and how they work. Then you will examine a few examples of the sort of procedures that you can create.

What Is a Procedure?

- A procedure is a sequence of commands executed by database engines, the operating system, or using the ODI tools.
- A procedure can have options that control its behavior.
- Procedures are reusable components that can be inserted into packages.
 - A package can contain a sequence of several procedures, each procedure being a step that itself contains a sequence of commands.



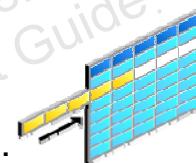
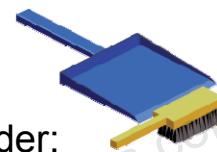
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

A procedure in ODI consists of a series of commands executed in sequence. Commands contain code that can be executed by database engines, the operating system where the Agent is running, or directly by ODI. You can also define options in the procedure to control its behavior at run time.

A very useful property of procedures is that they are reusable and can be inserted into packages. Thus, just as procedures comprise commands, a package can contain several procedures as steps.

Procedure: Examples

- Email Administrator procedure:
 1. Uses the “OdiSendMail” ODI tool to send an administrative email to a user. The email address is an option.
- Clean Environment procedure:
 1. Deletes the contents of the /temp directory by using the “OdiFileDelete” tool.
 2. Runs DELETE statements on these tables in order: CUSTOMER, CITY, REGION, and COUNTRY.
- Create and populate the RDBMS table:
 1. Run SQL statement to create an RDBMS table.
 2. Run SQL statements to populate table with rows.



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The following are some examples of the uses of procedures:

- **Email Administrator procedure:** This procedure contains a single command that calls the tool. You can then add an option to specify the email address. Thus, to change the email address of the administrator, you do not have to modify the procedure. You can even reuse the procedure in several places, each with different parameters to send emails to different people.
- You can create a procedure called Clean Environment.
- You can create an ODI procedure to create an RDBMS table and populate it with data.

Procedure: Examples

- Initialize Drive procedure:
 1. Connect to a network drive by using either a UNIX or Windows command (depending on an option).
 2. Create a /work directory on this drive.
- Email Changes procedure:
 1. Wait for 10 rows to be inserted into the INCOMING table.
 2. Transfer all data from the INCOMING table to the OUTGOING table.
 3. Dump the contents of the OUTGOING table to a text file.
 4. Email this text file to a user.

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Another example of a procedure is Initialize Drive. This consists of two commands:

1. First, a command to mount the drive is required. The text of this command depends on the current operating system. Therefore, an option is created to parameterize the procedure depending on the operating system.
2. Then, a work directory is created on the mounted drive. The ODI Tool OdiMkDir could be used here.

The last example for a procedure is called Email Changes. This procedure demonstrates the use of database connectivity by waiting for a table to change and then emailing the new rows.

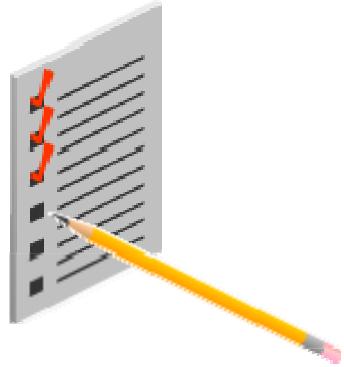
1. First, the procedure waits for the INCOMING table to be populated with 10 rows. It uses the “OdiWaitForData” tool to do this, and checks at regular intervals.
2. It transfers these rows into a new OUTGOING table by executing the required SQL.
3. It then dumps the contents of the OUTGOING table into a text file on disk. The “OdiSqlFileUnload” tool is useful here.
4. The procedure emails this text file to a user. You have seen how this is done.

Procedures can be useful when you combine several different types of commands.

Creating Procedures: Overview

To create a procedure:

1. Create a blank procedure.
2. Add some commands.
3. Add some options.
4. Run the procedure.



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

To create a procedure, you perform the following:

1. Create a blank procedure.
2. Add one or more commands that make the procedure do something useful.
3. Add options to make the procedure customizable. You do not need to define any options.
4. Save and run the procedure.

Each of these steps will be covered in greater detail in the following slides.

Agenda

- Procedures: Overview
- **Creating a Blank Procedure**
- Adding Commands
- Adding Options
- Running a Procedure



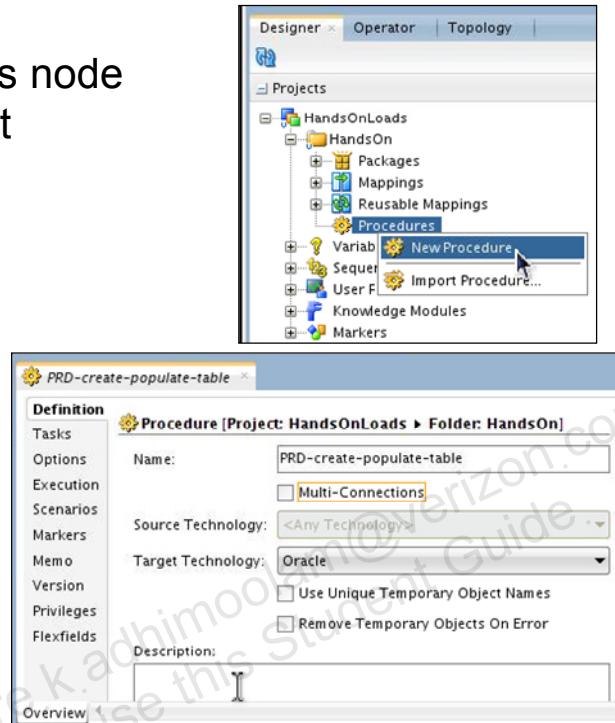
ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The first step is to create a blank procedure in ODI.

Creating a New Procedure

1. Right-click the Procedures node under a project and select New Procedure.
2. Enter:
 - The name
 - The description
3. Optionally, define the default:
 - Source technology
 - Target technology



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

It is very straightforward to create a blank procedure. Perform the following:

1. Navigate to the Project and Project Folder where you want to create the procedure. Right-click the Procedures node and select New Procedure.
2. Provide the procedure a meaningful name and a description. You may want to include any limitations the procedure has in the description.
3. A procedure can have a default target and source technology. You do not have to set these technologies. However, if you do, you will not have to set them individually for each command. You can always override these technologies for each individual step.

Note: Like mappings and packages, procedures are contained within project folders.

Agenda

- Procedures: Overview
- Creating a Blank Procedure
- **Adding Commands**
- Adding Options
- Running a Procedure



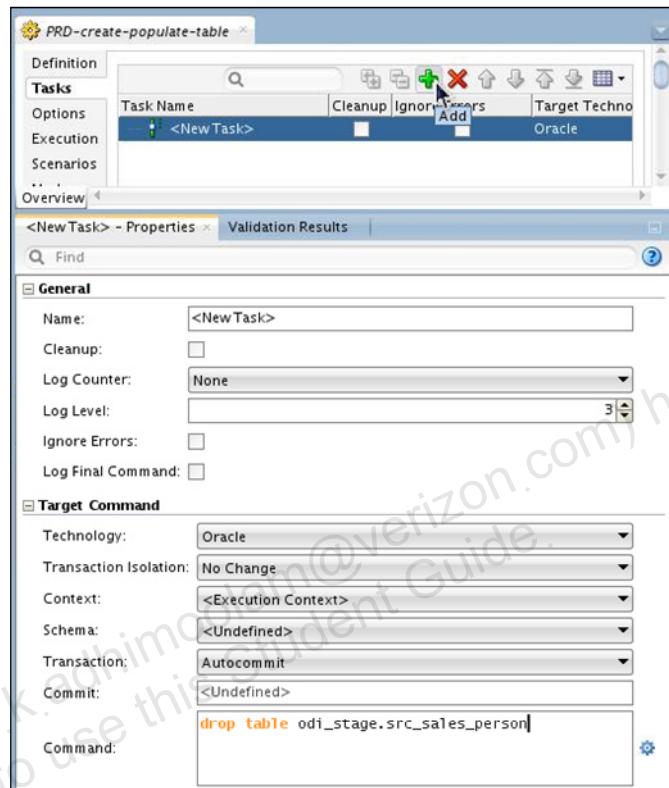
ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

With your procedure created, the next step is to add commands that make it do something useful.

Creating a Command

1. Click the Tasks tab.
2. Click Add Command (“+”).
3. Enter the task name.
4. Set Ignore Errors as appropriate.
5. For Target Command, set:
 - Technology
 - Context
 - Logical schema
 - Command code (using Expression Editor)
6. Repeat step 5 for Source Command (optional).



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

ORACLE

You can create a procedure with only one command, but you often want more than one. Remember that commands are always performed in order.

To add a command, perform the following:

1. Click the Tasks tab of the procedure window.
2. Click the Add Command (green plus) icon. A window for the command appears.
3. Enter a name for this individual command. This name appears in Operator when you execute the procedure, so you should try to make it specific and meaningful.
4. Selecting the Ignore Errors check box means that the procedure runs even if this command fails. This will be covered in detail in the next few slides.
5. Note that a command can execute code in two places (Source and Target) within the same command: There are Command on Target and Command on Source options. Click Command on Target first, and select the appropriate options:
 - Select the technology used in the command. This affects the code that you can use and how ODI generates its code.
 - The Schema and Context fields represent the logical schema and execution context for database queries. By keeping Context as *<Execution Context>*, the user can select the context at the time the procedure is executed.

- Finally, add the code for the command in the Command edit box. The best practice is to use the Expression Editor for this. Often, you want to use the ODI tools or refer to ODI objects. With the Expression Editor, you can drag the ODI tools directly into the code. It generates the correct tag for you. Note that the command must be written in the language appropriate for the technology. This can be a shell script on UNIX, some Jython code, or Oracle SQL.
6. If code is to be run on the source connection as well, click the Command on Source tab and repeat step 5. This is discussed in greater detail a few slides later.

You can now repeat steps 2 through 6 to create the other commands in the sequence.

Note: Source commands are explained in more detail in the section “Why Use a Source Command?”

How to Leverage Existing Code

You may wonder, “How can I leverage existing code that I have written, say, in PL/SQL?”

Here is how you can execute a PL/SQL procedure from an ODI procedure:

1. Create an ODI procedure.
2. Set Command on Target—select Technology as Oracle; select Schema.
3. Add a command.

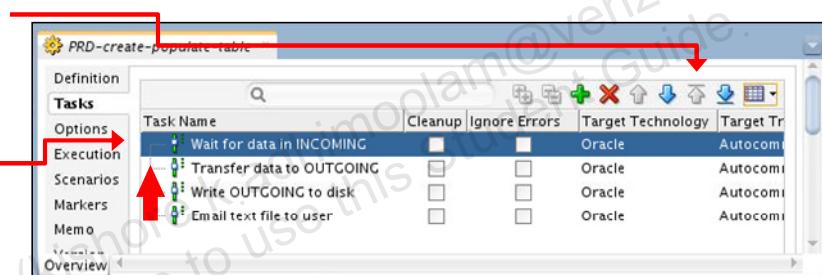
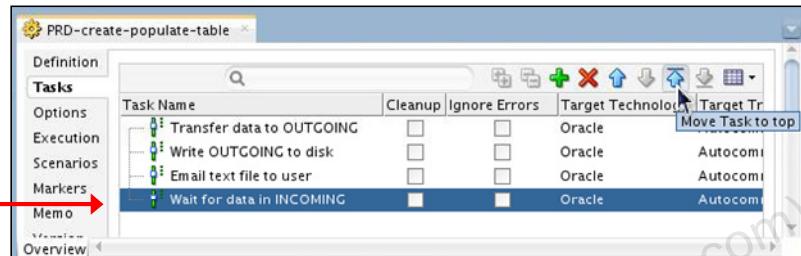
```
BEGIN  
Procedurename() ;  
END;
```

How do you pass parameters to a command?

```
BEGIN  
Procedurename(parametervalue1 , parametervalue2 ) ;  
END;
```

Arranging Tasks in Order

- The Details tab shows the tasks of your procedure.
- Tasks are executed from top to bottom.
 - In this example, “Wait for data in INCOMING” is executed last.
 - You want it to be executed first.
- To rearrange tasks: use up, down, top, bottom arrows.
- Now the procedure waits for data before attempting the transfer.



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

After creating your tasks, you need to place them in the correct order. You use the Details tab to do this. Tasks are always executed from top to bottom. Some tasks may be skipped, but the order of execution never changes.

Thus, in this example, “Transfer data to OUTGOING” is executed first, whereas “Wait for data in INCOMING” is executed last. This is incorrect. You want to wait for data in the INCOMING table before doing anything else. Thus, you need to move it to the top of the list.

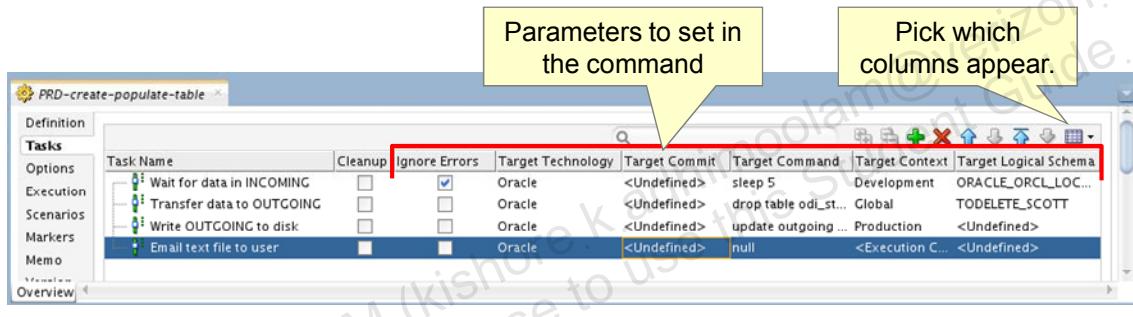
To change the order of tasks, use the up and down arrows. In this case, you click the up-arrow button three times. As you see, the waiting step is now the first step. Thus, the procedure waits for data before attempting to transfer the data from the INCOMING table to the OUTGOING table.

You should always double-check the order of your tasks before executing your procedure.

Which Parameters Should Be Set?

The following parameters should be set in the command:

- Technology: Overrides the default for the procedure
- Logical Schema: For DBMS technologies (Jython, OS, and ODI tools do not require a schema)
- Context: If you want to *ignore* the execution context
- Ignore Errors: If the command must not stop the procedure. A warning is issued only if the command fails.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The Technology parameter needs to be set only if no technology is set for the whole procedure, or if this command uses a different technology from the others.

You set Logical Schema only if the technology of the command is based on a DBMS. For example, if the command is executed by Jython, the operating system, or by ODI, no schema needs to be set.

Context should be set only if you want to always run this task in a certain context. Generally, you should leave it blank, so that the execution context is used instead. Whenever you execute a procedure, ODI asks you which context to run it in. That is what is meant by the execution context.

When Ignore Errors is set, the procedure continues even if an error is encountered in this step. This is useful for commands such as dropping a table, when the most likely reason for failure is that the table does not exist. In that case, it does not matter whether the command fails.

Valid Types of Commands

Examples of the types of commands that can be used in ODI procedures:

SQL Statement	Executed on any DBMS technology. DELETE, INSERT, and SELECT statements.
OS Commands	Executed on the OS technology. In OS-specific syntax, using shell commands or binary programs.
Oracle Data Integrator Tools	Executed on the Sunopsis API technology. Any ODI tool command call.
Jython Programs	Executed on the Jython technology. Can call Java code or precompiled objects.
Scripting Languages	Using Jython, Groovy, NetRexx, and Java BeanShell



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You should note that commands can be written in a range of different syntaxes, depending on the language of the technology that runs them.

For example, any statement that runs on a DBMS technology is normally written in SQL. You use statements such as `DELETE`, `INSERT`, and so on in the text for your command. The database functions that are available depend on the exact database engine you call.

Operating system commands depend similarly on the operating system. On a UNIX system, you might use commands such as `cat`, `ls`, or `mkdir`, depending on the shell. Similarly, you can embed the names of executable programs directly in the text to cause these commands to be executed.

ODI, as mentioned, has a large number of tools available for use. These are executed by using the simple ODI macro language. Some examples of these tools are discussed later.

You can also call programs coded in Jython by using the ODI Extensibility Engine, a built-in interpreter. In that case, you use Jython as the technology. The Extensibility Engine also supports other programming languages, such as Groovy, JavaScript, NetRexx, or Java BeanShell.

More Elements

In addition, you have access to the following ODI-specific elements that can be used within the commands:

Variables	They may be specified either in substitution mode #<variable>, or in bind mode :<variable>.
Sequences	They may be specified either in substitution mode #<sequence>, or in bind mode :<sequence>.
User Functions	Used in the same way as DBMS functions; replaced at the time of code generation by their implementation



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You can also gain access to ODI variables, sequences, and user functions within commands.

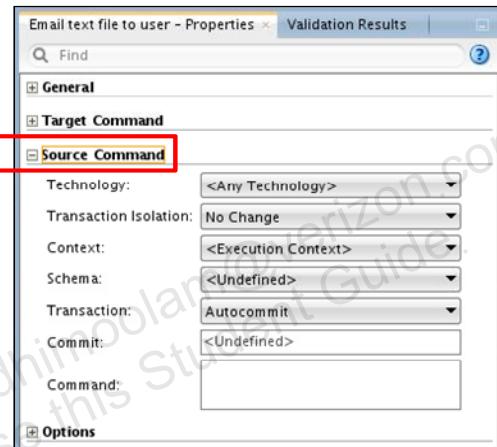
For example, to refer to variables, use the normal variable syntax—that is, the name of the variable preceded by a colon or the hash or pound sign.

Sequences are also used in the normal syntax—that is, the name of the sequence prefixed by either hash or colon.

You can similarly use the user functions that you defined. Use them as you would anywhere else, with the same syntax as DBMS functions. When the code is generated, they are replaced by the appropriate implementation for the technology.

Why Use a Source Command?

- The command sent to the source should return a result set, which is manipulated by the default command.
- Example: Transferring data from one table to another
 - Source command: SELECT statement
 - Target command: INSERT statement with source columns in bind mode



ORACLE

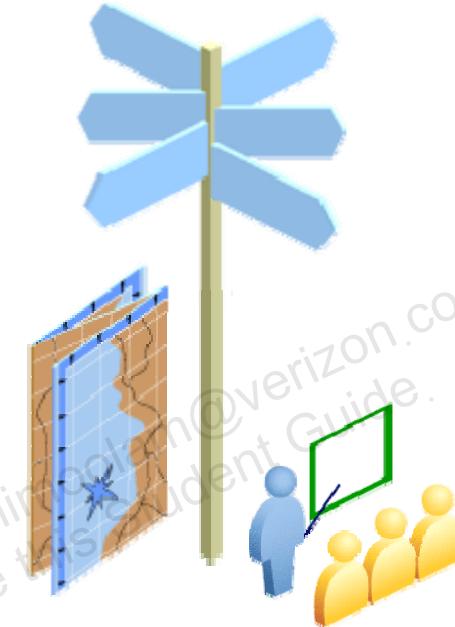
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You now look at the role of the source command in greater detail. This command is executed on the source connection for the command and should normally be a DBMS technology. It should return a result set, which is then passed through the agent to the target connection. The target connection is often referred to as the default connection, so commands executed on it are called default commands.

The simplest, most common example of using a source command involves transferring data from one table to another. This table-to-table transfer is effectively a simple mapping. Here, the source command is a SELECT statement that returns a result set. The default command is an INSERT statement. It uses the names of source columns in bind mode to insert the values. To see an example of this, look at the code for a “SQL to SQL” IKM.

Agenda

- Procedures: Overview
- Creating a Blank Procedure
- Adding Commands
- **Adding Options**
- Running a Procedure



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Now you have created a procedure that performs a series of tasks. However, it always performs the same tasks with the same parameters. To make the procedure more flexible, you can add options to it. These options function as parameters to the procedure. This means that you can use the same procedure in different environments with different parameters, without having to change any code.

Types of Options

- Options act like procedure parameters:
 - Check box, value, or text
- Options are used to control procedures:
 - Commands may or may not be executed, depending on the values of the check box options.
 - Value, text options, or check box can be used within the command to parameterize the code.
- Options have a default value. The value can also be specified when the procedure is used.

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Procedure options function as parameters. They always have a particular type, such as a check box, whose value can be either true or false.

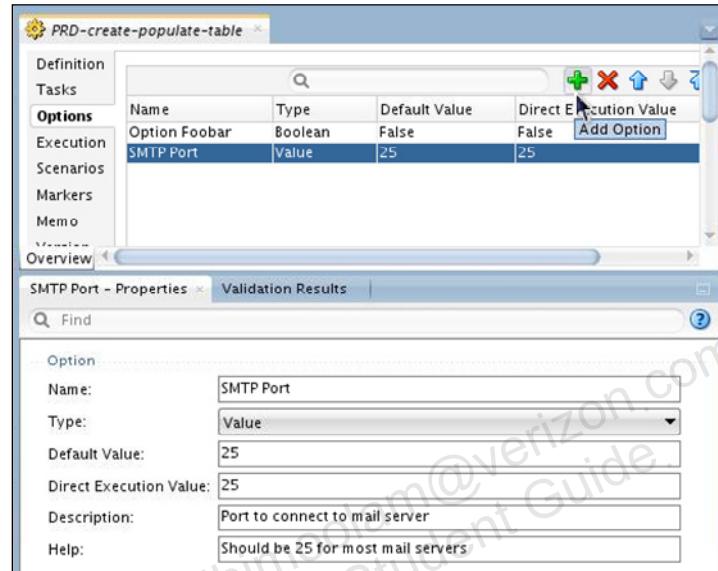
Options can be used in two different ways to control the way procedures are executed.

- Check box options serve to skip individual commands in the procedure. For example, you can have an option that, if false, completely skips a command, which sends an informative email.
- Value or text options are used in the text of commands to influence the behavior of the procedure at a smaller level. For example, the subject of the email can be a value option. This is then passed to the OdiSendMail tool.

In both cases, you define a default value for the option. This is the value that is used if no other value is specified when the procedure is used. When you call the procedure in a package, you can override any of the default values to customize the procedure for that particular context.

Creating a New Option

1. Select the Options tab.
2. Enter:
 - Name
 - Description
 - Help text
3. Select:
 - Type
 - Default value
4. Optional:
Re-order the displayed position up/down/top/bottom.



ORACLE®

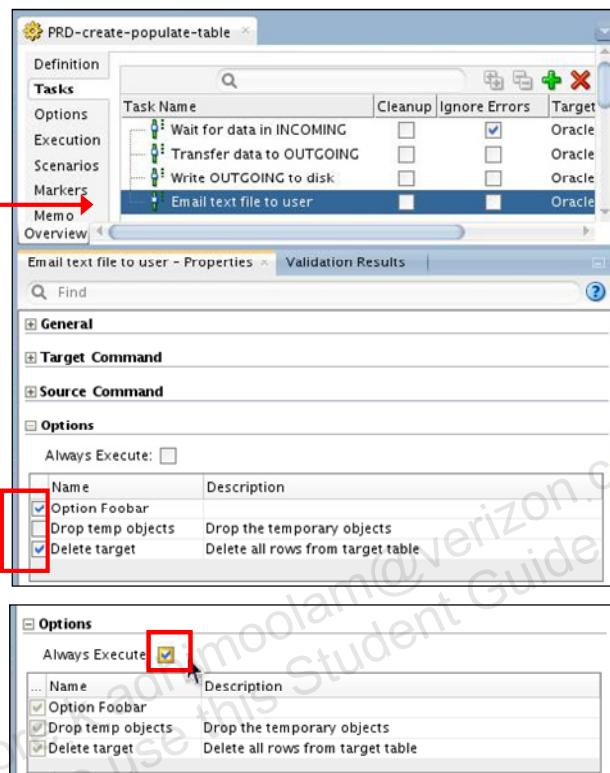
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Adding new options to a procedure:

1. Select the Options tab.
2. In Properties, enter the name as it appears in the list of options when you execute the procedure. Enter a description. The value that you enter in the Description field is displayed when you select the options that trigger a command in a procedure. It should be a short reminder of what the option does. Complete the Help field only when creating a Knowledge Module. It enables you to provide a longer description of the option, including any side effects or special notes.
3. Select the option type. This can be **Boolean** for an option that determines whether a given step is executed or not, **Value** for a numeric value, or **Text** for a text string. Complete the Default Value field. The default value specified for an option is the value that is used if you do not specifically set it in a package.
4. You can change the position within the list order by using the up/down/top/bottom blue arrows.

Making a Command Optional

1. Open a command and click the Options tab.
2. Select the option box if it should trigger the command execution.
3. If the command should run independently of any options, select the Always Execute check box.



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Check box options can be used to render certain commands optional. To do this, you perform the following:

1. Select the command you want to edit by double-clicking it in the Details window of your procedure. Click the Options tab.
2. Here, you see the list of check box options that trigger the command. That is, any option that is selected here causes this command to be executed if that option is enabled when the procedure is executed. Thus, to make one option determine whether the command executes, deselect Always Execute and select just that option.
3. Alternatively, if you want the command to be always executed, select the Always Execute check box. This has the same effect as manually selecting all options. However, if you add more options later, they are automatically selected if the Always Execute check box is selected.

Using an Option Value in a Command

In the command code, add the following code:

```
<%=odiRef.getOption("option_name")%>
```

- This code is replaced by the value of the option at run time.
 - Use quotation marks as appropriate.
- The value of the option is specified when the procedure is used within a package. Otherwise, the default value is used.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

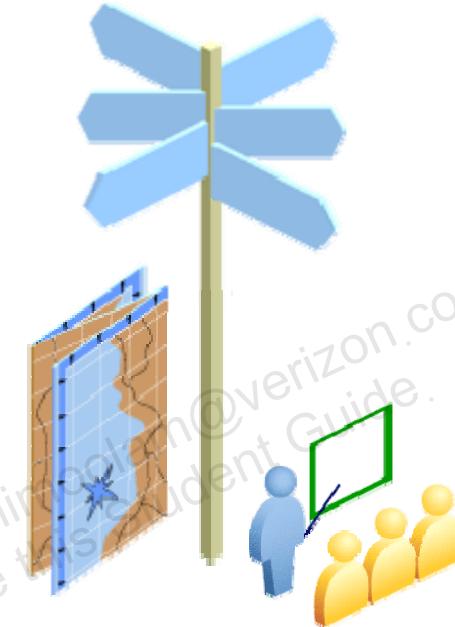
The other use of options is to change the code that is executed for the command. To do this, you use the `getOption` method defined on the `odiRef` object. The Expression Editor helps you get the syntax right. Remember to use the equal sign before “`odiRef`.”

When the procedure is executed, this code is directly substituted with the value of the option. You should therefore use the appropriate quotation marks with the substitution to make the substituted result sensible for your technology.

Procedures are usually executed from within packages. The value of the option is thus specified in a procedure step in a package. If the option is not specified, the default value is used instead.

Agenda

- Procedures: Overview
- Creating a Blank Procedure
- Adding Commands
- Adding Options
- **Running a Procedure**



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

After creating commands and making those commands use options, you now want to run the procedure. This can be done manually or through packages.

Procedures can also run within Load Plans. Load plans are covered in a subsequent lesson.

Procedure Execution

- A procedure can be executed manually for testing.
 - Default option values are used.
- Procedures are usually run from a package.
 - Add a procedure step by using drag-and-drop
 - Option values can be overridden.



ORACLE

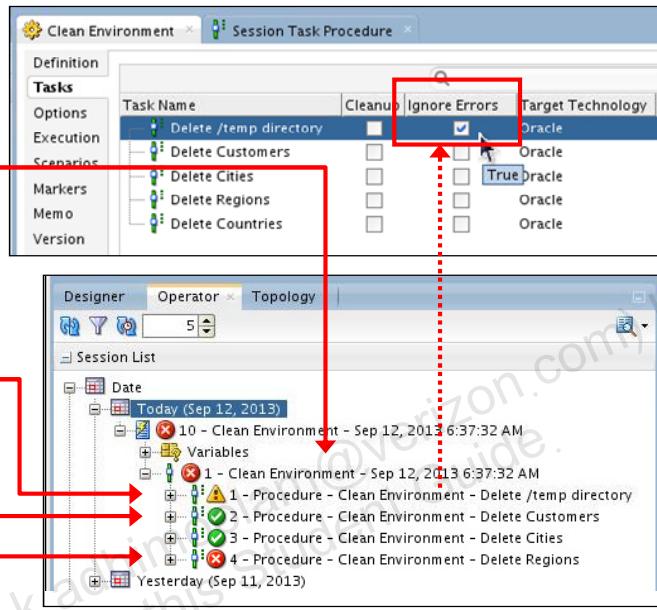
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

One of the most common ways to execute an ODI procedure is to run it from within a package. (This is discussed in detail in the next lesson). However, you can also execute a procedure manually for testing. You click the Execute button in the procedure window to do this. Default values for all options are used.

You can, however, override the default values for the options. Click the procedure step, and then click the Options tab from the Properties pane. Here, you can select the values of options that will be used when the procedure step is executed.

Using the Operator Navigator to View Results

- The procedure is executed as a session with one step.
- One task for each command:
 - Warning: Error ignored
 - Tasks completed successfully
 - Error that was not ignored
 - Tasks 5-n do not even appear



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

To follow the execution of the procedure, use the Operator Navigator. When you run a procedure manually, ODI creates a session for the procedure as a whole. The state of the session indicates whether the procedure completed successfully or not.

The session for the procedure is composed of a single step. If the procedure is part of a package, the session corresponds to the package and the step corresponds to the procedure.

Each command in the procedure then becomes a task executed by the agent. If a task generates an error, it is displayed as either a warning or an error. In your first example, the "Delete /temp directory" task has failed. However, the Ignore Errors option is enabled. Therefore, it is displayed as a warning, and execution continues.

The next two tasks have completed successfully. This is displayed by the success icons. However, the fourth task has also failed. This time, the Ignore Errors option was not enabled. Execution stops, and an error icon is displayed. Because execution has stopped, the fifth task never attempted to be run.

Note: A procedure used in a package is referenced by the package, not copied. Changes to the original procedure also apply to the package.

Quiz

Which of the following parameters must always be set in the command?

- a. Technology
- b. Logical Schema
- c. Context
- d. None of the above



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Answer: d

Explanation: The Technology parameter needs to be set only if no technology is set for the whole procedure, or if this command uses a different technology from the others. You set Logical Schema only if the technology of the command is based on a DBMS. Context should be set only if you want to always run this step in a certain context.

Quiz

You should always set the Technology parameter when creating a procedure step.

- a. True
- b. False

 ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Answer: b

Explanation: The Technology parameter needs to be set only if no technology is set for the whole procedure, or if this command uses a different technology from the others.

Summary

In this lesson, you should have learned how to:

- Describe the purpose of using an ODI procedure
- Create ODI procedures
- Add commands
- Provide options on your commands
- Execute and monitor ODI procedures

Checklist of Practice Activities

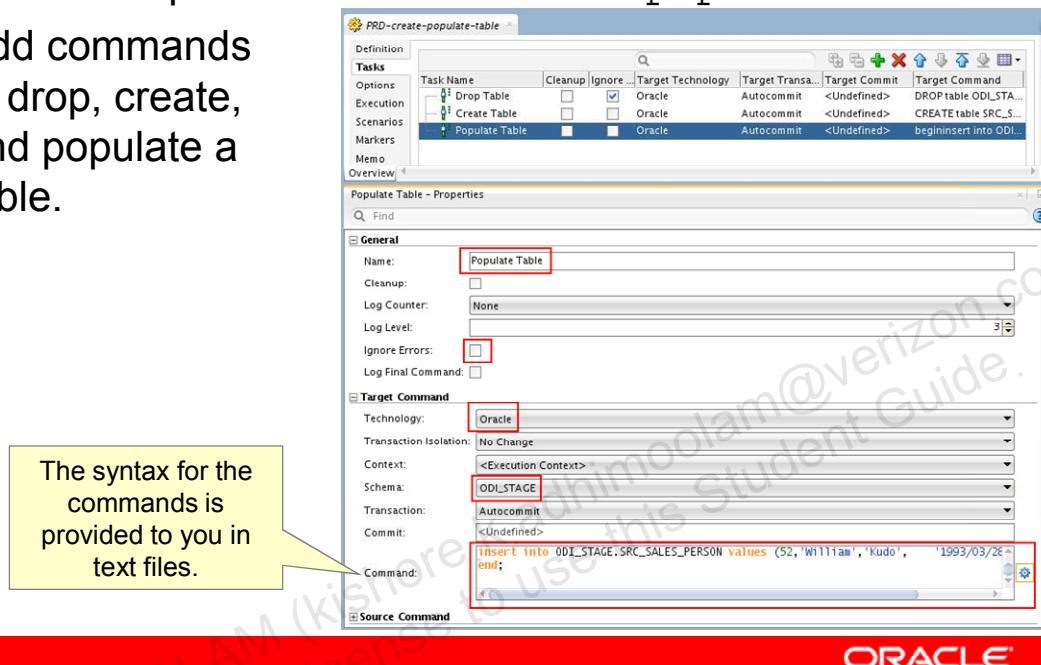
- ✓ 1-1: Logging In and Help Overview
- ✓ 2-1: Creating and Connecting to ODI Master and Work Repositories
- ✓ 3-1: Configuring Standalone Agents Using the Common Admin Model
- ✓ 4-1: Working with Topology
- ✓ 5-1: Setting Up a New ODI Project
- ✓ 6-1: Creating Models by Reverse-Engineering
- ✓ 7-1: Checking Data Quality in the Model
- ✓ 8-1: Creating ODI Mapping: Simple Transformations
- ✓ 9-1: Creating ODI Mapping: Complex Transformations
- ✓ 9-2: Creating ODI Mapping: Implementing Lookup
- ✓ 10-1: Creating ODI Mapping: Exporting a Flat File to a Relational Table
- ✓ 11-1: Using Native Sequences with ODI Mapping
- ✓ 11-2: Using Temporary Indexes
- ✓ 11-3: Using Sets with ODI Mapping
- ✓ 12-1: Creating and Using Reusable Mappings
- ✓ 12-2: Developing a New Knowledge Module
- **13-1: Creating an ODI Procedure**
- 14-1: Creating an ODI Package
- 14-2: Using ODI Packages with Variables and User Functions
- 15-1: Debugging Mappings
- 16-1: Creating and Scheduling Scenarios
- 17-1: Using Load Plans
- 18-1: Enforcing Data Quality with ODI Mappings
- 19-1: Implementing Changed Data Capture
- 20-1: Setting Up ODI Security
- 20-2: Integration with Enterprise Manager and Using ODI Console

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Practice 13-1: Creating an ODI Procedure

1. Create the project Procedure-CRT-TBL.
2. Create the procedure PRD-create-populate-table.
3. Add commands to drop, create, and populate a table.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In this practice, you use the same RDBMS schema, ODI data server, and ODI physical schema—all named ODI_STAGE—that you created in Practice 10-1. In the predefined project Procedure-CRT-TBL, create the procedure PRD-create-populate-table.

Next, you add commands to drop, create, and populate a table, using syntax provided in text files.

Finally, you execute the procedure and verify that the table was created and populated with records.

Note: A reset script is available from your instructor, to bring your ODI project and database to the equivalent of having completed Practice 10-1. Practice 13-1 and subsequent practices require completion of Practice 10-1. Speak to your instructor if you want to use this reset script.

14

Using ODI Packages

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Use Oracle Data Integrator (ODI) packages to create a complete workflow
- Create package steps of different types
- Execute a package
- Create package steps based on procedures, variables, tools, and models
- Define complex workflows in ODI packages, involving branches and loops



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In this lesson, you learn how to create packages that can run mappings and other tools automatically. By now, you should have a good understanding of the ODI mappings. These are the most important objects that will be used in a package.

In this lesson, you focus on using these mappings to create a complete data integration job. An ODI package contains all the necessary steps to run a complete job. You should also learn how to create several kinds of steps and then how to run the package.

Agenda

- **Packages: Overview**
 - 1. Creating and Naming a Package
 - 2. Adding Steps to the Package
 - 3. Arranging Package Steps in a Sequence
- Executing a Package
- Review of Package Steps
- Model, Submodel, and Datastore Steps
- Variable Steps
- Controlling the Execution Path

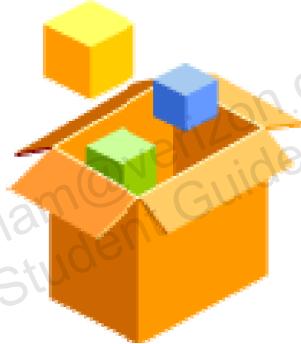


ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

What Is a Package?

- A package is an organized sequence of steps that makes up a workflow.
- Each package step performs a part of the entire job and these steps are organized using the Package Editor.



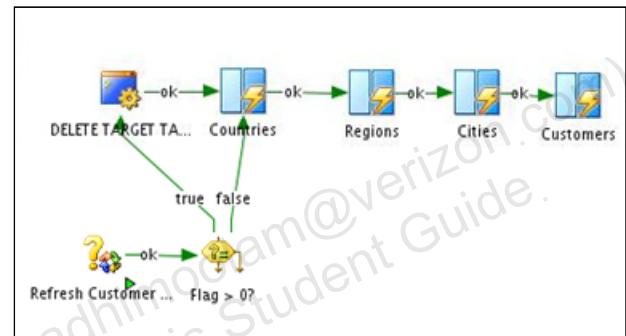
ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

An ODI package defines a complete data integration job. A job is made up of many smaller steps. Normally, you design these steps first, such as the ODI procedures and mappings. Other steps are created in the package.

Creating a Package

1. Create and name a blank package.
2. Create the steps that make up the package:
 - Drag mappings from the Projects view onto the Diagram tab.
 - Insert ODI tools from the toolbox.
3. Arrange the steps in order:
 - Define the first step.
 - Define the success path.
 - Set up error handling.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Other steps can be used to create a package, but in this lesson you focus on just two types of steps: mappings and tools.

The basic process for creating a package is as follows:

1. You make a blank package and name it.
2. Then, you load the package with the steps that it later executes. You can drag mappings from the Project view onto the package's Diagram tab. This creates a link rather than a copy to the mapping. Thus, you can keep working on your mapping, and your changes update the package. You can also add ODI tools into the package. Tools do useful things, such as sending email, copying files, or waiting for the data to arrive.
3. Finally, you arrange the steps in a meaningful order. You begin by defining the first step to be executed. Then, you tell ODI what to do next when the first step succeeds. You can also tell ODI what to do if any particular step in the package fails. Thus, you can link complex sequences of operations with error handling or error recovery.

These three steps are covered in greater detail in the following slides.

Agenda

- **Packages: Overview**
 - **1. Creating and Naming a Package**
 - 2. Adding Steps to the Package
 - 3. Arranging Package Steps in a Sequence
- Executing a Package
- Review of Package Steps
- Model, Submodel, and Datastore Steps
- Variable Steps
- Controlling the Execution Path



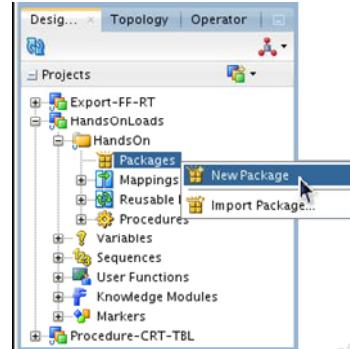
ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The first step is to create a package and give it a name.

Creating and Naming a Package

1. Navigate to the project and folder where you want to create your package.
2. Right-click the Packages node and select New Package.
3. Enter a name.
4. Enter a description.
5. Click the Diagram tab.



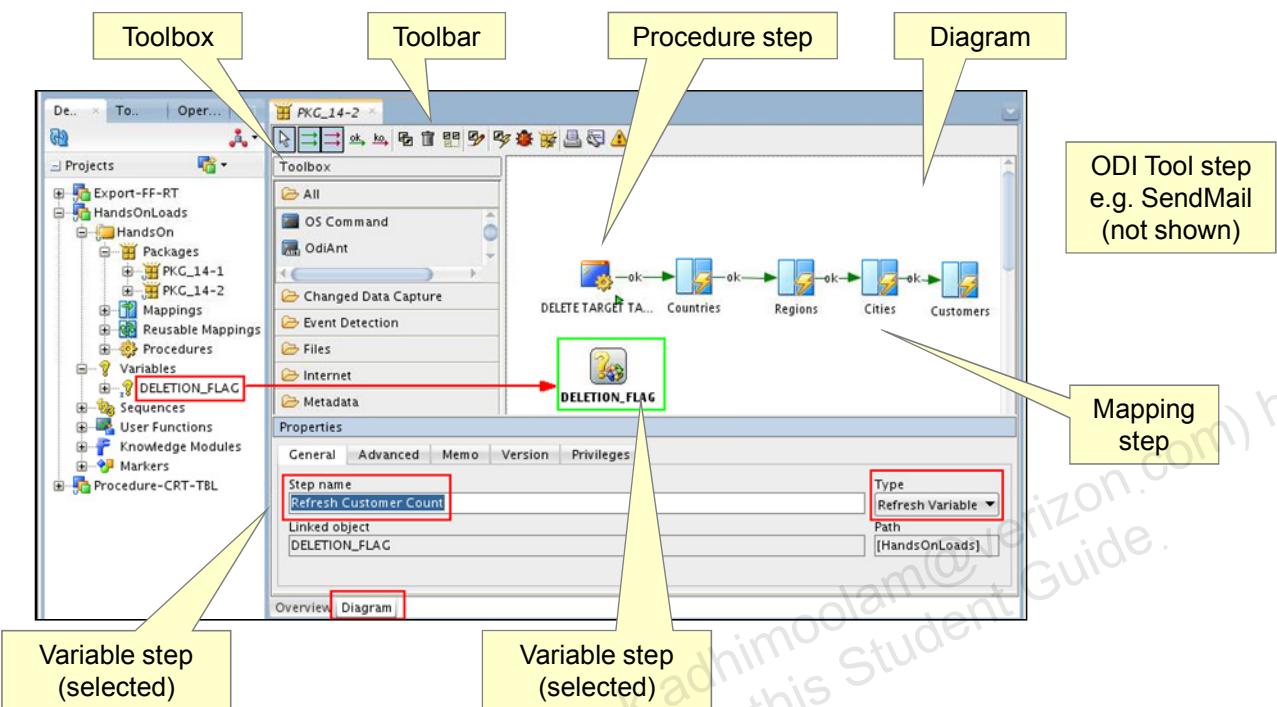
ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

To create a blank package, perform the following:

1. Locate the project and folder in the Projects view where you want to create the package. A package always belongs to a project, but you can use mappings from other projects in the package.
2. Right-click the Packages node and select New Package.
3. Provide a meaningful name to your package in the Name field.
4. Describe what the package does, by entering details in the Description field. This description appears in the package documentation and is useful when performing changes or maintenance.
5. To begin adding steps, click the Diagram tab.

Package Diagram



ODI Tool step
e.g. SendMail
(not shown)

Mapping step

Variable step
(selected)

Variable step
(selected)

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

This is how the Diagram tab of a package appears.

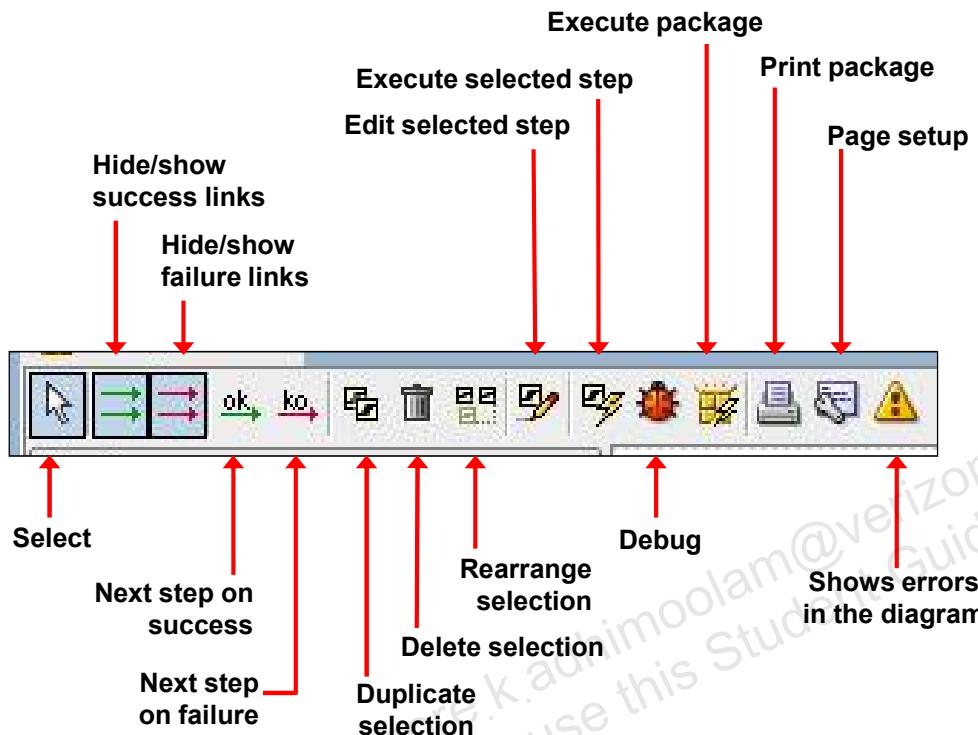
The Diagram represents the steps of the package as icons, with green links representing the success path between them. The currently selected step is highlighted by a box around it. The first step is marked with a green arrow.

At the bottom, you see the properties of the currently selected step. You can rename the step here. Different properties are displayed for different types of steps.

In Toolbox at the left, all the tools that you can use are categorized. You can also access the tools without categorization under All. Tools are covered in the next slide.

On the toolbar are buttons for sequencing packages. This is examined more closely in the next slide. You can move or undock the Toolbox and the Properties pane to improve your workspace.

Package Diagram Toolbar



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The following three tools are the most important. You use them frequently.

- “**Next step on success**”: You can draw a link from one step to the next with this. If the first step executes successfully, ODI executes the next step in the chain. For example, the first step might populate your ORDERS table, whereas the second can write the total number of orders to a file.
- “**Next step on failure**”: This tool does the opposite. You can draw a link from one step to another that should be run if the first fails. For example, you can create a failure link from the second step to an email step. If ODI is unable to write to the file, it sends an email informing you about the problem.

Linking steps are covered in detail in the slides that follow.

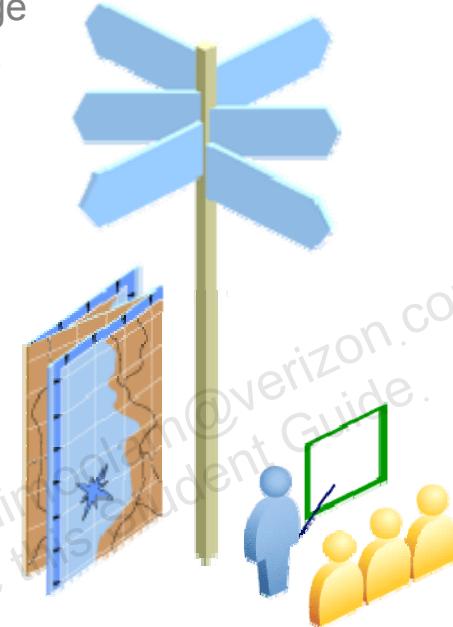
- “**Select**”: You can use this tool to stop drawing links and to lay out the diagram more clearly.

Note: The toolbar buttons are “sticky” (persistent) so you must select “Select” when you no longer want to use the last selected option.

- “**Hide/show success links**” and “**Hide/show failure links**” enable you to see certain aspects of the diagram more clearly by hiding links.
- “**Duplicate selection**” and “Delete selection” are self-explanatory and can also be accessed by right-clicking a step.
- “**Rearrange selection**” rearranges the diagram completely and automatically according to a predetermined algorithm.
- “**Execute selected step**” is useful to see the result of just one step. “Execute package” runs all steps starting from the first step. You can also use the Execute button in the lower-right corner of the window.
- “**Debug**” enters the step-by-step debugger (covered in the next lesson).
- “**Page setup**” and “**Print package**” enable you to configure and print a comprehensive report of all the steps in your package.
- Finally, “**Show errors in the diagram**” checks your diagram for errors, such as unlinked steps.

Agenda

- **Packages: Overview**
 - 1. Creating and Naming a Package
 - **2. Adding Steps to the Package**
 - 3. Arranging Package Steps in a Sequence
- Executing a Package
- Review of Package Steps
- Model, Submodel, and Datastore Steps
- Variable Steps
- Controlling the Execution Path



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

To make the package do something, you add steps on the Diagram tab.

Package Steps

- The three most common package steps are:
 - Mapping steps
 - ODI tool steps
 - Procedure steps
- Steps are created and sequenced in the package diagram.



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

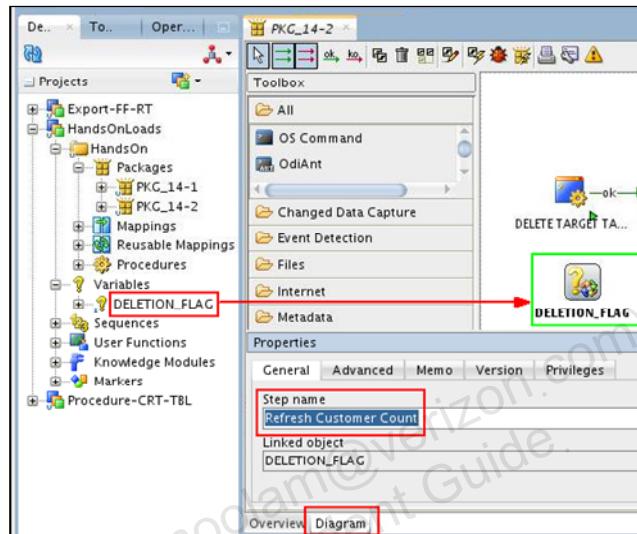
Steps

The most useful package steps for most situations are either mappings or ODI tools. Mappings, as you know by now, transfer data from one or more source datastores into a target datastore. ODI tools perform a much wider variety of tasks, including sending email or waiting for data. There are other steps, such as updating or testing variables, but they are not covered here.

Steps are created by dragging objects onto the package diagram, which is found on the Diagram tab. They are then sequenced by creating links from one step to the next.

Creating a Package Step

1. Expand the project and folder containing the mapping. Expand the Mappings node.
2. Drag a mapping, ODI tool, or procedure onto the package diagram.
 - The new step appears.
3. Optionally, change the step name in the Properties panel.



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

To create a package step, perform the following:

1. Find the mapping, ODI tool, or procedure that you want to add. To add a procedure, expand the project and folder nodes in the Projects view. Then expand the Procedures node.
2. Drag the procedure onto the package diagram. You see an icon representing the procedure step in the package.
3. You may want to change the name of the procedure step.

Note

- Mappings are reusable. When you create a mapping, you can use it several times in the same package. You can even copy it into several different packages simultaneously.
- You can use mappings from other projects in your package. However, the best practice is to avoid doing this because it makes it difficult to keep your project organized.
- You should save your work frequently when working on packages.

What Is an ODI Tool?

- ODI tools are macros that provide useful functions to handle files, send emails, use web services, and so on.
- Tools can be used as steps in packages.



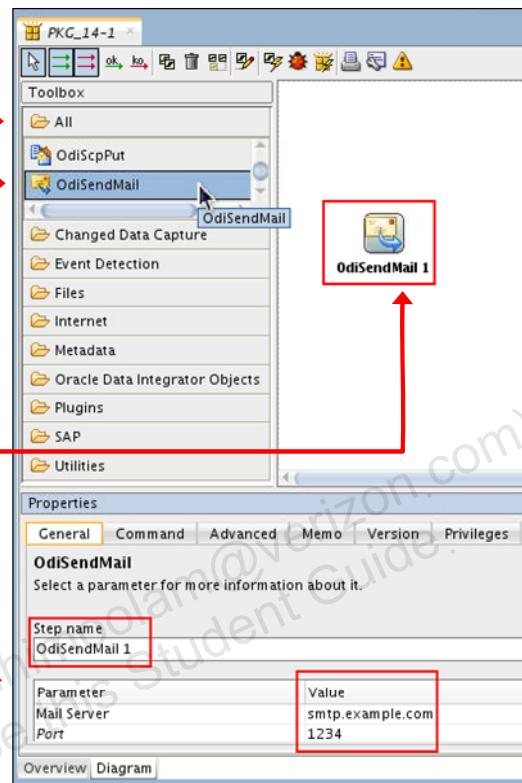
ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

An ODI tool can be used in packages to create completely automated tasks.

Creating an ODI Tool Step

1. In Toolbox, expand the group containing the tool that you want to add.
2. Click the tool.
3. Click the diagram.
 - A step named after the tool appears.
4. Change the step name in the Properties panel.
5. Set the tool's properties.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You can create a tool step as follows:

1. Find the tool that you want to add in Toolbox. Tools are arranged in different categories, such as Internet, Event Detection, and so on. When you know the name of a tool, it may be quicker to look directly in the “All” list, where all the tools are listed alphabetically.
2. Click the tool that you want to add. Each tool has a different icon.
3. Click in the diagram. The tool now appears as a step at that location. Tool steps are shown as the corresponding icons.
4. (Optional) Change the step name in the Properties panel. This helps distinguish between, for example, an email step that reports an error and an email step that sends the results of an operation. Otherwise, they appear as OdiSendMail1 and OdiReadMail2.
5. Configure the properties of the tool in the Properties panel. For example, for an OdiSendMail step, you must set the name of the mail server, the text to send, and so on.

Note: When you optionally change the name of a step (tool step or otherwise) you can see the changed name by pressing the Enter key.

Tool Steps: Best Practices



- Tool steps cannot be reused, but can be *duplicated*.
- To create a sequence of reusable tool commands, you must create a procedure.



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Unlike mapping steps, tool steps are simple commands that call the macros of ODI tools. You cannot reuse a tool step, but you can duplicate it.

To create reusable ODI tool commands, you must create a procedure. You can place several tool commands inside a single procedure. Then, you can reuse the same procedure in many different packages. You can, of course, duplicate steps within the same package if you want to perform the same action multiple times. But, they are copies of each other and so are maintained separately.

Examples of reusable tool step procedures include “end on failure” or “end on success” procedures that incorporate the setting of a status, sending of an email, and so on.

Agenda

- **Packages: Overview**
 - 1. Creating and Naming a Package
 - 2. Adding Steps to the Package
 - **3. Arranging Package Steps in a Sequence**
- Executing a Package
- Review of Package Steps
- Model, Submodel, and Datastore Steps
- Variable Steps
- Controlling the Execution Path



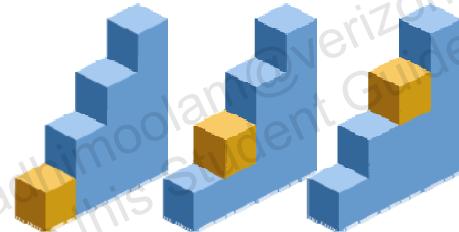
ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

After creating the package and placing steps into it, you arrange the steps in a sequence.

Sequencing Steps

- The first step must be defined.
 - Right-click and select First Step.
- After each step, the flow splits in two directions:
 - Success: **ok** (return code 0)
 - Failure: **ko** (return code not 0)



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

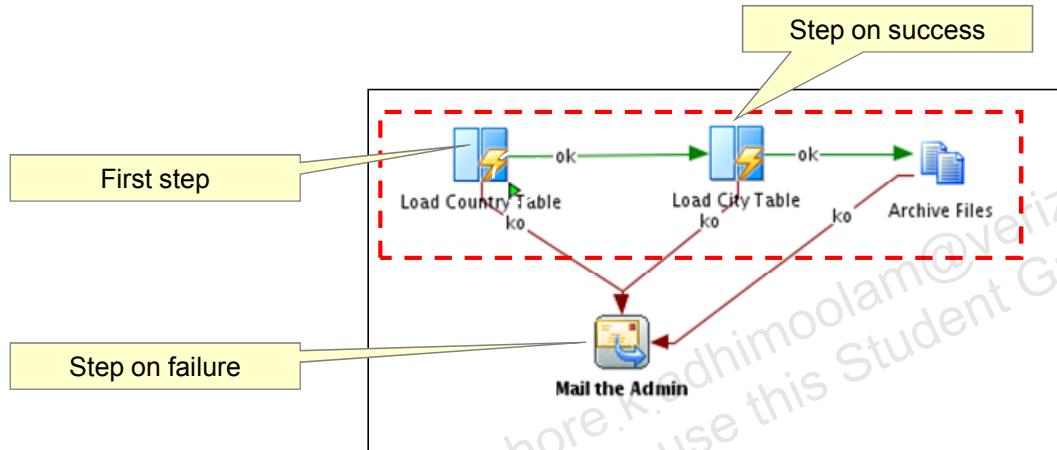
Every package must have a first step. This is where the execution of a package always begins. To define the first step, select a step, right-click and select First Step.

After that, the sequence of steps splits in two directions: If the step executes successfully, the execution follows the green “ok” link. If the step fails, it follows the red “ko” link. Success is defined by the step returning a code 0. Any return code other than 0 is treated as a failure.

Note: The first step you drag into the package diagram is always (initially) the First Step, unless you designate another step as the first one to execute. If you delete the step marked First Step, be sure to designate another step as the first one to execute. ODI displays an error message if no step is labeled First Step.

A Simple Package

- This package executes two mappings, and then archives some files.
- If any one of the three steps fails, an email is sent to the administrator.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

This is an example of a simple package. When it is executed, the first step, “Load Country Table,” is executed. If it fails, the step “Mail the Admin” is completed. You can see this by the red “**ko**” path (“**ok**” backwards), which indicates the path to follow after a step fails. The package then terminates. If it succeeds, however, the execution proceeds to “Load City Table.” This is shown by the green “**ok**” path. Again, if it fails, ODI sends an email to the administrator. Lastly, if the first two steps succeed, the “Archive Files” step is completed. This also sends an email if it fails.

Note

- A package can be designed to end at many different points depending on which steps succeed or fail. But there must always be one starting step. In this example, the package may end after archiving some files, or it may end after sending an email to the administrator. But it always starts by loading the Country table.
- If the email send is successful, the package status is “Success” even if a package step fails. An alternative embellishment is to add another step (like a bogus OS command) to have the package purposely fail after the bad email send step.

Sequencing Package Steps

1. Define the first step.
 - Right-click and select First Step.
2. Define the success path.
 - a. Select the “Next step on success” tool. 
 - b. Click the first step, the second step, and so on for the entire success path.
3. Set up error handling.
 - a. Select the “Next step on failure” tool. 
 - b. Click one step, and then click the next step.
 - c. Repeat to define the failure path wherever needed.
4. To delete links or steps:
 - a. Use the Select tool to select a link or step. 
 - b. Click the Delete selection button (or press the Delete key on the keyboard).

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The basic process for arranging package steps into a sequence is as follows:

1. Define the first step. Right-click a step and select First Step.
2. Define the success path for your package. Connect the normal sequence of steps into a continuous sequence. To do this, click the “Next step on success” tool, which resembles a green arrow with the word “**ok**” above it. Now, click all the steps in the sequence from the first to the last. Click the Select tool, which resembles a cursor, to stop sequencing.
3. You often add error-handling behavior. This works the same way. Click the red “**ko**” button representing “Next step on failure.” Now click a step and the corresponding error recovery step. This step can itself fail. For example, a mail server that is used to send reports can be down. If so, you may want to have a failure path from the `OdiSendMail` step to the `OdiFileAppend` step. Then report the error into a log file.
4. To delete unwanted links on steps, select the step or link with the Select tool. Then use “Delete selection” on the toolbar, or right-click and select Delete. You can also use the keyboard by pressing the Delete key.

Click the first step that you want to link and hold down the mouse button; drag the “**ok**” or “**ko**” line to the step to which you want to link and let go of the mouse button.

Note: All the steps in the diagram must be either on the success path or connected by a failure path to it. If there are unlinked steps, the yellow triangle will be highlighted. You can click the yellow triangle to check for the problem.

Agenda

- Packages: Overview
- **Executing a Package**
- Review of Package Steps
- Model, Submodel, and Datastore Steps
- Variable Steps
- Controlling the Execution Path



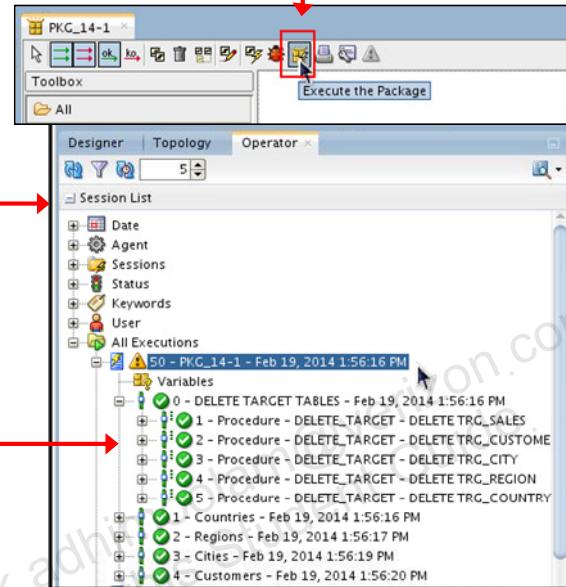
ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Now that you have created a package, you want to execute it. This is similar to executing a mapping. In this case, however, each mapping or tool step becomes a step in the session.

Executing a Package

1. Click the Execute button.



2. Open Operator Navigator.

- The package is executed as a session.
- Each package step is a step in Operator Navigator.
- Tool steps appear with a single task.
- Mapping steps show each command as a separate task.

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

To execute the package, click the Execute button in the package window. As with mappings, you can specify the context and the agent to run the package with.

When you receive a notification that the session has started, open the Operator Navigator. The package represents a session in the Operator Navigator. Mapping and tool steps are represented as steps. Mapping steps are then subdivided into tasks that complete the individual data transfer operations. Tool steps, however, have only one task.

Note: Be sure to test each mapping individually before using it in a package. Likewise, you can test individual steps in the package. To do this, right-click a step and select Execute Step. This is called “unit testing.” In some cases, it may not be possible to test an individual step, for example, when it requires a variable that is defined in the package. However, it is a good practice to test each step individually, whenever possible.

Agenda

- Packages: Overview
- Executing a Package
- **Review of Package Steps**
- Model, Submodel, and Datastore Steps
- Variable Steps
- Controlling the Execution Path



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

This section provides a quick summary of all the types of package steps available.

Basic Step Types

The following types of steps are straightforward. They complete a clearly defined operation when executed.

Mapping	Reusable reference to an existing mapping. Created by dragging.
Oracle Data Integrator Tool	Nonreusable call to an ODI tool
OS Command	Nonreusable call to an OS command
Procedure	Reusable reference to an existing procedure. Created by dragging.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The first group of steps is quite simple conceptually. They each represent an action that can be performed. When executed in a package, that action is completed. These should look familiar to you now.

- Mapping steps execute a mapping to move data from one place to another. You drag a mapping into the package to create a package step. This retains a reference to the original mapping.
- An ODI tool step calls an Oracle Data Integrator tool with some configured parameters. The step is not reusable, because it does not exist anywhere outside the package.
- An OS Command step is the same, but the command is sent directly to the operating system instead of the ODI tool engine. OS Command steps are also not reusable.
- The last for this group, the procedure step, calls the procedures that you have defined. You can configure the options for the procedure first. You can create this type of step by dragging a procedure to make a reference to it.

Advanced Step Types

- Drag an object into the package to create the step.
- This creates references, not copies.

Model Steps	Reverse-engineering, static control, or journalizing operations
Submodel Steps	Static control operations
Datastore Steps	Static control or journalizing operations
Variable Steps	Declare, set, increment, refresh, or evaluate the value of a variable. A variable can be used to create single steps.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

There are more complex types of steps that work on elements of models or variables, and each has several modes. All of the following types of steps are created by dragging an object into the package. This creates a reference, or link, rather than a copy of the object. It means that modifying the original object will affect the behavior of the package.

The first three types of steps perform operations on models, submodels, and datastores. A model is a collection of related datastores. A submodel is an optional additional level of hierarchy. You can set this by using the Models view.

There are three subtypes of model steps. You can reverse-engineer a model to update its structure from a database, perform a static data check on all the datastores in it, or configure journalizing behavior at run time.

Submodel steps have only one option: statically checking the data. Unlike static control in mappings, static control in packages can delete invalid rows that are detected.

Datastore steps allow both static control and journalizing configuration.

The other type of steps in this category is variable steps. These steps come in four types to declare, set or increment, refresh, and evaluate the value of a variable. Each variable can be used to create several different steps within the same package. This information will be covered in greater detail later.

Agenda

- Packages: Overview
- Executing a Package
- Review of Package Steps
- **Model, Submodel, and Datastore Steps**
 - Reverse (RKM)
 - Check (CKM)
 - Journalize (JKM)
- Variable Steps
- Controlling the Execution Path



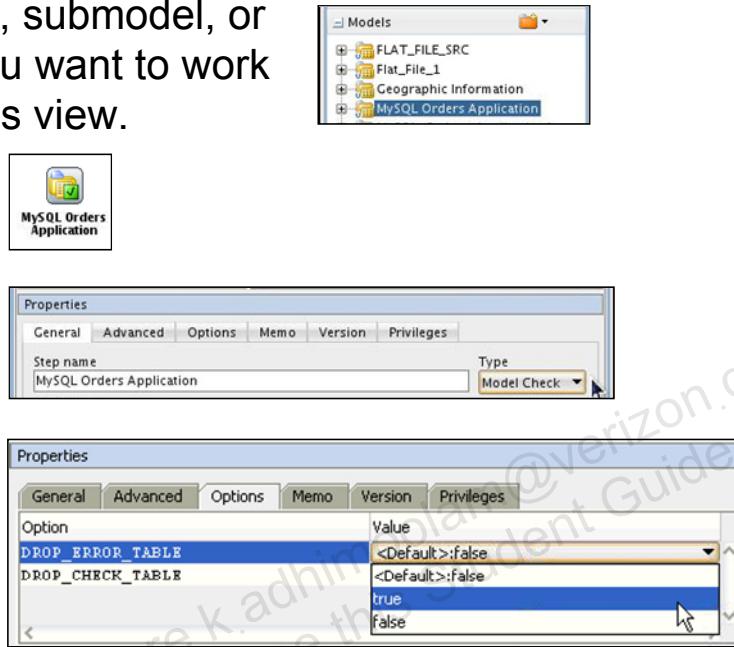
ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Now you look at a group of related steps: model steps, submodel steps, and datastore steps.

Creating Model, Submodel, and Datastore Steps

1. Select the model, submodel, or datastore that you want to work with in the Models view.
2. Drag it into the package.
3. Select the type of operation to perform.
4. Set the Options for the chosen operation.



ORACLE®

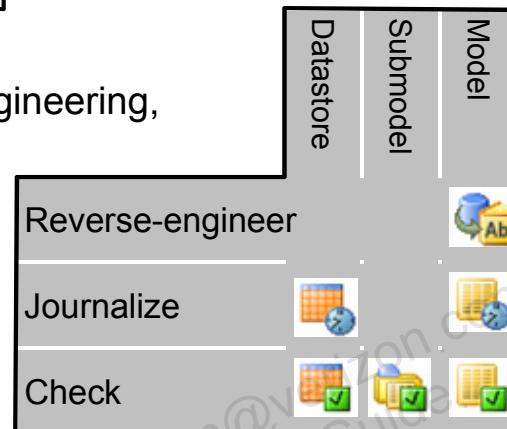
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The process for the three types of steps is basically the same. Perform the following:

1. Select the model, submodel, or datastore that you want to use. They are all found in the Models view.
2. Drag it onto the package diagram.
3. Select the type of operation that you want to perform on this object. The possibilities are reverse-engineering the model, configuring journalization options, and running a static data check. As you saw earlier, the operations available depend on the type of object. Each of these operation types has various options that can be set.
4. Click the Options tab of the Properties pane to set these options.

Models, Submodels, and Datastore Steps

- Reverse-Engineering type:
 - The reverse method defined for the model is used.
 - If using customized reverse-engineering, you must set the RKM options on the Options tab.
- Check type:
 - The static control strategy (CKM) set for the model and the datastores
 - Options for the CKM are set on the Options tab.
 - Select “Delete Errors from the Checked Tables” to remove erroneous records.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

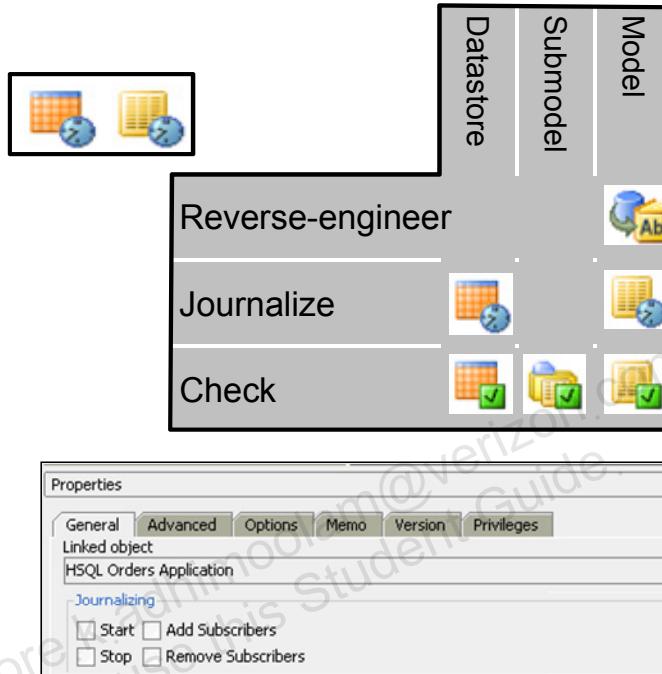
Now, the different operations are covered in greater detail. Models permit all three types of operations, and static data checks can be performed on all the three types. The only other combination possible is setting journalization options for a datastore.

- The **Reverse-Engineering** step type performs a reverse-engineering procedure on the linked model. This procedure updates the metadata for the model in the repository to synchronize it with the structure of the model in the database. Two reverse-engineering methods can be used: Standard and Customized. The method that has been defined directly on the model is the one that will be used here. If the model is set to use a customized method, the RKM selected on the model will be used. However, you must define the options for this RKM on the Options tab in the package step.
- The **Check** type runs a static data check on your datastore, submodel, or model. The Check Knowledge Module (CKM) is used. This is set on the Controls tab of the relevant object in the Models view. To set the relevant options for the CKM, click the Options tab in the Properties pane of the package step. For example, you may want to drop the existing `ERRORS` table when performing an automated static check. In this case, click the Options tab and set the `DROP_ERROR_TABLE` option to True. The “Delete Errors from the Checked Tables” check box appears on the General tab for check operations.

Models, Submodels, and Datastore Steps

Journalize step type:

- The journalizing type and strategy (JKM) set for the model are used in the step.
- The journalizing mode (simple/consistent set) determines which options are available.
- JKM-specific options are set on the Options tab.



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

When deselected, errors found during the check are copied to an `ERRORS` table, but not removed from the source datastores. However, when selected, they are removed from these datastores.

The **Journalize** step type enables you to reconfigure journalization for Changed Data Capture (CDC) for the given datastore or model. This type enables you to stop journalization, add subscribers, and so on.

You cannot change the journalization mode or the Journalizing Knowledge Module (JKM) used. These options should be defined on the model itself.

The choice of either the Simple mode or Consistent Set mode determines which options are shown. In particular, options for change consumption appear only if you use the Consistent Set mode. Use the Options tab to set options specific to the JKM, if any exist.

Note: Model steps, submodel steps, and datastore steps can have a major impact on your model when executed. For example, reverse-engineering at run time can create new datastores and add foreign keys or columns. Modifying journalization options can cause changes to be discarded. Even running a static check can delete invalid rows from your datastores. However, these capabilities can also be very useful. For example, static checks can regularly clean your data, or journalizing steps can reinitialize a CDC environment periodically. The best practice is to be very careful when using these capabilities.

Agenda

- Packages: Overview
- Executing a Package
- Review of Package Steps
- Model, Submodel, and Datastore Steps
- **Variable Steps**
- Controlling the Execution Path



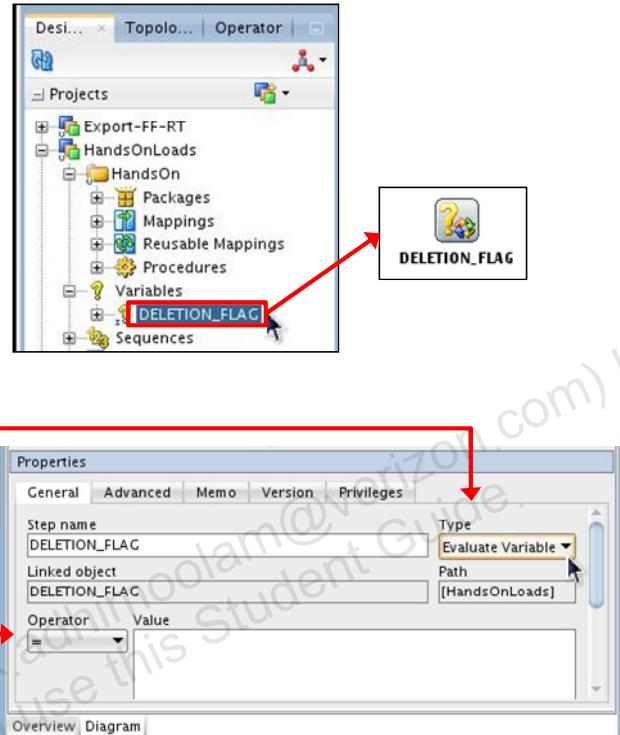
ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You now look at placing variables in packages. There are four types of variable steps that you can create this way.

Creating a Variable Step

1. Select the variable in:
 - The Projects view
 - Or the Others view
2. Drag it into the package.
3. Select the type of operation to perform.
4. Set the options for the operation to perform.



ORACLE

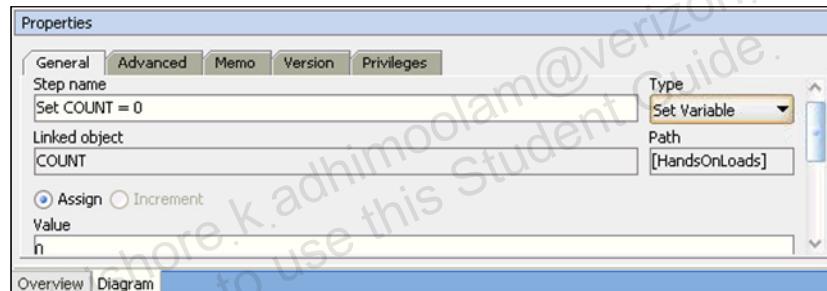
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Variable steps are created in a way very similar to procedure steps and model steps. Like model steps, you must select from several different types of steps. Perform the following:

1. Select the variable from the Projects view if it is a project variable. Global variables are found in the Global Objects view.
2. Drag the variable onto the package diagram.
3. On the General tab, select from one of the four available types of variable steps. These choices are covered in the next slide.
4. Set the options specific to the particular operation. For example, for the Evaluate Variable type, you set the value to compare against, as well as the operator for comparison, for example, `my_variable >= 10`.

Variable Steps

- Declare Variable step type:
 - It forces a variable to be taken into account.
 - Use this step for variables used in transformations, or in the topology.
- Set Variable step type:
 - It assigns a value to a variable or increments the numeric value of the variable.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

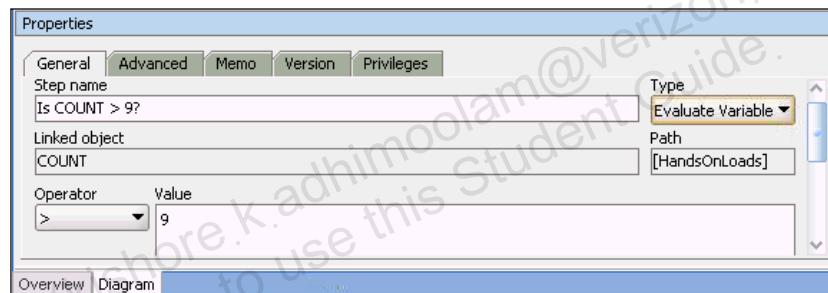
The four types of variable steps are as follows:

- The **Declare Variable step** is often placed at the start of a package. It explicitly defines the variable that is used in a mapping, procedure, or other package step. In general, variables are implicitly declared when they are referred to in a package. However, in certain complex situations, this cannot be guaranteed. Using this kind of step also improves readability, because you can visually display all the variables that are used in the package.
The variable's value may not be its default value when the step is executed. In this case, the variable's persistence type determines what happens. If it is "last value," the Declare Variable step does not change the stored value. If it is "not persistent," however, the stored value is discarded. It is then replaced by the default value of the variable.
- The **Set Variable step** is straightforward and can be used to perform two different actions. You can assign a specific value to a variable. For example, you can set an error counter to zero. You can also increment the value of a variable. For example, you can count the number of times a particular step is executed by putting a Set Variable step just before it with an increment of one.

Note: This slide shows a Set Variable in "Set" mode. After a Set Variable is inserted into the package, it can be instantiated again as an Increment variable (see option button disabled).

Variable Steps

- Refresh Variable step type:
 - It refreshes the value of the variable by executing the defined SQL query.
- Evaluate Variable step type:
 - It compares the variable value with a given value, according to an operator.
 - You can use another variable in the Value field.



ORACLE

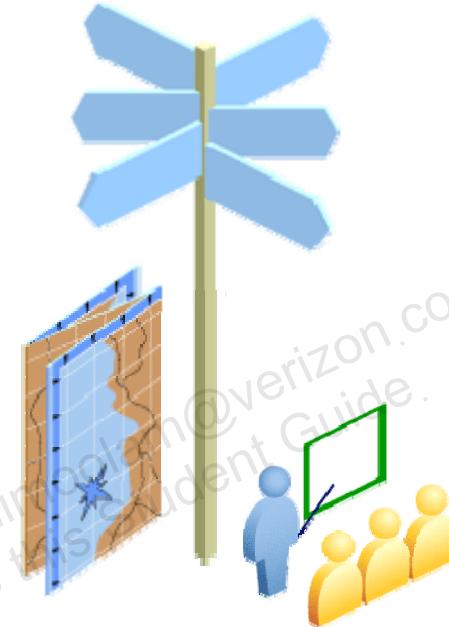
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The four types of variable steps (continued):

- The **Refresh Variable step** updates the variable value by executing the SQL expression of the variable. Note that variables are not automatically updated. For example, you have a variable with an expression that retrieves the number of rows in a table. In your package, you add more rows to that table. Before accessing the value of the variable again, you should include a Refresh Variable step to take into account the added rows. Note that it is not illegal to refresh a variable with no SQL expression. In this case, nothing happens.
- The **Evaluate Variable step** compares the value of the variable against a constant value or another variable. This comparison is used to create branches and loops later in this lesson. You can use any mathematical comparison, such as equals, greater than, less than, and so on. You can also use the SQL membership operator “IN.” The execution path of the package then splits according to the result of the comparison. Using the “ok” and “ko” tools, you define the “true” and “false” paths.

Agenda

- Packages: Overview
- Executing a Package
- Review of Package Steps
- Model, Submodel, and Datastore Steps
- Variable Steps
- **Controlling the Execution Path**



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Now that you have seen how to create the various types of package steps, you can use them to make parts of the package repeat several times in a loop, and branch as a result of a step.

Controlling Execution

- Each step may have two possible next steps:
 - Next step upon success
 - Next step upon failure
- If no next step is specified, the package stops.
- Execution can branch:
 - As the result of a step (success or failure)
 - Because of an Evaluate Variable step

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

After each step in a package is executed, two choices are possible for the next step:

- If the step is executed correctly, the “success” path is taken.
- If the step fails, the “failure” path is taken.
- If the next step does not exist, the execution of the package ends. For example, if a step fails and there is no failure step, the package stops. Similarly, if a step succeeds but does not define the next step on success, execution stops.

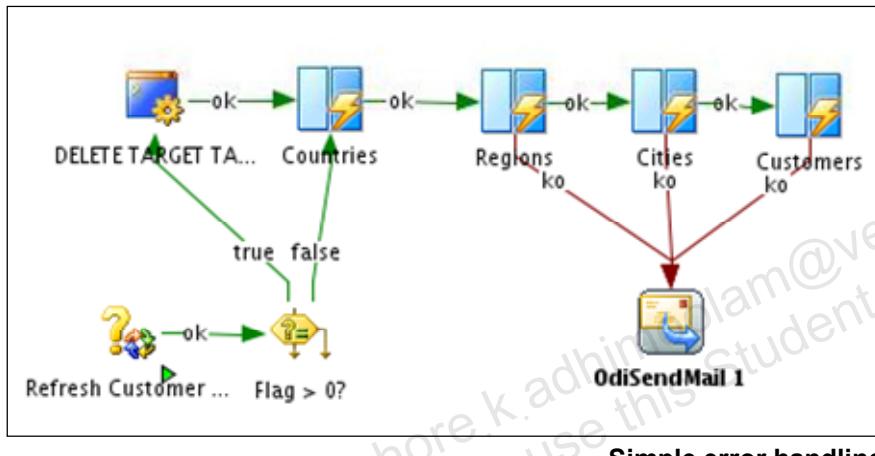
There are two distinct ways of making a branch:

- You can use the success and failure paths leading out of each step.
- You can use an Evaluate Variable step. This step performs a comparison on a variable and takes one of two different paths, depending on the result of the comparison.

Some examples of these control structures are covered in the next slide.

Error Handling

- Mappings fail when a fatal error occurs or when the number of permitted errors is exceeded.
- Procedures and other steps fail when a fatal error occurs.
- Try to take the possible errors into account.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

As stated earlier, steps have a failure path that is taken if the step fails. Failure, in this context, has several different causes.

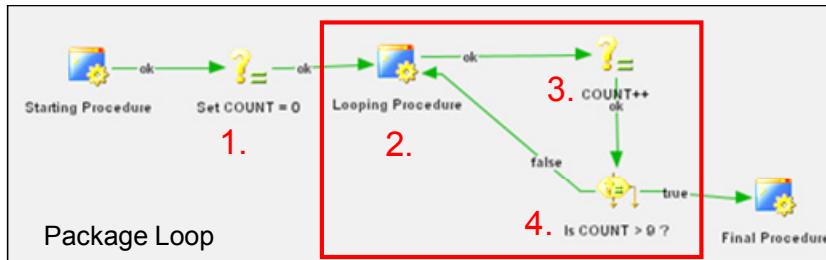
- Mappings fail when a fatal error occurs, which prevents them from executing completely or when the source datastore cannot be found on the database. However, they also fail when the predetermined maximum number of errors is exceeded during flow control.
- For example, you can specify a limit of 10 errors while processing a mapping. If more than 10 errors are detected, the mapping stops and is considered to have “failed.”
- Procedures and other steps fail when a fatal error occurs, which prevents them from continuing. Note that, for procedures, you can specify that errors for a particular step should be ignored. These errors will not cause the procedure to fail as a whole.

You should take into account the possibility of errors for every step. Every type of step can fail, so you should build error handling to respond to these situations.

A common solution is to link the failure path of every step to a common step that sends email to an administrator. You may want to consider adding error handling for the case where the mail server is down. For example, you can then write a file to a log file.

Creating a Loop

Loops need a counter variable.



1. Set the counter to an initial value.
2. Execute the step or steps to be repeated.
3. Increment the counter.
4. Evaluate the counter value and loop to step 2 if the goal has been reached.

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Now that you have seen how to create a branch, you can use a branch to create a loop.

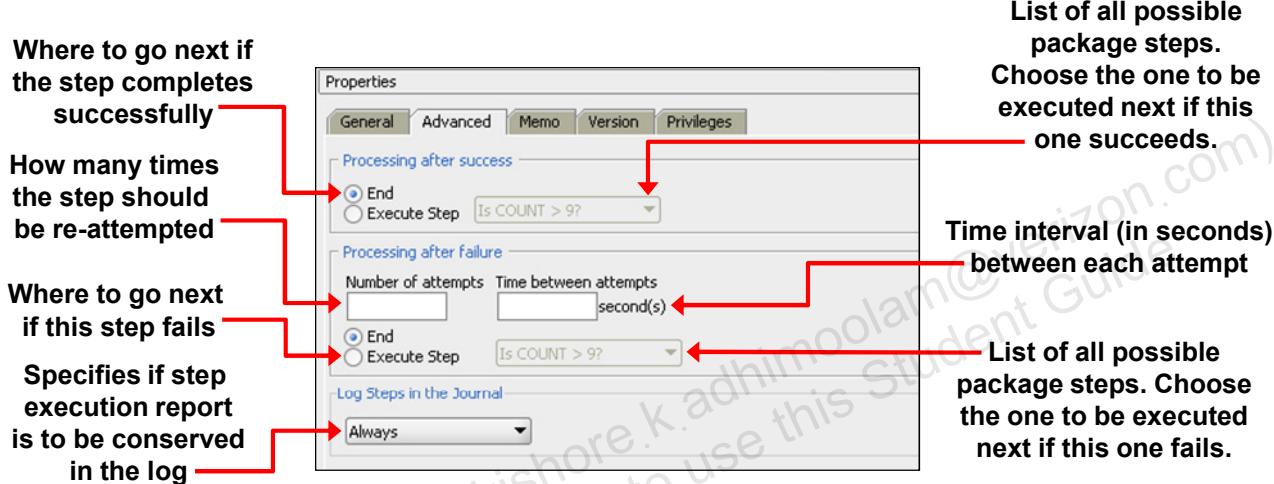
You need a variable to act as a “counter” for the loop. This variable usually starts at 0 or 1, and is incremented until it reaches the desired number of repetitions.

A loop usually looks like the one shown in the slide and has the following stages:

1. Set the counter to an initial value. In this case, you set the COUNT variable to 0. You do this by using the “Set Variable” step in “Assign” mode.
2. Then, you execute a procedure referred to as “Looping Procedure” here. This can be more than one step, including branches and even subloops.
3. Next, you increment the value of the counter to record the fact that you have executed the body of the loop one more time. You do this by using the “Set Variable” step in “Increment” mode.
4. Lastly, you evaluate the value of the counter by using the “Evaluate Variable” step. In this case, you test whether it is greater than nine. You link the “false” branch back to the start of the looping procedure. The “true” branch is connected to the next step of the package.

The Advanced Tab

- The Advanced tab of each step enables you to specify how the next step is determined.
- You can specify a number of automatic retries upon failure.



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

When you click any step in a package, you see an Advanced tab in the Properties pane. This tab enables you to indicate how the next step in the package is chosen. Most of the options are equivalent to drawing arrows in the diagram. In some cases, it may be more convenient or efficient to use this tab instead.

Note that you can specify how many times a step is automatically retried if it fails. You can also specify retries manually by setting up a loop connected to the failure path of the step. However, you might find the automatic retrial method more convenient.

Further, you can specify a time interval to wait between each attempt. This is useful in the case of attempting to connect to a server that may be down.

Quiz

Which of the following is *not* a way to make a branch?

- a. Use the success and failure paths leading out of the step.
- b. Use a Refresh Variable step.
- c. Use an Evaluate Variable step.
- d. None of the above



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Answer: b

Explanation: There are two distinct ways of making a branch:

- You can use the success and failure paths leading out of each step.
- You can use an Evaluate Variable step. This performs a comparison on a variable and takes one of two different paths, depending on the result of the comparison.

The Refresh Variable step is not used to create a branch.

Quiz

Variable steps are created by dragging an object into the package. This creates a reference to the variable.

- a. True
- b. False



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Answer: a

Explanation: Variable steps are created by dragging an object (variable) into the package. Dragging an object creates a reference, or link, rather than a copy of the object. It means that modifying the original variable will affect the behavior of the package.

Summary

In this lesson, you should have learned how to:

- Use Oracle Data Integrator packages to create a complete workflow
- Create package steps of different types
- Execute a package
- Create package steps based on procedures, variables, tools, and models
- Define complex workflows in ODI packages, involving branches and loops



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In this lesson, you learned about:

- Adding procedures to packages to create procedure steps. These are straightforward, because there is only one type of procedure step.
- Adding models, submodels, or datastores to packages. Here, you discussed three choices: reverse-engineering steps, static data check steps, and steps for reconfiguring journalization. Be careful with these choices because they can make major changes to your models.
- Variable steps. You learned that you can declare, set or increment, refresh, and evaluate a variable by selecting the appropriate type.
- Using variable steps to create complex package workflows. You also learned about the two different ways of branching: using the success and failure paths of any step, or using the “Evaluate Variable” step type. By combining the “Set Variable” step with the “Evaluate Variable” step, you can repeat steps in a loop.

Checklist of Practice Activities

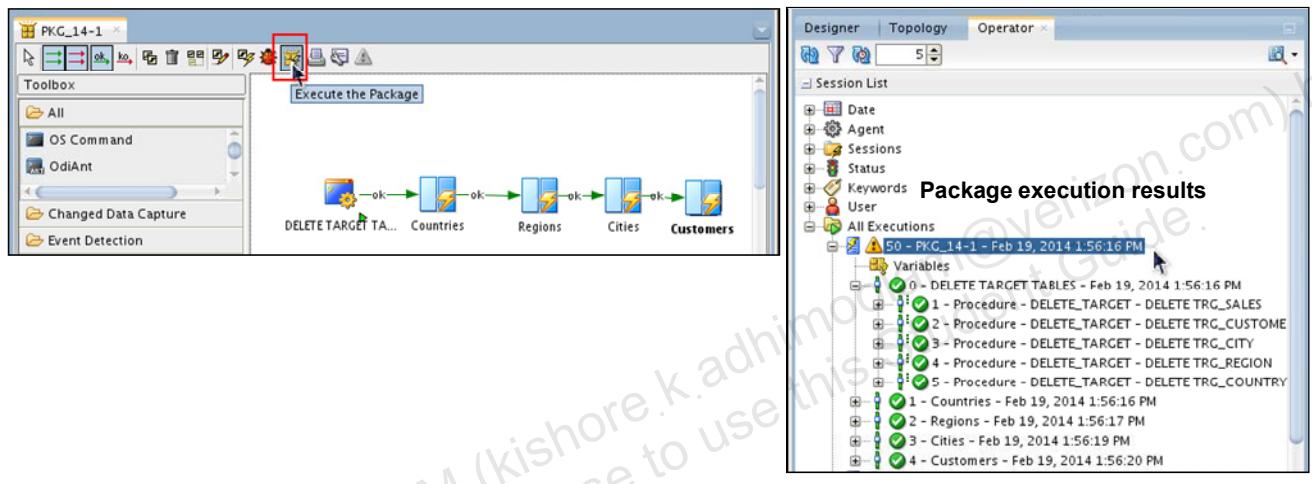
- ✓ 1-1: Logging In and Help Overview
- ✓ 2-1: Creating and Connecting to ODI Master and Work Repositories
- ✓ 3-1: Configuring Standalone Agents Using the Common Admin Model
- ✓ 4-1: Working with Topology
- ✓ 5-1: Setting Up a New ODI Project
- ✓ 6-1: Creating Models by Reverse-Engineering
- ✓ 7-1: Checking Data Quality in the Model
- ✓ 8-1: Creating ODI Mapping: Simple Transformations
- ✓ 9-1: Creating ODI Mapping: Complex Transformations
- ✓ 9-2: Creating ODI Mapping: Implementing Lookup
- ✓ 10-1: Creating ODI Mapping: Exporting a Flat File to a Relational Table
- ✓ 11-1: Using Native Sequences with ODI Mapping
- ✓ 11-2: Using Temporary Indexes
- ✓ 11-3: Using Sets with ODI Mapping
- ✓ 12-1: Creating and Using Reusable Mappings
- ✓ 12-2: Developing a New Knowledge Module
- ✓ 13-1: Creating an ODI Procedure
- 14-1: **Creating an ODI Package**
- 14-2: **Using ODI Packages with Variables and User Functions**
- 15-1: Debugging Mappings
- 16-1: Creating and Scheduling Scenarios
- 17-1: Using Load Plans
- 18-1: Enforcing Data Quality with ODI Mappings
- 19-1: Implementing Changed Data Capture
- 20-1: Setting Up ODI Security
- 20-2: Integration with Enterprise Manager and Using ODI Console

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Practice 14-1: Creating an ODI Package

1. Create a DELETE_TARGET procedure to delete the TRG_CUSTOMER, TRG_CITY, TRG_REGION, and TRG_COUNTRY tables.
2. Create and execute a package, PKG_14-1, that runs this new procedure and the four mappings that you defined in Lessons 8 and 9: MAP_8-1, MAP_8-2, MAP_8-3, and MAP_9-1.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

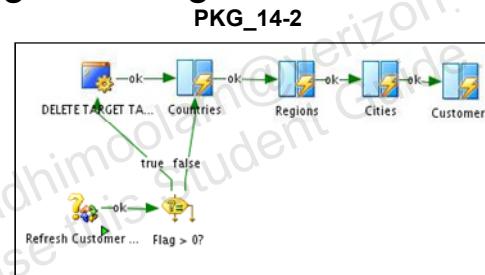
ORACLE

You need to delete TRG_SALES before TRG_CUSTOMER because an earlier practice has loaded it.

This package runs your new table deletion procedure, and then runs four table-creation mappings (that you defined earlier) that will re-create these same tables.

Practice 14-2: Using ODI Packages with Variables and User Functions

1. In the HandsOnLoads project, define a user function group named Conversion, containing a user function named DearConvert.
2. Use this function in the mapping MAP_9-1 to convert the values (0, 1, 2) to ("Mr," "Mrs," "Ms").
3. In Designer, create a numeric variable, DELETION_FLAG, for counting the number of lines in TRG_CUSTOMER.
4. Duplicate the PKG_14-1 package, naming the new package PKG_14-2.
 - Have the DELETE_TARGET procedure run only if the DELETION_FLAG variable is greater than 0.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Be mindful of curly double quotes versus straight single quotes on string values in code.

15

Step-by-Step Debugger

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Describe where the Debugger would be used in an ODI environment
- Start and stop a Debugger session
- Control the flow through a Debugger session
- Define the purpose of each icon on the Menu Bar

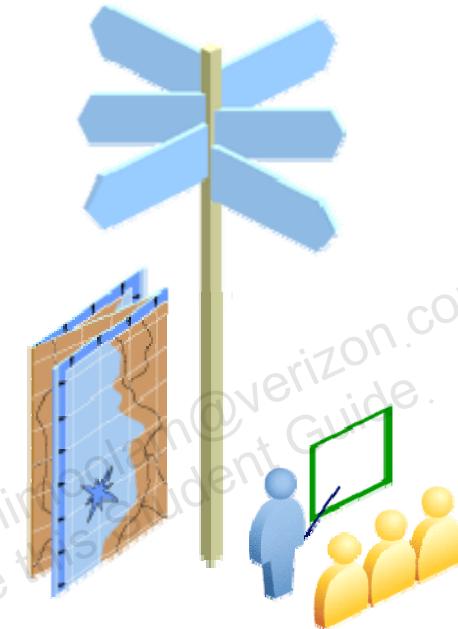


Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In this lesson, you learn how to operate the Debugger.

Agenda

- **Overview**
- Starting a Debug Session
- New Functions
- Menu Bar Icons

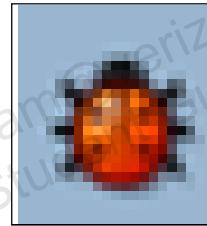


ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Overview

- Give control to a user over the execution flow of a session
- Define breakpoints on steps in packages
- Toggle breakpoints in sessions
- Query data through agents
- Start a session with the option to wait



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Using the Debugger, you can give control to the user over the execution flow of a session.

- A *session* can be:
 - Mappings
 - Packages
 - Procedures
 - Scenarios
- Control can be:
 - Pause
 - Resume
 - Run to Next Stepand so on

You can define breakpoints on steps in packages, and you can toggle breakpoints in sessions.

Query data through agents gives access to uncommitted data. You can also query variables.

Given that the Debugger is used to fix errors, you can start a session with the option to wait for user inputs on error.

Agenda

- Overview
- **Starting a Debug Session**
- New Functions
- Menu Bar Icons



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

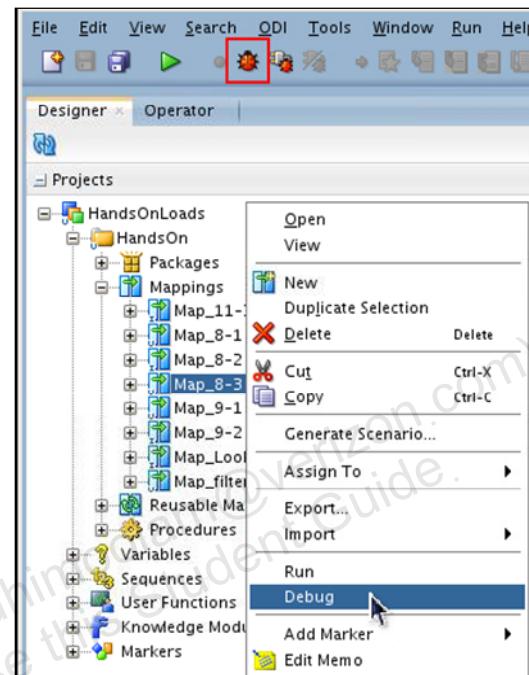
Process Overview

1. Start a session in Debug mode.
2. Specify Debug properties.
3. Control execution flow
(Pause, Resume, and so on).



Starting a Session in Debug mode

1. Start a session in Debug mode.
2. Specify Debug properties.
3. Control execution flow.



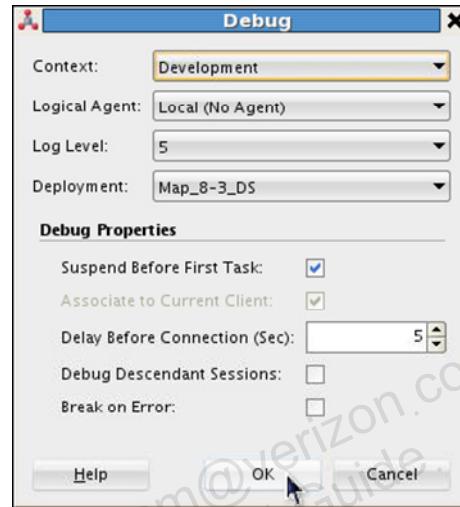
ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

To start a Debug session, right-click the mapping and then select Debug. Or select the mapping and then click Debug in the toolbar. This starts the new session for debugging.

Specifying Debug Properties

1. Start a session in Debug mode.
2. Specify Debug properties.
3. Control execution flow.



ORACLE

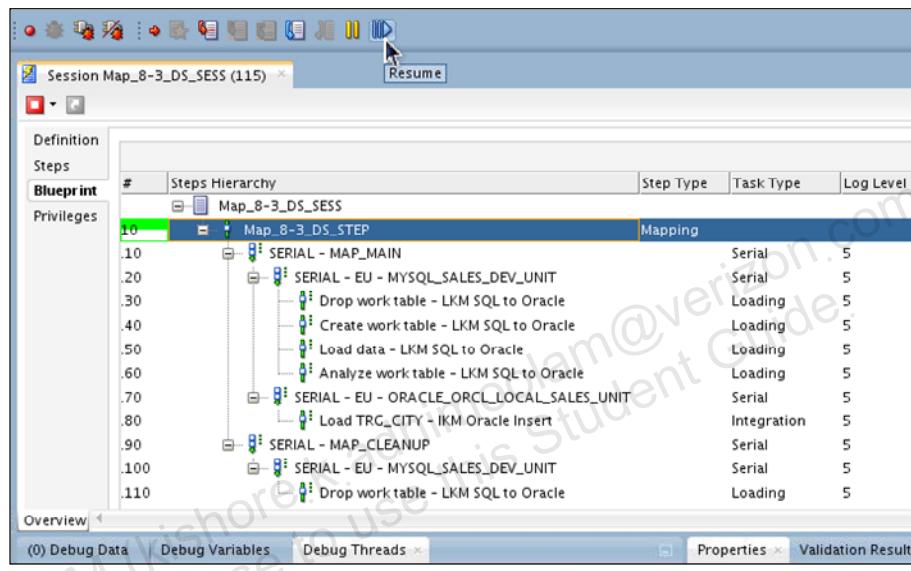
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In the Debug window, set the Context field to Development, and then specify other Debug properties. Click OK. Then click OK in the “Session started” pop-up window. In the Confirmation dialog box, click Yes to bring up the Debug windows (if they are not shown).

Note: A debugging session is executed on a context and an agent. You can select the name of the agent that executed the debugging session. By selecting Local (No Agent), you choose to use the built-in agent in Oracle Data Integrator Studio.

Control Execution Flow

1. Start a session in Debug mode.
2. Specify Debug properties.
3. Control execution flow.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

ORACLE

Control execution flow examples:

- Add breakpoints
- Pause
- Resume
- Navigate to steps

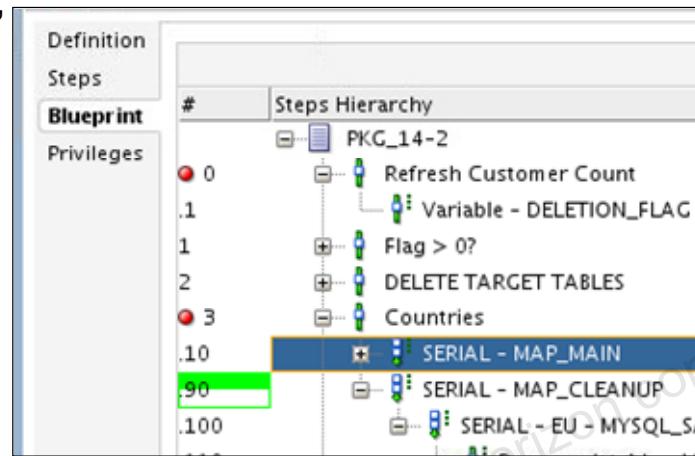
And so on

When controlling the execution flow, you can add breakpoints, pause and resume execution, and set breakpoint parameters. Debugging is performed in the blueprint of an active session, as shown in the slide. The blueprint shows all of the steps and the task hierarchy of the selected session, and allows you to go through individual commands.

When you start a session, the session editor opens in its Blueprint view. A blueprint is analogous to a SQL Execution Plan.

Screen Step Numbering

- Notice the “decimals.”
- Notice the + and –.
- Notice the “gaps.”



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Note that step .10 does not equal step .100, and that step .90 is before step .100; in other words these are not *decimals*, rather they are *labels*. Technically, they are 0.100, and 3.010, 3.090, and 3.100 to indicate children and parents, but it is not displayed that way.

The plus + collapsing step .10 (child of 3, so really 3.010) includes .10, .20 (really 3.020), .30, .40, .50, .60, .70, and .80. So in fact there is no gap between .10 and .90; you just cannot see the intermediate numbers.

Agenda

- Overview
- Starting a Debug Session
- **New Functions**
- Menu Bar Icons

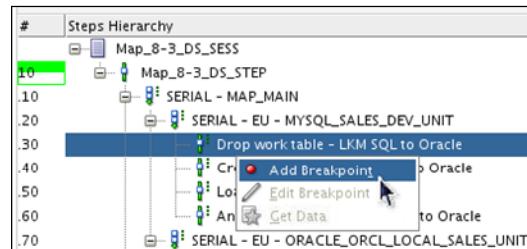


ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

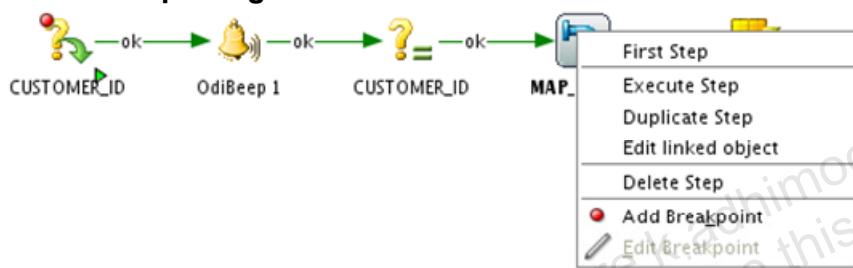
New Functionalities

Toggle breakpoints...



...in sessions

...in packages

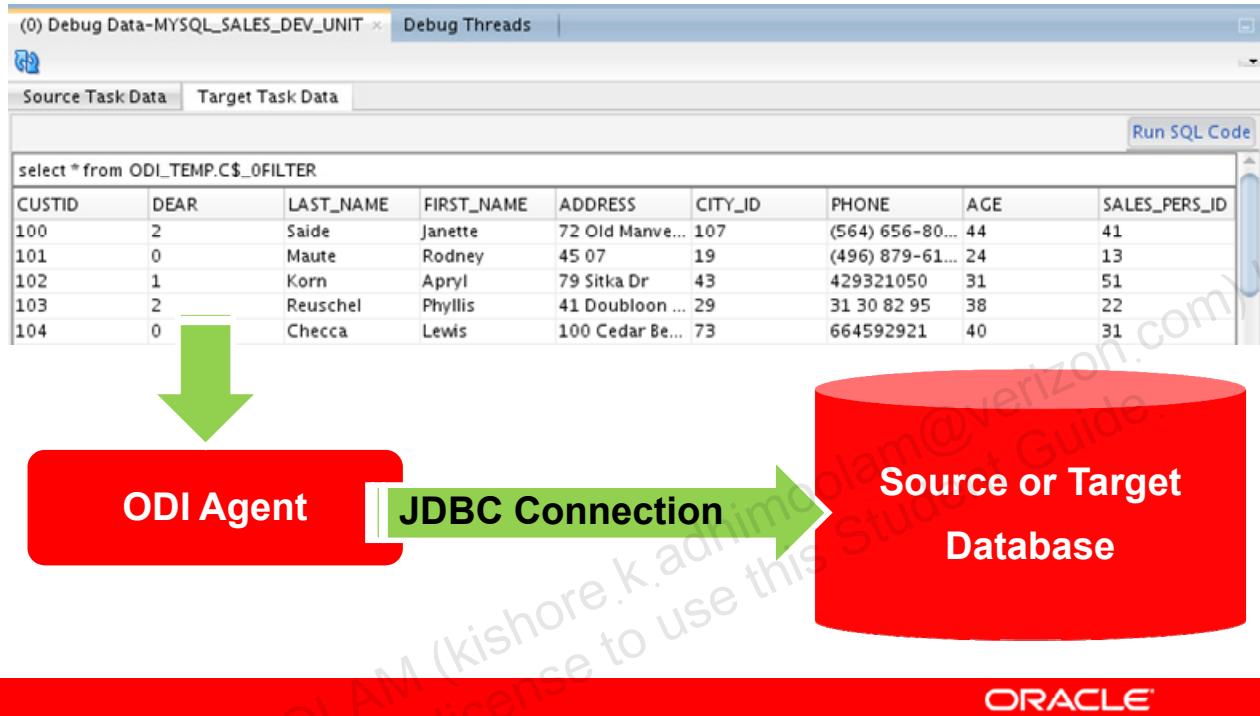


ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

New Functionalities

Access to uncommitted data during the session execution



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

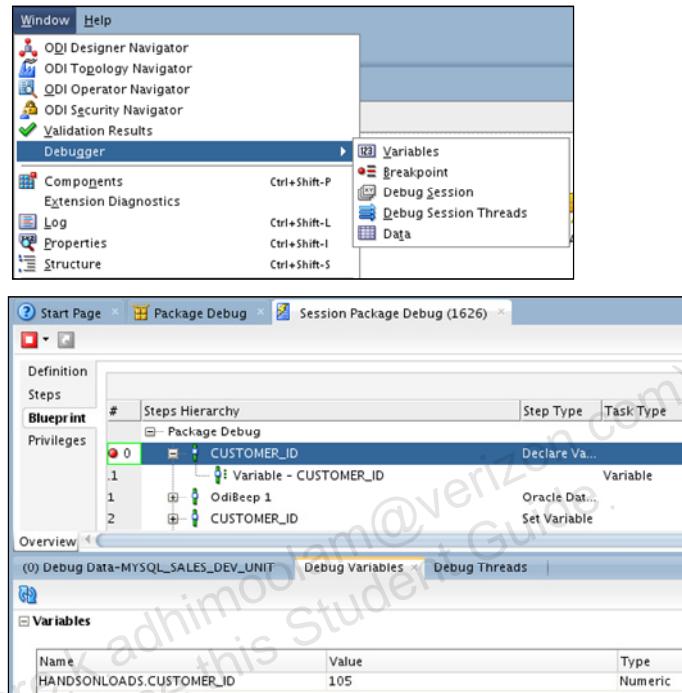
The step-by-step debugger provides users with access to uncommitted data in the source or target database during session execution.

New Functionalities

During session execution,
you can view:

- Variables
- Threads
- Data

and more



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Benefits for End Users

- Debuggability of developments:
 - Execute the mapping in Debug mode.
- Hot debuggability in Production:
 - Launch the scenario in Debug mode.
 - Follow the flow step by step.
 - View Variables as well as data being managed by ODI.
- Debug random errors in concurrent executions:
 - Start all scenarios with “Break on Error.”
 - Whenever an error occurs, look into the session(s) and the data being manipulated.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Reusability of blueprints:

- Mappings always get a new blueprint each time.
- Scenarios always use the same blueprint each time.

Agenda

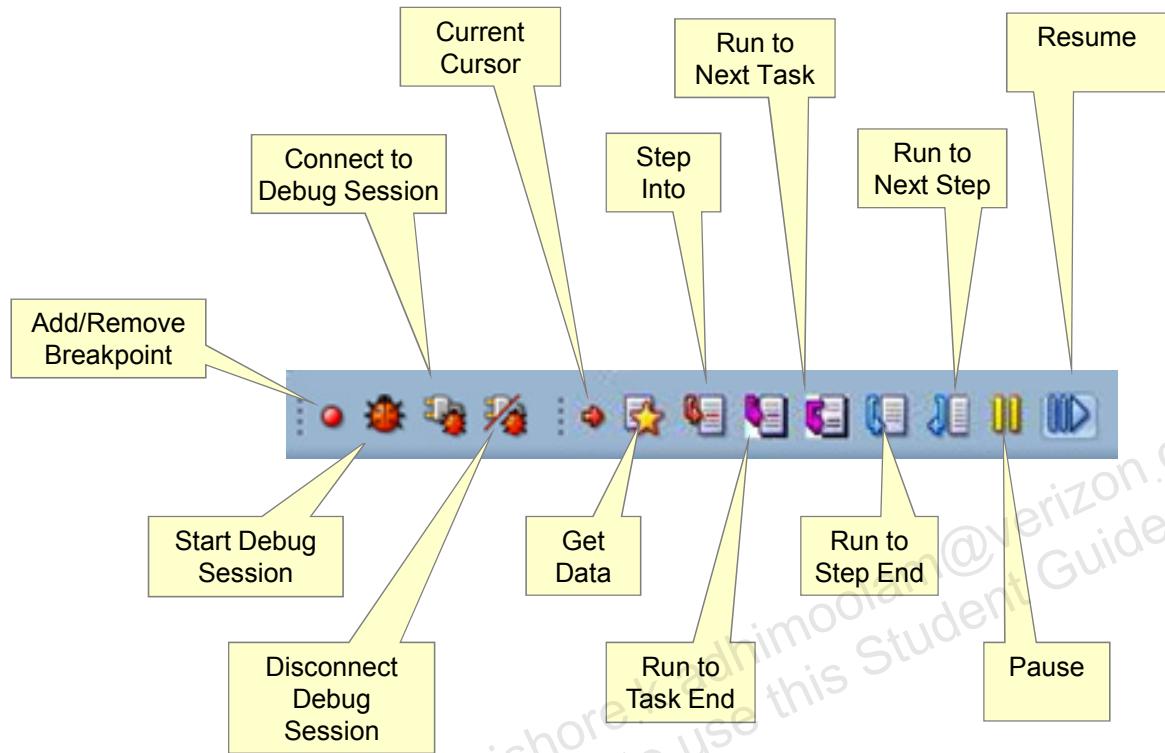
- Overview
- Starting a Debug Session
- New Functions
- **Menu Bar Icons**



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Debug Toolbar



ORACLE®

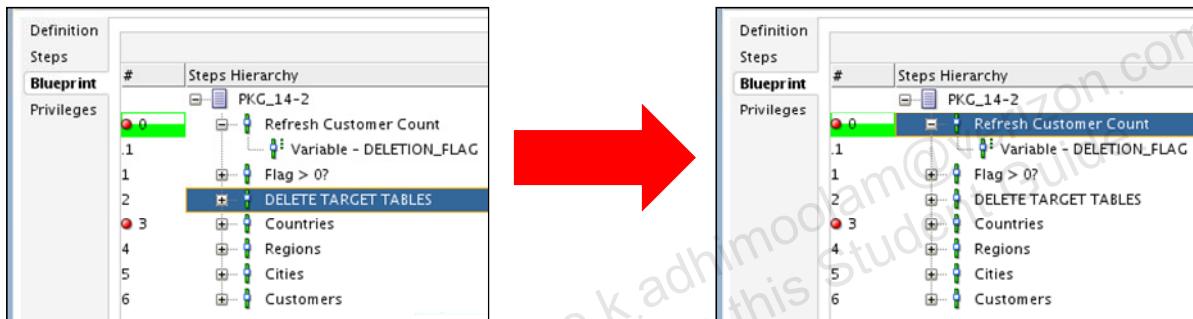
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Not all icons are visible at all times. If the icon is grayed out, then it is not available at that time.

Toolbar: Current Cursor



- Brings users back to the current step/task being executed

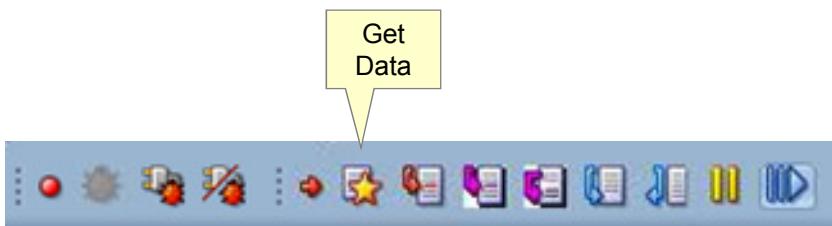


ORACLE

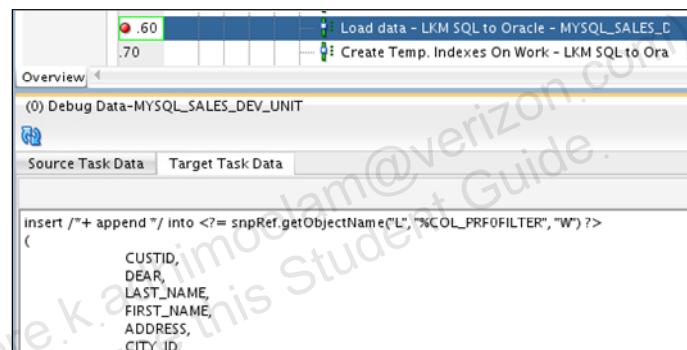
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You can use multiple cursors for in-session parallelism.

Toolbar: Get Data



- Populates Debug Data windows with executed code
- Users can run SELECT statements from Debug Data window.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

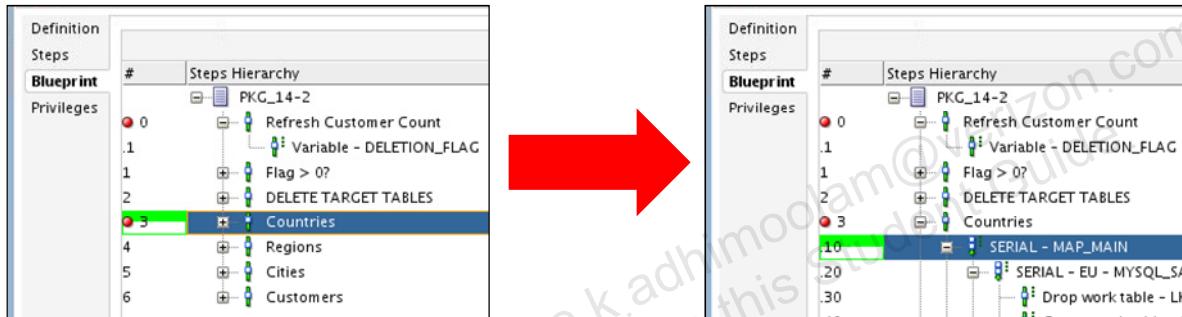
Get Data inserts the SQL code of the currently selected task into the SQL command field of the Debug Data window. Both the Source Task Data and Target Task Data windows are populated.

The Get Data command associates with the original cursor line; if you keep stepping, the data window remains attached to the original cursor line. You must click Get Data again to go to the new current cursor line.

Toolbar: Step Into



- Goes to next Child step/task and pauses execution



ORACLE

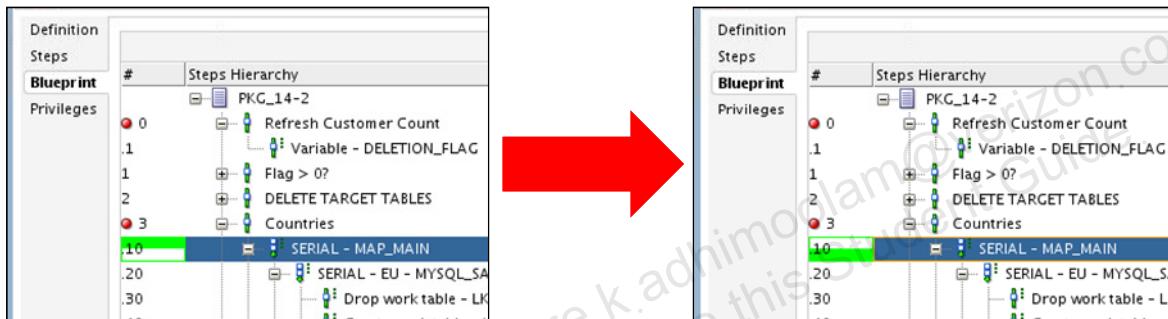
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Notice that the top half of the cursor on the left means that that step has *not* run. In between step 3 and 4 are many other tasks: .10, .20, .30, and so on. Expand by clicking the plus to see them.

Toolbar: Run to Task End



- Executes current task and pauses execution



ORACLE®

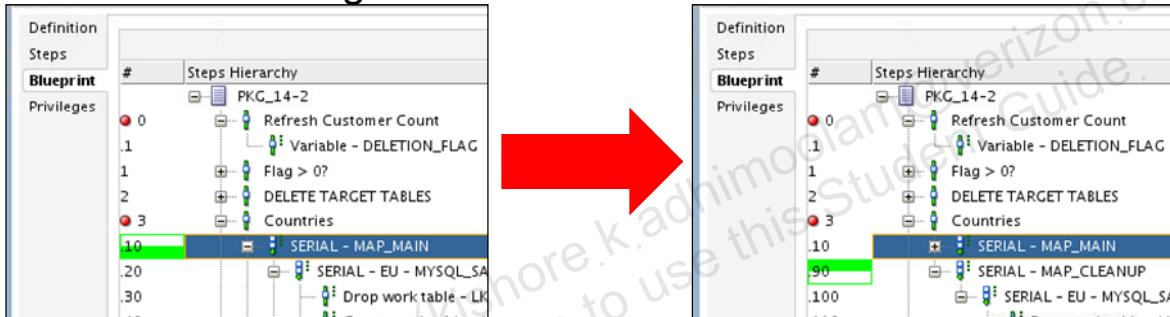
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Note that even though step .20 (really 3.020) is a child of .10 and a grandchild of 3, the labeling is not any new sequence; there is no labeling indentation. Same with .30 (really 3.030) being a great-grandchild of 3, there is no more indentation numerically, but there is an indentation graphically with the + and – to expand/collapse the view.

Toolbar: Run to Next Task



- Executes current task and goes to next task
- Goes to next task if current task has already been executed using Run to Task End



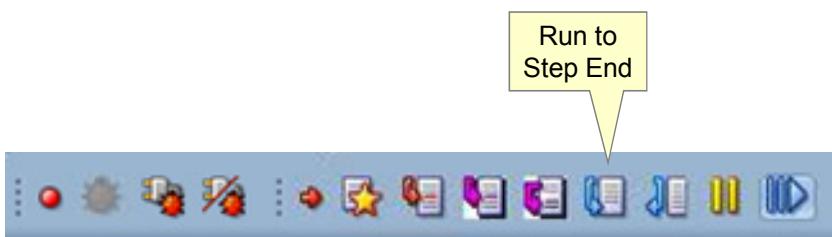
ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

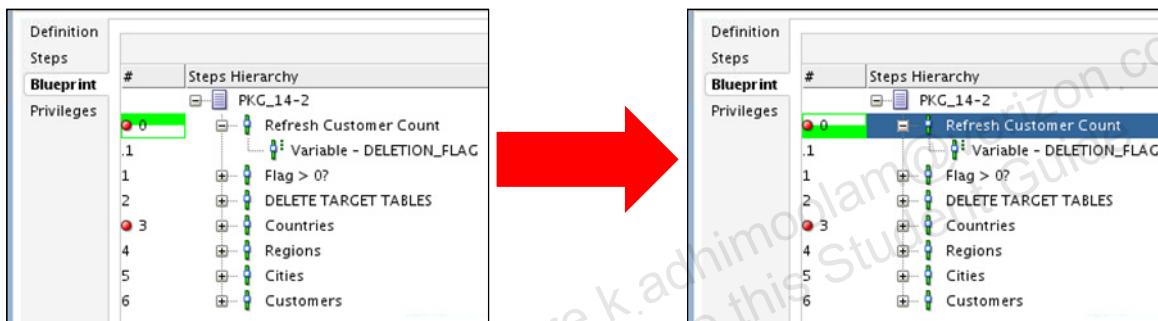
Tasks are the items within steps, the dot-labeled numbers.

Note that on the right task .10 has been collapsed, so tasks .20, .30, .40, and so on are not showing.

Toolbar: Run to Step End



- Executes current step and pauses execution

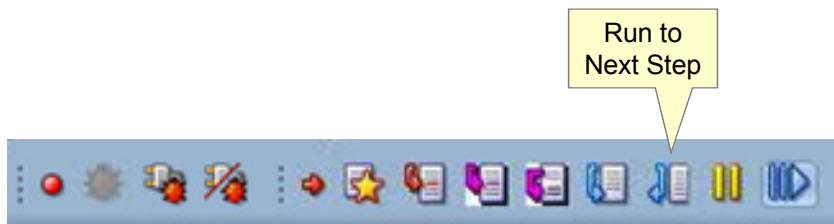


ORACLE

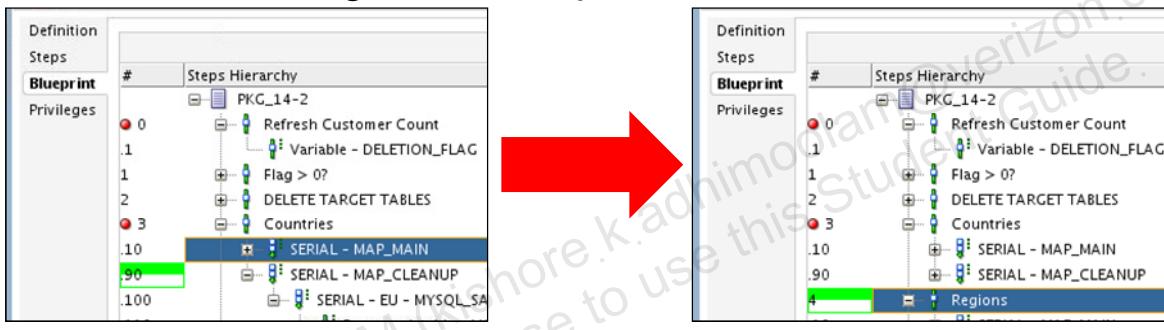
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Notice that the top half of the cursor on the left means that that step has *not* run. The bottom half of the cursor on the right means that that step *has* run.

Toolbar: Run to Next Step



- Executes current step and goes to next step
- Goes to next step if current step has already been executed using Run to Step End



ORACLE

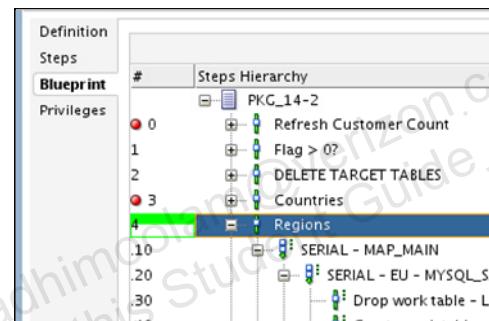
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Notice that it completed step .100 and .110 and went on to the next whole number.

Toolbar: Pause



- Pauses the current execution



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

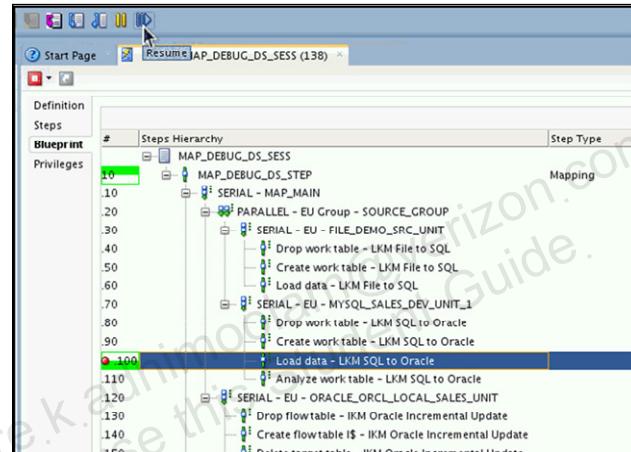
When execution is paused, you can modify the code of the tasks that have not yet been executed. That modified code is taken up when you resume execution.

Execution can be continued by stepping through the steps/tasks, or by resuming execution.

Toolbar: Resume



Resumes execution at the current cursor and continues until the session is finished or until a breakpoint is reached



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Summary

In this lesson, you should have learned how to:

- Describe where the Debugger would be used in an ODI environment
- Start and stop a Debugger session
- Control the flow through a Debugger session
- Define the purpose of each icon on the Menu Bar



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Checklist of Practice Activities

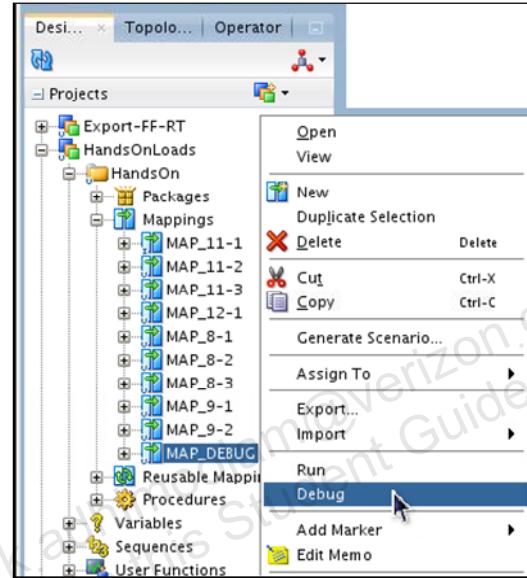
- ✓ 1-1: Logging In and Help Overview
- ✓ 2-1: Creating and Connecting to ODI Master and Work Repositories
- ✓ 3-1: Configuring Standalone Agents Using the Common Admin Model
- ✓ 4-1: Working with Topology
- ✓ 5-1: Setting Up a New ODI Project
- ✓ 6-1: Creating Models by Reverse-Engineering
- ✓ 7-1: Checking Data Quality in the Model
- ✓ 8-1: Creating ODI Mapping: Simple Transformations
- ✓ 9-1: Creating ODI Mapping: Complex Transformations
- ✓ 9-2: Creating ODI Mapping: Implementing Lookup
- ✓ 10-1: Creating ODI Mapping: Exporting a Flat File to a Relational Table
- ✓ 11-1: Using Native Sequences with ODI Mapping
- ✓ 11-2: Using Temporary Indexes
- ✓ 11-3: Using Sets with ODI Mapping
- ✓ 12-1: Creating and Using Reusable Mappings
- ✓ 12-2: Developing a New Knowledge Module
- ✓ 13-1: Creating an ODI Procedure
- ✓ 14-1: Creating an ODI Package
- ✓ 14-2: Using ODI Packages with Variables and User Functions
- 15-1: **Debugging Mapping**
- 16-1: Creating and Scheduling Scenarios
- 17-1: Using Load Plans
- 18-1: Enforcing Data Quality with ODI Mappings
- 19-1: Implementing Changed Data Capture
- 20-1: Setting Up ODI Security
- 20-2: Integration with Enterprise Manager and Using ODI Console

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Practice 15-1: Debugging Mappings

- Debugging a mapping
- Examining data



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Prerequisite: Complete Practice 9-1, because in Practice 15-1, you make a copy of the mapping created in 9-1.

Unauthorized reproduction or distribution prohibited. Copyright© 2015, Oracle and/or its affiliates.

KISHORE ADHIMOOLAM (kishore.k.adhimoolam@verizon.com) has
a non-transferable license to use this Student Guide.

16

Managing ODI Scenarios

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Describe the purpose of an ODI scenario
- Describe the properties of an ODI scenario
- Generate and regenerate an ODI scenario
- Execute ODI scenarios from different sources
- Prepare ODI scenarios for deployment
- Perform automated scenario management



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

This lesson focuses on releasing your Oracle Data Integrator (ODI) projects for production.

The goals for this lesson are:

- To understand how to generate deployable scenarios from your ODI objects
- To know how to generate, execute, and export scenarios
- To release generated scenarios so that they can be put into production

Agenda

- **Scenarios**
- Managing Scenarios
- Preparing for Deployment



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The first section of this lesson describes scenarios and how they enable you to release your work.

What Is a Scenario?

- A scenario is a frozen copy of the generated code, ready for execution in any context.
- It is the form in which all work in ODI should be released.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Remember that a mapping, procedure, or package can be modified at any time. A scenario is created by generating code from such an object. It thus becomes frozen, but can still be executed in a number of different environments or contexts. Because a scenario is stable and ready to be run immediately, it is the preferred form for releasing objects developed in ODI.

Properties of Scenarios

- Scenarios are “compiled” ODI objects.
 - Easily deployed
 - Not editable
- Scenarios have a unique name and version.
 - You can keep successive versions of the same scenario.
- Scenarios are generated from:
 - Packages
 - Mappings, procedures, and variables
- Scenarios can be started on different contexts.
 - The same scenario can be used in development, test, production, or on different sites.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Scenarios are created by generating code from ODI objects. This process is similar to compiling a program from source code. Scenarios are easily deployed on any repository and executed with an agent having access to the repositories.

Like compiled programs, scenarios cannot be modified. A scenario that works today will work tomorrow as well.

Because scenarios represent snapshots in the development of an object, they have a name and a version. You can retain several versions of the same scenario. In ODI, the different versions of a scenario are always displayed attached to the original object.

The most common way to create scenarios is to generate them from packages. However, you can also create scenarios from mappings, procedures, or even variables.

Scenarios are still context independent. You can generate a scenario to load data into a sales database, and then run the same scenario in different contexts to load the sales databases for different sites.

Note: Scenarios can be encrypted/decrypted. When encrypted, the generated code is obfuscated in the Operator Navigator logs. **Exception: Scenarios generated from variables cannot be encrypted.**

Agenda

- Scenarios
- **Managing Scenarios**
- Preparing for Deployment



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Now you see how to perform some practical tasks with scenarios.

Scenario-Related Tasks

Scenario-related tasks include:

- Generating and regenerating
- Exporting
- Executing scenarios
 - From the graphical user interface (from the Studio)
 - From a project by using the Sunopsis API tool OdiStartScen
 - From the command line (`./startscen.sh` or `startscen.bat`)
 - From the web service (`invokeStartScen` operation)
 - Scheduling
 - ODI Console
- Optionally encrypting the scenario



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In the rest of the lesson, you assume that you are generating a scenario from a package, which is the most common situation. You learn how to generate scenarios and regenerate them if they change.

After you generate a scenario, you can export it to a file. This enables you, for example, to transfer it to a remote site.

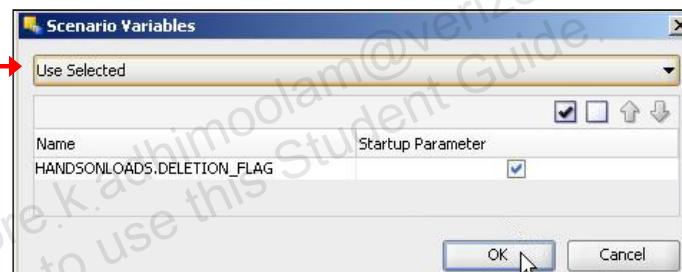
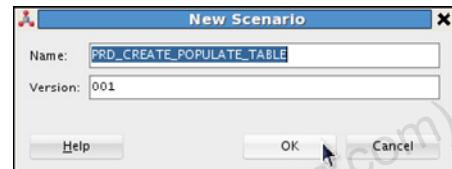
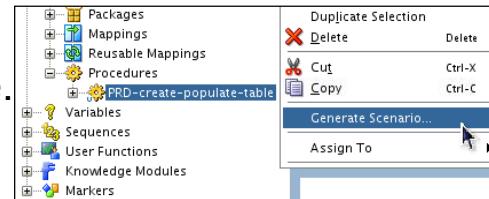
You learn how to execute scenarios, which can be done directly from the ODI GUI, from an operating system command line, or by using the Sunopsis API.

You also can start scenarios by using an external scheduler, or from ODI Console's web interface. However, these methods are not covered in this course.

You can optionally encrypt the scenario and produce a decryption key to later decrypt the scenario. Encrypting the scenario produces a set of random-appearing characters if you click the "Code" tab in the Operator Task editor windows after execution. This is a useful security feature.

Generating a Scenario

1. Right-click the package or procedure that you want to compile.
2. Select Generate Scenario.
3. Enter the scenario name and version, then click OK.
4. If the package uses variables, define which variables to use as parameters.
 - Use All, None, or a selection of variables.
 - Click OK, and ODI generates the scenario.



ORACLE

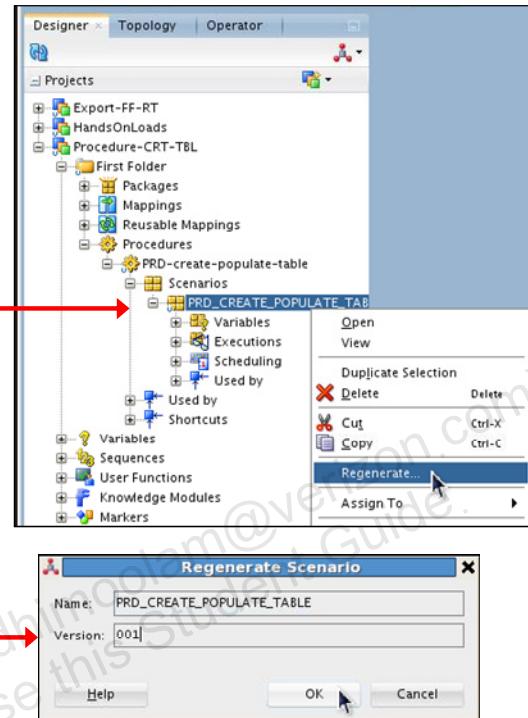
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

To generate a scenario for a package, perform the following:

1. Right-click the package.
2. From the context menu, select Generate Scenario.
3. Select a name and version for the scenario. You can choose any version numbering scheme you want. However, the combination of name and version must be unique for this package. Click OK.
4. If some variables are in the package, you must specify which ones to use as parameters. When you launch the scenario by using schedules or from Metadata Navigator, ODI asks you for the value of these parameters.
 - You can choose to use all variables as parameters or none of them.
 - If you want to specify which variables to use, select Selective Use.Click OK. ODI generates the scenario. Depending on the complexity of the package, the scenario generation may take a while.

Regenerating a Scenario

1. Right-click the scenario that you want to regenerate.
2. Select Regenerate.
3. Click OK.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Although you can store several versions of a scenario, at times you may want to regenerate an existing scenario after editing it. Regeneration replaces the selected version of the scenario with a freshly generated copy, *keeping* the version number.

To do this, right-click the scenario and select Regenerate. The parameters that you previously defined are used again. ODI generates the scenario. Again, this may take a while for complex scenarios.

Generation Versus Regeneration

- Regeneration:
 - Overwrites the scenario
 - Re-attaches any schedules for the current scenario
- Generation:
 - Preserves a history of scenarios
 - Requires schedules to be redefined



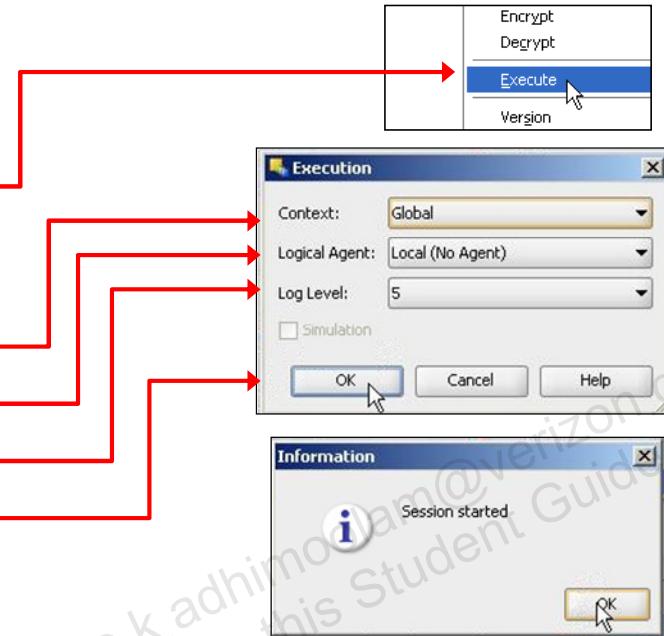
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The difference between generating a new version of a scenario and regenerating a scenario is as follows:

- Regeneration always overwrites an existing scenario that uses the same name and version number. However, the advantage is that any schedules that are attached to the old version of the scenario are retained.
- Generation, on the other hand, preserves a history of old versions of the scenario. However, you must redefine any schedules, because they remain attached to the old version.

Executing a Scenario from the GUI

1. Right-click the scenario that you want to run.
2. Select Execute.
3. Set the session parameters:
 - Context
 - Agent
 - Log level
4. Click OK.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

After a scenario has been generated, it has to be executed from ODI. You can do this in Designer or Operator, as follows:

1. Right-click the scenario.
2. Select Execute.
3. The session parameter window is the same as for executing any other kind of object. You specify the context to run the scenario in, the agent that will run it, and the log level. A higher log level means more information is retained in the execution log after the scenario completes.
4. Click OK to launch the session.

Executing a Scenario from a Command Line

- The shell script `startscen` starts an agent to execute one scenario.
- Syntax:

```
startscen.bat <Name> <Version> <context_code>
[<log_level>]
"[-v=<trace_level>]"
" [<variable>=<value>]"
```
- Example:

```
C:\> startscen.bat PRD-create-populate-table
001 DEVELOPMENT SESSION_NAME=Dev
```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You can also execute a scenario directly from an operating system command line.

ODI provides a shell script, `startscen`, to execute one scenario. The shell script is called `startscen.bat` for Windows platforms and `startscen.sh` for UNIX platforms. This script launches a special kind of agent that runs the scenario and shuts down.

To use this script, you must first configure the `odiparams` script. This script tells the agent how to connect to the repository where the scenario is stored.

To call the `startscen` script, you must identify the scenario that you want to run with its name and version. You must also provide the code for the context that you want to run it in. You can also specify a log level for retaining session information in the log after the execution, and a trace level, to write the execution detailed trace to the Console. This trace is useful for debugging or support purposes. Lastly, you can also supply values to any parameters used by the scenario.

In this example, you use `startscen` to launch version 001 of a scenario. You run the scenario in the `DEVELOPMENT` context. In the example, the variables or the optional trace levels are not specified.

Executing a Scenario from a Package

- ODI Tools has a component, `OdiStartScen`, to execute scenarios.
- This tool is useful for starting scenarios from package steps or procedures.
- You can achieve modular development with scenarios and this tool.
 1. Develop a package to perform a common task.
 2. Generate a scenario from this package.
 3. Call the scenario from other packages by using `OdiStartScen`.

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

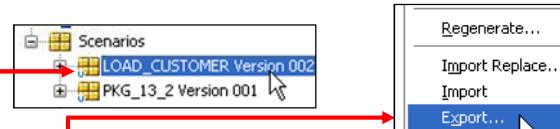
You can launch a scenario within an ODI project from a package step or from within a procedure. You can do this with ODI Tools by using the `OdiStartScen` tool. This enables you to achieve modular development—that is, you can reuse the logic in different packages.

To do this:

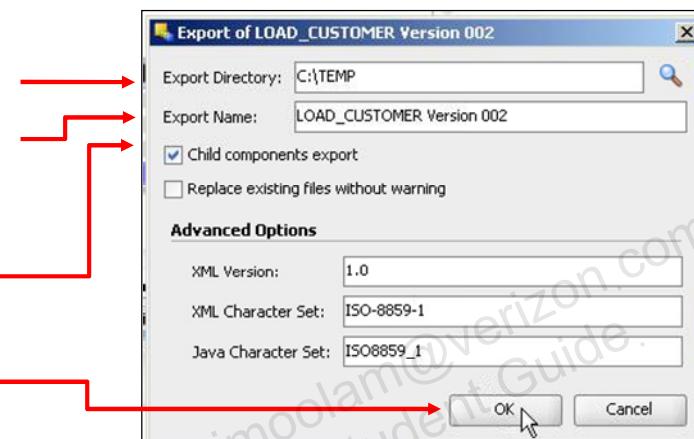
1. Develop a package that performs a common task, such as sending email to an administrator if errors are found.
2. Generate a scenario from this package.
3. When you want to perform the task in other packages, call the generated scenario with the `OdiStartScen` tool.

Exporting a Scenario

1. Right-click the scenario.
2. Select Export.



3. Specify:
 - Export directory
 - Export file name (without .xml)
 - Export child objects (always set to true)



4. Click OK.

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

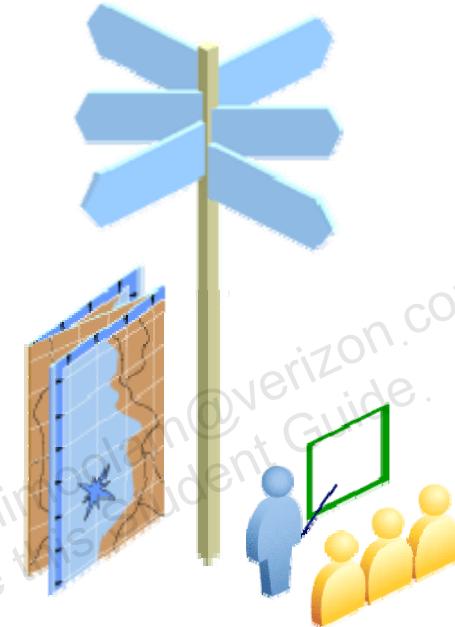
To export a scenario, perform the following:

1. Right-click the scenario.
2. Select **Export** from the context menu. The Export window appears.
3. Specify the directory where you want to export the scenario. You can use the browse button to select the directory by navigating to it. Specify the file name to export the scenario to. The **.xml** extension is automatically added to it. Select the **Child components export** check box. Always do so; otherwise, steps that make up the scenario are not exported.
4. Click **OK** to begin the export.

Note: If schedules are attached to a scenario, you are prompted to decide whether you want to export/import the schedules as well.

Agenda

- Scenarios
- Managing Scenarios
- **Preparing for Deployment**



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You now look at how to prepare scenarios for deployment. This is the final stage before the production team deploys the scenarios.

Preparing Scenarios for Deployment

1. Test your packages, mappings, and so on.
 - Unit and integration testing
2. Generate or regenerate scenarios.
3. Test the scenarios.
 - From the GUI and/or from the command line
4. Export the scenarios.
 - For delivery on remote sites
5. Set up schedules.
 - This can be done by the production team.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The basic process for preparing a scenario for deployment into a production environment is as follows:

1. Test your packages or mappings from within ODI. You should test that the individual mappings or procedures work, and then test that they work together in a package.
2. Generate your scenarios. If you fix or update scenarios, you are probably regenerating, then generating.
3. Test the scenarios. You can either execute them directly from the GUI or test them from a script file.
4. Now that the scenarios are ready, you may want to export them for deployment at the remote production sites.
5. The production team may then want to set up schedules to run the scenarios automatically at regular intervals.

Automating Scenario Management

- ODI provides API tools to automate scenario production:
 - OdiGenerateAllScen
 - OdiExportScen
 - OdiImportScen
 - OdiDeleteScen
 - OdiStartScen
- You can create packages to automate scenario deployment.

ORACLE®

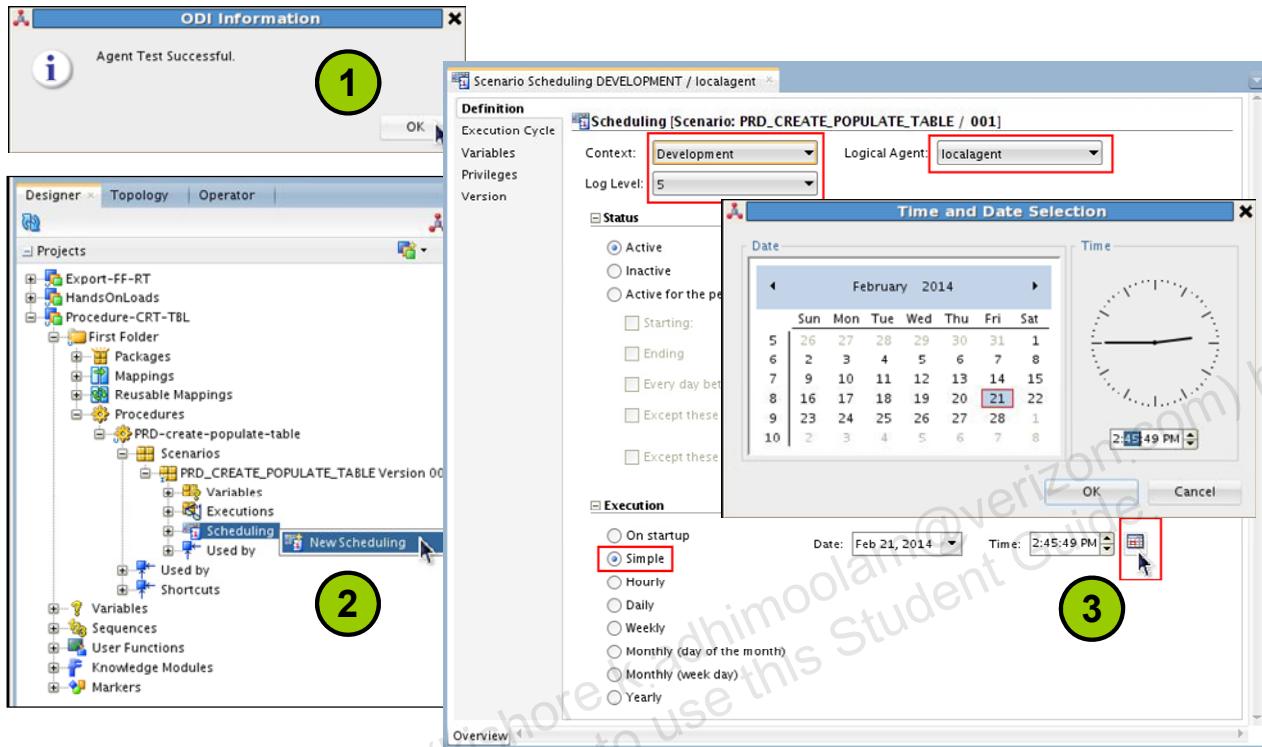
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

To speed up the whole process, you can use various tools provided by the Sunopsis API. You can combine these tools into packages or procedures.

- OdiGenerateAllScen generates or regenerates all scenarios in your project that match a certain criteria.
- OdiExportScen exports a scenario that you generated.
- OdiImportScen does the opposite. It loads an exported scenario from an Extensible Markup Language (XML) file into the repository. The last two tools are useful to automatically move scenarios between repositories.
- OdiDeleteScen removes a generated scenario from the repository.
- Lastly OdiStartScen runs a scenario.

By creating packages that use one or more of these tools, you can set up powerful automatic scenario delivery systems.

Scheduling the ODI Scenario



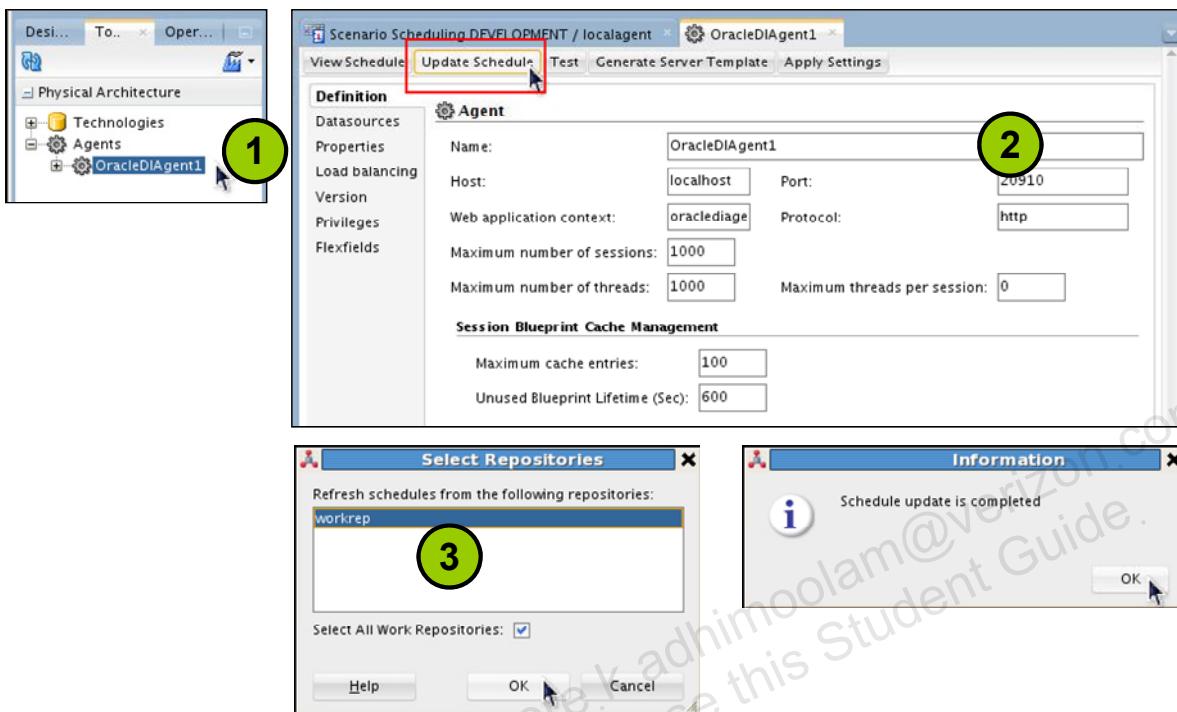
ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

After a scenario is successfully created, you can now execute the scenario directly, use the scenario within a package, or schedule the scenario in ODI. You can schedule the executions of your scenarios by using the Oracle Data Integrator built-in scheduler or an external scheduler. To schedule the scenario by using the built-in scheduler, do the following:

1. If it is not started, start the ODI Agent (execute the command: `agent -NAME=localagent`).
2. Insert Scheduling for your ODI scenario.
3. Schedule the scenario by setting an appropriate execution time for your scenario.

Scheduling the ODI Scenario



ORACLE

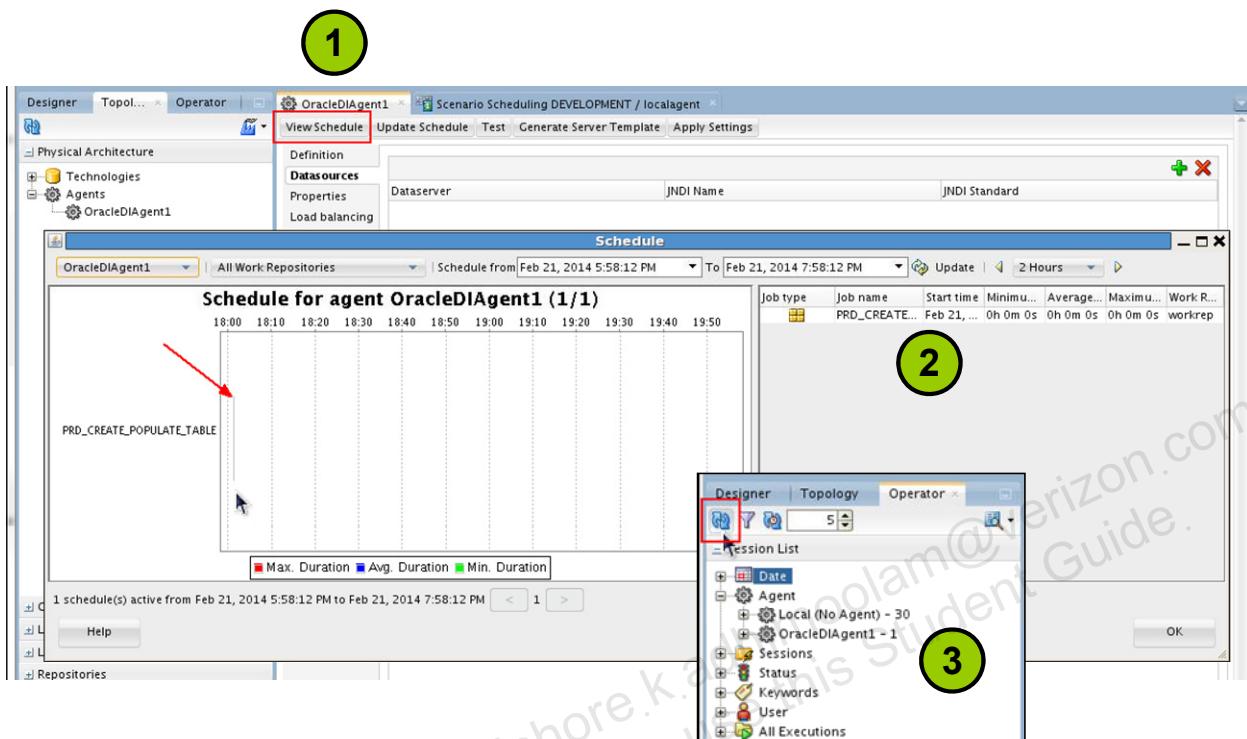
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

After your scenario is scheduled, update scheduling by using the ODI Agent in Topology Navigator.

1. In Topology > Physical Architecture, right-click your agent and select **Edit**.
2. Click the **Update Schedule** button.
3. Select the repository from which you refresh schedules. Click **OK** in the *ODI Information* window.

Note: Alternatively, right-click to display the agent menu and select “Update Schedule.”

Scheduling the ODI Scenario



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

After your schedule is updated, view the schedule and verify the execution of the scheduled scenario.

1. To view the schedule, click the **View Schedule** button.
2. View the screen that appears. It shows you the scheduling information.
3. To verify the execution of the scheduled scenario, click the ODI **Operator** tab to open ODI Operator. In ODI Operator, click the **Session List** tab. Wait until the scheduled execution time to view the execution results, and then click the **Refresh** icon. Expand: Agent > OracleDIAGent1 - 1 and view the execution results for your scheduled scenario. The number (-1) signifies that there is only one log entry (session) for this agent. That number will be incremented as more jobs are run on that agent.

Scheduling ODI Scenario with External Scheduler

- Use the command-line command `startscen` from the external scheduler.
- Use the web service interface for triggering the scenario execution.



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

To start a scenario with an external scheduler, you can either use the command-line command `startscen` from the external scheduler or use the web service interface for triggering the scenario execution. If a scenario completes successfully, the return code will be 0. If not, the return code will be different than 0. This code will be available in:

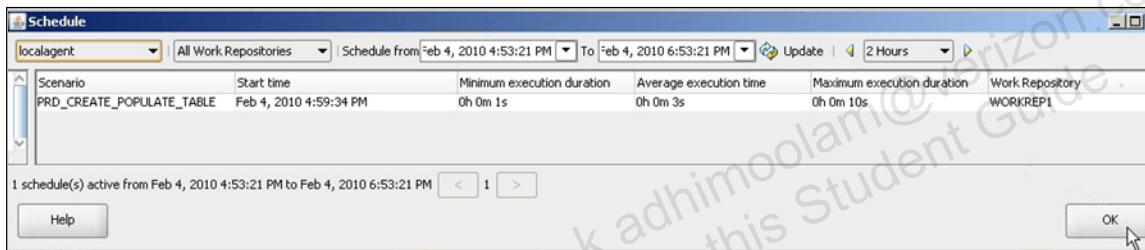
- The return code of the command-line call.
- The error message, if any, is available on the standard error output.
- The SOAP response of the web service call. The web service response includes also the session error message, if any.

See *Oracle Fusion Middleware Developer's Guide for Oracle Data Integrator 12c Release 1* for more information.

Note: You can also schedule execution of a scenario from a web service, for example, from a BPEL process within SOA.

Managing Schedules

- A schedule is always attached to one scenario.
- You can also import an already existing schedule during the scenario import.
- You can view the scheduled tasks of all your agents or you can view the scheduled tasks of one agent.
- Schedules can be also displayed in Operator Navigator.



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

See *Oracle Fusion Middleware Developer's Guide for Oracle Data Integrator 12c Release 1* for more information.

Quiz

Which of the following is *not* true about an ODI scenario?

- a. Scenarios are created by generating code from ODI objects.
- b. Scenarios cannot be modified.
- c. You can retain several versions of the same scenario.
- d. You can run the same scenario from only one context.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Answer: d

Scenarios are context-independent. You can generate a scenario to load data into a sales database and then run the same scenario in different contexts to load the sales databases for different sites.

Summary

In this lesson, you should have learned how to:

- Describe the purpose of an ODI scenario
- Describe the properties of an ODI scenario
- Generate and regenerate an ODI scenario
- Execute an ODI scenario from different sources
- Prepare ODI scenarios for deployment
- Perform automated scenario management



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Checklist of Practice Activities

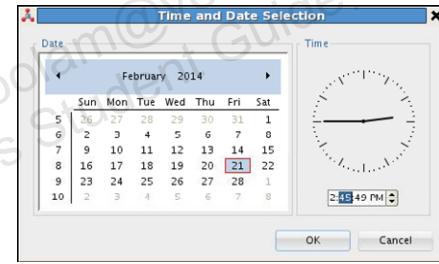
- ✓ 1-1: Logging In and Help Overview
- ✓ 2-1: Creating and Connecting to ODI Master and Work Repositories
- ✓ 3-1: Configuring Standalone Agents Using the Common Admin Model
- ✓ 4-1: Working with Topology
- ✓ 5-1: Setting Up a New ODI Project
- ✓ 6-1: Creating Models by Reverse-Engineering
- ✓ 7-1: Checking Data Quality in the Model
- ✓ 8-1: Creating ODI Mapping: Simple Transformations
- ✓ 9-1: Creating ODI Mapping: Complex Transformations
- ✓ 9-2: Creating ODI Mapping: Implementing Lookup
- ✓ 10-1: Creating ODI Mapping: Exporting a Flat File to a Relational Table
- ✓ 11-1: Using Native Sequences with ODI Mapping
- ✓ 11-2: Using Temporary Indexes
- ✓ 11-3: Using Sets with ODI Mapping
- ✓ 12-1: Creating and Using Reusable Mappings
- ✓ 12-2: Developing a New Knowledge Module
- ✓ 13-1: Creating an ODI Procedure
- ✓ 14-1: Creating an ODI Package
- ✓ 14-2: Using ODI Packages with Variables and User Functions
- ✓ 15-1: Debugging Mappings
- 16-1: **Creating and Scheduling an ODI Scenario**
- 17-1: Using Load Plans
- 18-1: Enforcing Data Quality with ODI Mappings
- 19-1: Implementing Changed Data Capture
- 20-1: Setting Up ODI Security
- 20-2: Integration with Enterprise Manager and Using ODI Console

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Practice 16-1: Creating and Scheduling Scenarios

1. Generate a scenario for the PRD-create-populate-table procedure.
2. In Topology Navigator, verify the connection to ODI Agent.
3. In Designer Navigator, schedule the scenario with ODI Agent.
4. Switch back to Topology Navigator to update the scheduling of the agent.
5. In Operator Navigator, after the scheduled execution time, verify that the scenario has executed.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In this practice, you learn how to create and schedule an ODI scenario to run the procedure you created in Practice 12-1. Start in the Designer Navigator's Projects tab.

17

Using Load Plans

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Describe the benefit of using load plans to load data most efficiently
- Use the load plan editor to define a load plan
- Design a sequence of hierarchical steps for the load plan
- Define restart behavior for handling errors in step execution



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

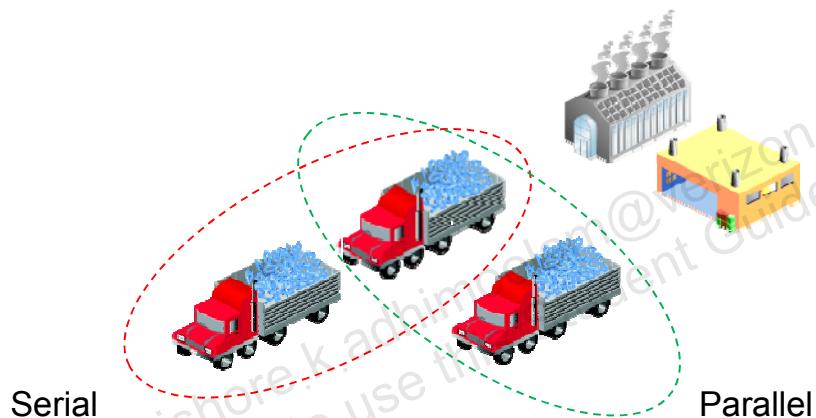
In this lesson, you learn the benefits of using load plans to organize Scenario executions. You see how to use the load plan editor to create, manage, and use load plans.

The topics for this lesson are:

- What are load plans?
- Load plan editor
- Load plan step sequence
- Defining restart behavior

Should You Organize Executions with Load Plans?

- A data warehousing team has learned that the execution of scenarios can now be organized by load plans.
- In very large data warehouses with thousands of tables to populate, can load plans execute hundreds of scenarios so that the data will flow from source to target most efficiently?



ORACLE

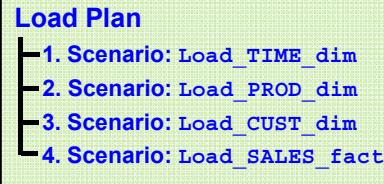
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

When ODI is used to populate very large data warehouses, it is common to have thousands of tables being populated using hundreds of scenarios. The execution of these scenarios has to be organized in such a way that the data throughput from the sources to the target is the most efficient within the batch window. Load plans help organize the execution of scenarios in a hierarchy of sequential and parallel steps for these types of use cases.

Load plans are objects that organize, at a high level, the execution of packages and scenarios. Load plans provide features for parallel, sequential, and conditional scenario execution, restartability, and exception handling. Load plans can be created and modified in production environments.

What Are Load Plans?

- Load plans are executable objects that contain a *hierarchy of steps* that can be executed conditionally, in parallel, or in series.
 - A load plan is the largest executable object in ODI, using **scenarios** in its steps.
- Packages, mappings, variables, and procedures can be added to load plans for executions in scenarios.
 - Load plans are not substitutes for packages or scenarios, but are used to organize, at a higher level, the execution of packages and scenarios.
- Load plans also support exception handling strategies for scenarios that end in error.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

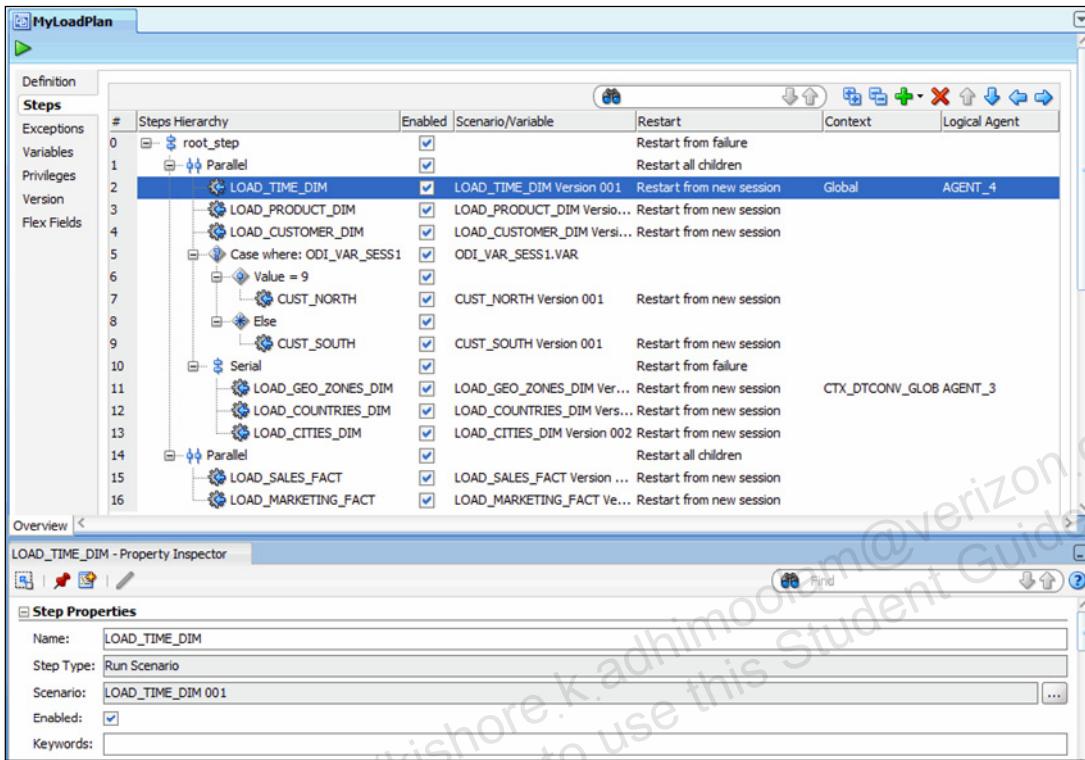
A load plan is an executable object in Oracle Data Integrator that can contain a hierarchy of steps that can be executed conditionally, in parallel, or in series. The leaves of this hierarchy are scenarios. The set of scenarios that makes up a load plan can involve the execution of packages, mappings, variables, and procedures.

Load plans also support exception handling strategies for scenarios that end in error.

Load plans can be started, stopped, and restarted from a command line, from Oracle Data Integrator Studio, Oracle Data Integrator Console, or a web service interface. They can also be scheduled using the runtime agent's built-in scheduler or an external scheduler. When a load plan is executed, a load plan instance is created. Each attempt to run this load plan instance is a separate load plan run.

A load plan can be modified in production environments and steps can be enabled or disabled according to the production needs. Load plan objects can be designed and viewed in the Designer and Operator Navigators. Various design operations (such as create, edit, delete, and so forth) can be performed on a load plan object if a user connects to a development Work Repository, but some design operations will not be available in an execution Work Repository. After it is created, a load plan is stored in the Work Repository. The load plan can be exported and then imported to another repository and executed in different contexts.

Load Plan Editor



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

This screenshot shows a sample load plan created in ODI for loading a data warehouse.

- Dimensions are loaded in parallel. This includes the `LOAD_TIME_DIM`, `LOAD_PRODUCT_DIM`, and `LOAD_CUSTOMER_DIM` scenarios, the geographical dimension and, depending on the value of the `ODI_VAR_SESS1` variable, the `CUST_NORTH` or `CUST_SOUTH` scenario.
- The geographical dimension consists of a sequence of three scenarios (`LOAD_GEO_ZONE_DIM`, `LOAD_COUNTRIES_DIM`, and `LOAD_CITIES_DIM`).
- After the dimensions are loaded, the two fact tables are loaded in parallel (`LOAD_SALES_FACT` and `LOAD_MARKETING_FACT` scenarios).

The load plan editor provides a single environment for designing load plans. The load plan steps are added, edited, and organized on the Steps tab of the load plan editor. The Steps Hierarchy table defines the organization of the steps in the load plan. Each row in this table represents a step and displays its main properties.

You can drag components such as packages, integration mappings, variables, procedures, or scenarios from the Designer Navigator into the Steps Hierarchy table for creating Run Scenario steps for these components.

You can also use the *Add Step Wizard* or the *Quick Step* tool to add Run Scenario steps and other types of steps into this load plan.

Load Plan Steps

Step Type	Description	Possible Child Steps
Serial	Defines a serial execution of its child steps. Child steps are ordered and a child step is executed only when the previous one is terminated. The root step is a Serial step.	<ul style="list-style-type: none">• Serial step• Parallel step• Run Scenario step• Case step
Parallel	Defines a parallel execution of its child steps. Child steps are started immediately in their order of Priority.	<ul style="list-style-type: none">• Serial step• Parallel step• Run Scenario step• Case step
Run Scenario	Launches the execution of a Scenario.	This type of step cannot have a child step.
Case	The combination of these steps allows conditional branching based on the value of a variable.	<ul style="list-style-type: none">• When step• Else step
When	Note: If you have several When steps under a Case step, only the first enabled When step that satisfies the condition is executed.	<ul style="list-style-type: none">• Serial step• Parallel step• Run Scenario step• Case step
Else	If no When step satisfies the condition or the Case step does not contain any When steps, the Else step is executed.	<ul style="list-style-type: none">• Serial step• Parallel step• Run Scenario step• Case step

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

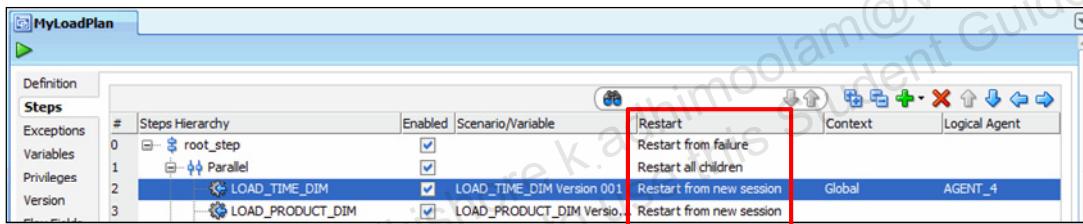
A load plan is made up of a sequence of several types of steps. Each step can contain several child steps.

Depending on the step type, the steps can be executed conditionally, in parallel, or sequentially.

By default, a load plan contains an empty root serial step. This root step is mandatory and the step type cannot be changed.

Defining the Restart Behavior

Step Type	Restart Value	Default?
Serial	Restart all children	
	Restart from failure	
Parallel	Restart all children	Yes
	Restart from failed children	
Run Scenario	Restart from new session	Yes
	Restart from failed step	
	Restart from failed task	



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The Restart Type option defines how a step in error restarts when the load plan is restarted.

You can define the Restart Type parameter in the Exception Handling section of the Properties Inspector.

Depending on the step type, the Restart Type parameter can take the values listed in this table.

Serial Step Type

- Restart all children:** When the load plan is restarted and if this step is in error, the sequence of steps restarts from the first one.
- Restart from failure:** When the load plan is restarted and if this step is in error, the sequence of child steps starts from the one that has failed.

Parallel Step Type

- Restart all children:** When the load plan is restarted and if this step is in error, all the child steps are restarted, regardless of their status. This is the default value.
- Restart from failed children:** When the load plan is restarted and if this step is in error, only the failed child steps are restarted in parallel.

Run Scenario Step Type

- **Restart from new session:** When restarting the load plan and this Run Scenario step is in error, start the scenario and create a new session. This is the default value.
- **Restart from failed step:** When restarting the load plan and this Run Scenario step is in error, restart the session from the step in error. All the tasks under this step are restarted.
- **Restart from failed task:** When restarting the load plan and this Run Scenario step is in error, restart the session from the task in error.

Are Load Plans Substitutes for Packages or Scenarios?

- Load plans are *not* substitutes for packages or scenarios. They are used to organize, at a higher level, the execution of packages and scenarios.
- Unlike packages, load plans provide native support for parallelism, restartability, and exception handling.
- Load plans are moved to production as-is, whereas packages are moved in the form of scenarios.
- Load plans can be created in production environments.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Load plan instances and load plan runs are similar to sessions. The difference is that when a session is restarted, the existing session is overwritten by the new execution. The new load plan run does not overwrite the existing load plan run. It is added after the previous load plan runs for this load plan instance. The load plan instance cannot be modified at run time.

A scenario is an atomic runtime component that is given to the production. When in production, the production administrator can orchestrate the various scenarios using load plans in order to optimize the time taken to execute the ETL jobs in the given time frame and ease the administration of all the ETL jobs.

Benefits of Using Load Plans

- Rely on load plans to orchestrate the execution of the various scenarios produced by ETL developers as part of their own projects.
 - Organize production globally instead of per project.
- Design load plans to handle the parallelism of scenarios in order to best use hardware resources.
- Design load plans to handle serial steps to avoid overload of the system with too many scenarios at a given time.
- Modify load plans to take into account the real execution times and tune the load on the systems.
- Rely on load plans to restart only what is needed to be restarted in case of failure.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The slide lists examples of how a production administrator can use load plans.

Handling Failed Load Plans

- It is possible to mark failed load plan steps as Complete.

Steps Hierarchy	Status	Duration	Start	End
root_step	✗	00:16	10:31:28	10:31:44
LOAD_COUNTRIES	✓	00:04	10:31:28	10:31:32
LOAD_REGIONS	✓	00:05	10:31:32	10:31:37
LOAD_CITIES	✓	00:03	10:31:37	10:31:40
LOAD_CUSTOMERS	✗	00:02	10:31:41	10:31:43

- Load plans provide enhanced handling of variables.
- Load plan variables that are not used in a load plan can now be hidden to improve the readability of load plans.

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

It is possible to change the execution status of a failed load plan step from **Error** to **Complete** on the Steps tab of the load plan editor.

This allows this particular load plan step to be ignored the next time the load plan run is restarted. This is useful, for example, when the error causing this load plan step to fail cannot be fixed at the moment. However, you want to execute the rest of the load plan regardless of this load plan step status. By changing the status to Done, the step will be ignored on the next execution.

Quiz

Which of the following statements are true?

- a. Load plans are not substitutes for packages or scenarios.
- b. Load plans organize the execution of packages and scenarios at a higher level.
- c. Unlike packages, load plans provide native support for parallelism, restartability, and exception handling.
- d. Load plans are moved to production as is, whereas packages are moved in the form of scenarios.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Answer: a, b, c, d

A load plan is an executable object in ODI that can contain a hierarchy of steps. The leaves of this hierarchy are scenarios. The set of scenarios that make up a load plan can involve the execution of packages, mappings, variables, and procedures.

Summary

In this lesson, you should have learned how to:

- Describe the benefit of using load plans to most efficiently load data
- Use the load plan editor to define a load plan
- Design a sequence of hierarchical steps for the load plan
- Define restart behavior for handling errors in step execution



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Checklist of Practice Activities

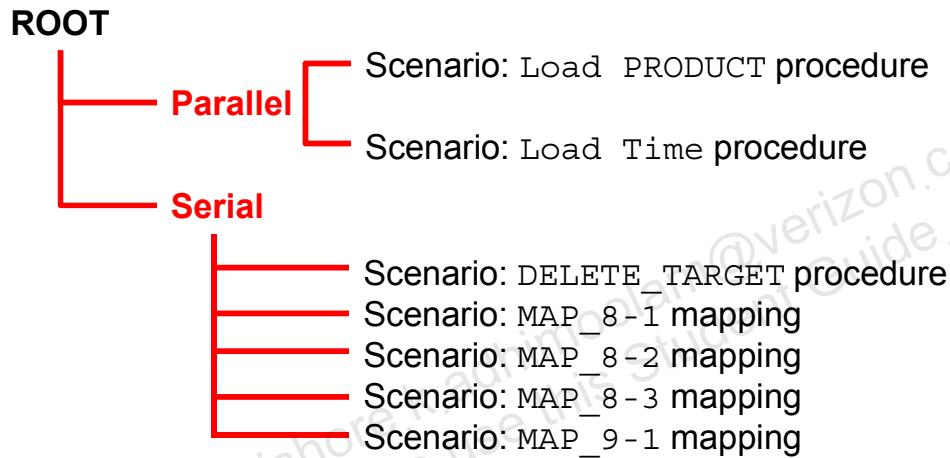
- ✓ 1-1: Logging In and Help Overview
- ✓ 2-1: Creating and Connecting to ODI Master and Work Repositories
- ✓ 3-1: Configuring Standalone Agents Using the Common Admin Model
- ✓ 4-1: Working with Topology
- ✓ 5-1: Setting Up a New ODI Project
- ✓ 6-1: Creating Models by Reverse-Engineering
- ✓ 7-1: Checking Data Quality in the Model
- ✓ 8-1: Creating ODI Mapping: Simple Transformations
- ✓ 9-1: Creating ODI Mapping: Complex Transformations
- ✓ 9-2: Creating ODI Mapping: Implementing Lookup
- ✓ 10-1: Creating ODI Mapping: Exporting a Flat File to a Relational Table
- ✓ 11-1: Using Native Sequences with ODI Mapping
- ✓ 11-2: Using Temporary Indexes
- ✓ 11-3: Using Sets with ODI Mapping
- ✓ 12-1: Creating and Using Reusable Mappings
- ✓ 12-2: Developing a New Knowledge Module
- ✓ 13-1: Creating an ODI Procedure
- ✓ 14-1: Creating an ODI Package
- ✓ 14-2: Using ODI Packages with Variables and User Functions
- ✓ 15-1: Debugging Mappings
- ✓ 16-1: Creating and Scheduling Scenarios
- **17-1: Using Load Plans**
- 18-1: Enforcing Data Quality with ODI Mappings
- 19-1: Implementing Changed Data Capture
- 20-1: Setting Up ODI Security
- 20-2: Integration with Enterprise Manager and Using ODI Console

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Practice 17-1: Using Load Plans

1. Use the load plan editor to create a load plan with a set of steps and substeps.
2. Include the definition of restart behavior.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In this practice, you learn how to create and use a load plan.

Unauthorized reproduction or distribution prohibited. Copyright© 2015, Oracle and/or its affiliates.

KISHORE ADHIMOOLAM (kishore.k.adhimoolam@verizon.com) has
a non-transferable license to use this Student Guide.

18

Enforcing Data Quality with ODI

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Describe why and when to enforce data quality
- List the different types of data quality business rules that Oracle Data Integrator (ODI) manages
- Describe Data Quality System within ODI
- Describe how data quality is managed in a mapping
- Implement flow control during the execution of a mapping
- Implement static control after the execution of a mapping
- Enforce constraints and review erroneous records
- Manage an error table



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

This lesson is an overview of monitoring and improving your data quality by using ODI. In this lesson, you learn why you should enforce data quality and in which situations to do so. The lesson will also cover the different types of rules that can be enforced with ODI and how to enforce these rules by using the various kinds of data quality control.

Agenda

- **Data Quality**
 - Why and When?
- Business Rules for Data Quality
- Enforcing Data Quality with ODI



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In the first few slides of this lesson, you find an answer to why and when you should enforce data quality.

Why Data Quality?

- Data quality is often neglected because of:
 - Development cost
 - Overconfidence in the actual data
- Monitoring data quality is critical.
 - Low-quality data propagates and contaminates your information system (IS).
- Data integration cannot be separated from data quality.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Companies often neglect monitoring and enforcing the overall quality of data processed by the integration processes. There are two reasons for this:

- The development of data-cleansing operations may be deemed too expensive.
- The IS teams often consider that the information residing in certain applications must closely conform to the defined business rules. This leads to the belief that any risk of incorrect data is minimal.

Data quality concerns become even more critical when new applications are integrated in the information system (IS). Indeed, data—even “bad” data—propagates right across the IS. It may then “contaminate” other applications unless a systematic and reliable approach to data quality is implemented.

When to Enforce Data Quality

- The IS can be broken into three subsystems:
 - Source application(s)
 - Data integration process(es)
 - Target application(s)
- Data quality should be managed in all three subsystems.
- ODI provides the solution for enforcing quality in all three subsystems.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Assume that every information system (IS) can be broken down into a number of subsystems, the largest elements being the source application(s) and the process(es) that transform and integrate data, and the target application(s).

In such a system, data quality can, and should, be managed in all of these subsystems. In the following slides, you learn how to achieve this by combining a number of approaches and technical solutions.

Data Quality in Source Applications

- Data is extracted from the source application.
 - Sources can also be targets in some processes.
- Quality is usually enforced at the application level.
 - GUI, Storage (DBMS constraints)
- Events may create inconsistent data due to:
 - Maintenance cycles
 - Deficient control mechanisms
- Data may be unsuitable for propagation. Data quality control is, therefore, important before integration.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In the context of ODI, source applications are the applications from which data is extracted by an integration process. In this respect, it is important to note that the source application of one integration process may also be the target application for a different process.

Data quality in source applications is usually checked by the application itself. This may be implemented by surface checks at the graphical user interface (GUI) level. Also, internal business rules may be enforced on the data that the source application manages before storing it permanently. This is the case for DBMS constraints.

However, events such as maintenance cycles may produce inconsistent data on these source applications. The control mechanisms mentioned previously can also be deficient.

Consequently, you should not implicitly assume that the quality of the data contained in the source applications is suitable for propagation through the IS.

Therefore, you are strongly advised to run control processes on the data in source applications to verify the integrity and quality of the stored information before allowing it to be injected into the integration processes.

Data Quality Control in the Integration Process

- Prevents poor data from being propagated across the IS
 - It is retained in the sources.
- “Data Quality Firewall”:
 - Valid data passes through the “firewall.”
 - Invalid data is rejected and stored in error tables.
 - Rejected data can be analyzed or fixed.
 - Fixed data can be processed and/or recycled.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Controlling data quality as part of the integration process ensures that potentially poor quality data is not propagated across the IS. Instead, it is retained in the source applications.

The idea behind such an approach is to implement a sort of “data quality firewall.” This is an integral part of the data integration process, allowing only validated data to pass through.

Conversely, any invalid data is blocked and stored in specific error tables, for analysis later and/or recycling.

Data Quality in the Target Applications

- The target applications can be strategic.
 - Data warehouse and data hub
 - Data quality in these target applications is critical.
- Controls may be bypassed depending on the:
 - Type of target application
 - Integration approach
- Checking data quality after integration is important.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In many integration situations, the target application is a data warehouse that is used as the basis of reports and analytics. Alternatively, it may be a data hub/operational datastore, both of which are particularly important for decision-making purposes.

The quality of data contained in these systems is, therefore, critical to the organization—any discrepancy can have consequences that go well beyond the application itself.

Furthermore, depending on the type of the target application, and the data integration approach implemented, the integrity and quality controls inherent to the target application may be bypassed. It is therefore equally important to verify the data managed by the target application subsequently. This process is similar to performing data control and verification in source applications.

Agenda

- Data Quality
- **Business Rules for Data Quality**
- Enforcing Data Quality with ODI



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You will now examine the different types of rules that can be enforced with ODI.

Data Quality Business Rules

- Are defined by designers and business analysts
- Are stored in the Metadata Repository
- May be applied to application data
- Are defined in two ways:
 - Automatically retrieved with other metadata
 - Rules defined in the databases
 - Obtained by reverse-engineering
 - Manually added by designers
 - User-defined rules



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

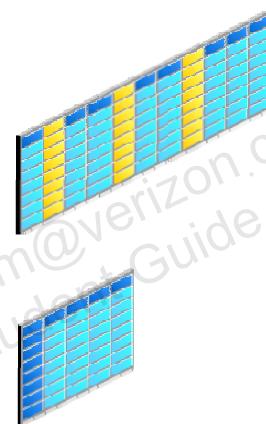
ODI enables application designers and business analysts to define business rules that target data quality maintenance. These rules are stored in the centralized ODI Metadata Repository. They may be applied to the application data to guarantee the overall integrity and consistency of information in the organization.

ODI enables data quality rules to be created in two ways:

- By automatically reverse-engineering metadata from source application data. ODI detects existing data quality rules defined at the database level (for example, foreign keys).
- It also enables application designers to define additional, user-defined rules in the repository.

From Business Rules to Constraints

- Deduplication rules:
 - Primary keys
 - Alternate keys
 - Unique indexes
- Reference rules:
 - Simple:
column A = column B
 - Complex:
column A = function (column B, column C)
- Validation rules:
 - Mandatory columns
 - Conditions



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The business rules for data quality can include:

- Deduplication rules (unique keys). Such rules can be used to identify and isolate inconsistent data sets, as shown in the following examples:
 - “The list of customers with the same email address”
 - “The list of products having the same item code and item family code”
 - These rules are defined in ODI as primary keys, alternate keys, or unique indexes.
- Simple and complex reference rules. Examples of these are:
 - “The list of customers that are linked to nonexistent sales reps”
 - “The set of orders that are linked to customers that are marked as Invalid”
 - These rules are defined in ODI as references.
- Validation rules. These rules enforce consistency at the record level. Example:
 - “The list of customers with an empty zip code”
 - “The list of web prospect responses with an invalid email address”
 - These rules are defined in ODI as conditions or mandatory columns.

Agenda

- Data Quality
- Business Rules for Data Quality
- **Enforcing Data Quality with ODI**
 - Enabling Static or Flow Control for a Mapping
 - Setting the Options
 - Selecting Which Constraints to Enforce
 - Reviewing Erroneous Records

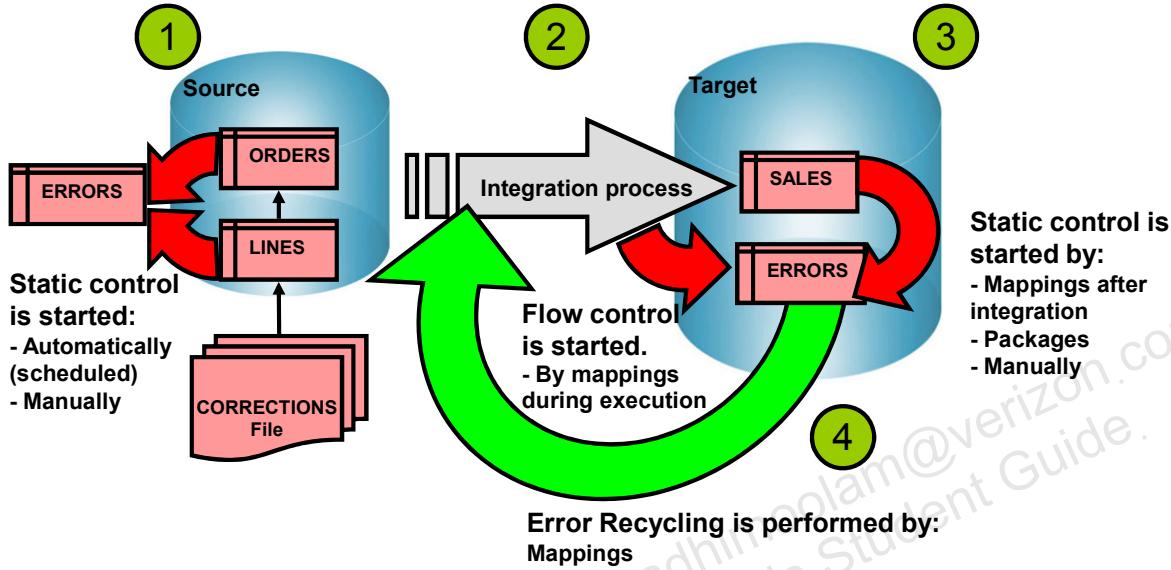


ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In the following slides, you learn to enforce the different types of rules with ODI.

Data Quality System: Overview



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You use a simple example of a source and one integration process pushing data to a target, to fit together the concepts that were just discussed and form the data quality system.

1. First, data quality is enforced on the sources by using static control. This static control enforces rules on the sources and isolates erroneous data into error tables. This control is triggered either manually or automatically. It can be scheduled regularly.
2. The integration process integrates data from the sources to the target. The integration unit of ODI—the mapping—enforces data quality on the data flow by using flow control before the data is integrated in the target. The rules enforced are those of the target. The invalid data is isolated in an error table and not inserted into the target.
3. You can perform static control on the data in the target. This operation can be launched by mappings after the integration stage—post-integration static control—or similarly to the controls on the source.
4. Finally, mappings recycle the erroneous data from the previous session into the flow. They are inserted back into the flow and are revalidated against the rules.

Depending on the strategy adopted, these invalid records may be targeted by an automatic data-cleansing mapping. Or, they may be edited and forwarded to the application end users to indicate the entries in the original application that require corrective measures.

Static and Flow Controls: Differences

- Static control (“static data check”):
 - Checks whether data contained in a datastore respects its constraints
 - Requires a primary key on the datastore
- Flow control (“dynamic data check”):
 - Enforces target datastore constraints on data in the flow
 - Requires an update key defined in the mapping
 - Enables recycling of erroneous data back into the flow



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Now that you know the similarities between static and flow controls, consider the differences:

- Static control operates on data that stays in static datastores.
 - Because data is left in the datastores, this check provides only information. Thus, you can think of it as a “static data check.”
 - You can perform this check at any time, or you can trigger a static control automatically after integration.
 - You need to define a primary key on the datastore that is being checked.
 - This control is suitable for enforcing data quality on sources and targets.
- Flow control enforces validity constraints during the course of integration.
 - It operates on a temporary table that has been constructed to closely resemble the target datastore. Thus, it ensures that the data respects the constraints of the target datastore.
 - It does not need a primary key, but still needs a defined way of uniquely referring to the individual rows.
 - It also enables recycling of any errors back into the next execution.
 - This control is suitable for enforcing quality on the integration process.

Data Quality Control: Properties

Static and flow checks:

- Can be triggered:
 - By a mapping (FLOW and/or STATIC)
 - By a package (STATIC)
 - Manually (STATIC)
- Require a Check Knowledge Module (CKM)
- Are monitored through Operator
- Copy invalid rows into the `ERRORS` table
 - Flow control then deletes them from flow.
 - Static control leaves them in datastores.
 - The `ERRORS` table can be viewed from Designer or any SQL tool.



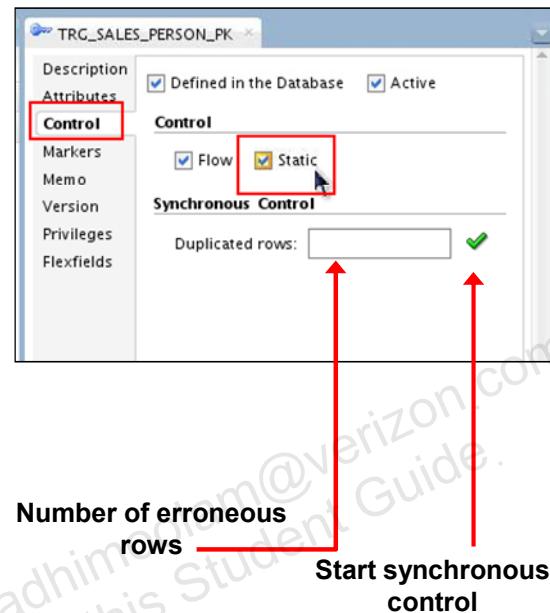
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Now that you know the types of constraints that make up the static and flow controls, you can see the other general properties.

- Static and flow controls can be performed by mappings. However, they occur at different times.
- Only static control can be launched as a step in a package.
- Similarly, only static control can be manually launched by the user in ODI.
- For both types of constraints, you must have an appropriate CKM defined either in the mapping or in the data model.
- The static and flow controls can be monitored through Operator. Here, you can see the result of each individual task. These tasks check each of the constraints that you have defined. Thus, you can see how many rows failed each check.
- Both controls reject erroneous rows into the `ERRORS` table. You can see this table through the Designer window. Alternatively, you can query it directly by SQL from another tool.

Synchronous Control

- “Quick” check
- Available on the Control tab of the Constraint window
 - For keys, references, and conditions
- Useful to check whether a rule is correct
- Requires a SQL-enabled system



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

A synchronous option is available to perform a “quick” check of one constraint. This option is available on the Control tab of the windows for keys, references, and conditions.

By clicking the Check button, you can perform a synchronous check on the constraint, which tells you the number of rows that violate the constraint. Use this button whenever possible to check whether your rule is correct. For example, if this check returns half of your rows as duplicates, you probably have made a mistake.

You should remember that synchronous checks query the data server directly to find the number of invalid rows. This feature works only with SQL-based systems. For instance, you cannot perform a synchronous check on a file-based datastore. Also note that, unlike a normal static check, the result of a synchronous check is not saved anywhere.

What Is a Constraint?

- A constraint is a statement that defines the rules enforced on the data in datastores.
- A constraint ensures the validity of the data in a given datastore and the integrity of the model as a whole.
- Constraints on the datastores are used in mappings to check the validity of the data before, during, and/or after integration in the target.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

A constraint in ODI is a general term for rules that are enforced on data in datastores. Although constraints are often duplicated by rules defined in databases, they act independently. They serve to ensure the validity of data in a given datastore, and thus the integrity of the model as a whole.

Constraints defined on a target datastore can be used by mappings to check the validity of data. These checks are generally implemented before integration into the target. This implementation prevents invalid data from ever reaching the target datastore.

What Can Be Checked?

Types of constraints:

- Uniqueness (primary/alternate key—PK/AK)
 - Is my customer number a unique value?
 - Do I have duplicated values in my invoice number field?
- Referential Constraint (foreign key—FK)
 - Does the customer number in my INVOICE table correspond to the customer number in my CUSTOMER table?
- Conditions (Check—CK)
 - The driver's license candidate must be older than 16 years.
 - The date of the last update must be more recent than the creation date.
- Mandatory Fields (Not null)
 - The name field must not be null.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

As mentioned in an earlier slide, a constraint in ODI refers to any rule for data quality that can be enforced. So, what are the types of constraints?

Uniqueness constraints are often the most critical. These constraints correspond to primary keys or alternate keys in databases. A uniqueness constraint ensures, for example, that an ID number given to a customer is not shared by any other customer. Similarly, it can check that the same number is not given to two different invoices.

Referential constraints enforce relationships between data items and correspond to foreign keys. For example, the customer number column in your INVOICE table must correspond to a valid customer number in your CUSTOMER table.

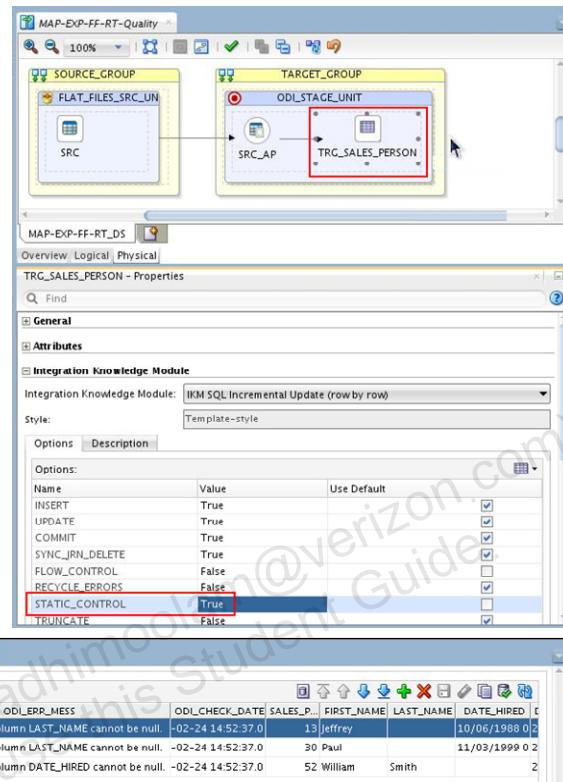
Conditions are sometimes known as check constraints. These are simple rules that can be checked only by looking at that row. For example, a candidate for a driving license must be older than 16 years. This can be checked only by looking at the relevant column. You can also compare between several columns in a row. For instance, the last update must be more recent than the creation date.

Mandatory fields correspond to “NOT NULL” columns in certain databases. This is the simplest kind of check: a given column passes if it contains any value other than NULL.

Enforcing Data Quality in a Mapping

The general process is as follows:

1. Enable Static/Flow Control.
2. Set the options.
3. Select the constraints to enforce:
 - Table constraints
 - Not null columns
4. Review the erroneous records.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

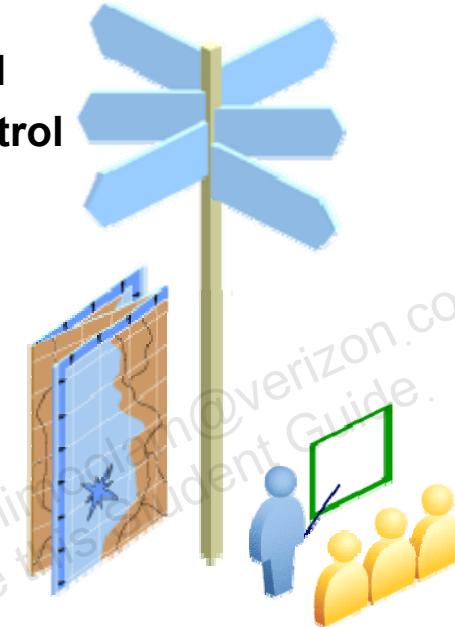
You have seen what data quality controls can do; you will now learn how to implement them in a mapping. Note that there are other ways of implementing quality controls. For example, you can run static controls manually. For now, however, static control is referred to in the context of a mapping. There are a few subtle differences between executing static controls in mappings and executing them manually or in other ways.

For this overview, you begin by defining the general steps. Assume that you have created the constraints that will be enforced.

1. First, you enable flow control or static control for the mapping on the Physical tab.
2. Then, you set the options for the Control Knowledge Module that you chose on the Controls tab.
3. Lastly, you tell ODI which constraints you would like to enforce for the given datastore. This is at the bottom of the Controls tab.
4. After execution, you should review the records that have been moved into the ERRORS table. You can do this in Operator or in the Models view in Designer.

Agenda

- Data Quality
- Business Rules for Data Quality
- **Enforcing Data Quality with ODI**
 - **Enabling Static or Flow Control for a Mapping**
 - Setting the Options
 - Selecting Which Constraints to Enforce
 - Reviewing Erroneous Records



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The first step in this general process is to enable static control or flow control for a mapping.

Setting Up Static or Flow Control

Three IKM options control how data quality constraints are enforced by a mapping:

- FLOW_CONTROL triggers a dynamic data check during execution.
- STATIC_CONTROL triggers a static data check after the mapping execution.
 - The CKM selected on the mapping performs both checks.
- RECYCLE_ERRORS forces invalid data detected in the previous executions to be recycled into the mapping flow.

Note: Not all IKMs support flow control.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

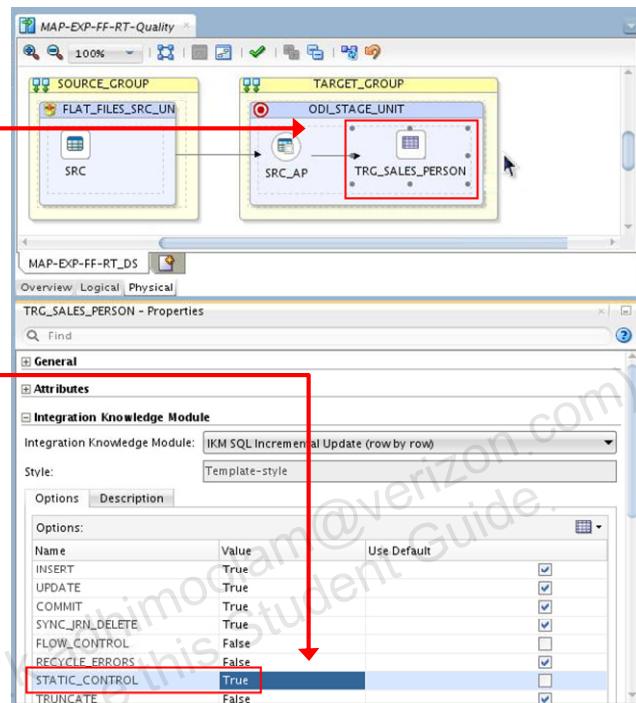
The basic options for static and flow controls are controlled by the Integration Knowledge Module (IKM) in the mapping. Thus, you click Target on the Physical tab to set these options.

- To enable a dynamic data check, you set the FLOW_CONTROL option to “Yes.” The IKM calls the selected CKM before inserting data into the target. This option removes invalid data from the data stream.
- For a static data check, you set the STATIC_CONTROL option to “Yes.” Now the IKM calls the selected CKM after inserting data into the target. This option does not remove invalid data from the target datastore. It reports invalid data in the ERRORS table.
- Finally, you can set the RECYCLE_ERRORS option. This option causes the bad data that was detected on previous executions to be reused. Imagine that you run your mapping with 100 rows of source data, and 10 errors are detected. These are moved to the ERRORS table. You run the same mapping again with the same data. The error rows are added to the input stream. This means that there are now 110 rows of input data. If nothing has changed, 20 bad rows are now in the ERRORS table.

Note: Not all IKMs support flow control, there are a number of new 12c IKMs that do not. The way to find out is to look for the FLOW_CONTROL IKM option. If it is not present, flow control is not supported.

Enabling Static or Flow Control

1. Click the Physical tab.
2. Select the target datastore.
 - The IKM Properties panel appears.
3. Set the FLOW_CONTROL and/or STATIC_CONTROL IKM options to “True.”
4. Set RECYCLE_ERRORS to “True,” if you want to recycle errors from previous runs.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The steps to enable static control or flow control for a mapping are as follows:

1. Click the Physical tab of the mapping.
2. Select the target. You now see the Properties panel for the IKM.
3. Change the values for either FLOW_CONTROL or STATIC_CONTROL to “Yes.” If there is no option for FLOW_CONTROL, it is because your Staging area is not on your target. Flow control is not possible in that case.
4. You can also set RECYCLE_ERRORS to “Yes.” This setting causes rows already in the ERRORS table to be reused. If you never empty the ERRORS table, you will possibly recycle the same error lines many times.

Agenda

- Data Quality
- Business Rules for Data Quality
- **Enforcing Data Quality with ODI**
 - Enabling Static or Flow Control for a Mapping
 - **Setting the Options**
 - Selecting Which Constraints to Enforce
 - Reviewing Erroneous Records



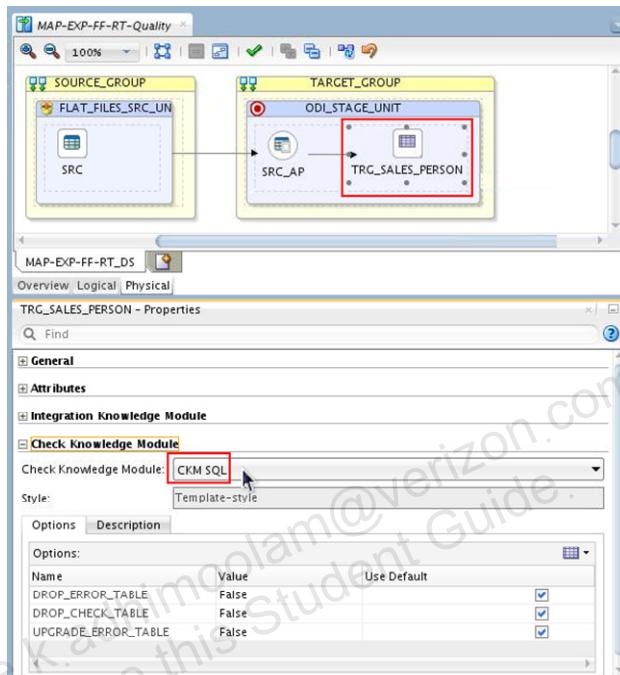
ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

After enabling the quality control, the next step is to set its options to define how it behaves.

Setting the Physical Options

1. Click the mapping's Physical tab.
2. Select a CKM.
3. Set the CKM options.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

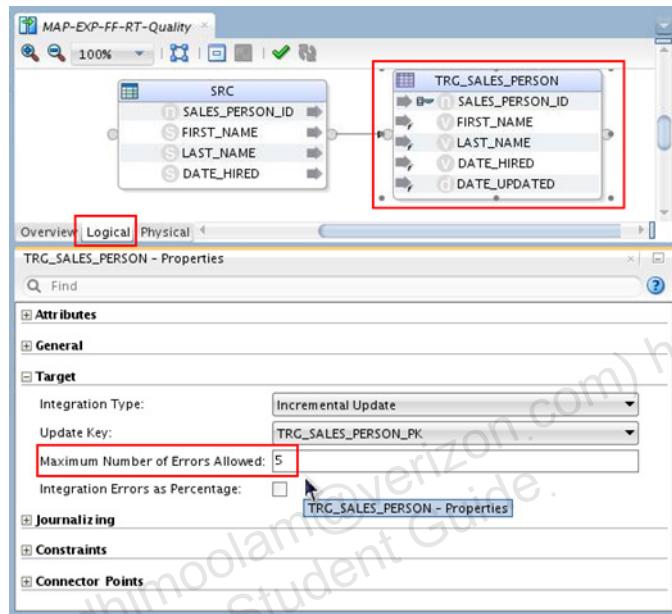
To configure the options for your data quality checks, perform the following:

1. Click the Controls tab—this is used to configure all static or flow control checks that you perform with this mapping. You may have noticed that the workflow for setting up a mapping starts at the Definition tab on the left, and then moves slowly to the right across Diagram, Flow, and onto Controls.
2. Select the CKM from the drop-down list; for example, for an Oracle server, select Oracle. Remember to import the Knowledge Modules into the project before using them.
3. Set specific options in the Option list. These generally change minor characteristics of the check. For example, if you want the ERRORS table to be deleted altogether, set DROP_ERROR_TABLE to "True."

At the bottom is the list of constraints to check. These constraints will be covered in greater detail in the following slides.

Setting the Logical Options

1. Click the mapping's Logical tab.
2. Select the target.
3. Set the Integration type.
4. Set the maximum number of errors allowed.
 - Leave blank to allow an unlimited number of errors.
 - To specify a percentage of the total number of integrated records, select the % check box.



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

To configure the options for your data quality checks, perform the following:

1. Click the Logical tab.
2. Select the Integration type from the drop-down list; for example, for an Oracle server, select Oracle. Remember to import the Knowledge Modules into the project before using them.
3. Normally, when rows with errors are detected, they are moved into the ERRORS table, and the processing continues. However, you can define a maximum number or proportion of errors. Suppose that you have a mapping that normally detects fewer than 10 errors. If it picked up more than 50 errors, for example, something must be very wrong with the input data. Therefore, you can set a limit to discover immediately if there is a problem. You can also express this limit as a percentage by selecting the % check box. Note that a row can generate several errors if it violates several constraints.

Agenda

- Data Quality
- Business Rules for Data Quality
- **Enforcing Data Quality with ODI**
 - Enabling Static or Flow Control for a Mapping
 - Setting the Options
 - **Selecting Which Constraints to Enforce**
 - Reviewing Erroneous Records



ORACLE

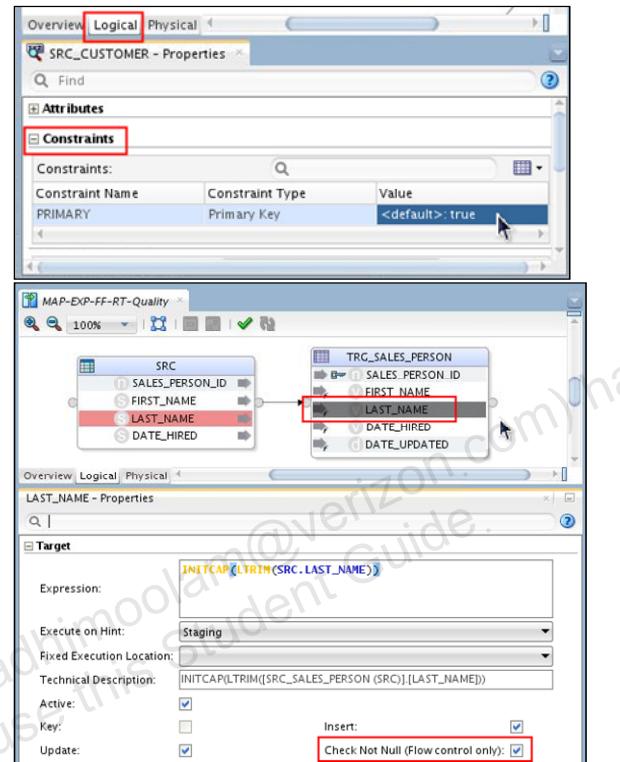
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The third step in setting up a quality control process is to select which constraints to enforce. You probably noticed this area on the Controls tab. You will now use it to select the constraints that will be enforced with flow control. Then you will see how to select which constraints to check with static control.

Selecting Which Constraints to Enforce

For flow control:

- For most constraints:
 1. Click the Logical tab.
 2. Select a datastore.
 3. Click the Constraints tab.
 4. For each constraint to enforce, select “true.”
- For Not Null constraints:
 1. Click the Logical tab.
 2. Select the target attribute you want to check for nulls.
 3. In the attribute Properties panel, select Not Null.



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

How you select which constraints to enforce depends on the type of constraint and on the type of quality control. First, you select the constraints to be enforced during flow control. For most constraints, you perform the following:

1. On a mapping, click the Logical tab.
2. Select a target datastore.
3. Click and expand the Constraints tab.
4. Select each constraint that you want to enforce (for example, PK) and select “true.”

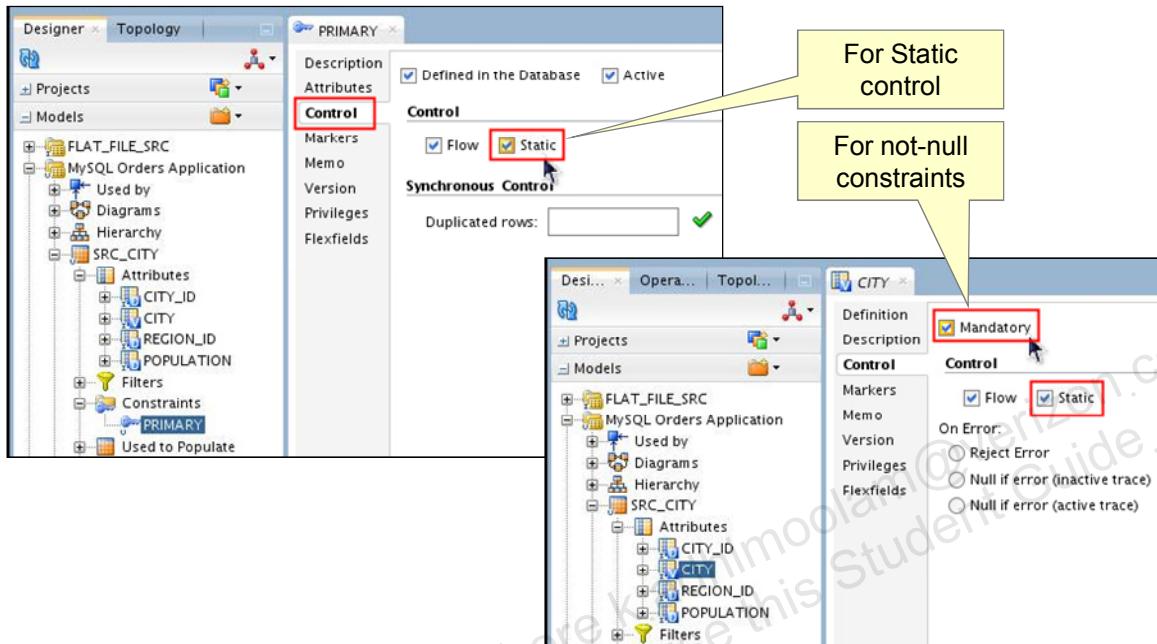
Mandatory, or not-null constraints, however, must be specified for each column of the target datastore.

Note: You can specify that a value is mandatory even if the database table accepts nulls.

1. Click the Logical tab.
2. Find the attribute/column in the target datastore that you want to make mandatory.
3. Now select the Not Null check box.

Note: When you deselect the “Check Not Null (Flow Control only)” check box, the Fix/Ignore dialog box always pops up. Be sure to click “Ignore” if you want your choice to remain. “Fix” will reset it to selected.

Selecting Which Constraints to Check



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You now select the constraints that are to be enforced during static control. To do so, perform the following:

1. Locate the relevant datastore in the Models view.
2. For constraints, such as references and primary keys, double-click the constraint.
3. Click the Controls tab, and select the Static check box.

Mandatory, or not-null constraints, however, must be specified for each column of the datastore.

1. To do this, double-click the name of the column under the Columns node.
2. Click the Controls tab.
3. Select the Mandatory check box.
4. Select the Static check box.

Differences Between Control Types

	Static Control		Flow Control
Launched through	Model	Mapping	Mapping
CKM defined on	Model	Mapping	Mapping
Options defined on	Model	Mapping	Mapping
Constraints defined on	Model	Mapping	Mapping
Invalid rows deleted	Possible	Never	Always



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You have seen some of the differences in the way static control and flow control work. This is because static control is a property of the model. Normally, you set your constraints on the model first. Then, when you create your mapping, these constraints are automatically copied.

This lesson focuses on data quality control in a mapping. Therefore, you have not seen how to set it up at the model level. However, you should know the differences, which are quite subtle.

The CKM that is used for a static check depends on how the static check is launched. For static control launched at the end of a mapping, it behaves like flow control: You choose the CKM on the Controls tab of the mapping. This is true also for defining other control options. If the static check is launched manually, or by a package, the CKM and options defined on the model are used instead.

However, to select the constraints that are checked, you must define them on the model for static control. The constraints that are selected in the mapping window apply only to flow control.

Another difference is whether invalid rows are deleted from the source table. For flow control, they are always deleted. For static control launched by a mapping, they are never deleted. When you launch static control manually or by a package, however, you have a choice.

Agenda

- Data Quality
- Business Rules for Data Quality
- **Enforcing Data Quality with ODI**
 - Enabling Static or Flow Control for a Mapping
 - Setting the Options
 - Selecting Which Constraints to Enforce
 - **Reviewing Erroneous Records**



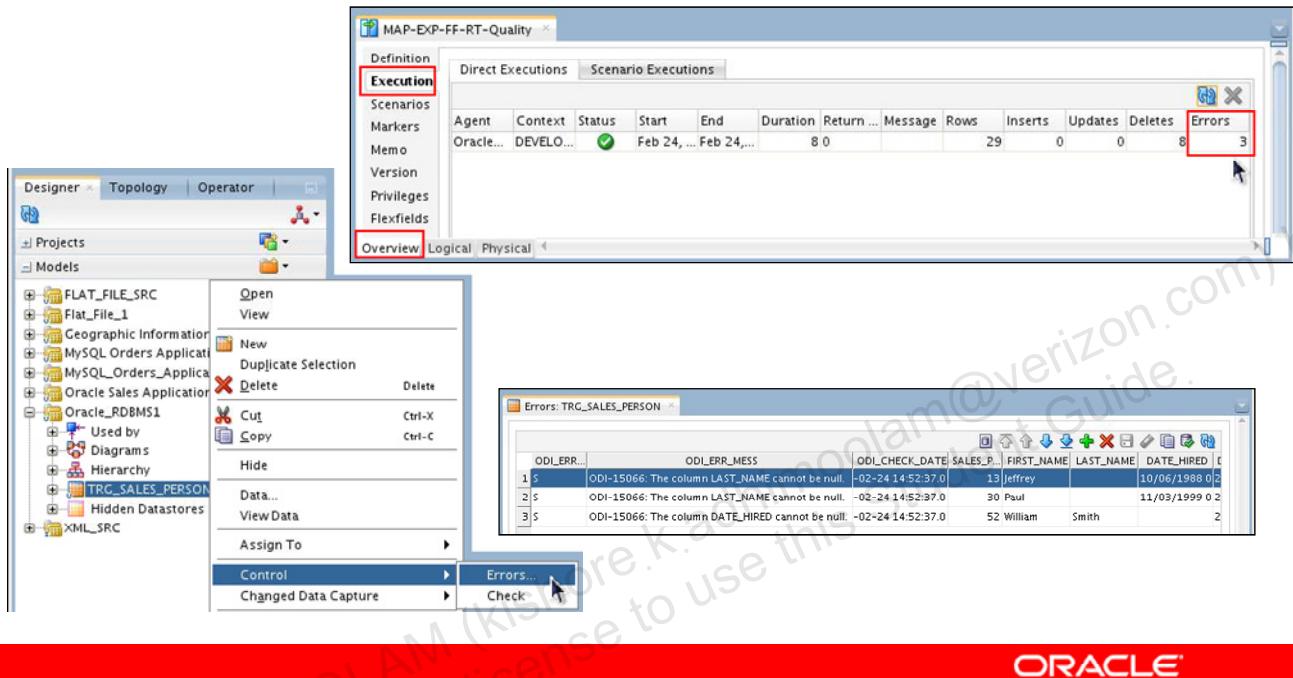
ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

After setting up the system of quality control, you now execute the mapping to perform these checks. The last step in the quality control process is to review erroneous records. These are the records that were detected by the CKM you chose. They have been isolated in an ERRORS table.

Reviewing Erroneous Records

To view the erroneous records, execute your mapping, and then:



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

To view the erroneous records, execute your mapping, and then perform the following:

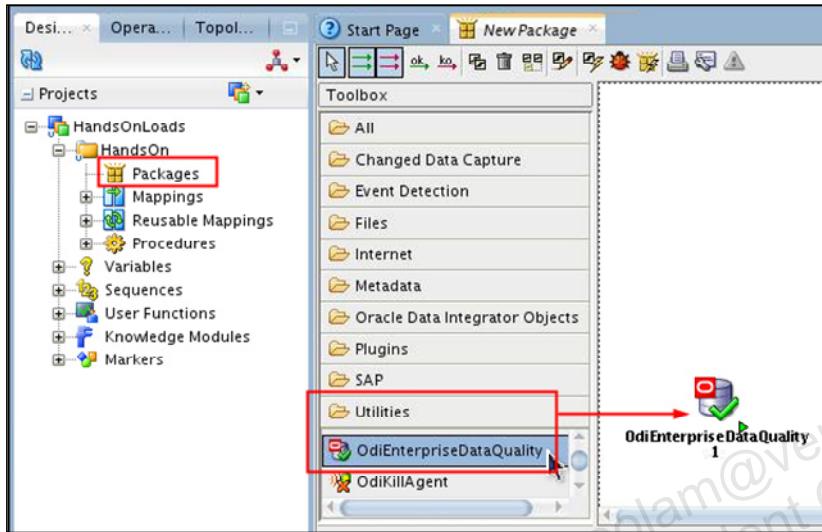
1. Click the Execution tab on the mapping. Note that there are two subtabs. When you run your mapping from Designer, you will find it on the Direct Executions tab.
2. Locate the most recent execution. You may need to click Refresh to see it. You can see the most recent execution by clicking the Start column. The number of errors is displayed under the No. of Errors column. You can also use Operator to view the number of errors.

To see the records that were rejected, you must exit the mapping window and then:

3. Select the target datastore from the Models view.
4. Right-click and select Control > Errors.
5. Review the erroneous rows.

Note: For best results, consider using the data quality tools and features in ODI. When you create mappings, consider enforcing constraints. These constraints do not have to exist in the database. Suppose that you regularly transfer detailed sales into a data warehouse. You would not expect to have any NULLs in the source data. However, you have not defined in the data warehouse that the sales cannot be NULL. You may still want to use ODI to enforce this rule. Thus, if some NULL figures appear in the source data, you will know about it sooner.

EnterpriseDataQuality Tool



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Oracle Enterprise Data Quality (EDQ / formerly Datanomic) is a comprehensive data quality solution providing data profiling and data quality capabilities.

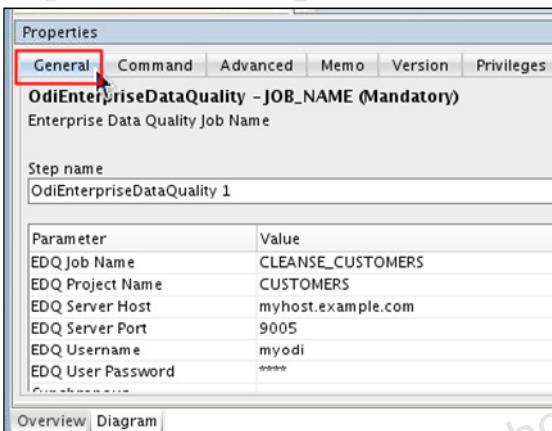
- EDQ and ODI are integrated using a package tool called **OdiEnterpriseDataQuality**, which allows customers to invoke EDQ Jobs from an ODI workflow.
- OdiEnterpriseDataQuality is shipped with ODI, while EDQ is a separate product.
- OdiEnterpriseDataQuality can be found in the Package Toolbox under Toolbox > Utilities.

Note: EDQ is the strategic data quality product at Oracle. Oracle Data Profiling and Oracle Data Quality products are now in controlled availability.

Using the EDQ Tool

The following command executes an EDQ Job called CLEANSE_CUSTOMERS located in the EDQ Project named CUSTOMERS.

The EDQ Server (a separate product) runs on *myhost.example.com* and JMX port 9005.



These parameters get composed into the Command on the next tab.

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The EnterpriseDataQuality Open Tool requires the following parameters:

- EDQ Job Name: Name of EDQ Job
- EDQ Project Name: Name of EDQ Project containing EDQ Job
- EDQ Server Host: Host name of machine running EDQ Server
- EDQ Server Port: JMX port of EDQ Server (default: 9005)
- EDQ Username/Password
- [optional] Synchronous/Asynchronous: Synchronous or asynchronous execution

Clicking the Command tab displays the following command composed of the parameters and their values:

```
OdiEnterpriseDataQuality "-JOB_NAME=CLEANSE_CUSTOMERS"
"-PROJECT_NAME=CUSTOMERS" "-HOST=myhost.example.com" "-PORT=9005"
"-USER=myodi" "-PASSWORD=g.yHwt.lzRgR3w3,ZmuNfGhLr"
```

The password is actually "myodi" but it has been encrypted.

Quiz

You can specify that a value is mandatory even if the database table accepts nulls.

- a. True
- b. False



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Answer: a

Explanation: Even if a database table accepts nulls, you can specify a value as mandatory for the purpose of ODI transformation.

Quiz

Business rules for the data quality can include:

- a. Deduplication rules (unique keys)
- b. Simple and complex reference rules that are defined in ODI as references
- c. Validation rules that enforce consistency at the record level
- d. All of the above



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Answer: d

Explanation: All of the above are examples of business rules for data quality.

Summary

In this lesson, you should have learned how to:

- Describe why and when to enforce data quality
- List the different types of data quality business rules that ODI manages
- Describe Data Quality System within ODI
- Describe how data quality is managed in a mapping
- Implement flow control during the execution of a mapping
- Implement static control after the execution of a mapping
- Enforce constraints and review erroneous records
- Manage an error table



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Checklist of Practice Activities

- ✓ 1-1: Logging In and Help Overview
- ✓ 2-1: Creating and Connecting to ODI Master and Work Repositories
- ✓ 3-1: Configuring Standalone Agents Using the Common Admin Model
- ✓ 4-1: Working with Topology
- ✓ 5-1: Setting Up a New ODI Project
- ✓ 6-1: Creating Models by Reverse-Engineering
- ✓ 7-1: Checking Data Quality in the Model
- ✓ 8-1: Creating ODI Mapping: Simple Transformations
- ✓ 9-1: Creating ODI Mapping: Complex Transformations
- ✓ 9-2: Creating ODI Mapping: Implementing Lookup
- ✓ 10-1: Creating ODI Mapping: Exporting a Flat File to a Relational Table
- ✓ 11-1: Using Native Sequences with ODI Mapping
- ✓ 11-2: Using Temporary Indexes
- ✓ 11-3: Using Sets with ODI Mapping
- ✓ 12-1: Creating and Using Reusable Mappings
- ✓ 12-2: Developing a New Knowledge Module
- ✓ 13-1: Creating an ODI Procedure
- ✓ 14-1: Creating an ODI Package
- ✓ 14-2: Using ODI Packages with Variables and User Functions
- ✓ 15-1: Debugging Mappings
- ✓ 16-1: Creating and Scheduling Scenarios
- ✓ 17-1: Using Load Plans
- 18-1: Enforcing Data Quality with ODI Mapping
- 19-1: Implementing Changed Data Capture
- 20-1: Setting Up ODI Security
- 20-2: Integration with Enterprise Manager and Using ODI Console

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Practice 18-1: Enforcing Data Quality with ODI Mappings

1. For a datastore, you verify that the primary key constraint is marked for static control and that two columns are marked mandatory with static control.
2. You then create a mapping using that datastore as a target.
3. On the mapping's Physical tab, you set STATIC_CONTROL to True. Ensure that the Knowledge Module is set to CKM SQL and that you set maximum allowed errors to 5.
4. For two attributes in the target datastore, you select the Check Not Null check box.
5. You run the mapping with Flow Control set to FALSE and verify that records with errors were loaded into the target datastore.
6. You rerun the mapping with Flow Control set to TRUE and verify that errors are excluded from the target datastore.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In this practice, you perform data quality control for the mapping created in Practice 10-1.

Note

- In step 3, the CKM SQL setting refers to a Knowledge Module that you will import.
- In step 4, you select the Check Not Null check box so that control errors will generate if the two columns are not loaded.
- In step 5, you verify that there are control errors on three records and that the three records with errors were loaded into the target datastore.

19

Working with Changed Data Capture

(CDC)

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Describe the purpose of Changed Data Capture (CDC) with Oracle Data Integrator (ODI)
- Describe what types of CDC implementations are possible with ODI
- Use CDC implementation techniques
- Perform journalizing
- Use the results of CDC



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

This lesson is an overview of the changed data capture techniques available in ODI to detect changes to source data. These CDC techniques enable you to perform data integration jobs that process only the source data that has changed.

Why Changed Data Capture?

- CDC:
 - Enables applications to process changed data only
 - Reduces the volume of data to be processed
- A CDC load processes only changes since the last load.
- CDC is extremely useful for near-real-time implementations, synchronization, and Master Data Management.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The purpose of CDC is to enable applications to process changed data only. CDC enables ODI to track changes in source data caused by other applications. When running integration mappings, ODI can avoid processing unchanged data in the flow. Each load will only process changes since the last load. The volume of data to be processed is dramatically reduced. CDC is extremely useful for near-real-time implementations, synchronization, and Master Data Management (providing processes for collecting, aggregating, and distributing nontransactional data throughout an organization).

CDC Techniques

Multiple techniques are available for CDC:

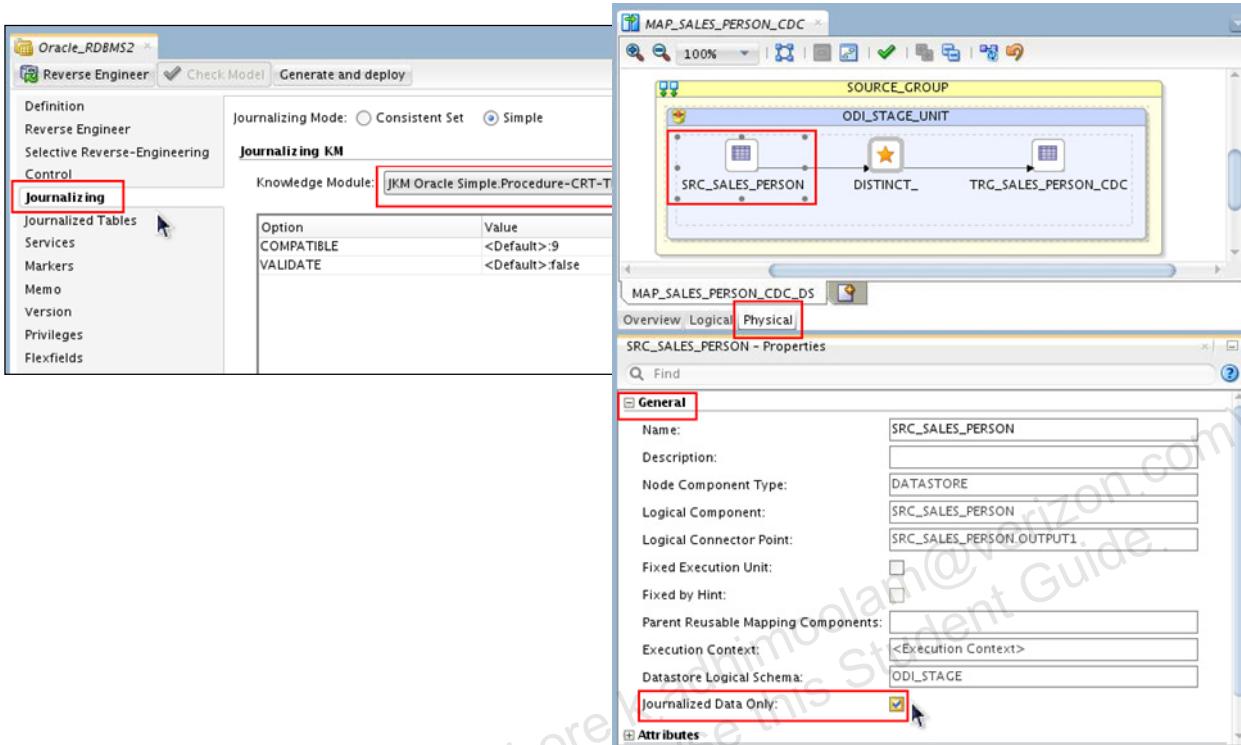
- **Trigger based:** ODI creates and maintains triggers to keep track of the changes.
- **Logs based:** ODI retrieves changes from the database logs (Oracle, AS/400).
- **Time stamp based:** Processes written with ODI can filter the data by comparing the time stamp value with the last load time (cannot process deletes)
- **Sequence number:** If the records are numbered in sequence, ODI can filter the data based on the last value loaded (cannot process updates and deletes).



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Note: In the cases of time stamp-based and sequence number techniques, the data model must have been designed properly.

Changed Data Capture in ODI



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

CDC in ODI is implemented through a family of JKMs: the Journalization KMs. These KMs are chosen and set in the model. The developer can choose from the mapping whether to use the full data set or only the changed data.

CDC is performed by journalizing models. Journalizing a model consists of setting up the infrastructure to capture the changes (inserts, updates, and deletes) made to the records of this model's datastores.

Journalizing Components

- Journals: Contain references to the changed records
- Capture processes: Capture the changes in the source datastores either by creating triggers on the data tables or by using database-specific programs to retrieve log data from data server log files
- Subscribers (applications, integration processes, and so on): Use the changes tracked on a datastore or on a consistent set



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Note

- Journals also contain the type of changes (insert/update, delete).
- See the documentation on Journalizing Knowledge Modules (JKMs) for more information about the capture processes used.
- Subscribers subscribe to a model's CDC to have the changes tracked for them. Changes are captured only if there is at least one subscriber to the changes. When all the subscribers have consumed the captured changes, these changes are discarded from the journals.

CDC Infrastructure in ODI

- Journal table: Created by the KM and loaded by specific steps implemented by the KM. This table has the following structure:
 - Primary key of the table being checked for changes
 - Time stamp to keep the change date
 - A flag to allow for a logical “lock” of the records
- Views are created to join this table with the actual data.



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

When other KMs need to select data, they will use the views instead of the tables.

Simple Versus Consistent Set Journalizing

- Simple journalizing enables you to journalize one or more datastores. Each journalized datastore is treated separately when capturing the changes (no consistency).
- Consistent Set journalizing guarantees the consistency of the captured changes (Consistency Window).



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Note: Consistent Set journalizing provides the guarantee of consistency of the captured changes in linked datastores. The set of available changes for which consistency is guaranteed is called the Consistency Window.

Limitations of Simple CDC Journalizing: Example

- As changed data is processed, more changes occur in the source environment; data transferred to the target environment may have missing references.
- Example: Process changes for orders and order lines
 1. Load all the new orders in the target (11,000 to 25,000).
 2. While these are being loaded, two new orders come in: 25,001, 25,002.
 3. Then load the order lines: By default, all the order lines are loaded, including order lines for orders 25,001 and 25,002.
 4. The order lines for 25,001 and 25,002 are rejected by the target database (invalid foreign keys).



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Note: In step 2, the last two orders are not processed as part of this load. They will be processed with the next load.

Consistent CDC Journalizing

The mechanisms put in place by Consistent CDC solves the issues faced with Simple CDC.

- Lock children records before processing the parent records.
- When new parent records and children records come in, both parent and children records are ignored.



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Consistent Set journalizing provides the guarantee that when you have an ORDER_LINE change captured, the associated ORDER change has been also captured, and vice versa. Changes in Consistency Window should be processed in the correct sequence (ORDER followed by ORDER_LINE) by designing and sequencing integration mappings into packages.

Although Consistent Set journalizing is more powerful, it is also more difficult to set up. Use it when referential integrity constraints need to be ensured when capturing the data changes. For performance reasons, Consistent Set journalizing is also recommended when a large number of subscribers are required.

Note: You cannot journalize a model (or datastores within a model) by using both Consistent Set and Simple journalizing.

Consistent CDC: Infrastructure

- Processing Consistent Set CDC consists of the next four phases:
 - Extend Window: Compute the consistent parent/child sets and assign a sequence number to these sets.
 - Lock Subscriber: For the application processing the changes, record the boundaries of the set of records to be processed.
 - After processing the changes, unlock the subscriber.
 - Purge the journal: Remove from the journal all the records that have been processed by all the subscribers.
- These steps can either be implemented in the KMs or done separately, as part of the Workflow management.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

For Lock Subscriber, the boundaries of the set of records to be processed are between sequence number xxx and sequence number yyy. Note that changes continue to happen in the source environment. Other subscribers can extend the window while you are processing the data.

Unlock the subscriber by recording the value of the last sequence number processed.

Note: All these steps can either be implemented in the Knowledge Modules or performed separately, as part of the Workflow management.

For more information about setting up Consistent Set CDC journalizing, see *Oracle Fusion Middleware Developer's Guide for Oracle Data Integrator 12c*.

Setting Up Journalizing

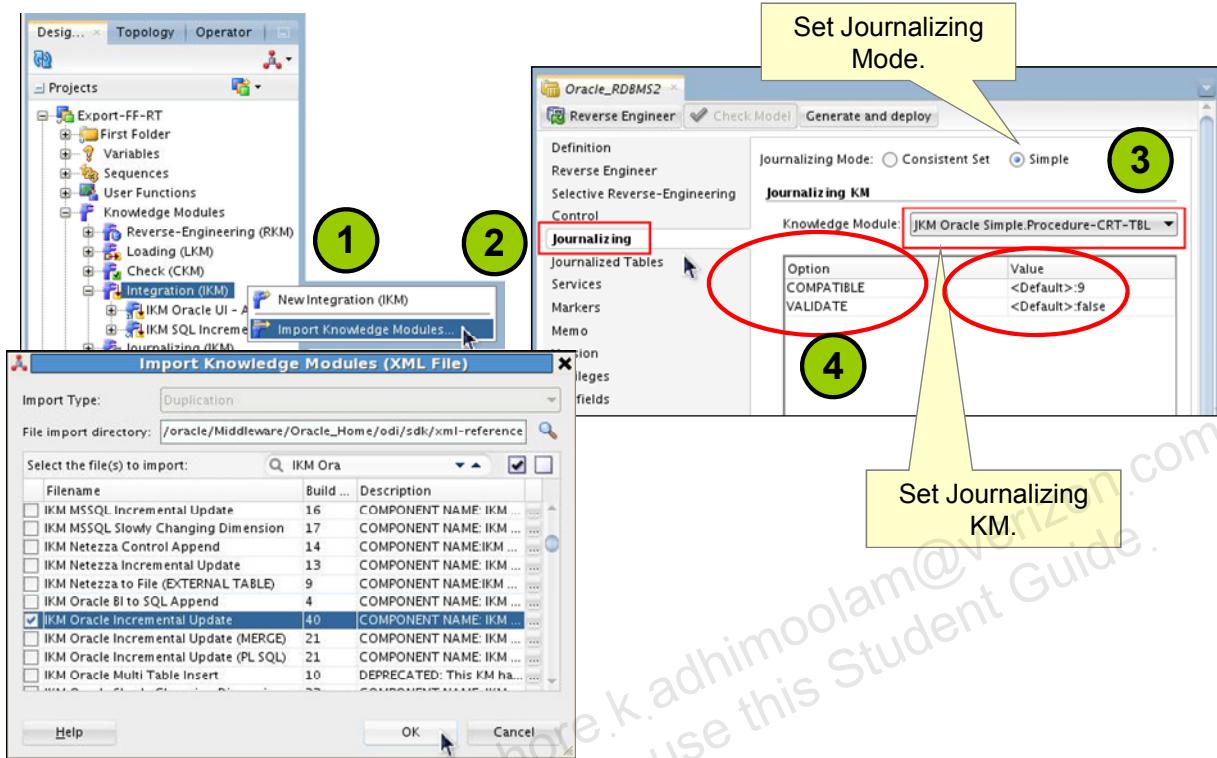
1. Set the CDC parameters.
2. Add the datastores to the CDC.
3. For Consistent Set journalizing, arrange the datastores in order.
4. Add subscribers.
5. Start the journals.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

This is the basic process for setting up CDC on an ODI data model. Each of these steps is described in more detail in the following slides.

Setting CDC Parameters: Example



ORACLE

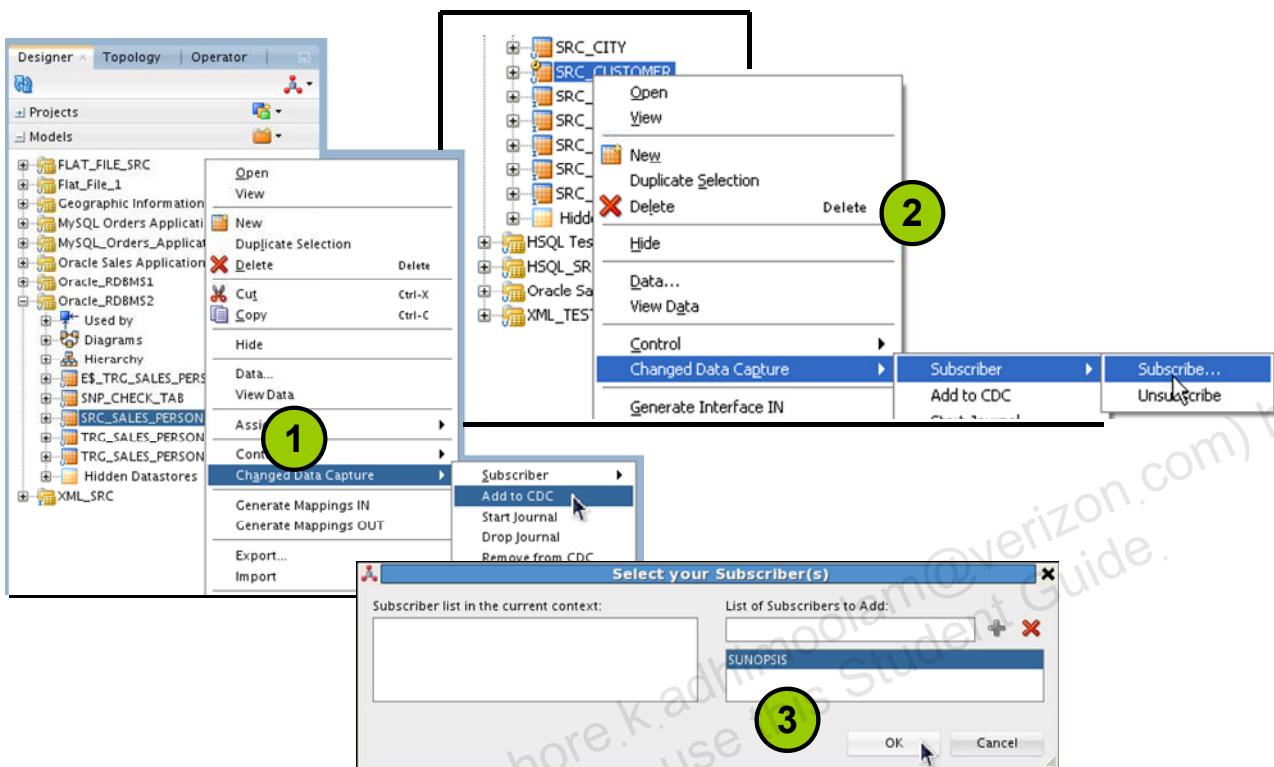
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Setting CDC Parameters

This includes selecting or changing the journalizing mode and JKM used for the model. If the model is already being journalized, stop journalizing with the existing configuration before modifying the data model journalizing parameters.

1. Import the correct JKM for your model.
2. Edit the data model that you want to journalize, and then click the Journalizing tab. Select the journalizing mode you want to set up: Consistent Set or Simple.
3. Select the Journalizing KM that you want to use for this model. Only Knowledge Modules suitable for the data model's technology and journalizing mode, and that have been previously imported into at least one of your projects, appear in the list.
4. Set the Options for this KM. Refer to the Knowledge Module's description for more information about the options. Save your changes.

Adding a Subscriber: Example



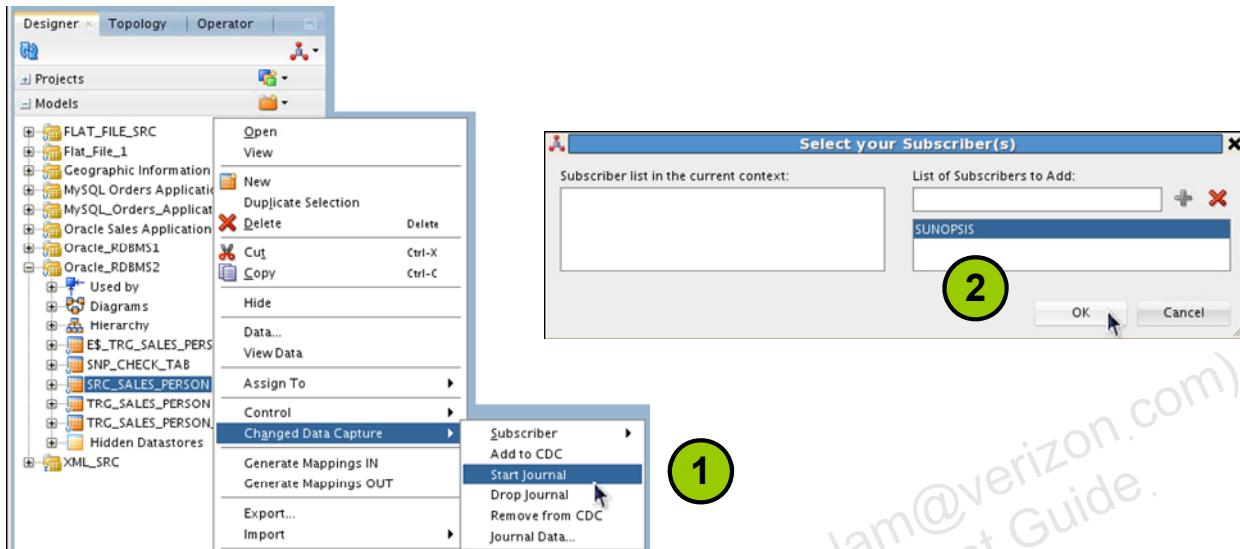
ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

For all the following steps, right-click a table to process just that table, or right-click the model to process all the tables of the model:

1. Add the table to the CDC infrastructure: Right-click a table and select Changed Data Capture > Add to CDC. For Consistent CDC, arrange the datastores in the appropriate order (parent/child relationship). In the model definition, click the Journalized Tables tab and click the Reorganize button.
2. Add the subscriber. (The default subscriber is SUNOPSIS.) Right-click a table and select Changed Data Capture > Subscriber.
3. Enter the subscriber name. Click the Add button, and then click OK.

Starting Journal: Example



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Starting the journals creates the CDC infrastructure if it does not exist yet. It also validates the addition, removal, and order changes for journalized datastores.

To start the journals:

1. Select the data model or datastore that you want to journalize. Right-click and select Changed Data Capture > Start Journal if you want to start the journals (or Changed Data Capture > Drop Journal if you want to stop them).
2. Select the subscriber, and then click OK. In this example, you use the default subscriber SUNOPSIS. Click OK to start execution.

A session starts or drops the journals. You can track this session from the Operator.

Note: Stopping the journals deletes the entire journalizing infrastructure and all captured changes are lost. Restarting a journal does not remove or alter any changed data that has already been captured.

Journalizing Status

- **OK:** Journalizing is active for this datastore in the current context, and the infrastructure is operational. 
- **No Infrastructure:** Journalizing is marked as active, but no appropriate journalizing infrastructure was detected in the current context. Journals should be started. 
- **Remnants:** Journalizing is marked as inactive in the model, but remnants of the journalizing infrastructure such as the journalizing table have been detected. 



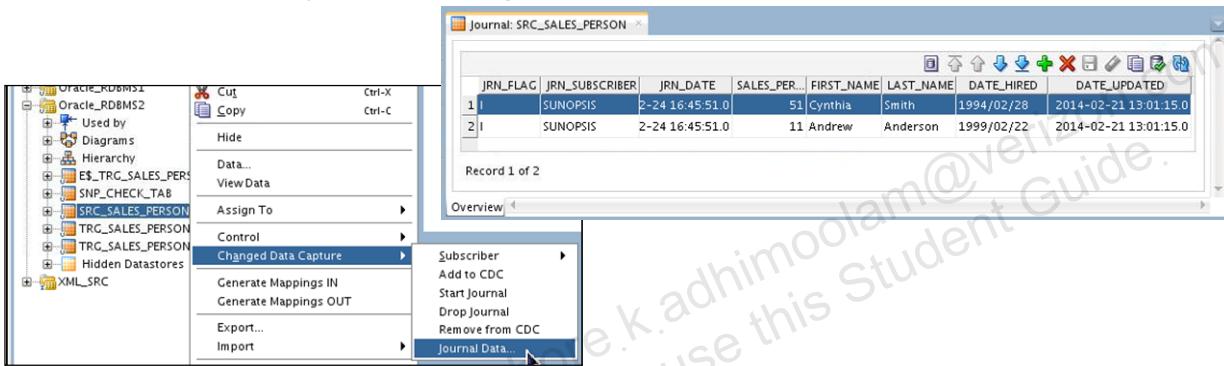
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Datastores in models or mappings have an icon marker indicating their journalizing status in Designer's current context.

- The No Infrastructure state may occur if the journalizing mode implemented in the infrastructure does not match the one declared for the model.
- The Remnants state may occur if the journals were not stopped and the table has been removed from CDC.

Viewing Data/Changed Data: Example

- In the model, right-click the table name and select Data to view the data or Changed Data Capture > Journal Data to view the changes.
- From the mapping, click the caption of the journalized source table and select or deselect *Journalized data only* to view only the changes or all the data.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Data and changed data can be viewed from the model and from the mappings.

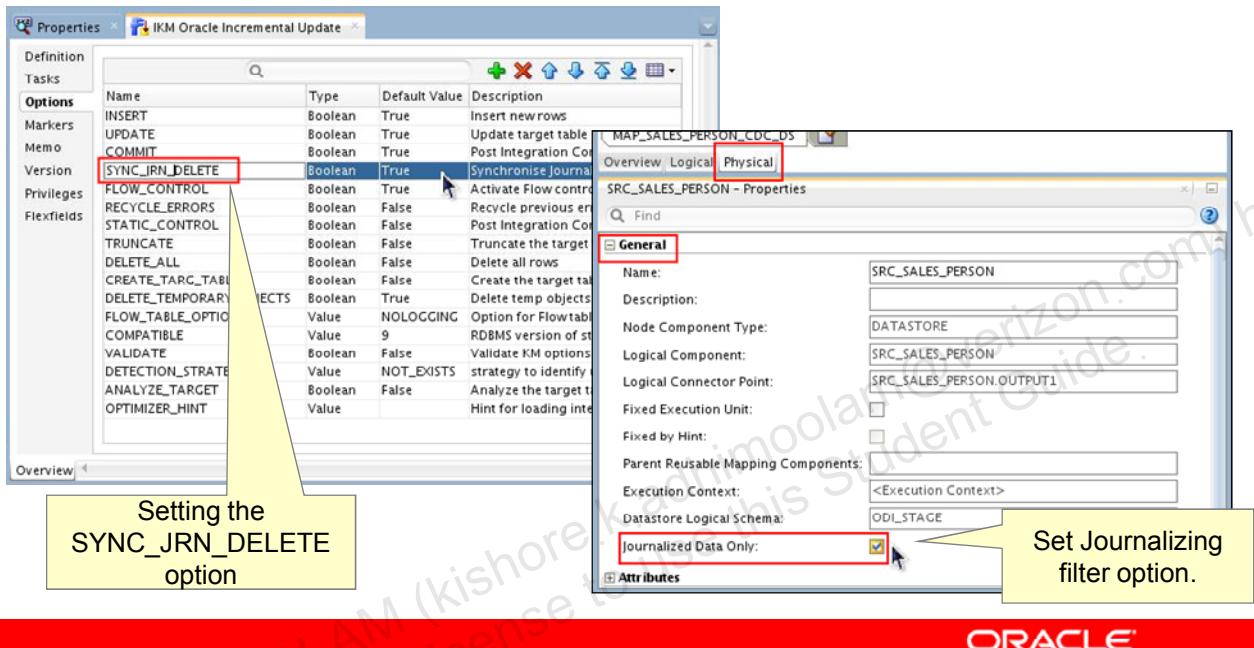
The changed data displays three extra columns for the changes details:

- **JRN_FLAG:** Flag indicating the type of change. It takes the value I for an inserted/updated record and D for a deleted record.
- **JRN_SUBSCRIBER:** Name of the subscriber
- **JRN_DATE:** Time stamp of the change

Journalized data is mostly used within integration processes. Changed data can be used as the source of integration mappings. The way it is used depends on the journalizing mode.

Using Changed Data

- Journalizing filter in mapping
- KM option SYNC_JRN_DELETE



ORACLE

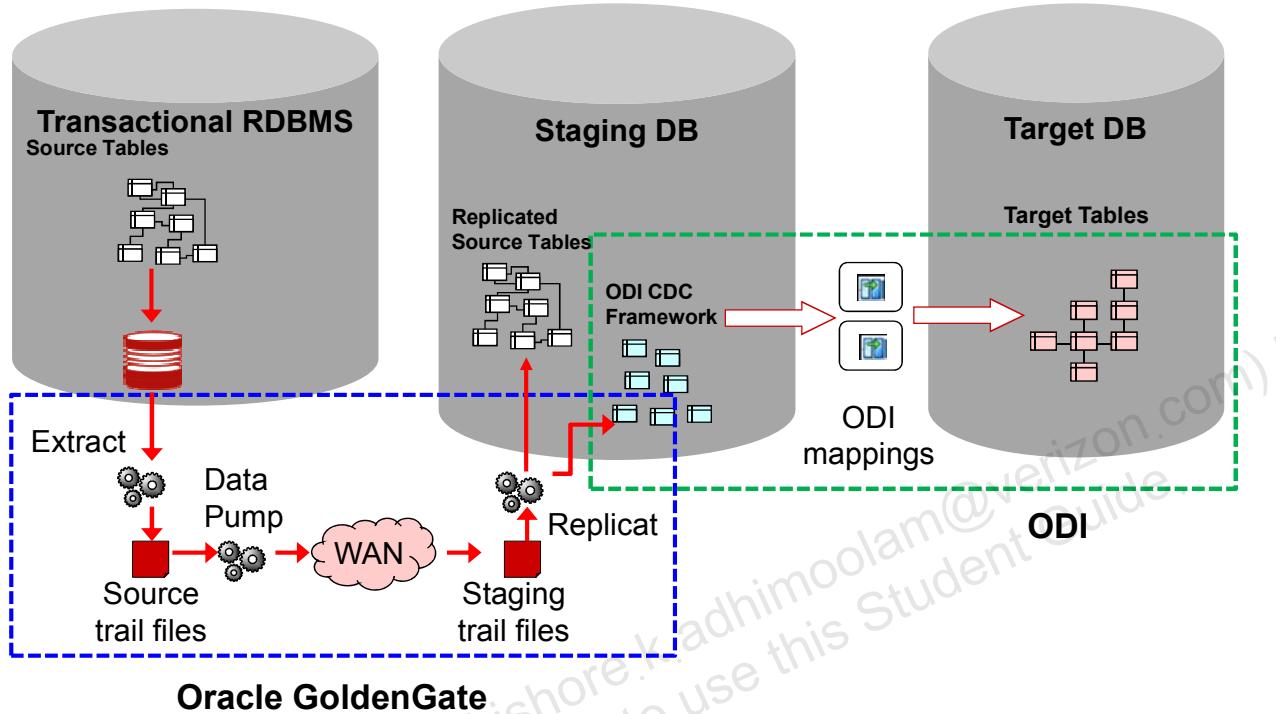
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Using changed data from Simple journalizing consists of designing mappings by using journalized datastores as sources.

- **Journalizing filter option:** When a journalized datastore is inserted into a mapping diagram, a Journalized Data Only check box appears in this datastore's property panel.
 - When you select this box, the journalizing columns (JRN_FLAG, JRN_DATE, and JRN_SUBSCRIBER) become available in the datastore.
 - A journalizing filter is also automatically generated on this datastore. This filter will reduce the amount of source data retrieved. It is always executed on the source. You can customize this filter (for instance, to process changes in a time range, or only a specific type of change).
- **Knowledge Module Option**
 - When processing journalized data, the SYNC_JRN_DELETE option of the Integration Knowledge Module must be set carefully. It invokes the deletion from the target datastore of the records marked as deleted (D) in the journals and that are not excluded by the journalizing filter. If this option is set to False, integration processes only inserts and updates.

Note: Keep in mind that only one journalized table can be used per mapping. Using changed data in Consistent journalizing is similar to Simple journalizing regarding mapping design. It requires extra steps before and after processing the changed data in the mappings, in order to enforce changes consistently within the set. For more details about processing change data in mappings with Consistent journalizing, refer to *Oracle Fusion Middleware Developer's Guide for Oracle Data Integrator Release 12c*.

Oracle GoldenGate Integration



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Oracle GoldenGate captures and replicates changes from the source system into a staging schema on the target data warehouse platform.

This staging schema contains a replicated copy of the source table and the structures that are used by ODI's own Changed Data Capture (CDC) framework. ODI then picks up the changes in the structures of the ODI CDC framework, transforms the data, perhaps joins the data to look up other tables, and then loads the data into the target schema.

This technology does not require ODI to be changed in any way. The GoldenGate CDC framework can be used with the existing ODI CDC framework.

Oracle GoldenGate Integration in ODI 12c

- With ODI 12c, you can further integrate ODI with OGG.
- Implementation:
 - OGG instances can now be managed in the topology.
 - You deploy OGG parameter files to distributed managers.
 - You can also configure OGG parameters graphically.
 - ODI 12c supports real-time and bulk processes using the same mapping through deployment specifications.

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Integration of Oracle GoldenGate as a source for the CDC framework has been improved in the following areas:

- Oracle GoldenGate source and target systems are now configured as data servers in the topology. Extract and replicate processes are represented by physical and logical schemas. This representation in the topology allows the separate configuration of multiple contexts, following the general context philosophy.
- Most Oracle GoldenGate parameters can now be added to extract and replicate processes in the physical schema configuration. The UI provides support for selecting parameters from lists. This minimizes the need for the modification of Oracle GoldenGate parameter files after generation.

A single mapping can now be used for journalized CDC load and bulk load of a target. This is enabled because the Oracle GoldenGate JKM uses the source model (and not the Oracle GoldenGate replication target). Multiple deployment specifications can be used in a single mapping for journalized load and bulk load.

- Oracle GoldenGate parameter files can now be automatically deployed and started to source and target Oracle GoldenGate instances through the JAgent technology.

Quiz

Simple journalizing provides the consistency of the captured changes in linked datastores.

- a. True
- b. False



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Answer: b

Explanation: Each journalized datastore is treated separately when capturing the changes and does not provide the guarantee of the consistency of the captured changes in linked datastores.

Quiz

The structure of the Journal table includes:

- a. The primary key of the table being checked for changes
- b. The time stamp to keep the change date
- c. A flag to allow for a logical “lock” of the records
- d. All of the above



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Answer: d

Explanation: All of the above are the features of the Journal table structure.

Summary

In this lesson, you should have learned how to:

- Describe the purpose of CDC with ODI
- Describe what types of CDC implementations are possible with ODI
- Use CDC implementation techniques
- Perform journalizing
- Use the results of CDC



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Checklist of Practice Activities

- ✓ 1-1: Logging In and Help Overview
- ✓ 2-1: Creating and Connecting to ODI Master and Work Repositories
- ✓ 3-1: Configuring Standalone Agents Using the Common Admin Model
- ✓ 4-1: Working with Topology
- ✓ 5-1: Setting Up a New ODI Project
- ✓ 6-1: Creating Models by Reverse-Engineering
- ✓ 7-1: Checking Data Quality in the Model
- ✓ 8-1: Creating ODI Mapping: Simple Transformations
- ✓ 9-1: Creating ODI Mapping: Complex Transformations
- ✓ 9-2: Creating ODI Mapping: Implementing Lookup
- ✓ 10-1: Creating ODI Mapping: Exporting a Flat File to a Relational Table
- ✓ 11-1: Using Native Sequences with ODI Mapping
- ✓ 11-2: Using Temporary Indexes
- ✓ 11-3: Using Sets with ODI Mapping
- ✓ 12-1: Creating and Using Reusable Mappings
- ✓ 12-2: Developing a New Knowledge Module
- ✓ 13-1: Creating an ODI Procedure
- ✓ 14-1: Creating an ODI Package
- ✓ 14-2: Using ODI Packages with Variables and User Functions
- ✓ 15-1: Debugging Mappings
- ✓ 16-1: Creating and Scheduling Scenarios
- ✓ 17-1: Using Load Plans
- ✓ 18-1: Enforcing Data Quality with ODI Mappings
- 19-1: **Implementing Changed Data Capture**
- 20-1: Setting Up ODI Security
- 20-2: Integrating with Enterprise Manager and Using ODI Console

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Practice 19-1: Implementing Changed Data Capture

1. Import the JKM Oracle Simple Knowledge Module.
2. Create a model, `Oracle_RDBMS2`, specifying this Knowledge Module on the Journalized tab.
3. Reverse-engineer the model.
4. Add the `SRC_SALES_PERSON` datastore to the CDC and start Journal by using the default subscriber `SUNOPSIS`.
5. Use Data Viewer to change some data in the table and verify that changed data has been captured.
6. Create a mapping to process changed data.

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In this practice, you perform changed data capture in a mapping by using the JKM Oracle Simple Knowledge Module.

You create a mapping by using the IKM Oracle Incremental Update Knowledge Module, specifying capture of journalized (changed) data only. You run this mapping to process changed data only.

20

Advanced ODI Administration

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Use ODI Security Navigator to set:
 - Security policy
 - Password policy
 - User parameters
- Manage ODI Reports
- Use methods of ODI Security integration
- Implement WebLogic Server, Enterprise Manager, and ODI Console integration with ODI



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

This lesson provides an overview of managing ODI security. You also learn how to integrate WebLogic Server and Enterprise Manager with ODI.

Agenda

- **Setting Up ODI Security**
- Managing ODI Reports
- ODI Integration with Java EE

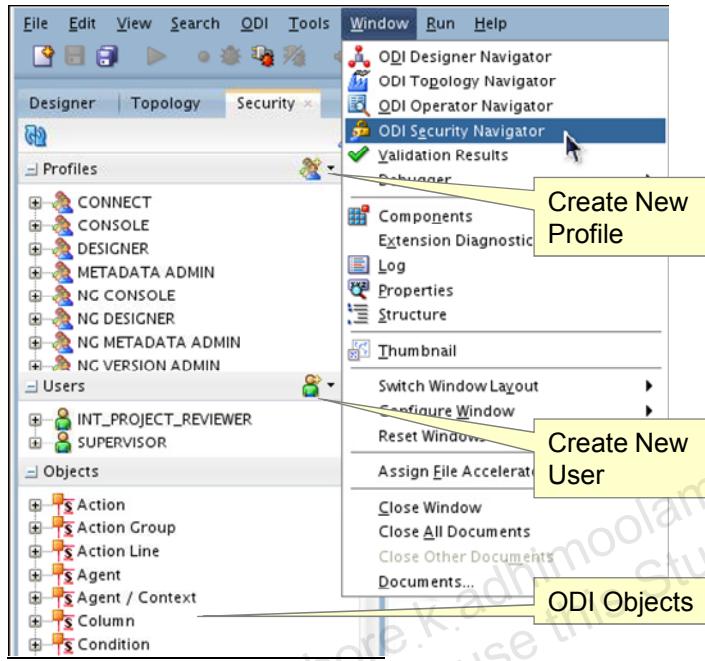


ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The first section of this lesson examines how to set up security in ODI.

Introduction to ODI Security Navigator



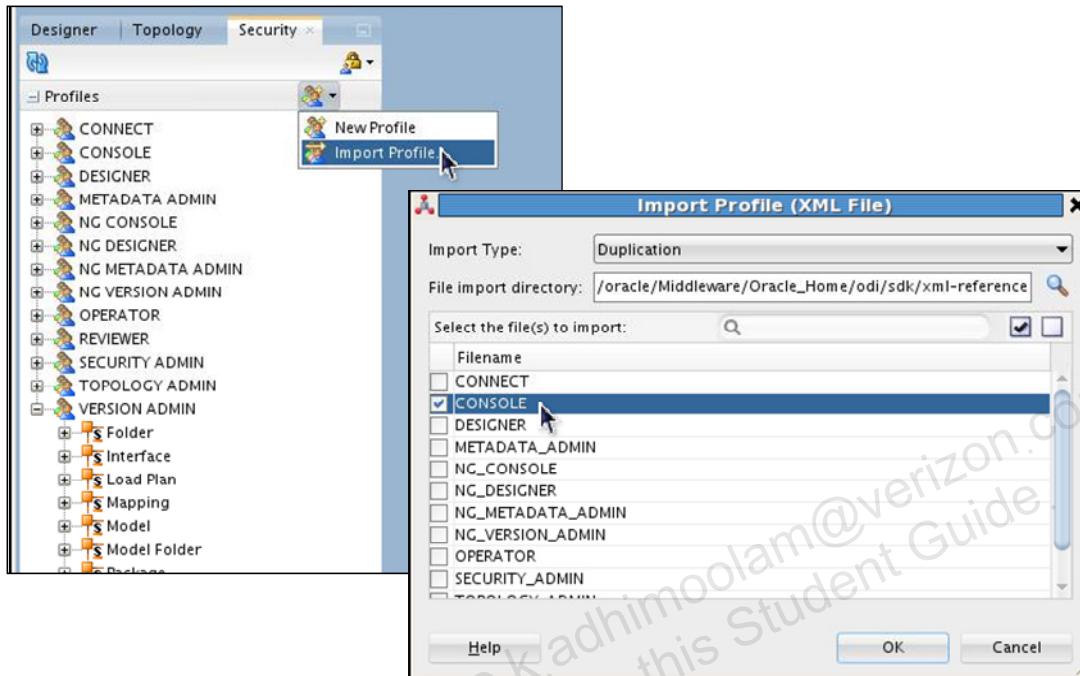
ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Using the Security Navigator tab, you can manage security in ODI. The Security Navigator module allows ODI users and profiles to be created. It is used to assign user rights for methods (edit, delete, and so on) on generic objects (data server, data types, and so on), and to fine-tune these rights on the object instances (Server1, Server2, and so on).

The Security Navigator stores this information in a Master Repository. This information can be used by all the other modules.

Introduction to ODI Security Navigator



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

- Security Navigator objects available to the current user are organized by these tree views:
 - The objects, describing each ODI elements type (datastore, model, and so on)
 - The users' profiles, users, and their authorizations
- You can perform the following operations in each tree view:
 - Insert or import root objects to the tree view by clicking the appropriate button in the frame title.
 - Expand and collapse nodes by clicking them.
 - Activate the methods associated with the objects (Edit, Delete, and so on) through the pop-up menus.
 - Edit objects by double-clicking them or by dragging them on the Workbench.
- The windows for the object being edited or displayed appear in the Workbench.

Note: Each tree view appears in floatable frames that may be docked to the sides of the main window. These frames can also be stacked up. When several frames are stacked up, tabs appear at the bottom of the frame window to access each frame of the stack. Tree view frames can be moved, docked, and stacked by selecting and dragging the frame title or tab. To lock the position of views, select Lock window layout from the Windows menu. If a tree view frame does not appear in the main window or has been closed, it can be opened by using the Windows > Show View menu.

Security Concepts: Overview

Objects, instances, and methods:

- **Object:** The representation of an element that can be handled through ODI (agents, models, datastores)
- **Instance** (object instance): Is attached to an object type (an object). For example, the `MY_PROJ_1` project is an instance of the project object type.
- **Method:** Type of action that can be performed on an object
- **Profiles:** Represents a generic rights model for working with ODI. An authorization by profile is placed on an object's method for a given profile.
- **User:** An ODI user who corresponds to the login name used for connecting to a repository



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

- **Objects** are the visible part of ODI object components (Java classes). It is necessary to combine the notions of object and object instance (or instances), which in ODI are similar to object-oriented programming notions.
- An example of an **instance**, `MY_PROJ_1`, is an instance of the project object type. Similarly, another instance of a project-type object is `YOUR_PROJ_2`.
- Each object has a series of **methods** that are specific to it. The notion of method in Data Integrator is similar to the one in object-oriented programming.

Security Concepts: Overview

Objects, instances, and methods:

- **Object:** The representation of an element that can be handled through ODI (agents, models, datastores)
- **Instance** (object instance): Is attached to an object type (an object). For example, the `MY_PROJ_1` project is an instance of the project object type.
- **Method:** Type of action that can be performed on an object
- **Profiles:** Represents a generic rights model for working with ODI. An authorization by profile is placed on an object's method for a given profile.
- **User:** An ODI user who corresponds to the login name used for connecting to a repository



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

- One or more **profiles** can be assigned to a user. An authorization by profile is placed on an object's method for a given profile. It allows a user with this profile to be given—either optionally or automatically—the right to this object through the method. The presence of an authorization by profile for a method, under an object in the tree of a profile, shows that a user with this profile is entitled (either optionally or automatically) to this object's instances through this method. The absence of authorization by profile shows that a user with this profile cannot, under any circumstance, invoke the method on an instance of the object.
- A **user** in the Security Navigator module represents an ODI user and corresponds to the login name used for connecting to a repository. A user inherits the following rights:
 - The profile rights they already have
 - Rights on objects
 - Rights on instances
- An authorization by the **user** is placed on a method of an object for a given user. It allows the user to be given, either optionally or automatically, the right to this object through the method.

Note: Objects and methods are predefined in ODI and must not be changed.

Defining Security Policies

1. Create appropriate profiles for your working methods, and give them generic rights to objects.
2. Create the users.
3. Give the users the generic profiles.
4. Optionally, you can define a password policy to encourage users to use a secured password.

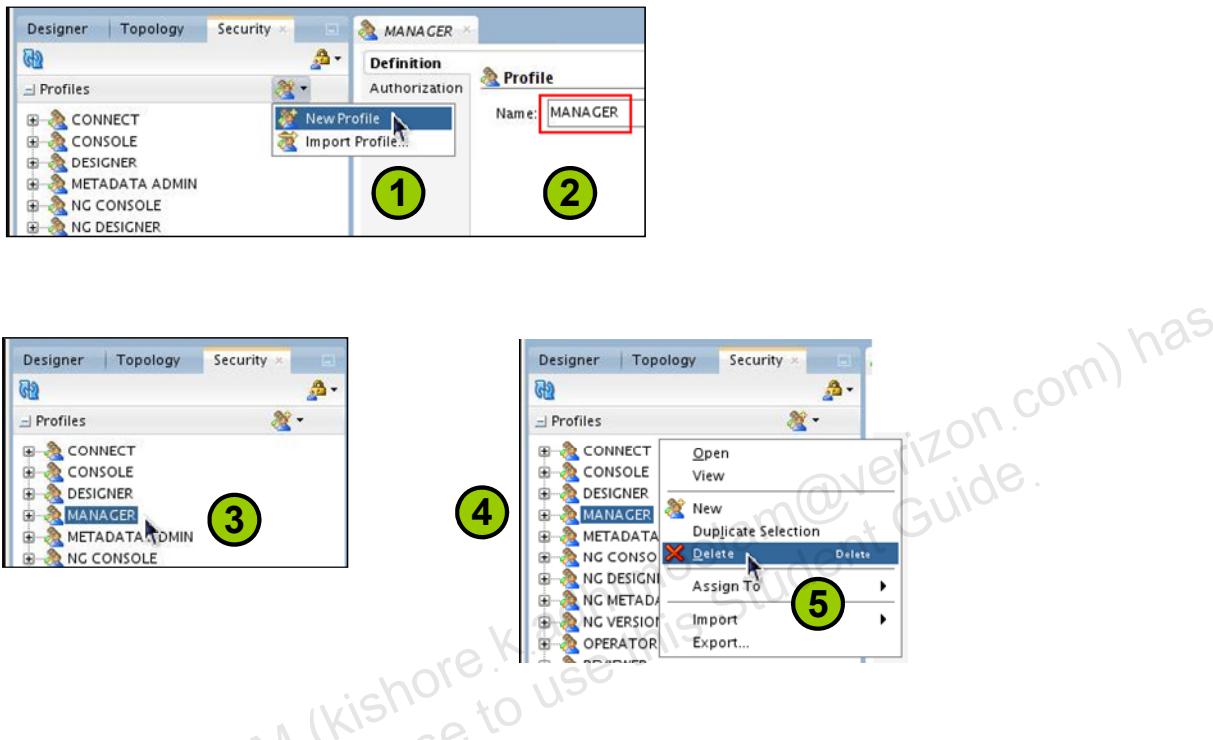


ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In step 1, you can use the generic profiles provided in ODI.

Creating Profiles



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

To create a new profile:

1. Select **Profiles** to display the tree. Right-click and select **New profile**.
2. Enter the name. Click the **Save** button.
3. Verify that your new profiles have been added to the tree view.

The profile is displayed in the tree.

Note: To delete a profile:

4. Select the profile to be deleted.
5. Right-click and select **Delete**. Click **OK**. The profile disappears from the tree.

Using Generic and Nongeneric Profiles

- Generic Profiles:
 - Have the Generic privilege option selected for all object methods
 - A user with such a profile is by default authorized for all methods of all instances of an object.
- Nongeneric Profiles:
 - Are not default authorized for all methods of all instances of an object
 - The administrator must grant the user the rights on the methods for each instance.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

If the security administrator wants a user to have rights on no instance by default, but wants to grant the rights by instance, the user must be given a nongeneric profile.

If the security administrator wants a user to have rights on all instances of an object type by default, the user must be given a generic profile.

Built-in Profiles

- **CONNECT:** Connects to Oracle Data Integrator
- **DESIGNER:** Performs development operations
- **METADATA_ADMIN:** Manages metadata
- **OPERATOR:** Manages runtime objects (for production users)
- **CONSOLE:** Views objects
- **SECURITY_ADMIN:** Edits security (for security administrators)
- **TOPOLOGY_ADMIN:** Edits topology (for ODI administrators)
- **VERSION_ADMIN:** Creates, restores, and edits versions and solutions

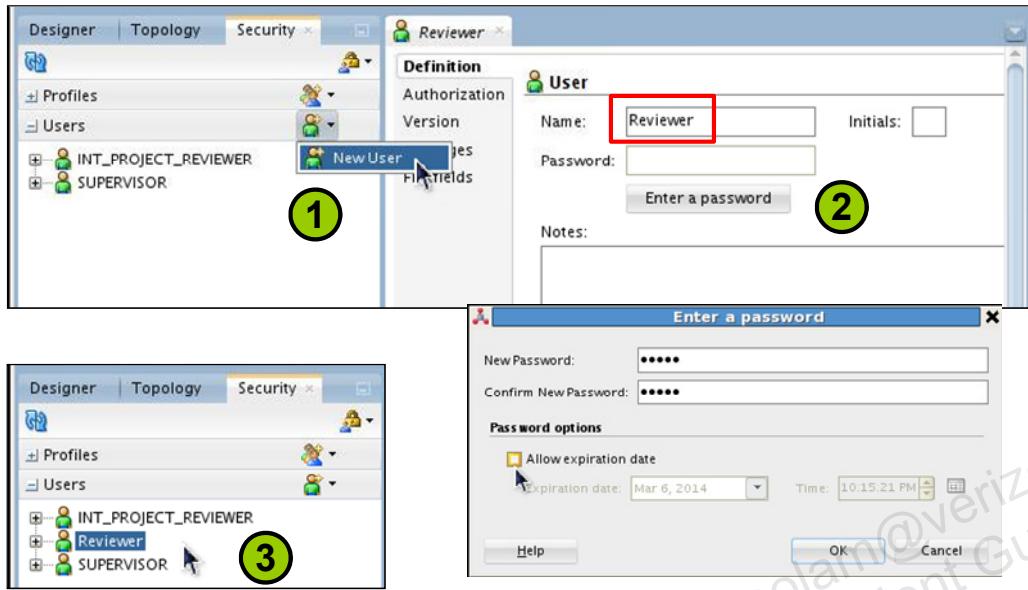


Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

ODI has some built-in profiles that security administrators can assign to the users they create. This slide shows some built-in profiles delivered with ODI. For a complete list of built-in profiles, see *Oracle Fusion Middleware Developer's Guide for Oracle Data Integrator 12c Release 1*.

- **CONNECT:** Must be granted with another profile
- **DESIGNER:** Use this profile for users who will work mainly on projects.
- **METADATA_ADMIN:** Use this profile for users who work mainly on models.
- **OPERATOR:** Use this profile for production users.
- **CONSOLE:** Use this profile for users who do not need to modify objects (was named REPOSITORY_EXPLORER in 11g)
- **SECURITY_ADMIN:** Use this profile for security administrators.
- **TOPOLOGY_ADMIN:** Use this profile for system or Oracle Data Integrator administrators.
- **VERSION_ADMIN:** Use this profile for those entitled to perform version management operations.

Creating Users



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Creating a New User

To create a new user:

1. Open the **Users** tab. Click the **New User** icon, and select **New User**.
2. Enter the name, the initials, and the user's password (by clicking **Enter a password**).
3. Click **OK**, and then click **Save**. Verify that your new user appears in the tree view.

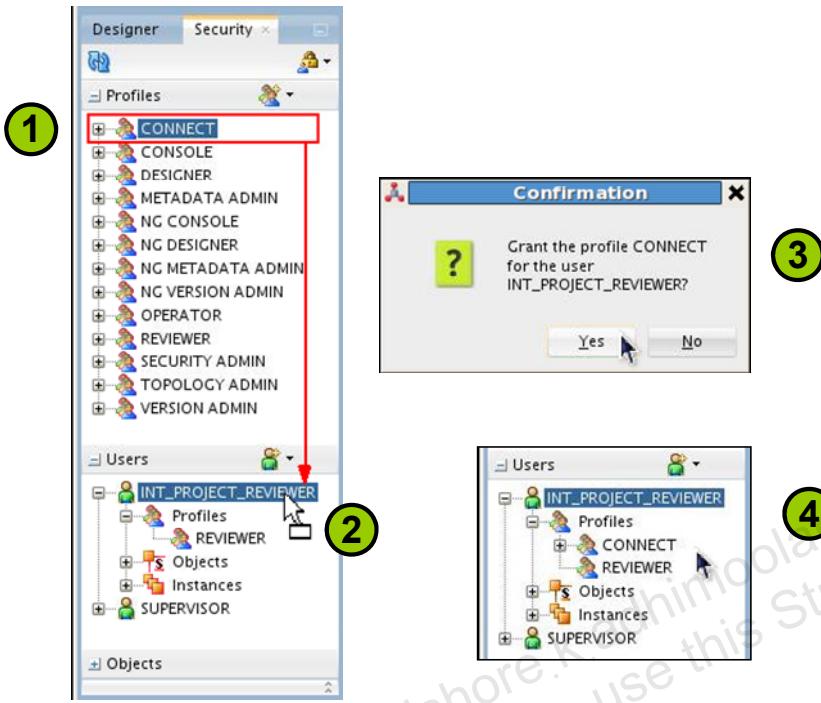
The user icon and information are displayed in the tree.

Note: To delete a user:

1. Select the user to be deleted.
2. Right-click and select **Delete**.
3. Click **OK**.

The user icon and information disappear from the tree.

Assigning a Profile to a User



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

To assign a profile to a user:

1. Expand the tree to display the user that you want to assign the profile to.
2. Select the profile that you want to assign, and then drag it onto the user branch in the tree.
3. Click **Yes** to confirm creating this profile for the user. Click the **Save** icon.
4. Expand the user's node and verify that the new profile was added under the Profiles node.

The profile is assigned to the user.

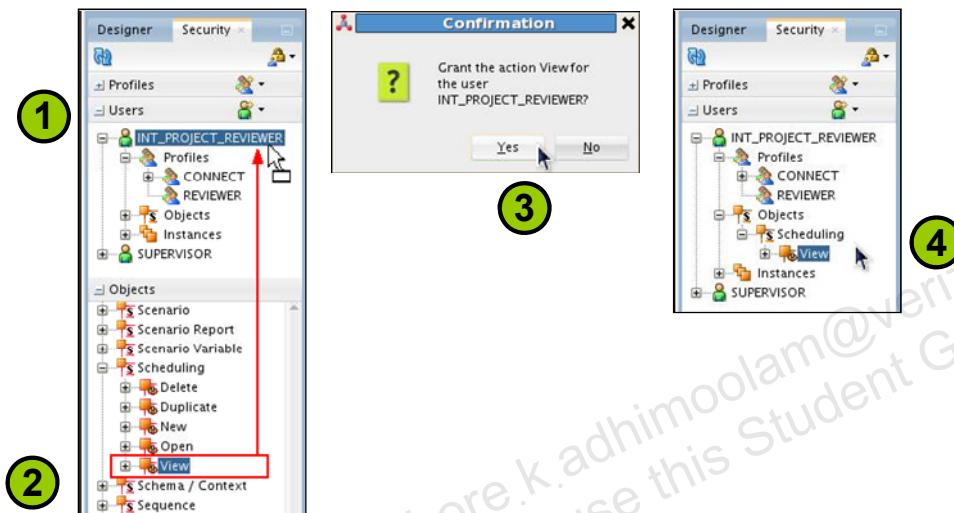
Note: To remove a profile from a user:

1. Expand the tree to display the profile that you want to delete, under the user branch.
2. Select the profile (under the user) to be deleted from the tree.
3. Right-click and select **Delete**.
4. Click **OK**.

The profile is removed from the user.

Assigning an Authorization by Profile or User

Select the method that you want to assign, then drag it onto the user or the profile.



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

To assign an authorization by profile or by user:

1. Expand the tree to display the user or the profile you want to assign the authorization to.
2. Under the object, select the method you want to assign, and then drag it onto the user or the profile.
3. Click **Yes**, and then click **Save**.
4. Verify that the new method is assigned to the user or profile.

The authorization is assigned to the user or the profile.

To assign authorizations on all the methods of an object to a profile or a user:

1. Expand the tree to display the user or the profile you want to assign the authorization to.
2. Select the object to which you want to assign all the methods, and then drag it onto the user or the profile.
3. Click **Yes**.

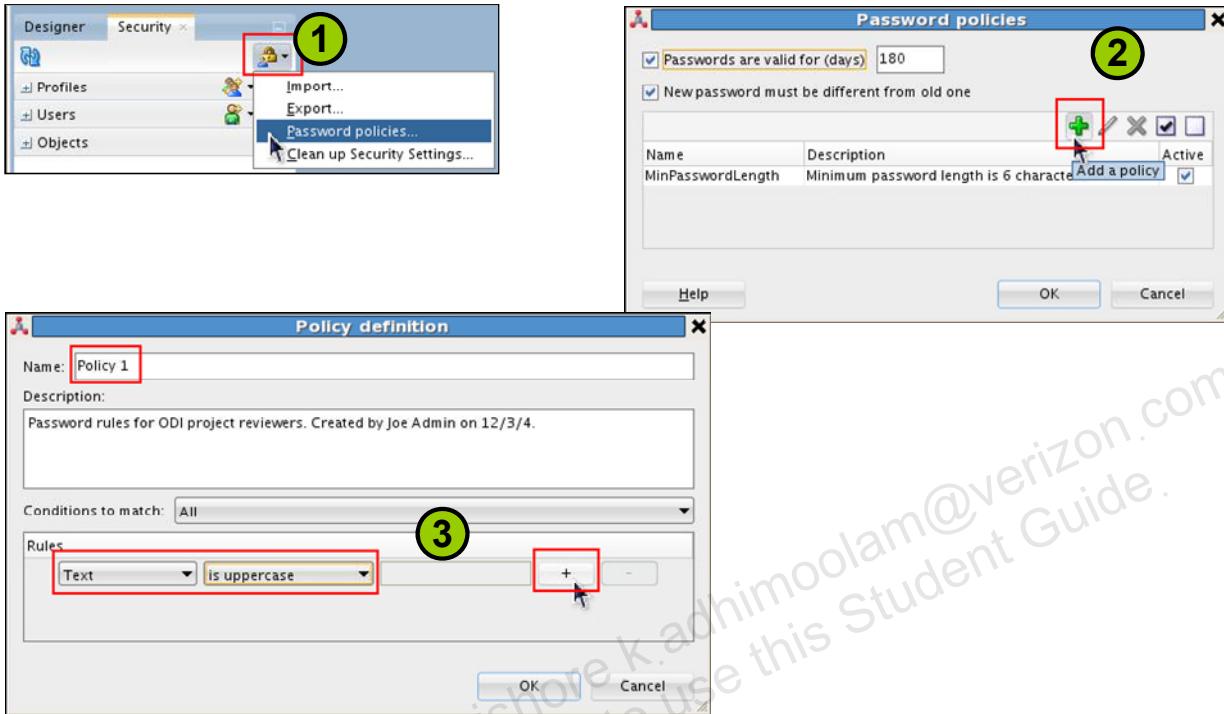
The authorizations on all the methods of this object are assigned to the user or the profile.

To delete an authorization by profile or user:

1. Select the method you want to delete under the user or profile branch.
2. Right-click and select **Delete**.
3. Click **Yes**.

The authorization is removed from the user or the profile.

Defining Password Policies



ORACLE

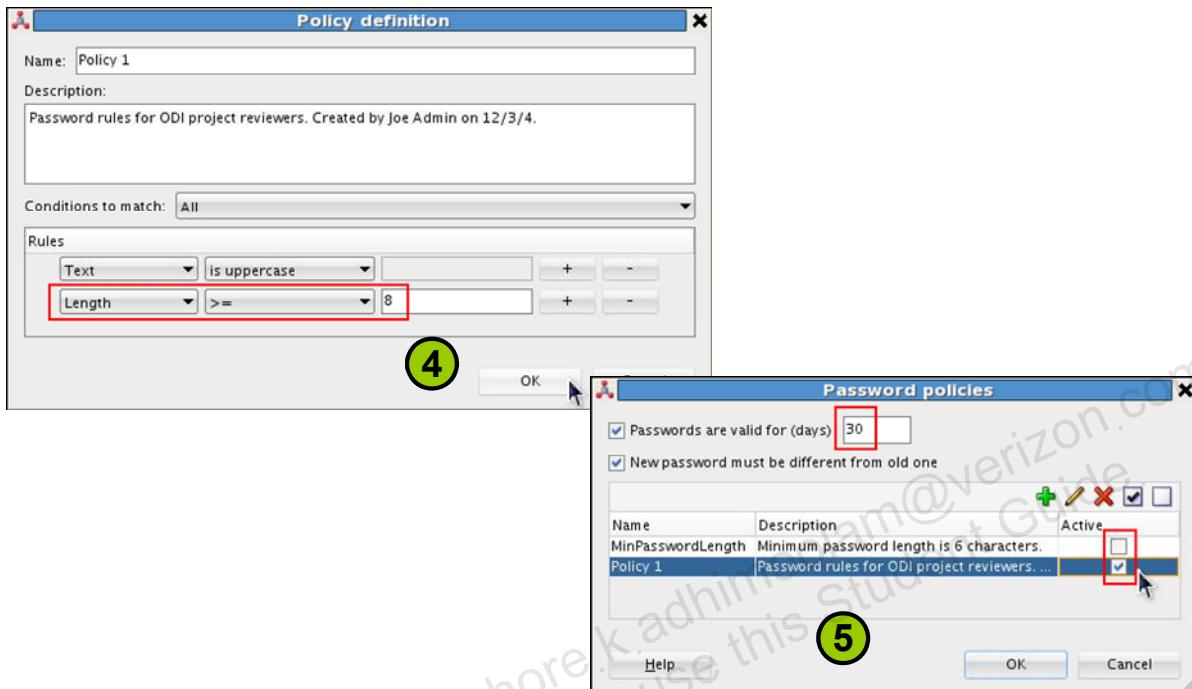
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The password policy consists of a set of rules applied on user passwords. This set of rules is checked when the password is defined by the user.

To define the password policy:

1. In *Security Navigator*, click the “Connect navigator” button, and then select **Password policy**. The “Password policies” window appears. In this window, a list of rules is displayed.
2. Click the **Add a policy** button. A new “Policy definition” window appears. A rule is a set of conditions that are checked on passwords.
3. Set a name and a description for this new rule. Click **Add rule** (the +).

Defining Password Policies

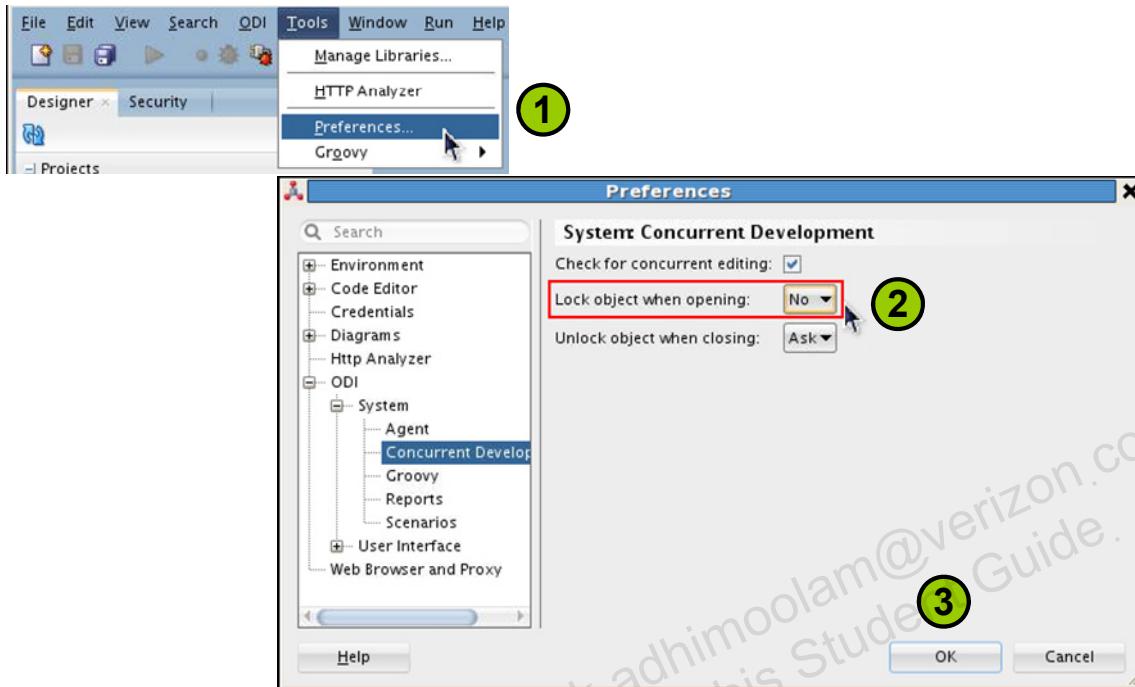


ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

4. Add conditions on the password value or length. You can define, for example, a minimum length for the passwords from this window. Select if you want at least one condition or all of them to be valid to consider that the password meets the rule. Click **OK**. You can add as many policies as necessary, and select the check boxes of the rules that you want active in your current policy. Only passwords that meet all the rules are considered valid for the current policy.
5. You can also define a period of validity for the passwords. Passwords older than this number of days will automatically expire; users have to change them. Click **OK** to update the password policy.

Setting User Preferences



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

ODI saves user parameters such as default directories, windows positions, and so on.

To set user parameters:

1. Select **Tools > Preferences** from the main menu bar.
2. In **Preferences**, change the value of parameters as required.
3. Click **OK** to save and close the window.

Note: A list of the possible user parameters is available in the reference manual: *Oracle Fusion Middleware Developer's Guide for Oracle Data Integrator Release 12c*.

ODI has a live, context-sensitive Help button that links you directly to the relevant documentation for this feature. The help shows what values are valid for each User Parameter. These parameters are specific to this user's workstation.

ODI Security Integration: Overview

- Implementing External Authentication (OPSS)
- Implementing External Password Storage
 - JPS Integration



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Oracle Data Integrator stores, by default, all the user information as well as the users' privileges into the Master Repository. A user who logs to Oracle Data Integrator logs against the Master Repository. This authentication method is called Internal Authentication. Oracle Data Integrator can optionally use Oracle Platform Security Services (OPSS) to authenticate its users against an external identity store, which contains enterprise users and passwords. Such an identity store is used at the enterprise level by all applications, in order to have centralized user and password definitions and Single Sign-On (SSO). In such a configuration, the repository contains only references to these enterprise users. This authentication method is called External Authentication.

Oracle Data Integrator stores, by default, all security information in the Master Repository. This password storage option is called Internal Password Storage. Oracle Data Integrator can optionally use JPS for storing critical security information. If you are using Java Provisioning Service (JPS) with Oracle Data Integrator, the data server passwords and contexts are stored in the JPS Credential Store Framework (CSF). This password storage option is called External Password Storage.

Implementing External Authentication (OPSS)

- Configuring ODI components for External Authentication
- Setting the Authentication mode
 - Setting up when creating the Master Repository with RCU
 - Switching the Authentication mode



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Configuring ODI Components for External Authentication

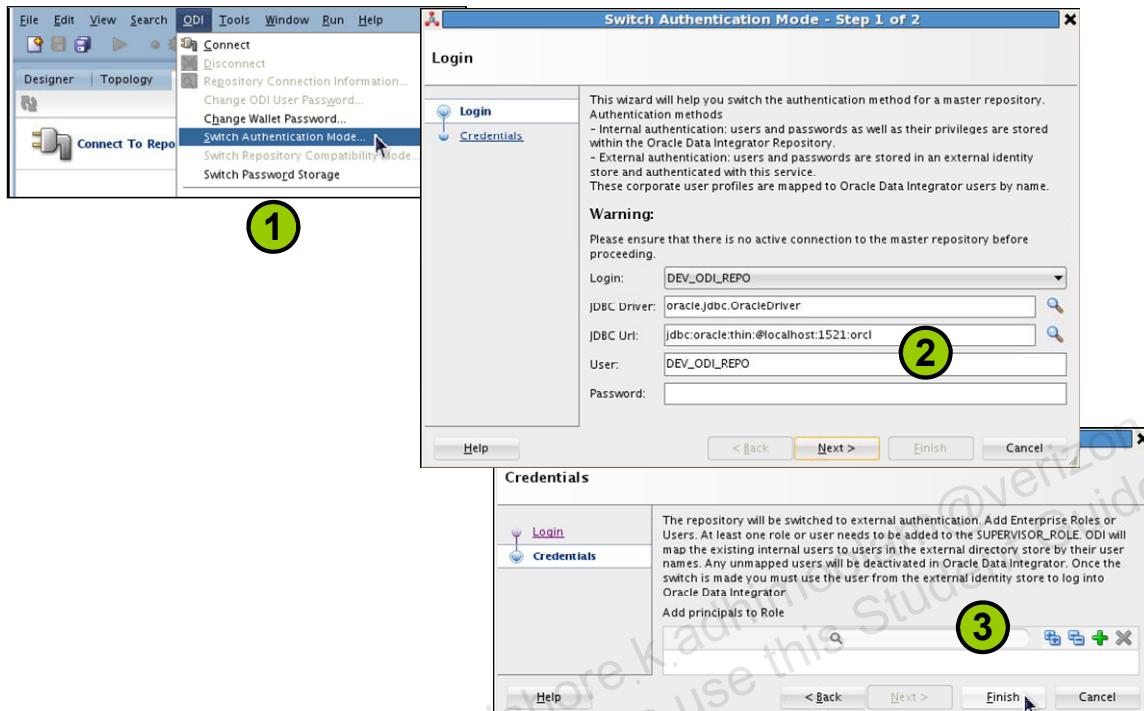
- To use the External Authentication option, you need to configure an Enterprise Identity Store (LDAP, Oracle Internet Directory, and so forth), and have this identity store configured for each Oracle Data Integrator component to refer by default to it. The configuration to connect and use the identity store is contained in an OPSS configuration file called `jps-config.xml`. Refer to the *Oracle Fusion Middleware Security Guide, 12c Release 1* for more information. Copy this file into the `ODI_HOME/client/odi/bin/` directory. The Studio reads the identity store configuration and authenticates it against the configured identity store.
- Oracle Data Integrator components deployed in a container (Java EE agent, Oracle Data Integrator Console) do not require a specific configuration. They use the configuration of their container. Refer to the *Oracle Fusion Middleware Security Guide, 12c Release 1* for more information about an OPSS configuration in a Java EE context.

Setting the Authentication mode:

- You can set or modify password storage in two ways:
 - Creating the Master Repository enables you to define the Authentication mode.
 - Switching the Authentication mode modifies the Authentication mode for an existing Master Repository.

Note: When you perform a password storage recovery, context and data server passwords are lost and need to be re-entered manually in Topology Navigator. If you are using External Authentication, users and password are externalized. Oracle Data Integrator privileges remain within the repository. Data servers and context passwords also remain in the Master Repository. You can externalize data server and context passwords by using the External Password Storage feature.

Implementing External Authentication (OPSS): Switching the Authentication Mode



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Switching the Authentication mode of the Oracle Data Integrator repository changes the way users authenticate. This operation must be performed by a Supervisor user. Use the Switch Authentication Mode wizard to change the user Authentication mode. Before launching the Switch Authentication Mode wizard, perform the following tasks:

1. Disconnect Oracle Data Integrator Studio from the repository. Shut down every component that uses the Oracle Data Integrator repository. From the ODI main menu, select **Switch Authentication Mode**. The Switch Authentication Mode wizard appears.
2. Specify the JDBC connectivity details of your Oracle Data Integrator Master Repository as defined when connecting to the Master Repository. Click **Next**.
3. Click **Finish**. The Authentication mode is changed.

Note: When switching from External to Internal Authentication, user passwords are not copied from the identity store to the repository. The passwords are nullified. All the user accounts are marked as expired and must be re-activated by a SUPERVISOR that is created during the switch. When switching from Internal to External Authentication, the users that exist in the repository and match a user in the identity store are automatically mapped. Users that do not match a user in the identity store are disabled. A Supervisor must edit each nonmatching user so that the user's name has a match in the identity store.

Implementing External Password Storage

There are four ways to set or modify password storage:

- Importing the Master Repository enables you to change the password storage.
- Creating the Master Repository enables you to define the password storage.
- Switching password storage modifies storage for an existing Master Repository.
- Recovering the password storage enables you to recover from a credential store crash.



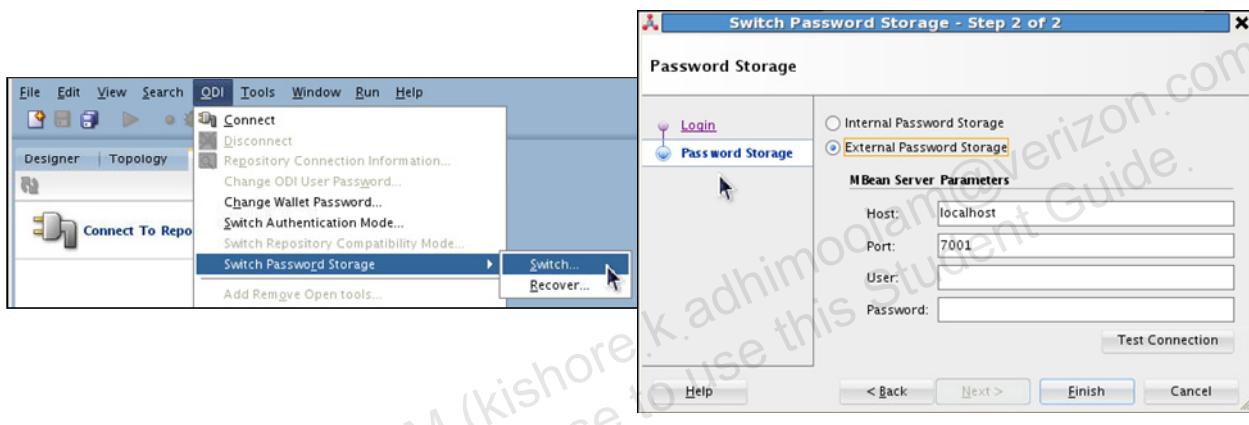
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

To use the External Password Storage option, you need to install a WebLogic Server instance configured with JPS, and all Oracle Data Integrator components (including the runtime Agent) need to have access to the remote credential store.

Implementing External Password Storage

Switching the password storage:

- **Disconnect** ODI Studio from the repository.
- **ODI** main menu: **Switch Password Storage > Switch**.
- Specify the login details of your ODI Master Repository.
- Select the **Password Storage mode** (Internal/External).



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

ORACLE

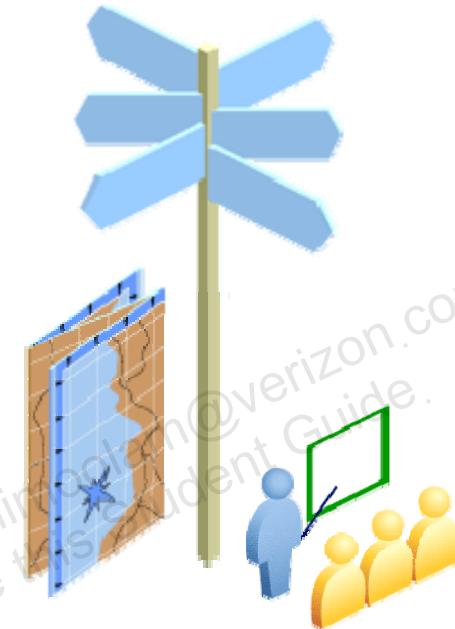
The login details of your Oracle Data Integrator Master Repository are defined when connecting to the Master Repository.

Select the password storage mode:

- Select **Internal Password Storage** if you want to store passwords in the Oracle Data Integrator repository.
- Select **External Password Storage** if you want to use JPS Credential Store Framework (CSF) to store the data server and context passwords.

Agenda

- Setting Up ODI Security
- **Managing ODI Reports**
- ODI Integration with Java EE



ORACLE

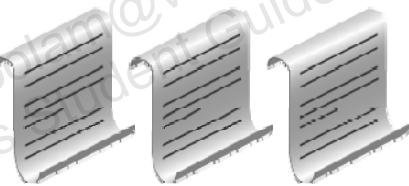
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The next section of this lesson examines the types of reports you can create in ODI.

Types of ODI Reports

With ODI, you can create the following PDF reports:

- Topology report
- Version comparison report
- ODI object report
- Execution simulation reports
- Diagram reports



ORACLE

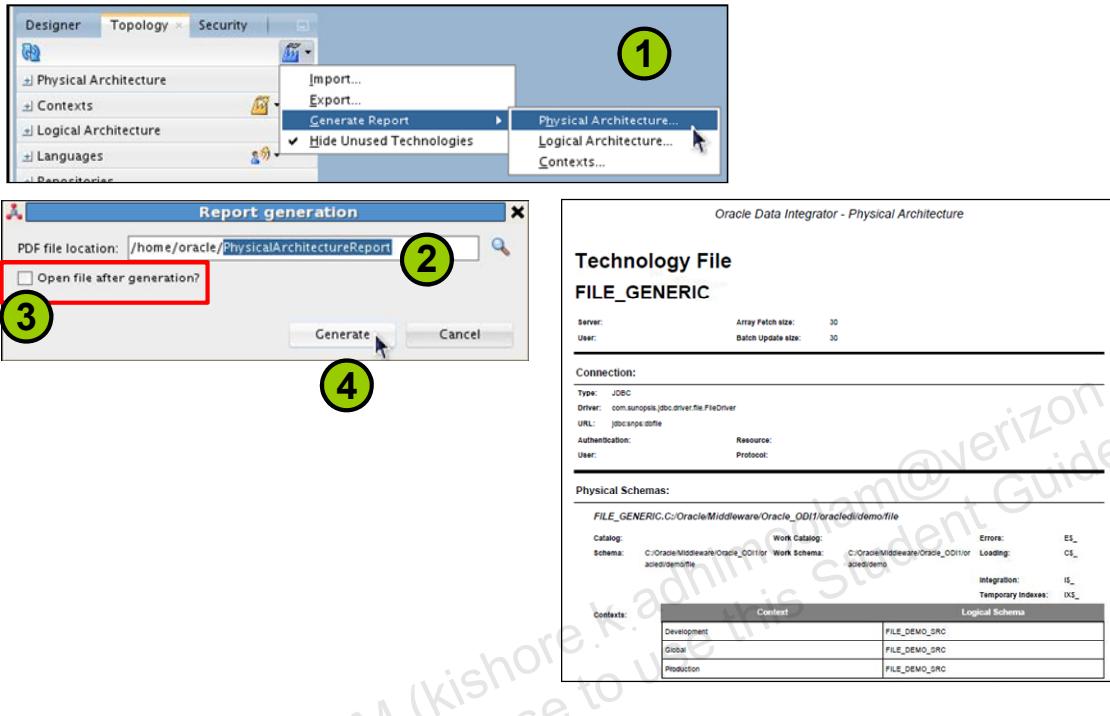
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In Oracle Data Integrator, you can print and share several types of reports with the PDF generation feature:

- Topology reports of the physical architecture, the logical architecture, or the contexts
- Reports of the version comparison results
- Reports of an ODI object
- Execution simulation reports
- Diagram reports (for diagrams created with Common Format Designer). For more information about this type of report, refer to *Oracle Fusion Middleware Developer's Guide for Oracle Data Integrator Release 12c*.

Tip: Most ODI prebuilt reports produce excellent documentation for collections of objects; for example, an entire Project, or the entire contents of a Model Folder, or a complete Topology. If you want to produce a report on an individual mapping, for example, that is easily accomplished by creating a new folder (call it "Print", for instance) and dragging that mapping temporarily into the new folder, then printing out the contents (one mapping) in the folder. Similarly, objects like datastores can be dragged into either Submodels or Model Folders and documentation printed.

Generating Topology Reports



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Using Oracle Data Integrator, you can generate Topology reports in PDF format of the physical architecture, the logical architecture, or the contexts.

To generate a topology report:

1. In the **Topology Navigator** tab, click the “Connect Navigator” icon. Select **Generate Report** and then the type of report that you want to generate:
 - Physical Architecture
 - Logical Architecture
 - Contexts
2. In the “Report generation” editor, enter the output PDF file location for your PDF report. Note that if no PDF file location is specified, the report in Adobe PDF format is generated in your default directory for PDF generation specified in the user parameters.
3. If you want to view the PDF report after generation, select the “Open file after generation?” option.
4. Click **Generate**.

The report is shown enlarged in the next slide.

Generated Topology Report: Example

Oracle Data Integrator - Physical Architecture

Technology Oracle

ORACLE_ORCL_LOCAL

Server:	ORCL	Array Fetch size:	30
User:	system	Batch Update size:	30

Connection:

Type:	JDBC
Driver:	oracle.jdbc.OracleDriver
URL:	jdbc:oracle:thin:@localhost:1521:ORCL
Authentication:	Resource:
User:	Protocol:

Physical Schemas:

ORACLE_ORCL_LOCAL.ORDERS			
Catalog:	Work Catalog:	Errors:	ES_
Schema:	ORDERS	Work Schema:	STAGING
		Loading:	CS_
		Integration:	IS_
		Temporary Indexes:	IXS_
Contexts:	Context	Logical Schema	
Development		ORACLE_ORCL_LOCAL_ORDERS	
Global		ORACLE_ORCL_LOCAL_ORDERS	
Production		ORACLE_ORCL_LOCAL_ORDERS	

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The slide shows an example report of the physical architecture of the ORACLE_ORCL_LOCAL.ORDERS physical schema. The table at the bottom shows each of the three related logical schemas, in terms of the three available contexts.

Version Comparison Report: Example

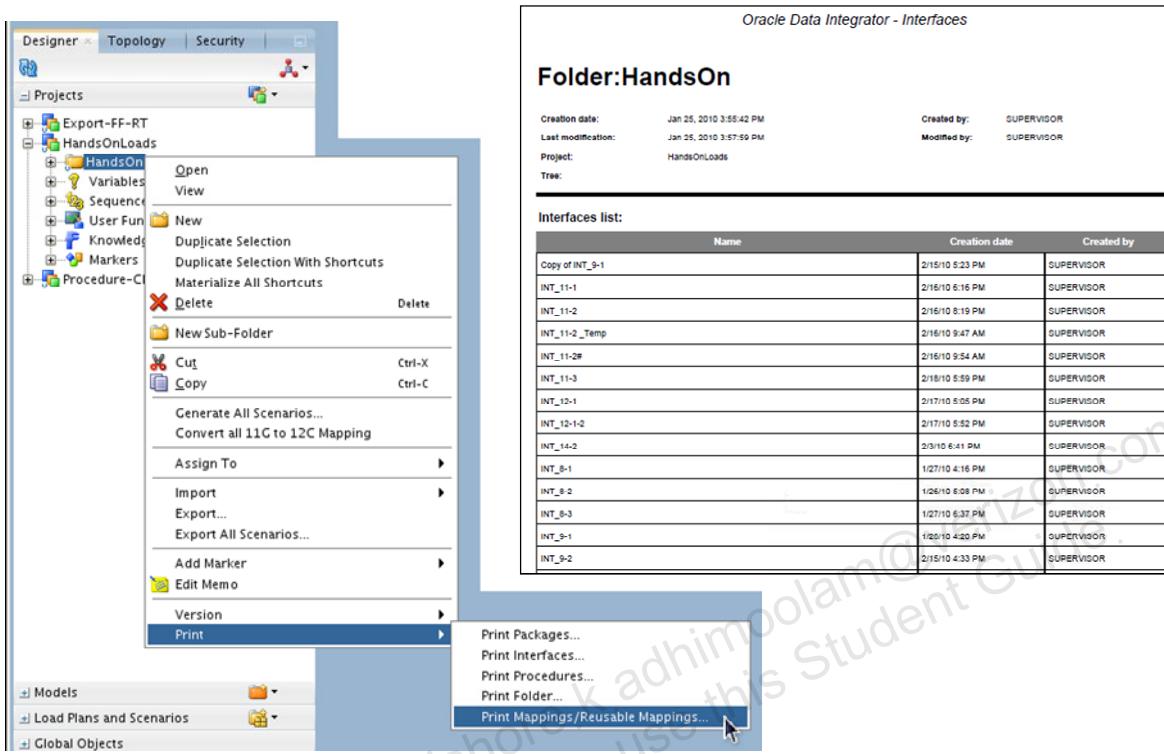
Version comparison report		
Interface : LD_SRC_AGE		
List of DataSet		
Name	1.0.0.0 (modified in the repository)	1.0.0.1
7001:Default	X	X
List of Interface Target Column		
Name	1.0.0.0 (modified in the repository)	1.0.0.1
7001:AGE_MAX	X	X
7001:AGE_RANGE	X	X
7001:AGE_MIN	X	X
List of Option Usage		
Name	1.0.0.0 (modified in the repository)	1.0.0.1
	X	X
	X	X
	X	X
Field details		
	1.0.0.0 (modified in the repository)	1.0.0.1
CommitWork	0	0
Distinct Rows	0	1
ErrorRep		

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

This slide shows a report of the version comparison results between versions 1.0.0.0 and 1.0.0.1 of the LD_SRC_AGE mapping. In the “Field details” section at the bottom of the screen, version 1.0.0.0 shows that the parameter for distinct rows is set to 0, while version 1.0.0.1 shows that parameter for distinct rows is set to 1.

Generating Object Reports



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In Designer Navigator, you can generate different types of reports depending on the type of object. You can generate the following reports for different type of objects:

- **Project:** Knowledge Modules
- **Project Folder:** Reports for Folder, Packages, Mappings, Procedures
- **Model Folder:** Model Folder report
- **Model:** Model report
- **Sub-model:** Sub-models report

To generate an Object report:

1. In Designer Navigator, select the object for which you want to generate a report.
2. In the “Report generation” editor, enter the output PDF file location for your PDF report. Note that if no PDF file location is specified, the report in Adobe PDF format is generated in your default directory for PDF generation specified in the user parameters.
3. If you want to view the PDF report after generation, select the “Open file after generation?” option.
4. Click **Generate**.

Agenda

- Setting Up ODI Security
- Managing ODI Reports
- **ODI Integration with Java EE**

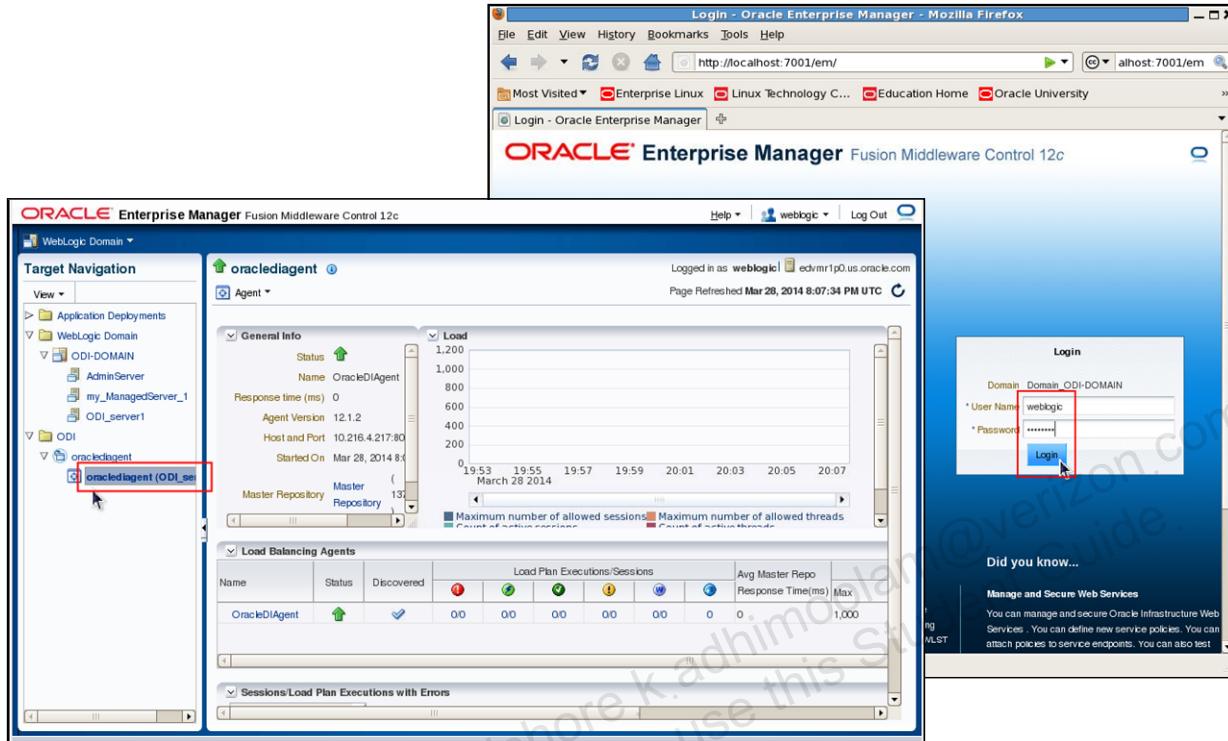


ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The last section of this lesson examines how to use the ODI Java EE agent to integrate ODI with Enterprise Manager.

Integration of ODI with Enterprise Manager



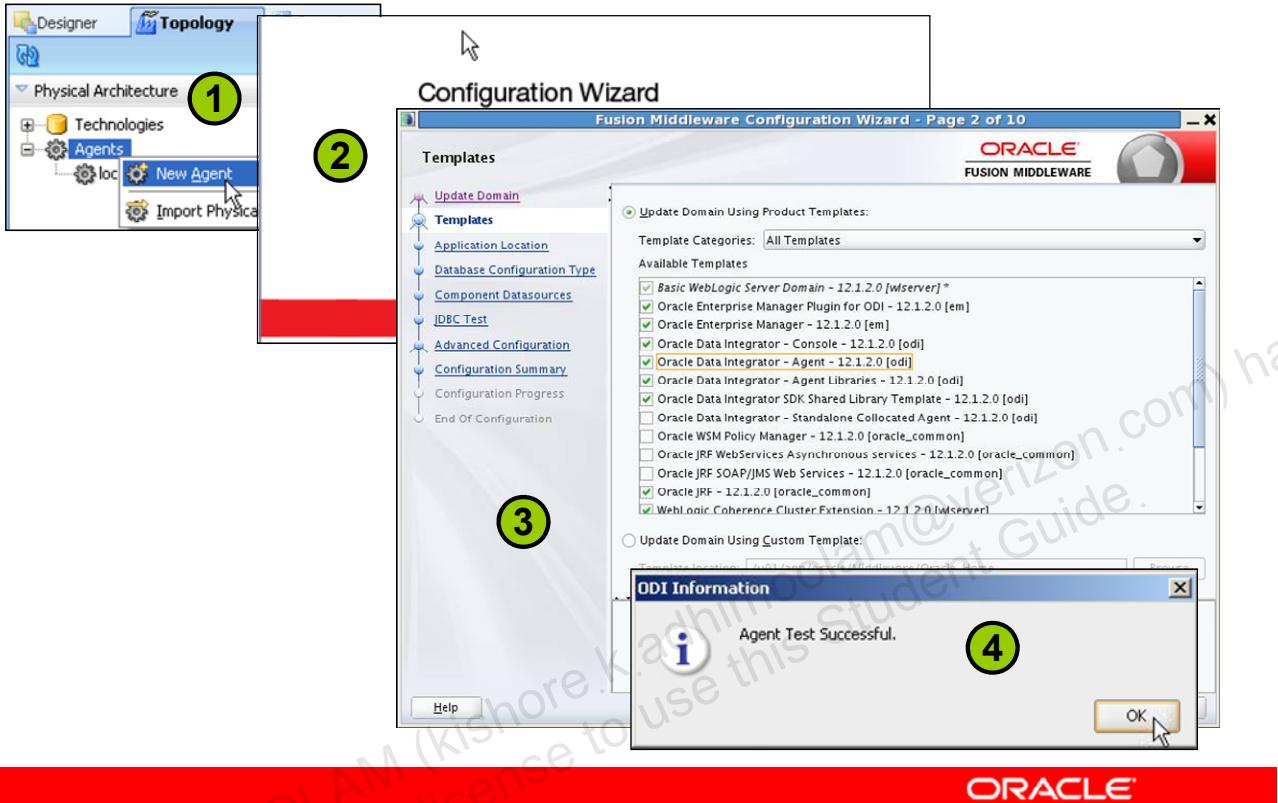
ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Oracle Data Integrator provides an extension integrated into the Fusion Middleware Control Console (Enterprise Manager). Oracle Data Integrator components can be monitored as a domain through this console, and administrators can have a global view of these components along with other Fusion Middleware components from a single administration console. To implement integration with Enterprise Manager:

- ODI Java EE agent must be deployed and configured with the existing WebLogic Server domain
- Enterprise Manager and the ODI Enterprise Manager Plug-in must be deployed in the WebLogic Server domain that has the ODI Java EE agent deployed and configured

Java EE Agent and Enterprise Manager Configuration with WebLogic Domain: Overview

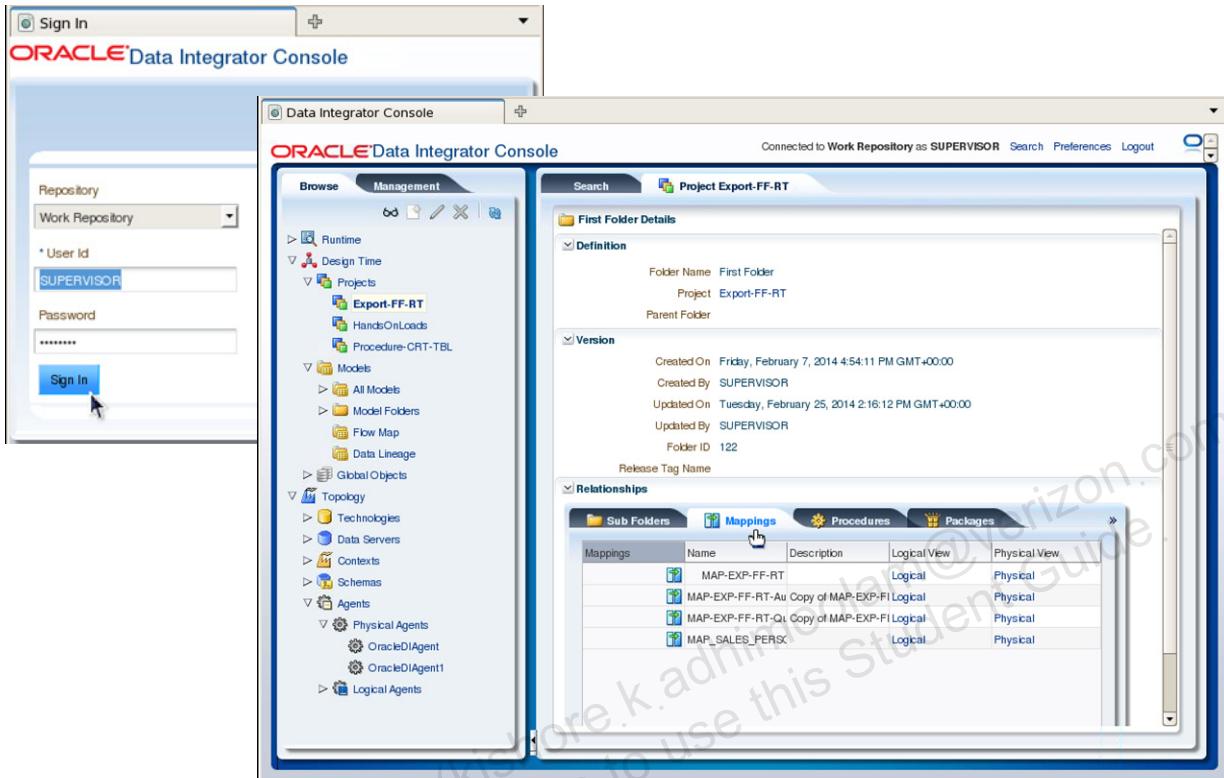


Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

1. In Topology Navigator, open Physical Architecture and then create a new ODI Agent named OracleDIAgent1.
2. To deploy and configure domains with WebLogic Server (WLS), you start the config.sh file from the ODI Home Install directory. Configuration Wizard also enables you to extend the existing WebLogic domain to support Java EE agent and Enterprise Manager plug-ins.
3. Security has to be set up for the Java EE application to have access to the ODI repository. The entry is created within the credential store, which allows the Java EE agent to authenticate itself so that it can use the required resources. This user must be a user who is already set up in ODI Security. In this practice, use the user SUPERVISOR.
4. From ODI Designer, test the connectivity of your configured ODI Java EE agent.

Note: Oracle JRF is included automatically.

Using ODI Console: Example



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

ODI Console enables an ODI developer to browse ODI objects and manage the ODI environment through the web service.

To start the Oracle Data Integrator Console, in a browser enter:

`http://<hostname>:<Port>/odiconsole`

Example:

<http://localhost:8002/odiconsole>

To log in, enter SUPERVISOR for user ID, and SUNOPSIS for password. Browse ODI objects in the tree view, select a node, and then click the View icon. The object or group of objects is displayed on the tab.

Quiz

Which of the following statements regarding ODI security concepts is not true?

- a. The objects are the visible part of ODI object components (Java classes).
- b. A method is a type of action that can be performed on an object. Each object has a series of methods that are specific to it.
- c. An authorization by a user is placed on a method of an object for a given use.
- d. Objects and methods can be changed in ODI.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Answer: d

Explanation: Objects and methods are predefined in ODI and must not be changed.

Summary

In this lesson, you should have learned how to:

- Use ODI Security Navigator to set:
 - Security policy
 - Password policy
 - User parameters
- Manage ODI reports
- Use methods of ODI Security integration
- Implement WebLogic Server, Enterprise Manager, and ODI Console integration with ODI



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Checklist of Practice Activities

- ✓ 1-1: Logging In and Help Overview
- ✓ 2-1: Creating and Connecting to ODI Master and Work Repositories
- ✓ 3-1: Configuring Standalone Agents Using the Common Admin Model
- ✓ 4-1: Working with Topology
- ✓ 5-1: Setting Up a New ODI Project
- ✓ 6-1: Creating Models by Reverse-Engineering
- ✓ 7-1: Checking Data Quality in the Model
- ✓ 8-1: Creating ODI Mapping: Simple Transformations
- ✓ 9-1: Creating ODI Mapping: Complex Transformations
- ✓ 9-2: Creating ODI Mapping: Implementing Lookup
- ✓ 10-1: Creating ODI Mapping: Exporting a Flat File to a Relational Table
- ✓ 11-1: Using Native Sequences with ODI Mapping
- ✓ 11-2: Using Temporary Indexes
- ✓ 11-3: Using Sets with ODI Mapping
- ✓ 12-1: Creating and Using Reusable Mappings
- ✓ 12-2: Developing a New Knowledge Module
- ✓ 13-1: Creating an ODI Procedure
- ✓ 14-1: Creating an ODI Package
- ✓ 14-2: Using ODI Packages with Variables and User Functions
- ✓ 15-1: Debugging Mappings
- ✓ 16-1: Creating and Scheduling Scenarios
- ✓ 17-1: Using Load Plans
- ✓ 18-1: Enforcing Data Quality with ODI Mappings
- ✓ 19-1: Implementing Changed Data Capture
- 20-1: **Setting Up ODI Security**
- 20-2: **Integrating with Enterprise Manager and Using ODI Console**

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Practice 20-1: Setting Up ODI Security

1. In ODI Security Navigator, duplicate the Repository Explorer profile and name this new profile *Reviewer*.
2. Grant three methods to the Reviewer profile: “Compare with version,” “Duplicate,” and “Export from Object Model.”
3. Create a new ODI user, `INT_PROJECT_REVIEWER`, and assign the profile of Reviewer to this user.
4. To connect to ODI repositories, assign the CONNECT generic profile to this new user.
5. Grant the View method for the Scheduling object to the `INT_PROJECT_REVIEWER` user.
6. Define user parameters to enable specific ODI functionality for the `INT_PROJECT_REVIEWER` user.
7. Define password policies to enforce passwords to be uppercase and greater than eight characters.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In this practice, you explore the security features of ODI Security Manager. You create a profile, grant three methods (actions) to that profile, and create a new ODI user to whom you assign that profile.

You then assign another profile (generic) to your new user, and then grant a specific method (View) for a specific object (Scheduling) for your new user.

Next, you define user parameters enabling specific functionality for your new user.

Finally, you define password policies that control the case sensitivity and length of user-defined passwords.

Practice 20-2: Integration with Enterprise Manager and Using ODI Console

1. Deploy and configure ODI Java EE agent and EM FMW, and ODI Console with existing WebLogic ODI - DOMAIN.
2. Connect to the WLS and managed server (odi_server1).
3. Connect to Enterprise Manager to administer ODI.
4. Use ODI Console to explore ODI environment.

