Oracle® Analytics

Managing Metadata Repositories for Oracle Analytics Server

F24225-11
August 2023

Oracle Analytics Managing Metadata Repositories for Oracle Analytics Server,

F24225-11

Copyright © 2020, 2023, Oracle and/or its affiliates.

Primary Author: Jenny Smalling

Contributing Authors: Stefanie Rhone, Hemala Vivek, Nick Fry

Contributors: Oracle Analytics Server development, product management, and quality assurance team

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

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

If this is software, software documentation, data (as defined in the Federal Acquisition Regulation), or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software," "commercial computer software documentation," or "limited rights data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

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

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

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

Preface

Audience xxiii
Documentation Accessibility xxiii
Diversity and Inclusion xxiii
Related Resources xxiv
Conventions xxiv

Part I Get Started with Metadata

1 Introduction to Building Your Metadata Repository

About Oracle BI Server Architecture 1-1
About Layers in the Oracle BI Repository 1-2
Analyze Your Business Model Requirements 1-4
Identify the Content of the Business Model 1-5
- Identify Logical Fact Tables 1-5
- Identify Logical Dimension Tables 1-6
- Identify Dimensions 1-7
  - About Dimensions with Multiple Hierarchies 1-8
- Identify Lookup Tables 1-8

Identify the Data Source Content for the Physical Layer 1-9
- About Types of Physical Schemas in Relational Data Sources 1-9
- About Cubes in Multidimensional Data Sources 1-10
- Identify the Data Source Table Structure 1-10

Guidelines to Design a Repository 1-11
- Design Strategies for Structuring the Repository 1-11
- Design Tips for the Physical Layer 1-11
- Design Tips for the Business Model and Mapping Layer 1-12
  - Model Outer Joins 1-14
- Design Tips for the Presentation Layer 1-14

Topics of Interest in Other Guides 1-15

2 Before You Begin

Open the Administration Tool 2-1
Set Administration Tool Options 2-1
- Oracle BI Administration Tool General Options 2-2
- Oracle BI Administration Tool Repository Options 2-4

Edit, Delete, and Reorder Objects in the Repository 2-6
About Name Requirements for Repository Objects 2-6
Change Icons for Repository Objects 2-6
Sort Objects in the Administration Tool 2-7
About the Oracle BI Server Command-Line Utilities 2-7
About Options in NQSConfig.INI 2-9
Download Repository Command 2-10
- What You Need to Know Before Using the Command 2-11

Use Online and Offline Repository Modes 2-12
- Edit Repositories in Offline Mode 2-12
  - Open Repositories in Offline Mode 2-13
  - Publish Offline Changes 2-13
- Edit Repositories in Online Mode 2-13
  - Open Repositories in Online Mode 2-14
  - Publish Online Changes 2-15
  - Guidelines for Using Online Mode 2-15
- Check Out Objects 2-16
- Check In Changes 2-17
- About Read-Only Mode 2-17
- Open a MDX XML Repository 2-18

Check the Consistency of a Repository or a Business Model 2-18
- About the Consistency Check Manager 2-18
- Run the Consistency Check Manager 2-20
- Use the validaterpd Utility to Check Repository Consistency 2-20
- Common Consistency Check Messages 2-22

Part II Build Your Metadata Repository

3 Set Up and Use the Multiuser Development Environment

About the Multiuser Development Environment 3-1
- About the Multiuser Development Process 3-2

Set Up Projects 3-3
- About Projects 3-3
- Create Projects 3-4

Set Up the Multiuser Development Directory 3-5
Identify the Multiuser Development Directory 3-5
Copy the Master Repository to the Multiuser Development Directory 3-6
Set Up a Pointer to the Multiuser Development Directory 3-6
Make Changes in a Multiuser Development Environment 3-7
About Changing and Testing Metadata 3-7
Make Changes to a Repository Using Projects 3-8
About Repository Project Checkout 3-8
Check Out Projects 3-9
Use the extractprojects Utility 3-9
Refresh the Local Project Extract 3-11
Make Changes to an Entire Repository 3-11
About Multiuser Development Options 3-11
Publish Changes to Multiuser Development Repositories 3-12
About the Multiuser Development Merge Process 3-12
How Are Multiuser Merges Different from Standard Repository Merges? 3-13
Publish to the Network 3-14
Enforce Consistent Repositories When Publishing Changes 3-14
Branch in Multiuser Development 3-15
About Branching 3-15
Use the Multi-Team, Multi-Release Model 3-17
Synchronize Branches 3-17
View and Delete History for Multiuser Development 3-17
View Multiuser Development History 3-18
Delete Multiuser Development History 3-18
Set Multiuser Development Options 3-18

4 Use a Source Control Management System for Repository Development

About Using a Source Control Management System with the Administration Tool 4-1
About MDS XML 4-2
Set Up Your System for Repository Development Under Source Control Management 4-4
Create an SCM Configuration File 4-4
Create an MDS XML Repository and Check In Files to the SCM System 4-6
Save an Existing Repository File in MDS XML Format 4-6
Create a New Repository in MDS XML Format 4-6
Link to Source Control Files to Convert Your Repository (Small Repositories Only) 4-7
Use Source Control Management in Day to Day Repository Development 4-7
Update, Save, and Check In Changes for Repositories Under Source Control 4-7
Handle Errors 4-8
Test Repositories Under Source Control 4-9

View the Source Control Log 4-9
Use Source Control Management with MUD 4-10
Put the MUD Master Repository and MUD Log File Under Source Control 4-10
Check In New Versions of the MUD Master and MUD Log File to Source Control 4-10
Manually Check In the Updated MUD Master Repository and Log File 4-11
Use a Script to Check In the Updated MUD Master Repository and Log File 4-11

5 Import Metadata and Working with Data Sources

About Importing Metadata and Working with Data Sources 5-1
Create an Oracle BI Repository 5-2
Perform Data Source Preconfiguration Tasks 5-2
Set Up ODBC Data Source Names (DSNs) 5-3
Set Up Oracle Database Data Sources 5-4
Oracle 12c Database In-Memory Data Sources 5-4
Oracle 12c on Exadata Data Sources 5-4
Advanced Oracle Database Features Supported by Oracle BI Server 5-5
Oracle Database Fast Application Notification and Fast Connection Failover 5-5
Additional Oracle Database Configuration for Client Installations 5-6
Configure Oracle BI Server When Using a Firewall 5-6
DataDirect Drivers and Oracle Database 5-6
About Setting Up Oracle OLAP Data Sources 5-7
Java Data Sources 5-7
Load Java Data Sources 5-7
About Setting Up Oracle TimesTen In-Memory Database Data Sources 5-7
Configure TimesTen Data Sources 5-8
Improve Use of System Memory Resources with TimesTen Data Sources 5-8
Configure Oracle BI Server to Access the TimesTen DLL on Windows 5-9
About Setting Up Essbase Data Sources 5-10
About Setting up Cloudera Impala Data Sources 5-10
Obtain Windows ODBC Driver for Cloudera 5-10
Import Cloudera Impala Metadata Using the Windows ODBC Driver 5-10
About Setting Up Apache Hive Data Sources 5-11
Obtain Windows ODBC Driver for Client Installation 5-11
Limitations on the Use of Apache Hive 5-12
About Setting Up Hyperion Financial Management Data Sources 5-14
Perform Additional Hyperion Configuration for Client Installations 5-16
Set Up Oracle RPAS Data Sources 5-16
Set Up Teradata Data Sources 5-16
Avoid Spool Space Errors for Queries Against Teradata Data Sources 5-18
Enable NUMERIC Data Type Support for Oracle Database and TimesTen 5-18

Configure Essbase to Use a Shared Logon 5-19
Configure SSO for Essbase, Hyperion Financial Management, or Hyperion Planning Data Sources 5-19
Import Metadata from Relational Data Sources 5-20
Import Metadata from Multidimensional Data Sources 5-21
Multidimensional Data Source Connection Options 5-23
About Importing Metadata from Oracle RPAS Data Sources 5-24
About Importing Metadata from XML Data Sources 5-25
About Using XML as a Data Source 5-26
Import Metadata from XML Data Sources Using the XML Gateway 5-27
Examples of XML Documents Generated by the Oracle BI Server XML Gateway 5-28
About Using HTML Tables as a Data Source 5-33
Import Metadata from XML Data Sources Using XML ODBC 5-35
Example of an XML ODBC Data Source 5-36
Examples of XML Documents 5-36
About Using a Standby Database 5-39
Configure a Standby Database 5-40
Create the Database Object for the Standby Database Configuration 5-41
Create Connection Pools for the Standby Database Configuration 5-41
Update Write-Back Scripts in a Standby Database Configuration 5-42
Set Up Usage Tracking in a Standby Database Configuration 5-42
Set Up Event Polling in a Standby Database Configuration 5-42
Set Up Oracle BI Scheduler in a Standby Database Configuration 5-43

6 Work with ADF Data Sources

What Are ADF Business Components? 6-1
About Operational Reporting with ADF Business Components 6-2
About Importing ADF Business Components 6-2
About Specifying a SQL Bypass Database 6-3
Set Up ADF Data Sources 6-3
Create a WebLogic Domain for ADF Business Components 6-4
Deploy OBIEEBroker as a Shared Library in Oracle WebLogic Server 6-4
Deploy the Application EAR File to Oracle WebLogic Server from JDeveloper 6-5
Set Up a JDBC Data Source in the WebLogic Server 6-8
Set the Logging Level for the Deployed Application in Oracle WebLogic Server 6-9
Import Metadata from ADF Data Sources 6-9
Perform an Initial Import from ADF Data Sources 6-10
Use Incremental Import to Propagate Flex Object Changes 6-11
Automatically Map Flex Object Changes to the Logical Model 6-12
Customize the Mapping Behavior 6-13
Manually Map Flex Object Changes to the Logical Model 6-13

Automatically Map Flex Object Changes Using the biserverextender Utility 6-14
Configure SSL in Oracle WebLogic Server 6-15
Configure One-Way SSL in Oracle WebLogic Server 6-15
Configure Two-Way SSL in Oracle WebLogic Server 6-15
Enable the Ability to Pass Custom Parameters to the ADF Application 6-17
Propagate Labels and Tooltips from ADF Data Sources 6-18
What Are Labels and Tooltips? 6-18
About the Session Variable Naming Scheme for UI Hints 6-19
About Determining the Physical Column for a Presentation Column 6-20
About Initializing Session Variables Automatically for Propagating UI Hints 6-20
Use UI Hints from an Oracle ADF Data Source When Creating Analyses 6-21
Use XML Code in Initialization Blocks to Query UI Hints 6-21
ADFQuery Element Reference 6-22

7 Set Up Database Objects and Connection Pools

Set Up Database Objects 7-1
About Database Types in the Physical Layer 7-1
Create a Database Object Manually in the Physical Layer 7-2
Database General Properties Reference 7-3
When to Allow Direct Database Requests by Default 7-3
SQL Features Supported by a Data Source 7-4
View Database Properties 7-6
Review Supported Database Features 7-6
About Connection Pools 7-6
About Connection Pools for Initialization Blocks 7-7
Create or Change Connection Pools 7-8
Set Connection Pool Properties in the General Tab 7-8
Common Connection Pool Properties in the General Tab 7-9
Multidimensional Connection Pool Properties in the General Tab 7-12
Set Connection Pool Properties in the Connection Scripts Tab 7-14
Set Connection Pool Properties in the XML Tab 7-14
Search Script Example 7-15
Set Connection Pool Properties in the Write Back Tab 7-16
Connection Pool Properties in the Miscellaneous Tab 7-17
Specify Application Properties for JDBC (Direct Driver) or JDBC (JNDI) Data Sources 7-18
EXECUTE PHYSICAL DATABASE 7-19
Set Up Persist Connection Pools 7-19
Remove the Persist Connection Pool Property 7-20
About Setting the Buffer Size and Transaction Boundary 7-20
List Connection Pool Command 7-20

Update Connection Pool Command 7-21
Use the BIServerT2PProvisioner.jar Utility to Change Connection Pool Passwords 7-23

8 Work with Physical Tables, Cubes, and Joins

About Working with the Physical Layer 8-1
Work with the Physical Diagram 8-2
Create Physical Layer Folders 8-4
- Create Physical Layer Catalogs and Schemas 8-4
  - Create Catalogs 8-4
  - Create Schemas 8-5
- Use a Variable to Specify the Name of a Catalog or Schema 8-5
- Set Up Display Folders in the Physical Layer 8-6

Work with Physical Tables 8-6
- About Tables in the Physical Layer 8-6
- About Physical Alias Tables 8-8
- Create and Manage Physical Tables and Physical Cube Tables 8-10
  - Create Physical Tables 8-10
  - Create Alias Tables 8-12
  - Set Physical Table Properties for XML Data Sources 8-12
- Create and Manage Columns and Keys for Relational and Cube Tables 8-12
  - Create and Edit a Column in a Physical Table 8-13
  - Specify a Primary Key for a Physical Table 8-14
  - Delete Physical Columns for All Data Sources 8-14
  - View Physical Column Properties 8-14
- View Data in Physical Tables or Columns 8-15

Work with Multidimensional Sources in the Physical Layer 8-15
- About Physical Cube Tables 8-15
- About Measures in Multidimensional Data Sources 8-16
  - About Externally Aggregated Measures 8-16
- About Working with Physical Dimensions and Physical Hierarchies 8-17
  - Work with Physical Dimension Objects 8-17
  - Work with Physical Hierarchy Objects 8-17
- View Members in Physical Cube Tables 8-19

Work with Essbase Data Sources 8-20
- About Using Essbase Data Sources 8-20
  - About Incremental Import 8-22
- Work with Essbase Alias Tables 8-23
  - Determine the Value to Use for Display 8-23
  - Explicitly Define Columns for Each Alias 8-24
- Model User-Defined Attributes 8-24

Associate Member Attributes to Dimensions and Levels 8-24
Model Alternate Hierarchies 8-25
Model Measure Hierarchies 8-26
Improve Performance by Using Unqualified Member Names 8-27
Work with Hyperion Financial Management and Hyperion Planning Data Sources 8-27
Import Metadata from Hyperion Financial Management Data Sources 8-27
Import Metadata from Hyperion Planning Data Sources 8-28
About Query Support for Hyperion Financial Management and Hyperion Planning Data Sources 8-29
Work with Oracle OLAP Data Sources 8-29
About Importing Metadata from Oracle OLAP Data Sources 8-30
Work with Oracle OLAP Analytic Workspace (AW) Objects 8-30
Work with Oracle OLAP Dimensions, Hierarchies, and Levels 8-31
Work with Oracle OLAP Cubes and Columns 8-32
Work with Physical Foreign Keys and Joins 8-33
About Physical Joins 8-33
About Primary Key and Foreign Key Relationships 8-34
About Complex Joins 8-34
About Multi-Database Joins 8-34
About Fragmented Data 8-35
Define Physical Joins with the Physical Diagram 8-36
Define Physical Joins with the Joins Manager 8-37
Deploy Opaque Views 8-37
About Deploying Opaque Views 8-38
Deploy Opaque View Objects 8-38
Use the Create View SELECT Statement 8-38
Undeploy a Deployed View 8-40
When to Delete Opaque Views or Deployed Views 8-40
When to Redeploy Opaque Views 8-41
Use Hints in SQL Statements 8-41
How to Use Oracle Hints 8-41
About the Index Hint 8-42
About the Leading Hint 8-42
About Performance Considerations for Hints 8-42
Create Hints 8-43
Display and Update Row Counts for Physical Tables and Columns 8-43
Display Row Counts in the Physical Layer 8-44

9 Work with Logical Tables, Joins, and Columns

About Working with the Business Model and Mapping Layer 9-1
Create the Business Model and Mapping Layer 9-2

Create Business Models 9-2
Automatically Create Business Model Objects 9-2
Automatically Create Business Model Objects for Multidimensional Data Sources 9-3
Duplicate a Business Model and Subject Area 9-3
About Working with the Business Model Diagram 9-3
Create and Manage Logical Tables 9-4
Create Logical Tables 9-5
Enable Data Driven Fragment Selection in Logical Table Sources 9-6
Specify a Primary Key in a Logical Table 9-6
Review Foreign Keys for a Logical Table 9-6
Define Logical Joins 9-6
Define Logical Joins with the Business Model Diagram 9-7
Define Logical Joins with the Joins Manager 9-8
Create Logical Joins with the Joins Manager 9-9
Create Logical Foreign Key Joins with the Joins Manager 9-9
Specify a Driving Table 9-10
Factors That Determine Join Trimming 9-11
Identify Physical Tables That Map to Logical Objects 9-14
Create and Manage Logical Columns 9-15
Create Logical Columns 9-15
Base the Sort for a Logical Column on a Different Column 9-16
Enable Double Column Support by Assigning a Descriptor ID Column 9-16
Create Derived Columns 9-17
Configure Logical Columns for Multicurrency Support 9-17
Set Default Levels of Aggregation for Measure Columns 9-19
Set Up Dimension-Specific Aggregate Rules for Logical Columns 9-20
Specify Dimension-Specific Aggregation Rules for Multiple Logical Columns 9-21
Define Aggregation Rules for Multidimensional Data Sources 9-21
Associate an Attribute with a Logical Level in Dimension Tables 9-23
Move or Copy Logical Columns 9-23
Enable Write Back On Columns 9-24
Set Up Display Folders in the Business Model and Mapping Layer 9-26
Model Bridge Tables 9-27
Create Joins in the Physical Layer for Bridge and Associated Dimension Tables 9-28
Model the Associated Dimension Tables in a Single Dimension 9-28
Model the Associated Dimension Tables in Separate Dimensions 9-29
Model Binary Large Object (BLOB) Data and Character Large Object (CLOB) Data 9-29

10 Work with Logical Dimensions

About Working with Logical Dimensions 10-1
Create and Manage Dimensions with Level-Based Hierarchies 10-2
About Level-Based Hierarchies 10-2
About Using Dimension Hierarchy Levels in Level-Based Hierarchies 10-5
Manually Create Dimensions, Levels, and Keys with Level-Based Hierarchies 10-6
Create Dimensions in Level-Based Hierarchies 10-6
Create Logical Levels in a Dimension 10-7
Associate a Logical Column and Its Table with a Dimension Level 10-7
Identify the Primary Key for a Dimension Level 10-11
Select and Sort Chronological Keys in a Time Dimension 10-11
Add a Dimension Level to the Preferred Drill Path 10-12
Add Sequence Numbers to a Time Dimension's Logical Level 10-12
Rules for Automatically Created Dimensions with Level-Based Hierarchies 10-13
Automatically Create Dimensions with Level-Based Hierarchies 10-14
Populate Logical Level Counts Automatically 10-14
Create and Manage Dimensions with Parent-Child Hierarchies 10-15
About Parent-Child Hierarchies 10-15
About Levels and Distances in Parent-Child Hierarchies 10-16
About Parent-Child Relationship Tables 10-17
Create Dimensions with Parent-Child Hierarchies 10-18
Define Parent-Child Relationship Tables 10-19
Model Aggregates for Parent-Child Hierarchies 10-20
Store Facts for Parent-Child Hierarchies 10-20
Aggregate Parent-Child Hierarchies 10-22
Add the Parent-Child Relationship Table to the Model 10-23
Maintain Parent-Child Hierarchies Based on Relational Tables 10-24
Model Time Series Data 10-24
About Time Series Functions 10-24
About the AGO Function 10-25
About the TODATE Function 10-26
About the PERIODROLLING Function 10-27
Create Logical Time Dimensions 10-28
Select the Time Option in the Logical Dimension Dialog 10-29
Set Chronological Keys for Each Level 10-30
Create AGO, TODATE, and PERIODROLLING Measures 10-31

11 Manage Logical Table Sources (Mappings)

About Logical Table Sources 11-1
How Fact Logical Table Sources Are Selected to Answer a Query 11-1

How Dimension Logical Table Sources Are Selected to Answer a Query 11-2
Change the Default Selection Criteria for Dimension Logical Table Sources 11-2
Consistency Among Data in Multiple Sources 11-3
Create Logical Table Sources 11-3
Set Priority Group Numbers for Logical Table Sources 11-4
Define Physical to Logical Table Source Mappings and Creating Calculated Items 11-7
Unmap a Logical Column from Its Source 11-9
Define Content of Logical Table Sources 11-9
Verify Joins from Dimension Tables to Fact Tables 11-10
Joins from Dimension Tables to Fact Tables 11-11
Logical Table Source Options Reference 11-12
About WHERE Clause Filters 11-13
About Working with Parent-Child Settings in the Logical Table Source 11-14
Set Up Aggregate Navigation by Creating Sources for Aggregated Fact Data 11-14
Set Up Fragmentation Content for Aggregate Navigation 11-15
Specify Fragmentation Content for Single Column, Value-Based Predicates 11-15
Specify Fragmentation Content for Single Column, Range-Based Predicates 11-16
Specify Multicolumn Content Descriptions 11-16
Specify Parallel Content Descriptions 11-17
Specify Unbalanced Parallel Content Descriptions 11-19
Specify Fragmentation Content for Aggregate Table Fragments 11-19
Specify the Aggregate Table Content 11-20
Define a Physical Layer Table with a Select Statement to Complete the Domain 11-21
Specify the SQL Virtual Table Content 11-21
Create Physical Joins for the Virtual Table 11-22

12 Create and Maintain the Presentation Layer

About the Presentation Layer 12-1
Create and Customize the Presentation Layer 12-1
About Creating Subject Areas 12-2
Automatically Create Subject Areas Based on Logical Stars and Snowflakes 12-3
About Removing Columns 12-3
Rename Presentation Columns to User-Friendly Names 12-3
Export Logical Keys in the Subject Area 12-4
Set an Implicit Fact Column in the Subject Area 12-4
Maintain the Presentation Layer 12-4
Work with Subject Areas 12-5
Work with Presentation Tables and Columns 12-6
Create and Manage Presentation Tables 12-6
Reorder Presentation Layer Tables 12-7

About Presentation Columns 12-7
Change the Presentation Column Name 12-8
Reorder Presentation Columns 12-8
Nest Folders in Answers and BI Composer 12-8
Work with Presentation Hierarchies and Levels 12-9
Create and Manage Presentation Hierarchies 12-10
Model Dimensions with Multiple Hierarchies in the Presentation Layer 12-11
Edit Presentation Hierarchy Objects 12-13
Create and Manage Presentation Levels 12-14
Set Permissions for Presentation Layer Objects 12-15
Generate a Permission Report for Presentation Layer Objects 12-16
Sort Columns in the Permissions Dialog 12-16
Create Aliases (Synonyms) for Presentation Layer Objects 12-16
Control Presentation Object Visibility 12-17

13 Create and Persist Aggregates for Oracle BI Server Queries

About Aggregate Persistence 13-1
Aggregate Persistence Improvements 13-2
About Aggregate Persistence Errors 13-3
Identify Query Candidates for Aggregation 13-4
Use Oracle BI Summary Advisor to Identify Query Candidates for Aggregation 13-4
About Oracle BI Summary Advisor 13-5
Gather Summary Advisor Statistics 13-5
Generate and Use Summary Advisor Recommendations 13-6
About Measure Subset Recommendations 13-6
Set Up the Statistics Database 13-6
Columns in the S_NQ_SUMMARY_ADVISOR Table 13-7
Turn On Usage Tracking 13-7
Turn On Summary Advisor Logging 13-8
Generate an Aggregate Specification Script 13-9
Summary Advisor Stop Criteria Run Constraints 13-11
Use the nqaggradvisor Utility to Run the Oracle BI Summary Advisor 13-12
Use the Aggregate Persistence Wizard to Generate the Aggregate Specification 13-13
Use Model Check Manager to Check for Modeling Problems 13-15
About Model Check Manager 13-16
Run Model Check Manager 13-16
Resolve Model Errors 13-17
Check Models Using the validaterpd Utility 13-17
Write the Create Aggregates Specification Manually 13-19
What Constraints Are Imposed During the Create Process? 13-19

Write the Create Aggregates Specification 13-20
Delete Statement for Aggregate Specification 13-20
Create Statement for Aggregate Specification 13-21
Multiple Aggregates in Aggregate Specification 13-21
Where Clause for Aggregate Specification 13-21
Add Surrogate Keys to Dimension Aggregate Tables 13-22
About the Create/Prepare Aggregates Syntax 13-23
About Surrogate Key Output from Create/Prepare Aggregates 13-23
Run the Aggregate Specification Script 13-24
Lifecycle Use Cases for Aggregate Persistence 13-25
Use Double Buffering to Refresh Highly Available Aggregates 13-27
Create Aggregates on TimesTen Sources 13-28
Enable PL/SQL for TimesTen 13-28
Enable Performance Enhancement Features for TimesTen 13-28

14 Apply Data Access Security to Repository Objects

About Data Access Security 14-2
Where to Find Information About Security Tasks 14-2
Row-Level Security 14-3
Set Up Row-Level Security 14-4
Data Filters 14-5
Set Up Data Filters in the Repository 14-6
Specify a Functional Group for an Application Role 14-7
Set Up Row-Level Security in the Database 14-8
Object Permissions 14-9
Set Up Object Permissions 14-10
About Permission Inheritance for Users and Application Roles 14-11
Overview of User and Application Role Commands 14-13
Rename Application Role Command 14-13
Delete Application Role Command 14-15
Rename Users Command 14-16
Delete Users Command 14-18
Set Query Limits 14-19
Access the Query Limits Functionality in the Administration Tool 14-20
Limit Queries By the Number of Rows Received 14-20
Limit Queries By Maximum Run Time and Restricting to Particular Time Periods 14-21
Allow or Disallow Direct Database Requests 14-22
Allow or Disallow the Populate Privilege 14-22
About Applying Data Access Security in Offline Mode 14-23

Set Up Placeholder Application Roles for Offline Repository Development 14-23

15 Complete Oracle BI Repository Setup

Save the Repository and Check Consistency 15-1
Use nqcmd to Test and Refine the Repository 15-2
nqcmd Command Line Arguments 15-2
Upload Repository Command 15-4
Make the Repository Available for Queries 15-6
Create Data Source Connections to the Oracle BI Server for Client Applications 15-6
Publish to the User Community 15-6

16 Set Up Data Sources on Linux

About Setting Up Data Sources on Linux 16-1
Settings for Data Source Connections Using Native Gateways 16-2
Sample obis.properties Entries for Oracle Database 16-2
Configure Data Source Connections Using Native Gateways 16-3
About Updating Row Counts in Native Databases 16-4
Troubleshoot OCI Connections 16-4
Use DataDirect Connect ODBC Drivers on Linux 16-5
Configure Oracle Analytics Server to Use DataDirect 16-5
Additional DataDirect Configuration for Oracle Essbase 16-6
Configure the DataDirect Connect ODBC Driver for DB2 Database 16-6
Configure the DataDirect Connect ODBC Driver for MySQL Database 16-8
Configure the DataDirect Connect ODBC Driver for Sybase ASE Database 16-9
Configure the DataDirect Connect ODBC Driver for Informix Database 16-11
Configure the DataDirect Connect ODBC Driver for Cloudera Impala Database 16-12
Configure Impala 1.3.x to Include a LIMIT Clause 16-14
Modify the Impala DefaultOrderByLimit Alternate Methods 16-14
Configure the DataDirect Connect ODBC Driver for Apache Hive Database 16-15
Configure Database Connections Using Native ODBC Drivers 16-17
Set Up Oracle TimesTen In-Memory Database on Linux 16-18
Configure Essbase Data Sources on Linux 16-19

17 Manage Oracle BI Repository Files

Compare Repositories 17-1
Compare Repositories Using the Compare Dialog 17-1
Compare Repositories Using comparerpd 17-2
Turn Off Compare Mode 17-3
Equalize Objects 17-3

About Equalizing Objects 17-4
View the Upgrade ID for Repository Objects 17-4
Use the Equalize Objects Dialog 17-5
Use the equalizerpds Utility 17-6
About Values for TypeName 17-7
Merge Repositories 17-8
Perform Full Repository Merges 17-9
About Full Repository Merges 17-9
Perform Full Repository Merges With a Common Parent 17-11
Perform Full Repository Merges Without a Common Parent 17-15
Perform Patch Merges 17-16
About Patch Merges 17-16
Generate a Repository Patch 17-17
Apply a Repository Patch 17-17
Query and Manage Repository Metadata 17-20
Query Related Objects 17-20
Repository Query Options 17-21
Query the Repository 17-21
Construct a Filter for Query Results 17-22
Query Filter Examples 17-22
Configure the Repository for Large Complex Queries 17-24
Query Data Models Remotely 17-24
Change the Oracle BI Repository Password 17-24
Change the Oracle BI Repository Password Using the Administration Tool 17-25
Change the Oracle BI Repository Password Using the obieerpdpwdchg Utility 17-25

18 Use Expression Builder and Other Utilities

Use Expression Builder 18-1
About the Expression Builder Dialogs 18-1
About the Expression Builder Toolbar 18-2
About the Categories in the Category Pane 18-2
Set Up an Expression 18-3
Navigate Within Expression Builder 18-5
Build an Expression 18-5
About the INDEXCOL Conversion Function 18-6
Use Administration Tool Utilities 18-6
Use the Replace Column or Table Wizard 18-7
Use the Event Tables Utility 18-7
Use the Externalize Strings Utility 18-8
Use the Rename Wizard 18-9

Use the Update Physical Layer Wizard 18-9
Generate Documentation of Repository Mappings 18-10
Generate a Metadata Dictionary 18-11
Provide Access to Metadata Dictionary Information 18-12
Remove Unused Physical Objects 18-13
Persist Aggregates 18-14
Use the Convert Presentation Folders Utility 18-14
Generate a List of Logical Column Types 18-14
Use the biservergentypexml Utility to Generate a List of Logical Column Types 18-15
Compare Logical Column Types 18-16
Fix Upgrade IDs 18-16
Set Permissions In Bulk 18-16
Use the Calculation Wizard 18-17
Associate S_NQ_ACCT Record with the Query Log 18-18

19 Use Variables in the Oracle BI Repository

Work with Repository Variables 19-1
About Repository Variables 19-1
About Static Repository Variables 19-1
About Dynamic Repository Variables 19-2
Create Repository Variables 19-3
Use Repository Variables in Expression Builder 19-3
Work with Session Variables 19-4
About Session Variables 19-4
About System Session Variables 19-4
About Nonsystem Session Variables 19-6
Create Session Variables 19-6
Work with Initialization Blocks 19-7
About Using Initialization Blocks with Variables 19-8
Initialize Dynamic Repository Variables 19-8
Initialize Session Variables 19-8
About Row-Wise Initialization 19-9
Create Initialization Blocks 19-10
Create Session Variable Initialization Blocks 19-10
Assign a Name and Schedule to Initialization Blocks 19-11
Select and Test the Data Source and Connection Pool 19-12
Variable Order in Initialization Blocks 19-17
Associate Variables with Initialization Blocks 19-18
Establish Execution Precedence 19-19
When Processing of Session Variable Initialization Blocks Can't Be Deferred 19-19

Enable and Disable Initialization Blocks 19-20
Work with Multi-Source Session Variables 19-21
Example to Illustrate the Creation and Usage of Multi-Source Session Variables 19-21
List Repository Variables Command 19-23
Update Repository Variables Command 19-25

20 Manage the Repository Lifecycle in a Multiuser Development Environment

Plan Your Multiuser Development Deployment 20-1
About Business Organization and Governance Process Best Practices 20-2
About Technical Team Roles and Responsibilities 20-2
Multiuser Development Architecture 20-3
About Multiuser Development Concepts 20-4
About Multiuser Development Styles 20-5
Multiuser Development Sandbox Architecture 20-10
Multiuser Development and Lifecycle Management Architecture 20-12
Understand the Multiuser Development Environment 20-14
About Multiuser Development Environment Task Flow 20-15
About Multiuser Development Projects 20-16
How to Create Branches 20-17
How to Create a Main Branch 20-17
How to Create a Side Branch 20-17
How to Create a Delegated Administration Branch 20-19
Which Merge Utility Should I Use? 20-20
MUD Tips and Best Practices 20-20
Best Practices for Branching 20-21
Best Practices for Setting Up Projects 20-21
Best Practices for Three-Way Merges 20-22
Best Practices for MUD Merges 20-22
Best Practices for Two-Way Merges 20-23
Best Practices for Production Migration 20-24
Best Practices for Application Roles and Users 20-25
Troubleshoot Multiuser Development 20-25

21 MUD Case Study: Eden Corporation

About the Eden Corporation Fictional Case Study 21-1
Phase I - Initiate Multiuser Development (MUD) 21-3
Start Initiative S 21-4
Set Up MUD Projects 21-5

First Developer Checks Out 21-6
Second Developer Checks Out 21-9
First Developer Publishes Changes to the Master MUD Repository 21-10
Second Developer Publishes Changes to the Master MUD Repository 21-10
MUD Administrator Test Migration Activities 21-11
Phase I Test 21-11
Phase I Migrate to Production 21-12
Phase I Summary 21-13
Phase II - Branch, Fix, and Patch 21-13
Set Up the Second Branch 21-14
Developers Check Out Projects 21-14
Patch Fix for the Main Branch 21-14
Finish and Merge Phase II Branch 21-18
Phase II Summary 21-18
Phase III - Independent Semantic Model Development 21-19
Security Considerations for Multiple Independent Semantic Models 21-19
HR Semantic Model Developer Builds Content 21-20
Phase III Summary 21-21

22 Merge Rules

About the Merge Process 22-1
Merge Rules and Behavior for Full Merges 22-2
Special Merge Algorithms for Logical Table Sources and Other Objects 22-3
Merge Objects that Use the Vector Merge Algorithm 22-3
Merge Logical Table Sources 22-5
Merge Security Filters 22-5
Infer the Use Logical Column Property for Presentation Columns 22-5
Merge Aliases 22-6
Merge Rules and Behavior for Multiuser Development Merges 22-6
Merge Rules and Behavior for Patch Merges 22-7
Use Patchrpd to Automate the Patch Process 22-7

23 Delete Unwanted Objects from the Repository

About the Object Pruning Utility 23-1
Use the Object Pruning Utility 23-1
Create the Input File 23-1
Run the prunerpd Utility 23-2
Deletion Rules for the Object Pruning Utility 23-3

24 Data Types Supported by Oracle Analytics Server

Supported Data Types 24-1
- Textual Data 24-1
- Numeric Data 24-1
- Date and Time Data 24-2
- Binary Data 24-2

Use the NQSGetSQLDataTypes Procedure to Access Data Type Information 24-2
Data Type Limitations 24-2
- Floating Point Limitations 24-4

Other Oracle BI Server Limitations 24-5
Data Type Mapping in Oracle Database and Oracle Analytics Server 24-5

25 Exchange Metadata with Databases to Enhance Query Performance

About Exchanging Metadata with Databases 25-1
Generate the Import File 25-1
- Run the Generator 25-2
- About the Metadata Input File 25-4
- About the Output Files 25-5
- Troubleshoot Errors from the Generator 25-5
- Metadata Conversion Rules and Error Messages 25-6
  - Conversion Rules for Oracle Databases 25-6

Use Materialized Views in the Oracle Database with Oracle Analytics Server 25-7
- About Using the SQL Access Advisor with Materialized Views 25-7
- Deploy Metadata for Oracle Database 25-7
  - Run the SQL File for Oracle Database 25-8
  - Define Constraints for the Existence of Joins 25-8
  - Create the Query Workload 25-8
  - Create Materialized Views 25-10

26 XML Schema Files for ADF Mapping Customizations

app_segment_rule.xsd XML Schema File 26-1
app_segment_rules_*.xml Example 26-5
mapping_rules.xsd XML Schema File 26-9
mapping_rules_*.xml Example 26-11

27 Administration Tool Keyboard Shortcuts

Menu Keyboard Shortcuts 27-1
Dialog Keyboard Shortcuts 27-2

Physical Diagram and Business Model Diagram Keyboard Shortcuts 27-3

Part III Reference

28 Expression Editor Reference

SQL Operators 28-1
Conditional Expressions 28-3
Functions 28-4
- Aggregate Functions 28-5
- Analytics Functions 28-8
- Date and Time Functions 28-9
- Conversion Functions 28-11
- Display Functions 28-12
- Evaluate Functions 28-13
- Mathematical Functions 28-14
- Running Aggregate Functions 28-16
- Spatial Functions 28-17
- String Functions 28-17
- System Functions 28-21
- Time Series Functions 28-22

Constants 28-24
Types 28-24
Variables 28-24

Preface

The Oracle Analytics Server Foundation Suite is a complete, open, and integrated solution for all enterprise business intelligence needs, including reporting, ad hoc queries, OLAP, dashboards, scorecards, and what-if analysis. The Oracle Analytics Server Foundation Suite includes Oracle Analytics Server.

Oracle Analytics Server is a comprehensive set of enterprise business intelligence tools and infrastructure, including a scalable and efficient query and analysis server, an ad-hoc query and analysis tool, interactive dashboards, proactive intelligence and alerts, and an enterprise reporting engine.

The components of Oracle Analytics Server share a common service-oriented architecture, data access services, analytic and calculation infrastructure, metadata management services, semantic business model, security model and user preferences, and administration tools. Oracle Analytics Server provides scalability and performance with data-source specific optimized request generation, optimized data access, advanced calculation, intelligent caching services, and clustering.

This guide contains information about building an Oracle BI metadata repository and includes topics on setting up and connecting to data sources, building the Physical layer, Business Model and Mapping layer, and Presentation layer, and how to use the multiuser development environment.

Audience

The intended audience is anyone who plans to design and build a metadata repository using the Oracle BI Administration Tool such as a Business Intelligence strategist, metadata provider, or ETL developer.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and

partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Related Resources

For a full list of guides, refer to the Books tab on Oracle Analytics Server Help Center.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Videos and Images

Your company can use skins and styles to customize the look of the application, dashboards, reports, and other objects. It is possible that the videos and images included in the product documentation look different than the skins and styles your company uses.

Even if your skins and styles are different than those shown in the videos and images, the product behavior and techniques shown and demonstrated are the same.

Part I

Get Started with Metadata

This part introduces you to Oracle Analytics Server metadata.

This part includes the following chapters.

Chapters

- Introduction to Building Your Metadata Repository
- Before You Begin

1

Introduction to Building Your Metadata Repository

This chapter explains how to plan and design your metadata repository, including how to plan your business model, how to work with the physical content for your business model, and general repository design guidelines.
To effectively plan and build your metadata repository, you need to have experience with SQL queries and be familiar with reporting and analysis. You should also have experience with industry-standard data warehouse modeling practices, and be familiar with general relational entity-relationship modeling.

This chapter contains the following topics:

- About Oracle BI Server Architecture
- About Layers in the Oracle BI Repository
- Identify the Data Source Content for the Physical Layer
- Guidelines for Designing a Repository

About Oracle BI Server Architecture

The Oracle BI Server component processes user requests and queries in underlying data sources.

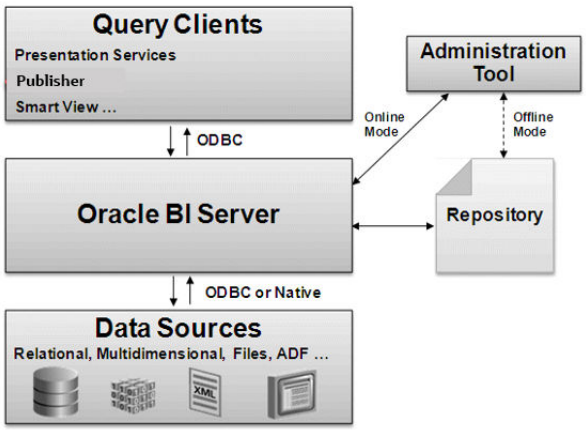
The Oracle BI Server maintains the logical data model and provides client access to the model using ODBC connectivity or native APIs, such as OCI for the Oracle Database.

The Oracle BI Server uses the metadata in the Oracle BI repository to perform the following two tasks:

- Interpret Logical SQL queries and write corresponding physical queries against the appropriate data sources.
- Transform and combine the physical result sets and perform final calculations.

The Administration Tool client is a Windows application that you can use to create and edit your Oracle BI repository. The Administration Tool can connect directly to the repository in offline mode, or it can connect to the repository through the Oracle BI Server. Some options are only available in online mode. See Use Online and Offline Repository Modes.

The image shows how the Oracle BI Server interacts with query clients, data sources, the Oracle BI repository, and the Administration Tool.



The example shows how the Oracle BI Server interprets and converts Logical SQL queries.

Logical Requests Are Transformed Into Complex Physical Queries

Assume the Oracle BI Server receives the following simple client request:

```
SELECT
"D0 Time"."T02 Per Name Month" saw_0,
"D4 Product"."P01 Product" saw_1,
"F2 Units"."2-01 Billed Qty (Sum All)" saw_2
FROM "Sample Sales"
ORDER BY saw_0, saw_1
```

The Oracle BI Server can then convert the Logical SQL query into a sophisticated physical query, as follows:

```
WITH SAWITH0 AS (
select T986.Per_Name_Month as c1, T879.Prod_Dsc as c2,
   sum(T835.Units) as c3, T879.Prod_Key as c4
from
   Product T879 /* A05 Product */ ,
   Time_Mth T986 /* A08 Time Mth */ ,
   FactsRev T835 /* A11 Revenue (Billed Time Join) */
where ( T835.Prod_Key = T879.Prod_Key and T835.Bill_Mth = T986.Row_Wid)
group by T879.Prod_Dsc, T879.Prod_Key, T986.Per_Name_Month
)
select SAWITH0.c1 as c1, SAWITH0.c2 as c2, SAWITH0.c3 as c3
from SAWITH0
order by c1, c2
```

About Layers in the Oracle BI Repository

Layers in the Oracle BI Repository define the objects and their relationships.

An Oracle BI Repository has the following layers:

- Physical layer

  This layer defines the objects and relationships that the Oracle BI Server needs to write native queries against each physical data source. You create this layer by importing tables, cubes, and flat files from your data sources.

Separating the logical behavior of the application from the physical model provides the ability to federate multiple physical sources to the same logical object, enabling aggregate navigation and partitioning, as well as, dimension conformance and isolation from changes in the physical sources. This separation also enables the creation of portable Oracle BI Applications.

- Business Model and Mapping layer

  This layer defines the business or logical model of the data and specifies the mapping between the business model and the physical schemas. This layer determines the analytic behavior seen by users, and defines the superset of objects and relationships available to users. Business Model and Mapping layer hides the complexity of the source data models.

  Each column in the business model maps to one or more columns in the Physical layer. At run time, the Oracle BI Server evaluates Logical SQL requests against the business model, and then uses the mappings to determine the best set of physical tables, files, and cubes for generating the necessary physical queries. The mappings often contain calculations and transformations, and might combine multiple physical tables.

- Presentation layer

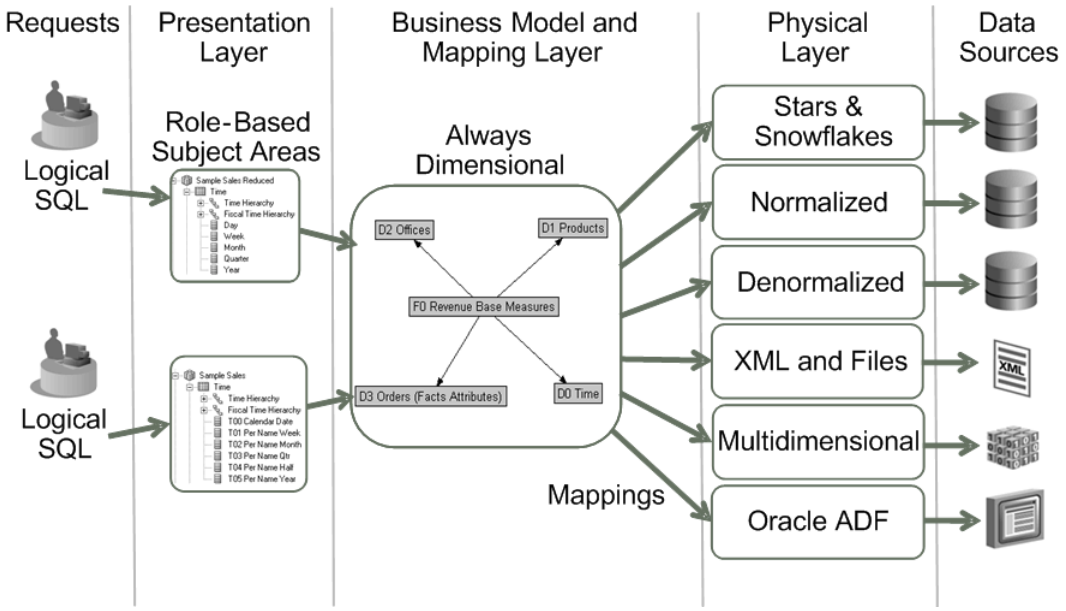
  This layer provides a way to present customized, secure, role based views of a business model to users. It adds a level of abstraction over the Business Model and Mapping layer and provides the view of the data seen by users building requests in Presentation Services and other clients.

  You can create multiple subject areas in the Presentation layer that map to a single business model, effectively breaking up the business model into manageable pieces.

Before you build repository layers in the Administration Tool, create a high-level design of the Business Model and Mapping layer based on the analytic requirements of your users. After you've a conceptual design to work toward, you can then build your metadata objects.

You should create the Physical layer objects first, next create the Business Model and Mapping layer objects, and then create Presentation layer objects. You can work on each layer at any stage. After you complete all three layers, you can set up security when you're ready to begin testing the repository.

The figure shows how a Logical SQL query traverses the layers of an Oracle BI Repository.



A single Oracle BI Repository can contain two or more independent semantic models, rather than a single, integrated, enterprise-wide model. A semantic model consists of one business model, its related objects in the Presentation and Physical layers, and additional related objects like variables, initialization blocks, and application roles. A semantic model is also known as a Common Enterprise Information Model (CEIM).

See About Multiuser Development Styles for a visual representation of multiple semantic models.

Analyze Your Business Model Requirements

You must thoroughly analyze your business data to understand and plan your business model requirements.

Planning your business model is the first step in developing a usable data model for decision support. After planning, you can begin to create your repository.

You must understand the requirements of the business model before you can determine the components of the Physical layer.

In a decision support environment, the objective of data modeling is to design a model that presents business information in a manner that parallels business analysts' understanding of the business structure. A successful model allows the query process to become a natural process by enabling analysts to structure queries in the same intuitive fashion as they would ask business questions. This model must be one that business analysts inherently understand and that answers meaningful questions correctly.

Unlike visual SQL tools such as Publisher, the business model defines the analytic behavior of your BI application. In contrast, the Physical layer only provides the components used to assemble a physical query mapped from business model logic.

This requires breaking down your business into several components to answer the following questions:

- What kinds of business questions are analysts trying to answer?
- What are the measures required to understand business performance?
- What are all the dimensions under which the business operates? Or, in other words, what are the dimensions used to break down the measurements and provide headers for the reports?
- Are there hierarchical elements in each dimension, and what types of relationships define each hierarchy?

After you've answered these questions, you can identify and define the elements of your business model.

Identify the Content of the Business Model

To determine what content to include in your business model, you must first identify the logical columns on which users need to query.

Then, you must identify each column's role as either a measure column or a dimensional attribute. Finally, you need to arrange the logical columns in a dimensional model based on the relevant roles, relationships between columns, and logic.

Businesses are analyzed by relevant dimensional criteria, and the business model is developed from these relevant dimensions. These dimensional models form the basis of the valid business models to use with the Oracle BI Server.

Although not all dimensional models are built around a star schema, it's a best practice to use a simple star schema in the business model layer. In other words, the dimensional model should represent some measurable facts that are viewed in terms of various dimensional attributes.

After you analyze your business model requirements, you need to identify the specific logical tables and hierarchies that you need to include in your business model.

This section contains the following topics:

- Identify Logical Fact Tables
- Identify Logical Dimension Tables
- Identify Dimensions
- Identify Lookup Tables

Identify Logical Fact Tables

Logical fact tables in the Business Model and Mapping layer contain measures that have aggregations built into their definitions.

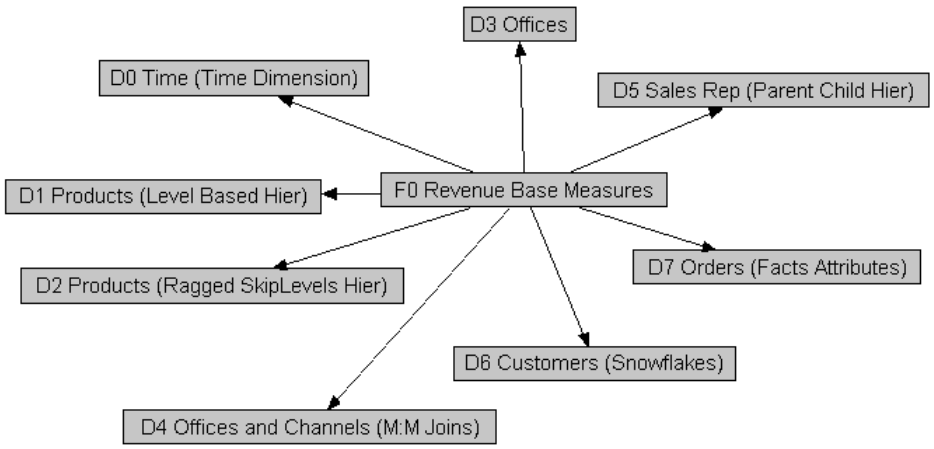
Logical fact tables are different from physical fact tables in relational models. Physical tables in rational models define facts at the lowest possible grain. Logical fact table can contain measures of different grains,

You must define measures aggregated from facts in a logical fact table. Measures are calculated data such as dollar value or quantity sold. You can specify measures in terms of dimensions. For example, you might want to determine the sum of dollars for a given product in a given market over a given time period.

Each measure has its own aggregation rule such as `SUM`, `AVG`, `MIN`, or `MAX`. A business might want to compare values of a measure and need a calculation to express the comparison. You can specify aggregation rules to specific dimensions. You can define complex, dimension-specific aggregation rules in the Oracle BI Server.

You don't explicitly label tables in the Business Model and Mapping layer as fact tables or dimension tables. The Oracle BI Server treats tables at the *one* end of a join as dimension tables, and tables at the *many* end of a join as fact tables.

The image shows the many-to-one joins to a fact table in a Business Model diagram. In the Business Model diagram, all joins have an arrow, indicating the *one* side, pointing away from the Fact-Pipeline table; no joins are pointing to it. For an example of this in a business model, open a repository in the Administration Tool, right-click a fact table, select **Business Model Diagram**, and then select **Whole Diagram**.



Identify Logical Dimension Tables

Dimension tables contain attributes that describe business entities such as Customer Name, Region, Address and Country.

A business uses facts to measure performance using established dimensions such as by time, product, and market. Every dimension has a set of descriptive attributes. Dimension tables contain primary keys that identify each member.

Dimension table attributes provide context to numeric data, for example, by providing the ability to categorize Service Requests. Attributes stored in a service requests dimension table could include Service Request Owner, Area, Account, and Priority.

Dimensions in the business model are conformed dimensions, that is those dimensions that are consistent across. If a specific data source has five different instances of a specific Customer table, the business model should only have one Customer table. To achieve conformance, all five physical source instances of Customer are mapped to a single Customer logical table, with transformations in the logical table source as necessary. Conformed dimensions hide the complexity of the Physical layer from users, and enable combining data from multiple fact sources at different grains. Conformed dimensions enable combining multiple data sources.

The business model uses business keys for a dimension and level keys instead of generated surrogate keys. For example, you would use *Customer Name* with values

like *Oracle* instead of *Customer Key* with values like *1076823*. Using business keys in the business model ensures that all sources for that dimension can conform to the same logical dimension table with the same logical key and level key.

Generated surrogate keys can exist in the Physical layer where the keys are useful for their query performance advantages on joins. The Business Model and Mapping layer doesn't have surrogate key columns.

Identify Dimensions

Dimensions are categories of attributes by which the business is defined.

Common dimensions are time periods, products, markets, customers, suppliers, promotion conditions, raw materials, manufacturing plants, transportation methods, media types, and time of day. Within a given dimension, there are many attributes. For example, the time period dimension can contain the attributes day, week, month, quarter, and year. Exactly what attributes a dimension contains depends on the way the business is analyzed.

Dimensions contain hierarchies that are sets of top-down relationships between members within a dimension. There are two types of hierarchies:

- level-based hierarchies (structure hierarchies)
- parent-child hierarchies (value hierarchies)

In level-based hierarchies, members of the same type occur only at a single level, while members in parent-child hierarchies all have the same type. Oracle Analytics Server supports a time dimension level-based hierarchy that provides functionality for modeling time series data. In level-based hierarchies, levels roll up from a lower level to higher level, for example, months can roll up into a year. These roll ups occur over the hierarchy elements and span natural business relationships.

In parent-child hierarchies, the business relationships occur between different members of the same real-world type such as the manager-employee relationship in an organizational hierarchy tree. Parent-child hierarchies don't have explicitly named levels. There isn't a limit to the number of implicit levels in a parent-child hierarchy.

To define your hierarchies, you define the **contains** relationships in your business to drive roll up aggregations in all calculations, as well as drill-down navigation in reports and dashboards. For example, if month rolls up into year and an aggregate table exists at the month level, you can use the table to answer questions at the year level by adding up all of the month-level data for a year.

You must use the right type of hierarchy for your needs. To determine the appropriate type to use, consider the following:

- Are all the members of the same type such as employee, assembly, or account, or are they of different types that naturally fall into levels such as year-quarter-month, continent-country-state/province, or brand-line-product?
- Do members have the same set of attributes? For example, in a parent-child hierarchy like *Employees*, all members might have a Hire Date attribute. In a level-based hierarchy like Time, the Day type might have a Holiday attribute, while the Month type doesn't have the Holiday attribute.
- Are the levels fixed at design time (year-quarter-month), or can runtime business transactions add or subtract levels? For example, if you can add a level when the current lowest-level employee hires a subordinate, who then is the new lowest level.

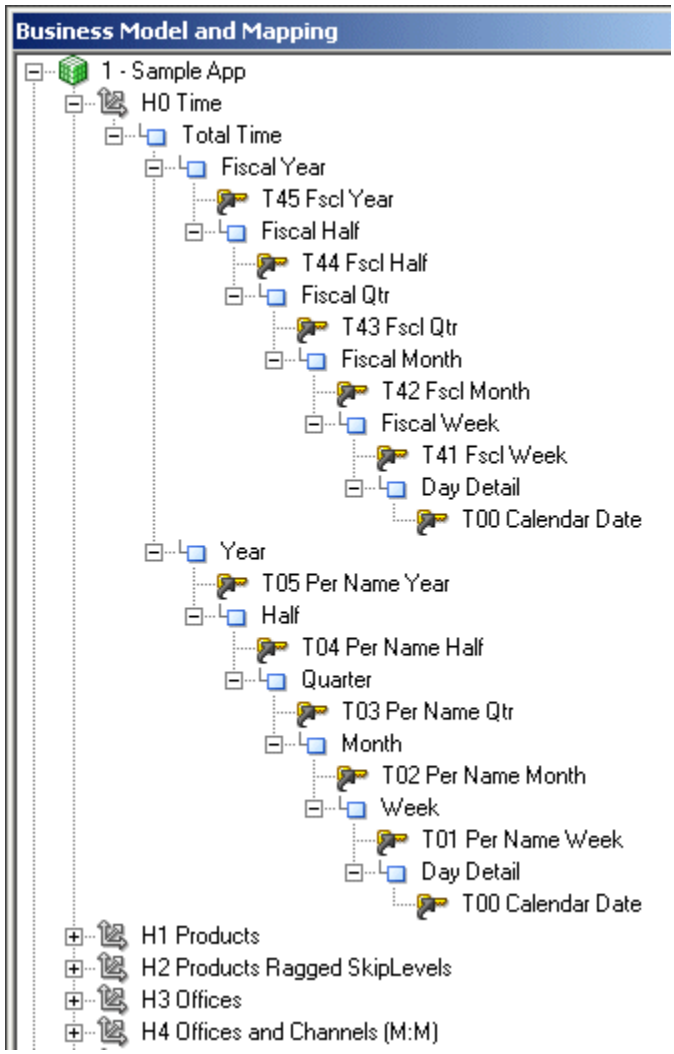
- Are there constraints in your primary data source that require a certain hierarchy type? If your primary data source is modeled in one way, you might need to use the same hierarchy type in your business model, regardless of other factors.

About Dimensions with Multiple Hierarchies

Dimensions can contain multiple hierarchies. Dimensions with multiple hierarchies must always end with the same leaf table.

For example, time dimensions often have one hierarchy for the calendar year, and another hierarchy for the fiscal year.

The image shows a dimension with multiple hierarchies in the Business Model and Mapping layer of the Oracle BI Administration Tool.



Identify Lookup Tables

When you need to display translated field information from multilingual schemas, you create a logical lookup table that corresponds to a lookup table in the Physical layer.

A lookup table stores multilingual data corresponding to rows in the base tables. Before using a specific logical lookup table, you must designate the table as a lookup

table in the General tab of the Logical Table dialog. See Localizing Metadata Names in the Repository in *Administering Oracle Analytics Server*.

You can use lookup tables to display one set of values to users, while using a different, corresponding set of values in the physical query. You can use the lookup table to provide human-readable values that are looked up in a different data source.

Identify the Data Source Content for the Physical Layer

After you've determined the requirements for your business model, you can look at what data source content you need in the Physical layer.

Unlike the Business Model and Mapping layer that's always dimensional, each physical model mirrors the shape of the source, for example, normalized, and cube.

This section contains the following topics:

- About Types of Physical Schemas in Relational Data Sources
- About Cubes in Multidimensional Data Sources
- Identify the Data Source Table Structure

About Types of Physical Schemas in Relational Data Sources

You can successfully model any physical schema in the Oracle BI repository, regardless of its type, because you can break down the model of any physical source into overlapping subsets that are dimensional.

There are four types of physical schemas (models):

- Star Schemas

  A star schema is a set of dimensional schemas (stars) that each have a single fact table with foreign key join relationships to several dimension tables. When you map a star to the business model, you first map the physical fact columns to one or more logical fact tables. Then, for each physical dimension table that joins to the physical fact table for that star, you map the physical dimension columns to the appropriate conformed logical dimension tables.

- Snowflake Schemas

  A snowflake schema is similar to a star schema, except that each dimension is made up of multiple tables joined together. Like star schemas, you first map the physical fact columns to one or more logical tables. Then, for each dimension, you map the snowflake physical dimension tables to a single logical table. You can achieve this by either having multiple logical table sources, or by using a single logical table source with joins.

- Normalized Schemas

  Normalized schemas distribute data entities into multiple tables to minimize data storage redundancy and optimize data updates. Before mapping a normalized schema to the business model, you need to understand how the distributed structure is understood in terms of facts and dimensions.

  After analyzing the structure, you pick a table that has fact columns and then map the physical fact columns to one or more logical fact tables. Then, for each dimension associated with that set of physical fact columns, you map the distributed physical tables containing dimensional columns to a single logical table. Like with snowflake schemas, you can achieve this by having multiple logical table sources, or by using a single logical

table source with joins. Mapping normalized schemas is an iterative process because you first map a certain set of facts, then the associated dimensions, and then you move on to the next set of facts.

When a single physical table has both fact and dimension columns, you may need to create a physical alias table to handle the multiple roles played by that table.

- Fully Denormalized Schemas

  This type of dimensional schema combines the facts and dimensions as columns in one table (or flat file), and is mapped differently than other types of schemas. When you map a fully denormalized schema to the star-shaped business model, you map the physical fact columns from the single physical fact table to multiple logical fact tables in the business model. Then, you map the physical dimension columns to the appropriate conformed logical dimension tables.

About Cubes in Multidimensional Data Sources

Cubes are made up of measures and are organized by dimensions.

Cube are already dimensional, each cube maps to the logical fact and dimension tables in the business model.

- Measures in multidimensional cubes and relational fact columns map to logical measures in the Business Model and Mapping layer. Measures in multidimensional cubes include calculations and aggregations. Relational fact columns require applying the calculations and aggregations in the business model. The Oracle BI Server takes advantage of the pre-aggregated data and powerful calculations in the cube.
- Multidimensional physical objects and relational physical objects map to logical dimensions in the Business Model and Mapping layer. Dimensional and hierarchical semantics are built into multidimensional data sources. The Oracle BI Server takes advantage of the hierarchy and dimensional support in the cube during import and at query time.

Identify the Data Source Table Structure

The Oracle BI Administration Tool provides an interface to map logical tables to the underlying physical tables in your data sources.

Before you can map the tables, you need to identify the contents of the physical data sources as it relates to your business model. To do this correctly, you need to identify the following contents of the physical data source:

- Identify the contents of each table
- Identify the detail level for each table
- Identify the table definition for each aggregate table. This lets you set up aggregate navigation. The following detail is required by the Oracle BI Server:
  - The columns by which the table is grouped (the aggregation level)
  - The type of aggregation (`SUM`, `AVG`, `MIN`, `MAX`, or `COUNT`)

  To set up aggregate navigation in a repository, see Manage Logical Table Sources (Mappings).
- Identify the contents of each column

- Identify how each measure is calculated
- Identify the joins defined in the database

To acquire this information about the data, you could refer to any available documentation that describes the data elements created when the data source was implemented, or you might need to spend some time with the DBA for each data source to get this information. To fully understand all the data elements, you might also need to ask people in the organization who are users of the data, owners of the data, or the application developers of applications that create the data.

Guidelines to Design a Repository

After analyzing your business model needs and identifying the database content that your business requires, you can complete your repository design.

This section contains some design best practices that can help you implement a more efficient repository.

You shouldn't make performance tuning changes until importing and testing your databases. Run performance tuning tasks during the final steps in completing the setup of your repository. See Complete Oracle BI Repository Setup.

This section contains the following topics:

- Design Strategies for Structuring the Repository
- Design Tips for the Physical Layer
- Design Tips for the Business Model and Mapping Layer
- Design Tips for the Presentation Layer

Design Strategies for Structuring the Repository

Use these recommended design strategies for structuring your Oracle BI repository.

- If you work in online mode, save backups of the online repository before and after every completed unit of work. If needed, use **Copy As** on the **File** menu to make an offline copy containing the changes.
- Use the Physical Diagrams in the Administration Tool to verify sources and joins.
- Decide whether you want to set up row-level security controls in the database, or in the repository. This decision determines whether you share connection pools and cache, and may limit the number of separate source databases you want to include in your deployment. See Applying Data Access Security to Repository Objects.

Design Tips for the Physical Layer

The Physical layer contains information about the physical data sources.

The most common way to create the schema in the Physical layer is by importing metadata from databases and other data sources. If you import metadata, many of the properties are configured automatically based on the information gathered during the import process. You can also define other attributes of the physical data source, such as join relationships, that might not exist in the data source metadata.

The Physical layer can contain data sources of many different types, including multidimensional, relational, and XML sources.

For each data source, there is at least one corresponding connection pool. The connection pool contains data source name (DSN) information used to connect to a data source, the number of connections allowed, timeout information, and other connectivity-related administrative details. See About Connection Pools.

The following is a list of tips to use when designing the Physical layer:

- You should use table aliases in the Physical layer to eliminate extraneous joins, including the following:
  - Eliminate all physical joins that cross dimensions (inter-dimensional circular joins) by using aliases.
  - Eliminate all circular joins (intra-dimensional circular joins) in a logical table source in the Physical Model by creating physical table aliases.

    A circular join involves using different joins from the same table to get results, for example, you've a Customer table that's used to look up ship-to addresses, and you use a different join to the Customer table to look up bill-to addresses. You can avoid the circular joins by creating an alias table in the Physical layer so that only one table instance is used for each purpose, with separate joins.

  If you don't eliminate circular joins, you could get erroneous report results. In addition, query performance is negatively impacted by circular joins.
- You should use alias tables to create separate physical joins when you need the join to perform as an inner join in one logical table source, and as an outer join in another logical table source.
- You might import some tables into the Physical layer that you might not use right away, but that you don't want to delete. To identify tables that you do want to use right away in the Business Model and Mapping layer, you can assign aliases to physical tables before mapping them to the business model layer.

  To display the original name of a table that has an assigned alias, select **Tools**, select **Options**, select **General**, and then select **Display original name for alias in diagrams**.
- Use an opaque view only if there is no other solution to your modeling problem. You should create a physical table or a materialized view. Opaque views prevent the Oracle BI Server from generating optimized SQL because opaque views contain fixed SQL statements that are sent to the underlying data source.

Design Tips for the Business Model and Mapping Layer

The Business Model and Mapping layer organizes information by business model. In this layer, each business model is effectively a separate application.

The logical schema defined in each business model must contain at least two logical tables. You must define relationships between all the logical tables. See About Layers in the Oracle BI Repository and Work with Logical Tables, Joins, and Columns.

When designing the Business Model and Mapping layer:

- Create the business model with one-to-many logical joins between logical dimension tables and the fact tables wherever possible. The business model should resemble a simple star schema in which each fact table is joined directly to its dimensions.

- Join every logical fact table to at least one logical dimension table. When the source is a fully denormalized table or flat file, you must map its physical fact columns to one or more logical fact tables, and its physical dimension columns to logical dimension tables.
- Associate every logical dimension table with a dimensional hierarchy. This rule holds true even if the hierarchy has only one level such as a scenario dimension (actual, forecast, or plan).
- Map all appropriate fact sources map to the appropriate level in the hierarchy using aggregation content when creating level-based measures. You set up aggregation content in the Levels tab of the Logical Column dialog for the measure.

  Set up aggregation content in the Levels tab of the Logical Column dialog for level-based measures. For measures that aren't level-based, leave the Logical Level field blank.
- Create aggregate sources as separate logical table sources. For fact aggregates, use the Content tab of the Logical Table Source dialog to assign the correct logical level to each dimension.
- Create a unique level key for each dimension level in a hierarchy. Each logical dimension table must have a unique primary key. The key is also used as the level key for the lowest hierarchy level.
- Ensure that each logical level of a dimension hierarchy contains the correct value in the field named **Number of elements at this level** to prevent problems with aggregate navigation. Fact sources are selected on a combination of the fields selected as well as the levels in the dimensions to which they map. By adjusting these values, you can alter the fact source selected by the Oracle BI Server. See Create Logical Levels in a Dimension.

Logical Fact Tables

- Logical fact tables can contain measures of different grains. Don't use the grain as a reason to split up logical fact tables.
- Logical fact tables shouldn't contain any keys, except when you need to send Logical SQL queries against the Oracle BI Server from a client that requires keys. In this case, you need to expose those keys in both the logical fact tables, and in the Presentation layer.
- All columns in logical fact tables are aggregated measures, except for keys required by external clients, or dummy columns used as a divider. Other non-aggregated columns should exist in a logical dimension table.
- You can use multiple logical fact tables in a single business model. For Logical SQL queries, the multiple logical fact tables behave as if they're one table. Reasons to have multiple logical fact tables include:
  - To assign projects, see Set Up Projects.
  - To automatically create small subject areas in the Presentation layer, see Automatically Create Subject Areas Based on Logical Stars and Snowflakes.
  - For organization and simplicity of understanding.

Renaming columns in the Business Model and Mapping layer automatically creates aliases (synonyms) for Presentation layer columns that have the property **Use Logical Column Name** selected. This occurs because Presentation layer columns with this option selected are automatically renamed so that the logical column and presentation column names are in sync. Renaming Presentation layer columns directly when **Use Logical Column Name** isn't selected creates an alias.

Calculations

- You can define calculations in the following ways:
  - Before the aggregation, in the logical table source. For example:

    ```
    sum(col_A *( col_B))
    ```

  - After the aggregation, in a logical column derived from two other logical columns. For example:

    ```
    sum(col A) * sum(col B)
    ```

  You can also define post-aggregation calculations in Answers or in Logical SQL queries.

Model Outer Joins

Use these guidelines on how to model outer joins.

- Due to the nature of outer joins, queries that use them are usually slower. Because of this, define outer joins only when necessary. Where possible, use ETL techniques to eliminate the need for outer joins in the reporting SQL.
- Outer joins are always defined in the Business Model and Mapping layer. Physical layer joins don't specify inner or outer.
- You can define outer joins by using logical table joins, or in logical table sources. Which type of outer join you use is determined by whether the physical join maps to a business model join, or to a logical table source join.
- If you must define an outer join, try to create two separate dimensions, one that uses the outer join and one that doesn't. Make sure to name the dimension with the outer join in a way that clearly identifies it, so that client users can use it as little as possible.
- Avoid using more than one outer join. Instead, to achieve the same effect as a logical outer join, Oracle recommends that the logical join be an inner join and that the analysis designer at design time selects the **Include Null Value** option in the corresponding analysis.

Design Tips for the Presentation Layer

You set up the user view of a business model in the Presentation layer.

The names of folders and columns in the Presentation layer can appear in localized language translations. The Presentation layer is the appropriate layer in which to set user permissions. See Create and Maintain the Presentation Layer.

In this layer, you can do the following:

- You can show fewer columns than exist in the Business Model and Mapping layer. For example, you can exclude the key columns because they have no business meaning.
- You can organize columns using a different structure from the table structure in the Business Model and Mapping layer.
- You can display column names that are different from the column names in the Business Model and Mapping layer.

- You can set permissions to grant or deny users access to individual subject areas, tables, and columns.
- You can export logical keys to ODBC-based query and reporting tools.
- You can create multiple subject areas for a single business model.
- You can create a list of aliases (synonyms) for presentation objects that are used in Logical SQL queries. You can change presentation column names without breaking existing reports.

The following is a list of tips to use when designing the Presentation layer:

- Because there isn't an automatic way to synchronize all changes between the Business Model and Mapping layer and the Presentation layer, it's best to wait until the Business Model and Mapping layer is relatively stable before adding customizations in the Presentation layer.
- There are many ways to create subject areas, such as dragging and dropping the entire business model, dragging and dropping incremental pieces of the model, or automatically creating subject areas based on logical stars or snowflakes. See About Creating Subject Areas. Dragging and dropping incrementally works well if certain parts of your business model are still changing.
- It's a best practice to rename objects in the Business Model and Mapping layer rather than the Presentation layer, for better maintainability. Giving user-friendly names to logical objects rather than presentation objects ensures that you can use the names in multiple subject areas. Also, it ensures that the names persist even when you need to delete and re-create subject areas to incorporate changes to your business model.
- Members in a presentation hierarchy aren't visible in the Presentation layer. You can see hierarchy members in Answers .
- You can use the Administration Tool to update Presentation layer metadata to give the appearance of nested folders in Answers. See Nest Folders in Answers and BI Composer.
- When setting up data access security for a large number of objects, consider setting object permissions by role rather than setting permissions for individual columns. See Apply Data Access Security to Repository Objects.
- When setting permissions on presentation objects, you can change the default permission by setting the `NQSConfig.INI` file. See NQSConfig.INI File Configuration Settings in *Administering Oracle Analytics Server*

Topics of Interest in Other Guides

Some topics of interest to metadata repository builders are covered in other guides.

Topic	Where to Go for More Information
Starting and stopping Oracle Analytics Server processes	*Administering Oracle Analytics Server*
Using the Oracle BI ServerXML API to work with your repository	*XML Schema Reference for Oracle Business Intelligence Enterprise Edition*
Using the Oracle BI Server web services	*Integrator's Guide for Oracle Business Intelligence Enterprise Edition*

Topic	Where to Go for More Information
Setting up and managing query caching	*Administering Oracle Analytics Server*
Managing configuration settings that affect repository development in Fusion Middleware Control and `NQSConfig.INI`	*Administering Oracle Analytics Server*
Managing users, groups, and application roles	*Managing Security for Oracle Analytics Server*
Moving from test to production environments	*Administering Oracle Fusion Middleware*
Managing Oracle Analytics Server Application Module metadata in BI Archive (BAR)	*Administering Oracle Analytics Server*
Setting up DSNs for the Oracle BI Server	*Integrator's Guide for Oracle Business Intelligence Enterprise Edition*
Localizing Oracle Analytics Server deployments	*Administering Oracle Analytics Server*
Information about the SA System subject area	*Scheduling Jobs Guide for Oracle Business Intelligence Enterprise Edition*
Managing logging	*Administering Oracle Analytics Server*
Managing usage tracking	*Administering Oracle Analytics Server*
General information about managing Oracle WebLogic Server	*Administering Oracle Fusion Middleware*

2

Before You Begin

These topics provide an overview of the Oracle BI Administration tool, and explains other concepts that you must know before building a metadata repository.
This chapter contains the following topics:

- Open the Administration Tool
- Set Administration Tool Options
- Edit, Delete, and Reorder Objects in the Repository
- About Name Requirements for Repository Objects
- Change Icons for Repository Objects
- Sort Objects in the Administration Tool
- About the Oracle BI Server Command-Line Utilities
- About Options in NQSConfig.INI
- Download Repository Command
- Use Online and Offline Repository Modes
- Check the Consistency of a Repository or a Business Model

Open the Administration Tool

Learn how to open the Oracle BI Administration Tool.

The Oracle BI Administration Tool requires administrator privileges on the machine on which it's installed. Before installing or running the tool, ensure that you're logged in with administrator privileges.

- Do one of the following:
  - Choose **Start**, expand **Programs**, select **Oracle Business Intelligence**, and then select **BI Administration**.
  - Launch the Administration Tool from the `admintool` utility located in *`ORACLE_HOME`*`/bi/bitools/bin`.
  - Don't use double-click to open a repository file. The resulting Administration Tool window isn't initialized to your Oracle instance, resulting in errors.

Set Administration Tool Options

Use these steps to set preferences and options for the Oracle BI Administration Tool.

1. In the Administration Tool, select **Tools**, then select **Options**.
2. In the Options dialog, on the General tab, select the options to use.

3. On the Repository tab, select **Show tables and dimensions only under display folders** or **Hide level based measure**.
4. On the Sort Objects tab, specify which repository objects appear in the Administration Tool in alphabetical order.
5. On the Source Control tab, create or edit a configuration file to integrate with a source control management system, or change the status of an MDS XML repository.
6. On the Cache Manager tab, select the columns you want to display in the Cache Manager.
7. Select an item to change the order of columns in the Cache Manager, then use the **Up** and **Down** buttons to change its position.
8. On the Multiuser tab, specify the path to the multiuser development directory and the name of the local developer for this Administration Tool.
9. On the More tab, you can set the scrolling speed for Administration Tool dialogs. To set the scrolling speed, position the cursor on the slider.
10. Click **OK** when you're finished setting preferences.

Oracle BI Administration Tool General Options

The table describes some of the Oracle BI Administration Tool options available in the Options dialog on the General tab.

Option	Action When Selected
Display qualified names in diagrams	Displays fully qualified names in the Physical Diagram and Business Model Diagram. For example, selecting this option displays *"B - Sample Fcst Data"..."B02 Market"* rather than *B02 Market* in the Physical Diagram. Selecting this option can help identify objects by including the name of the parent database or business model, but it can also make the diagram harder to read because the fully qualified names are longer. If you choose not to select this option, you can still see fully qualified names by moving the cursor over an object in the diagram, or by selecting an object in the diagram and then viewing the text in the status bar.
Display original names for alias in diagrams	Displays the names of original physical tables rather than the names of alias tables in the Physical diagram. Select this option when you want to identify the original table rather than the alias table name.
Show Wizard introduction page	Displays the Calculation Wizard introduction page. The introduction page also contains an option to suppress its display in the future. Use the Calculation Wizard to create new calculation columns that compare two existing columns and to create metrics in bulk (aggregated), including existing error trapping for NULL and divide by zero logic. See Use the Calculation Wizard.

Option	Action When Selected
Check out objects automatically	Automatically checks out an object when you double-click it. If you don't select this option, you're prompted to check out objects before you can edit them. This option only applies when the Administration Tool is open in online mode. See Edit Repositories in Online Mode.
Show row count in physical view	Displays row counts for physical tables and columns in the Physical layer. Row counts aren't initially displayed until they're updated. Row counts aren't shown for items that are stored procedure calls from the **Table Type** list in the General tab of the Physical Table dialog. Row counts aren't available for XML, XML Server, or multidimensional data sources. When you're working in online mode, you can't update row counts on any new objects until you check them in.
Prompt when moving logical columns	Lets you ignore, specify an existing, or create a new logical table source for a moved column.
Remove unused physical tables after Merge	Runs a utility to clean the repository of unused physical objects. It might make the resulting repository smaller.
Allow import from repository	When selected, the **Import from Repository** option on the **File** menu becomes available. By default, the **Import from Repository** option on the **File** menu is disabled. It's recommended that you create projects in the repository that contain the objects that you want to import, and then use repository merge to bring the projects into your current repository. See Merge Repositories.
Allow logical foreign key join creation	When selected, provides the capability to create logical foreign key joins with the Joins Manager. This option is provided for compatibility with previous releases and is generally not recommended.
Skip Gen 1 levels in Essbase drag and drop actions	When selected, excludes Gen 1 levels when you drag and drop Essbase cubes or dimensions from the Physical layer to the Business Model and Mapping layer. See Work with Essbase Data Sources.
Hide unusable logical table sources in Replace wizard	By default, the Replace Wizard shows all logical table sources, even ones that aren't valid for replacement. When this option is selected, unusable logical table sources are hidden in the Replace Wizard screens. Click **Info** for details on why a logical table source that maps to that column doesn't appear in the list. Selecting this option might result in the Wizard page loading more quickly, especially for large repositories.

Option	Action When Selected
Allow first Connection Pool for Init Blocks	Selecting this option isn't a best practice and might cause performance issues. By default, when you select a connection pool for an initialization block, the first connection pool under the database object in the Physical layer doesn't display as available for selection. This behavior ensures that you can't use the same connection pool for initialization blocks that you use for queries. If the same connection pool is used for initialization blocks and for queries, then queries might be blocked whenever initialization blocks run. Alternatively, initialization blocks used for authentication might be blocked by long-running queries, causing delayed or suspended logins. Select this option to change the default behavior and allow the first connection pool to be selected for initialization blocks. See About Connection Pools for Initialization Blocks.
Show Upgrade ID in Query Repository	Upgrade IDs aren't displayed by default in the Query Repository dialog. When this option is selected, Upgrade IDs are displayed as a column in the Query Repository results. In addition, you can set a filter on Upgrade ID to search for a particular value. This option is useful for MDS XML format repositories in which the Upgrade ID is included in the file name.
Extender For BIAPPS	The availability of this option depends on your configuration.
Show Tenant Info in Online Login	If you're working in a multi-tenant environment, then select this option to show the **Tenant info** field in the Open Online dialog.
Display Translation Key in the presentation tree	Select this option to instead display the translation key values for all presentation objects.
Edit presentation names	By default, the presentation object names are read-only. Select this option to allow the names of presentation objects to be modified.
Drag and drop: Show only hierarchal columns	For Essbase data sources, selecting this option hides presentation columns and shows only hierarchal columns in Answers.

Oracle BI Administration Tool Repository Options

You can set preferences for the repository in the Oracle BI Administration Tool.

Repository tab options include the following:

- **Show tables and dimensions only under display folders**

  You can create display folders to organize objects in the Physical and Business Model and Mapping layers. They have no metadata meaning. After you create a display folder, the selected objects appear in the folder as a shortcut and in the database or business model tree as an object. You can hide the objects so that only the shortcuts appear in the display folder.

  See Set Up Display Folders in the Physical Layer and Set Up Display Folders in the Business Model and Mapping Layer.

- **Hide level based measure**

By default, each level of a dimension hierarchy in the Business Model and Mapping layer shows both dimension columns that are assigned to that level, and level-based measures that have been fixed at that level. Level-based measures are objects that aren't part of the dimension table, but that have been explicitly defined as being at a particular level.

Hiding level-based measures in dimension hierarchies can reduce clutter. The measures are still visible in the logical fact tables.

See Level-Based Measure Calculations.

- **System logging level**

  This option determines the default query logging level for the internal BISystem user. The BISystem user owns the Oracle BI Server system processes and isn't exposed in any user interface.

  A query logging level of 0 (the default) means no logging. Set this logging level to 2 to enable query logging for internal system processes like event polling and initialization blocks.

  See Managing the Query Log in *Administering Oracle Analytics Server*.

- **LDAP**

  If you're using any alternative LDAP servers, the Oracle BI Server maintains an authentication cache in memory for user identifiers and properties to improve performance when using LDAP to authenticate large numbers of users. Disabling the authentication cache can slow performance when authenticating hundreds of session. The authentication cache isn't used for Oracle WebLogic Server's embedded directory server.

  Properties for the authentication cache include:

  – **Cache refresh interval**

    The interval at which the authentication cache entry for a logged on user is refreshed.

  – **Number of Cache Entries option (authentication cache)** and **Number of Cache Entries**

    The maximum number of entries in the authentication cache, pre-allocated when the Oracle BI Server starts. If the number of users exceeds this limit, cache entries are replaced using the LRU algorithm. If this value is 0, then the authentication cache is disabled.

  You need to specify some additional LDAP properties when you're using a secure connection to your LDAP server. In other words, provide the following information when you've selected **SSL** on the Advanced tab of the LDAP Server dialog:

  – **Wallet directory**

    The location of the Oracle wallet that holds the client certificate and Certificate Authority (CA) certificate.

  – **Password** and **Confirm password**

    The password for the Oracle wallet.

  The authentication cache properties and Oracle wallet properties are shared for all defined LDAP server objects.

See Set Administration Tool Options.

Edit, Delete, and Reorder Objects in the Repository

Learn how to edit objects in the repository.

This section provides information about editing, deleting, and reordering objects.

- To edit objects, double-click an object, or right-click an object and select **Properties**. Then, complete the fields in the dialog that's displayed. In some dialogs, you can click **Edit** to open the appropriate dialog.
- To delete objects, select one or more objects and click **Delete**, or press the delete key. You can also right-click an object and select **Delete**.
- To reorder objects, drag and drop an object to a new location. Note the following:
  - Reordering is only available for certain objects and in certain dialogs.
  - In some dialogs, you can use an up or down arrow to move objects to a new location.
  - In the Oracle BI Administration Tool main window, you can drag and drop an object onto its parent to duplicate the object. For top-level objects like business models and subject areas, drag and drop the object onto white space to duplicate it.

About Name Requirements for Repository Objects

You can learn about the repository object naming requirements.

Repository object names can include multi-byte characters. All repository object names must follow these requirements:

- Names that are 128 characters or less
- Names that don’t contain leading or trailing spaces
- Names that don’t contain characters such as single quotes, hash marks, question marks, or asterisks

Change Icons for Repository Objects

In the Oracle BI Administration Tool, you can change the icon that represents a particular object in the repository.

Changing the icon for a particular object doesn't have any functional effect, and isn't visible in analytics editor or other clients. Changing the icon is a useful way to visually distinguish objects for the convenience of repository developers.

For example, you can:

- Use a special icon for objects that are in the Business Model and Mapping layer, but not the Presentation layer, for easier maintenance of the repository.
- Mark objects that are logical calculations with a separate icon.
- Choose an icon to visually distinguish tables in the Presentation layer that appear as nested folders in Answers.

- Use an icon to denote objects in a logical table that pertain to a specific functional area, or that are sourced from a particular logical table source.

You can only change the icon for individual objects. You can't globally change the icon for all objects of a particular type.

1. In the Administration Tool, right-click an object in the Physical, Business Model and Mapping, or Presentation layer, for example, a particular logical table.
2. Select **Set Icon**.
3. In the Select Icon dialog, select the icon you want to use for that object and click **OK**.

Sort Objects in the Administration Tool

Many dialogs in the Administration Tool show lists of objects, such as a list of physical columns in the Physical Table dialog, a list of logical levels for **Preferred Drill Path** in the Logical Level dialog, and a list of presentation hierarchies in the Presentation Table dialog.

You can click the header to sort the objects in ascending or descending order. An up arrow or down arrow icon is displayed next to the header name, indicating how the list has been sorted.

Each list also has a default order that's persisted from session to session. The default order appears when you view a list in a dialog for the first time each session. The default order is displayed when there is no ascending or descending arrow icon in the header. Click the header three times to toggle between ascending, descending, and default order. In some cases, the default order is the ascending or descending order.

Some dialogs provide the capability to move items up or down in a list. In these dialogs, if you click **Up** or **Down** while the list is sorted in ascending or descending order, the selected item moves, and the resulting order becomes the new default order. Clicking the header eliminates any manually determined order.

About the Oracle BI Server Command-Line Utilities

You can use command-line utilities with the Oracle BI Server to make programmatic changes to your repository file, run sample queries, delete unwanted repository objects, and perform other tasks.

The table describes the Oracle BI Server command-line utilities.

> **Important:**
>
> When using tools such as `nqcmd`, `biserverxmlcli`, and `comparerpd`, you must edit the input to match the format expected by SQL, for example, don't include a single quote in your XML content.

Utility Name	Description	Where to Go for More Information
`biserverextender`	Use to import flex object changes from ADF data sources and map them to the Business Model and Mapping layer and Presentation layer.	Automatically Map Flex Object Changes Using the biserverextender Utility

Utility Name	Description	Where to Go for More Information
`biservergentypexml`	Used to compare data types of logical columns between a particular repository and a generated list of logical column types to ensure that the types match as expected.	Generate a List of Logical Column Types Compare Logical Column Types
XML utilities (`biserverxmlgen`, `biserverxmlexec`, `biserverxmlcli`)	Use to leverage the Oracle BI Server XML API for metadata migration, programmatic metadata generation and manipulation, metadata patching, and other functions. The XML utilities include: • `biserverxmlgen`: generates XML from an existing Oracle BI repository file. Also includes an option to generate repositories in MDS XML format. • `biserverxmlexec`: runs the XML in offline mode to create or modify a repository file. Also includes an option to run XML in MDS XML format. • `biserverxmlcli`: runs the XML against the Oracle BI Server.	*XML Schema Reference for Oracle Business Intelligence Enterprise Edition*
`comparerpd`	Used to compare two repositories and generate a CSV diff file, an XML patch file, or an MDS XML diff.	Compare Repositories Using comparerpd
`deleteapproles`	Use to upload a JSON file containing a list of application roles to delete from a specific server instance.	Delete Application Role Command
`deleteusers`	Used to upload a JSON file containing a list of users to delete from a specific server instance.	Delete Users Command
`downloadrpd`	Use to download the repository to work offline on diagnostics and development tasks.	Download Repository Command
`equalizerpds`	Use to equalize objects in two repositories that have the same name, but different Upgrade IDs. Running this utility before merging repositories prevents unintended renaming during the merge.	Equalize Objects
`externalizestrings`	Use to localize the names of Presentation layer subject areas, tables, hierarchies, columns and their descriptions.	Configure Oracle Analytics Server to Use DataDirect
`extractprojects`	Use to extract projects from a given repository.	Use the extractprojects Utility
`listConnectionPool`	Use to create a list of connection pools in JSON format for a specific server instance.	List Connection Pool Command
`listrpdvariable`	Use to create a list of repository variables in JSON format for a specific service instance.	List Repository Variables Command
`mhlconverter`	Use to convert MUD history files from .mhl format to .xml format so that you can check in the files under source control.	Check In New Versions of the MUD Master and MUD Log File to Source Control

Utility Name	Description	Where to Go for More Information
nqaggradvisor	Use to invoke the Oracle BI Summary Advisor to generate an aggregate specification script that's run to create aggregates. This utility is only available for Oracle Analytics Server running on an Oracle Exalytics machine.	Using the nqaggradvisor Utility to Run the Oracle BI Summary Advisor
nqcmd	Use to run test queries against the repository. Connects using an Oracle BI Server ODBC DSN.	Using nqcmd to Test and Refine the Repository
nqlogviewer	Use to view the query log.	*Administering Oracle Analytics Server*
obieerpdpwdchg	Used to change the Oracle BI repository password.	Changing the Oracle BI Repository Password Using the obieerpdpwdchg Utility
patchrpd	Use to apply an XML patch file. This utility is especially useful for patching repository files on Linux systems.	Using patchrpd to Apply a Patch
prunerpd	Use to delete unwanted repository objects from your repository file, such as databases, tables, columns, initialization blocks, and variables.	Delete Unwanted Objects from the Repository
renameapproles	Use to upload a JSON file containing information about the application roles to rename for a specific server instance.	Rename Application Role CommandRename Application Role Command
renameusers	Used to upload a JSON file containing a list of information about users to rename for a specific server instance.	Rename Users Command
sametaexport	Use to generate the information necessary for the Oracle Database SQL Access Advisor to preaggregate relational data and improve query performance.	Exchange Metadata with Databases to Enhance Query Performance
updateConnectionPool	Use to upload a modified JSON file containing updated connection pool values to a specific server instance.	Update Connection Pool Command
updaterpdvariable	Use to upload a JSON file or a modified JSON file containing variable information to a specific server instance.	Update Repository Variables Command
uploadrpd	Use to upload the repository to the Oracle BI Server and include changes in the Oracle Analytics Server archive (BAR) file.	Upload Repository Command
validaterpd	Use to check the consistency of a repository.	Using the validaterpd Utility to Check Repository Consistency

About Options in NQSConfig.INI

Many configuration settings that affect the Oracle BI Administration Tool and repository development are managed in the `NQSConfig.INI` configuration file.

Repository developers must be familiar with the `NQSConfig.INI` configuration settings to effectively work with the Administration Tool and with their repositories.

Common configuration settings that affect repository development include:

- `LOCALE`

  Set `LOCALE` in `NQSConfig.INI` to specify the place (geographical, political. or cultural) to return the data from the server, and to determine the localized names of days and months.

- `DATE_TIME_DISPLAY_FORMAT`, `DATE_DISPLAY_FORMAT`, and `TIME_DISPLAY_FORMAT`

  Set these options in `NQSConfig.INI` to control the display of data/time formats.

- `DEFAULT_PRIVILEGES`

  Set `DEFAULT_PRIVILEGES` in `NQSConfig.INI` to specify the default privilege, *NONE* or *READ*, granted to users and application roles for repository objects without explicit permissions set.

See *Administering Oracle Analytics Server* for full information about `NQSConfig.INI` configuration settings.

Download Repository Command

Use the `downloadrpd` command to download the repository used by the service instance.

The Download Repository command extracts the repository from the archive (BAR) file for the service instance. Oracle recommends only working with the downloaded repository for offline diagnostic and development purposes such as testing. In all other repository development and maintenance situations, you should use BAR to utilize BAR's repository upgrade and patching capabilities and benefits.

You run the utility through a launcher script, `datamodel.sh` on Linux and `datamodel.cmd` on Windows.

If the domain is installed in default folder then the location of the launcher script looks like the following:

Oracle_Home`/user_projects/domains/`*Domain_Name*`/bitools/bin/datamodel.sh` or `datamodel.cmd` on Windows.

See What You Need to Know Before Using the Command.

> **Important:**
>
> You must have BI Service Administrator privileges to run the `downloadrpd` command and issue any of the commands. You must also have membership in the Administrators group in WebLogic security.

Syntax

The `downloadrpd` command takes the following parameters:

```
downloadrpd -O RPDname [-W RPDpwd] -SI service_instance -U cred_username [-P
cred_password] [-S hostname] [-N port_number] [-SSL] [-H]
```

Where

`O` specifies the name of the repository that you want to download.

`W` specifies the password for the repository. If you don't supply the password, you're prompted for the password when the command is run. For security purposes, Oracle recommends that you include a password in the command, only when using automated scripting to run the command.

`SI` specifies the name of the service instance.

`U` specifies a valid user's name to be used for authentication.

`P` specifies the password corresponding to the user's name that you specified for `U`. If you don't supply the password, a prompt displays for the password when the command is run.

`S` specifies the host name. Only include this option when you're running the command from a client installation.

`N` specifies the port number. Only include this option when you're running the command from a client installation.

`SSL` specifies to use SSL to connect to the Oracle WebLogic Server to run the command. Only include this option when you're running the command from a client installation.

`H` displays the usage information and exits the command. Use `-H` or run `.sh` without any parameters to display the help content.

What You Need to Know Before Using the Command

You can learn about the download and upload repository commands, and to the list and update connection pool, rename and delete users and application roles, list and update repository commands.

System Privileges

For either the server installation or client installation, you must have BI Service Administrator privileges to run the command line utility and issue any of the commands.

Passwords in Commands

The commands provide options for including a user's password and a repository passwords. If you don't supply passwords, then you're prompted for passwords when you run the command.

For security purposes, Oracle recommends that you include passwords in the command only if you're using automated scripting to run the command.

Trust Store Key File for SSL

WebLogic Server provides Secure Sockets Layer (SSL) support for encrypting data transmitted between WebLogic Server clients and servers, Java clients, Web browsers, and other servers. When using SSL, you must use the WebLogic Server's trusted keys file if the server is using a self-signed certificate. This is the case when a domain is first created, as the server's identity certificate is generated when the domain is created.

If you replace the WebLogic Server's default self-signed identity certificate with a certificate signed by a recognized signing authority, then the standard Java trusted certificate list validates it and the extra settings aren't needed.

The location of the WebLogic Server's trusted key file is:

ORACLE_HOME/wlserver/server/lib/DemoTrust.jks

The default password for the DemoTrust.jks file is:

DemoTrustKeyStorePassPhrase

The location of the trusted key file and its password are passed to the system properties javax.net.ssl.trustStore and javax.net.ssl.trustStorePassword. For example,

```
java \
-Djavax.net.ssl.trustStore=$ORACLE_HOME/wlserver/server/lib/DemoTrust.jks \
-Djavax.net.ssl.trustStorePassword=DemoTrustKeyStorePassPhrase \
-jar bi-commandline-tools.jar <args…>
```

Upon installation, the data-model-cmd.sh and data-model-cmd.cmd scripts are delivered with the trusted key file locations included. For server installations, you don't need to update the trusted key file locations.

For client installation, you must put the trusted keys file in the correct location. Oracle recommends that you copy and paste the files from the WebLogic Server to the proper location.

Hostname, Port Number, and Use of SSL

For server installations, the command line utility by default queries the endpoint manager which provides the host name, port number, and whether SSL is available. The user doesn't need to include these options in the command.

For client installation, you must include the S host name, N port number, and SSL, use SSL to connect to the WebLogic Server to run the command options in the commands.

Use Online and Offline Repository Modes

You can open a repository for editing in either online or offline mode. The tasks you can perform depend on the mode in which you opened the repository.

This section contains the following topics:

- Editing Repositories in Offline Mode
- Editing Repositories in Online Mode
- Checking Out Objects
- Checking In Changes
- About Read-Only Mode

Edit Repositories in Offline Mode

Use offline mode to view and modify a repository while it isn't loaded into the Oracle BI Server.

If you attempt to open a repository in offline mode while it's loaded into the Oracle BI Server, the repository opens in read-only mode. Only one Administration Tool session at a time can edit a repository in offline mode. See About Read-Only Mode.

You don't need to enter a user name and password to open a repository in offline mode. You only need to enter the repository password.

This section contains the following topics:

- Open Repositories in Offline Mode
- Publish Offline Changes

Open Repositories in Offline Mode

Use these steps to open an Oracle BI repository in offline mode.

If the server is running and the repository you're trying to open is loaded, the repository opens in read-only mode. If you want to edit the repository while it's loaded, you must open it in online mode. Also, if you open a repository in offline mode and then start the server, the repository becomes available to users. Any changes you make become available only when the server is restarted.

When you open an Oracle BI repository in the Administration Tool in offline mode, the title bar displays the name of the open repository.

1. In the Administration Tool, select **File**, select **Open,** and then select **Offline**.
2. Navigate to the repository to open, and then select **Open**.
3. In the Open Offline dialog, enter the repository password, and then click **OK**.

Publish Offline Changes

Use these steps to publish changes made to your repository in offline mode.

See Upload Repository Command.

1. Publish the repository using the upload repository command.

   You can't upload MDS XML format repositories. To publish changes made to MDS XML repositories, you must first convert the repository to Oracle BI repository format.
2. Restart all Oracle BI Server instances. You don't need to restart other BI system components.
3. In Presentation Services, click the **Reload Files and Metadata** link from the Administration page.

Edit Repositories in Online Mode

Use online mode to view and modify a repository while it's loaded into the Oracle BI Server.

The Oracle BI Server must be running to open a repository in online mode. There are certain things you can do in online mode that you can't do in offline mode. In online mode, you can perform the following tasks:

- Manage user sessions
- Manage the query cache

- Manage clustered servers
- Use the Oracle BI Summary Advisor (Oracle Exalytics Machine deployments only)

This section contains the following topics:

- Open Repositories in Online Mode
- Publish Online Changes
- Guidelines for Using Online Mode

Open Repositories in Online Mode

Use these steps to open a repository in online mode.

The Oracle BI Server data source names (DSNs) that have been configured on your computer are displayed in the Open Online Repository dialog. If no additional DSNs have been configured for this version of the Oracle BI Server, you might see only the default DSN that's configured for you during installation.

See Integrating Other Clients with Oracle Business Intelligence in *Integrator's Guide for Oracle Business Intelligence Enterprise Edition* for information about how to create an ODBC DSN for the Oracle BI Server.

The user name that you provide must have the Manage Repositories permission. See *Managing Security for Oracle Analytics Server*.

For multitenancy, provide the details in the form `tenantguid:servicename`, for example `1234101:service1`. Contact the tenant administrator to obtain the GUID and service name. See *Administering Oracle Analytics Server* for information on GUIDs for tenants in the Identity Store. The Oracle BI Server uses the details that you specify to open the repository that's appropriate for your tenant.

If you expect to work extensively with the repository and check out many objects, use the **Load all objects on startup** option to loads all objects immediately, rather than as selected. The initial connect time might increase slightly, but opening items in the tree and checking out items is faster.

Leave the **Tenant info** field blank if multitenancy isn't configured.

1. In the Administration Tool, select **File**, select **Open**, and then select **Online**.
2. In the Open Online Repository dialog, provide a valid user name and password.
3. In a multitenant environment, specify the details for your tenant in the **Tenant info** field.
4. Optional: Select the **Load all objects on startup** option.
5. Select the appropriate DSN and click **OK**.

When you open a repository in the Administration Tool in online mode, the title bar displays the DSN for the Oracle BI Server to which you're connected rather than the name of the current repository.

Publish Online Changes

When performing a single-node deployment, changes made using the Oracle BI Administration Tool, in online mode are available after reloading the metadata in Presentation Services.

In a clustered deployment, Oracle BI Server consumes these changes automatically, but you must restart all destination Oracle BI Servers for them to get the latest changes, and then reload metadata in Presentation Services by clicking the **Reload Files and Metadata** link from the Administration page.

See Using nqcmd to Test and Refine the Repository.

You must run the RollingRestart procedure directly against the source Oracle BI Server. The DSN created upon install for each Oracle BI Server is clustered by default, so you must manually create a non-clustered DSN for the source Oracle BI Server to run the procedure against.

See Integrating Other Clients with Oracle Business Intelligence in *Integrator's Guide for Oracle Business Intelligence Enterprise Edition* for how to create an ODBC DSN for the Oracle BI Server.

You can restart the Oracle BI Servers the following ways:

- Use the start and stop scripts. See Start and Stop Your System in *Administering Oracle Analytics Server*.
- Use the `RollingRestart` ODBC procedure, and enter the following in `nqcmd`:

  ```
  call RollingRestart(timeout);
  ```

  where *timeout* is the number of seconds to wait for each destination Oracle BI Server to restart before moving on to the next one.

  For example:

  ```
  call RollingRestart(300);
  ```

  In this example, the system waits five minutes for each Oracle BI Server to restart. If the given Oracle BI Server restarts sooner, the system moves on to the next one immediately.
- To restart the destination servers using Fusion Middleware Control, first use the Cluster Manager in the Oracle BI Administration Tool, in online mode, to determine which Oracle BI Server is the source, and which are the destination servers. Use the Process tab of the Availability page Fusion Middleware Control to restart the destination Oracle BI Servers. See Using Fusion Middleware Control to Start and Stop Oracle Business Intelligence System Component Processes in *Administering Oracle Analytics Server*.

It's a best practice to avoid making other configuration changes in Fusion Middleware Control or the configuration files when using the `RollingRestart` ODBC procedure or when restarting the destination Oracle BI Servers in Fusion Middleware Control. Because only the destination servers are restarted, a situation might result where the source Oracle BI Server has a different set of configuration settings loaded than the destination Oracle BI Servers. If this occurs, restart the source Oracle BI Server.

Guidelines for Using Online Mode

Use online mode only for small changes that don't require running consistency checks.

Running consistency checks against the full online repository can take a long time. Instead, make more complex changes that require consistency checks in offline mode against a project extract of the repository.

The table provides guidelines for when to perform online and offline edits.

Mode	Use This Mode For...	Example Use Cases
Online	• Small changes that are required to fix things in a running system • Changes that need to be deployed quickly	• Renaming Presentation layer metadata • Reorganizing Presentation layer metadata • Setting the logging level for an application role
Offline	• Full-scale development or customization activities that require running consistency checks multiple times and iterating	• Customizing existing fact or dimension tables • Adding new fact or dimension tables

You should limit the number of concurrent online users. The best practice is to have only one user working in online mode at a time. Even when users have different objects checked out, dependencies between the objects could cause conflicts when the changes are checked in. Only one user should make online changes in a single business model at a time.

If you must have multiple concurrent users in online mode, don't have more than five users. For situations where you need more than five users, use the multiuser development environment. See Set Up and Use the Multiuser Development Environment.

Even with a single user making changes, be aware that online mode is riskier than offline mode because you're working against a running server. If you check in changes that aren't consistent, it might cause the Oracle BI Server to shut down. When you work in online mode, make sure to have a backup of the latest repository so that you can revert to it if needed. You can also use **Undo All Changes** available on the File menu to roll back all changes made since the last check-in.

Check Out Objects

When you're working in a repository open in online mode, you're prompted to check out objects when you attempt to perform various operations.

- To check out objects, do one of the following:
  - Select the objects you want to check out and click **Yes** to check out the objects.
  - If you're performing a task in a wizard, Checkout displays a summary of the objects that you need to check out to complete the operation. Click **Next** to check out the objects and complete the task.

Check In Changes

When you're working in a repository that was opened in online mode, you're prompted to perform a consistency check before checking in the changes you make to a repository.

If you've made changes to a repository and then attempt to close the repository without first checking in your changes, a dialog opens automatically asking you to select an action to take. If you move an object from beneath its parent and then attempt to delete the parent, you're prompted to check in changes before the delete is allowed to proceed.

Use the Check in Changes dialog to make changes available immediately for use by other applications. Applications that query the Oracle BI Server after you've checked in the changes recognize the changes immediately. Applications that are currently querying the server recognize the changes the next time they access any items that have changed.

If the Administration Tool detects an invalid change, a message is displayed to alert you to the nature of the problem. Correct the problem and perform the check-in again.

In some cases, you might see the error, `97005 (Transaction Failed)`. This error occurs when the Oracle BI Server doesn't accept the changes. You can check the server log files to determine the cause of the problem.

You must save changes to persist the changes to disk. You must check in changes before you can save, but you don't need to save to check in changes.

- In the Administration Tool, select **File**, then select **Check In Changes**.

About Read-Only Mode

Only one component, the Oracle BI Server or a single Oracle BI Administration Tool client in offline mode can have a repository open in read/write mode at a time.

If a second component opens a repository that's already in use, the repository is opened in read-only mode.

For example, assume the Oracle BI Server loads a repository in read/write mode. Any Administration Tool clients connecting to that repository in online mode also get read/write mode because they're accessing the repository through the Oracle BI Server. However, Administration Tool clients opening that repository in offline mode get read-only mode because the repository is already open for read/write through the Oracle BI Server.

If the Administration Tool client opens a repository offline in read/write mode, when the Oracle BI Server starts, the server and any Administration Tool client are also opened in read-only mode.

To enable the server to load the repository in read/write mode, you must first close the Administration Tool client that has the repository locked, and then restart the Oracle BI Server.

The Administration Tool opens a repository in read-only mode when Oracle Analytics Server has been clustered, and the Administration Tool is connected in online mode to a dependent server. The cluster’s controlling BI Server holds a lock on the repository. To avoid this lockout situation when running in a clustered environment, ensure that the Oracle BI Server ODBC data source name (DSN) used by the Administration Tool is configured to point to the cluster controllers rather than to a specific Oracle BI Server.

Open a MDX XML Repository

Use these steps to open a MDS XML file.

When you open a MDS XML format repository in the Administration Tool, the title bar displays the format and root folder location, for example, MDS XML `C:\Root_Folder`.

1. In the Administration Tool, select **File**, select **Open**, and then select **MDS XML**.
2. Select the root folder location for the MDS XML files and click **OK**.
3. If this is the first time you've opened this MDS XML repository, specify whether this repository is a standalone MDS XML repository, or whether it's under source control.
4. Click **OK**.

Check the Consistency of a Repository or a Business Model

Repository metadata must pass a consistency check before you can make the repository available for queries.

The Consistency Check Manager lets you enable and disable rules for consistency checks, find and fix inconsistent objects, and limit the consistency check to specific objects. You can also use the `validaterpd` utility to check the validity of all metadata objects.

> **Note:**
>
> The Model Check Manager identifies modeling problems that affect Oracle BI Summary Advisor and aggregate persistence performance and results. Run Model Check Manager before running Oracle BI Summary Advisor or the Aggregate Persistence Wizard. See Using Model Check Manager to Check for Modeling Problems.

This section contains the following topics:

- About the Consistency Check Manager
- Checking the Consistency of Repository Objects
- Using the validaterpd Utility to Check Repository Consistency
- Common Consistency Check Messages

About the Consistency Check Manager

The Consistency Check Manager checks the validity of your repository to ensure that it can load at run time, and to identify any syntax or semantic errors that may cause queries to fail.

Running a consistency check might result in updates to your repository metadata when you run the **Global Consistency Check** or the **Check Consistency** option

against an object. You must save the repository when using those options. For example, invalid objects are deleted during Consistency Checks. This behavior might result in deleted expressions and filters on logical table sources and logical columns. Invalid references can occur when objects were deleted in the Physical layer without properly accounting for the references in the Business Model and Mapping layer objects.

The **Show Consistency Check** option is read-only and doesn't implement changes in the repository.

The Consistency Check Manager doesn't check the validity of objects outside the metadata using the connection. It only checks the consistency of the metadata and not the mapping to the physical objects outside the metadata. If the connection isn't working or objects were deleted in the database, the Consistency Check Manager doesn't report these errors.

The Consistency Check Manager identifies application roles that defined in the Administration Tool, but that weren't added to the policy store. Messages about placeholder application roles only appear when you perform a consistency check in online mode. The set of consistency check messages returned for your repository might contain different results depending on whether you've opened the repository in offline or online mode.

If you use lookup tables to store translated field names with multilingual schemas, the consistency checking rules are relaxed for the lookup tables. See Localize Oracle Analytics Server in *Administering Oracle Analytics Server*.

The consistency checker returns the following types of messages:

- Errors

  These messages describe errors that you must fix. Use the information in the message to correct the inconsistency, then run the consistency checker again. The following is an example of an error message:

  ```
  [38082] Type of Hierarchy '"0RT_C41"..."0RT_C41/MDF_BW_Q02"."Product Hierarchy for
  Material MARA"' in Cube Table '"0RT_C41"..."0RT_C41/MDF_BW_Q02"' needs to be set.
  ```

  If you disable an object and it's inconsistent, a message is displayed, asking if you want to make the object unavailable for queries.

- Warnings

  These messages indicate conditions that you might need to fix. For example, you might receive a warning message about a disabled join that was intentionally disabled to eliminate a circular join condition. Other messages may warn of inconsistent values, or feature table changes that don't match the defaults. The following is an example of a warning message:

  ```
  [39024] Dimension '"Paint"."MarketDim"' has defined inconsistent values in its
  levels' property 'Number of elements'.
  ```

> **Note:**
>
> After upgrading from a previous software version and checking the consistency of your repository, you might notice messages that you hadn't received in previous consistency checks. The inconsistencies are the result of issues that were undetected before the upgrade, not new errors.

Run the Consistency Check Manager

Use the Oracle BI Administration Tool to run the consistency checker on all of the repository objects, on a specific physical database or data source, physical database, business model, or subject area.

You can view the Consistency Check Manager's results without performing a global consistency check, by selecting the **Show Consistency Checker** from **Tools** menu. If you've checked consistency in the current session, the messages from the last check appear in the Messages pane.

If a disabled object is inconsistent, you're prompted to make the object unavailable for queries. If an object isn't consistent, the Consistency Check Manager appears and displays a list of messages.

1. In the Administration Tool, open a repository.
2. Do one of the following:
   - From the **File** menu, select **Check Global Consistency** , then select **Report Only**. This option reviews all of the objects in the repository and generates a list of errors.
   - From the **File** menu, select **Check Global Consistency** , then select **Auto-fix**. This option reviews all of the objects in the repository and automatically fixes any errors where possible. When this option is chosen, a list of all fixes is logged to the following file:
     `orainst\servers\obis1\logs\`*`username`*`_NQSAdminTool.log`.
   - In the repository, select an object, right-click, and select **Check Consistency**. This option reviews all of the objects in the repository and automatically fixes any errors where possible. When this option is chosen, a list of all fixes is logged to the following file:
     `orainst\servers\obis1\logs\`*`username`*`_NQSAdminTool.log`

Review the output. If you've chosen to automatically fix problems in the repository, you must save the fixed repository. If your repository is read-only the auto-fix option is disabled and the right-click **Check Consistency** option generates a report without making fixes.

Use the validaterpd Utility to Check Repository Consistency

You can use the Oracle BI Server `validaterpd` utility to check the validity of all metadata objects in a repository.

Running this utility performs the same validation checks as the Consistency Check Manager in the Administration Tool.

The `validaterpd` utility is available on both Windows and Linux systems. You can run `validaterpd` against a binary Oracle BI repository file, against an XML file based on the Oracle BI Server XML API, or against a set of MDS XML documents.

You must provide the full path names to your repository files, both the input files and the output files, if they're located in a different directory.

The location of the `validaterpd` utility is:

`BI_DOMAIN``/bitools/bin`

Using `validaterpd` with the `-L` option checks your repository metadata for issues that might affect the success of Oracle BI Summary Advisor or the aggregate persistence engine. See Checking Models Using the validaterpd Utility to learn about using `validaterpd` with the `-L` option.

Syntax

The `validaterpd` utility takes the following parameters:

```
validaterpd {-R repository_name | -I input_file_pathname |
-D MDS_XML_document_directory} [-P repository_password] {-O output_txt_file_name |
-C output_csv_file_name | -X output_xml_file_name} [-8] [-F fixed_rpd_name|-E] [-S] [-
B]
```

Where:

repository_name is the name and path of the binary Oracle BI repository file that you want to validate.

input_file_pathname is the name and path of the XML input file that you want to validate.

MDS_XML_document_directory is the location of the input MDS XML documents.

repository_password is the password for the repository that you want to validate.

The *repository_password* argument is optional. If you don't provide the password argument, you're prompted to enter the password when you run the command. To minimize the risk of security breaches, Oracle recommends that you don't provide password arguments either on the command line or in scripts. The password argument is supported for backward compatibility only, and is removed in a future release. For scripting purposes, you can pass the password through standard input.

output_txt_file_name is the name and path of a text file where the validation results are recorded.

output_csv_file_name is the name and path of a csv file where the validation results are recorded.

output_xml_file_name is the name and path of an XML file where the validation results are recorded.

Specify `-M` to specify that you want to run MDS XML documents. If you specify `-D`, the `-M` argument isn't needed. You only need to specify `-M` when you've a single MDS XML file that contains all the object definitions.

`-8` specifies UTF-8 encoding in the output file.

Specify `-F` to create a new version of the repository in Oracle BI repository file format that includes automatic fixes for some internal validation errors. For *fixed_rpd_name*, provide the name and path of a binary Oracle BI repository file where you want to save the fixes. When this option is chosen, a list of all fixes is logged to the following file: `orainst\servers\obis1\logs\`*username*`_NQSAdminTool.log`.

Specify `-E` to save the changes into the input repository (`-R` must also be specified). When this option is chosen, a list of all fixes is logged to the following file: `orainst\servers\obis1\logs\`*username*`_NQSAdminTool.log`.

Specify `-S` to check server errors and navigation spaces only.

Specify `-B` to skip checks for business models availability.

Examples

The following example generates an output file called *results.txt* that contains validation information for the repository called *repository.rpd*, and saves a fixed version to *fixed_repository.rpd*:

```
validaterpd -R repository.rpd -O results.txt -F fixed_repository.rpd
Give password: my_rpd_password
```

The following example generates an output file called *results.csv* that contains validation information for the repository contained in the MDS XML documents located at C:\MDS_dir:

```
validaterpd -D C:\MDS_dir -C results.csv
Give password: my_rpd_password
```

Common Consistency Check Messages

Review the table to get information about some commonly seen consistency check warnings and errors.

The table provides a partial list only and doesn't show all possible warnings and errors.

Validation Rule Example	Type	Description
[14031] The content filter of a source for logical table: FACT_TABLE_NAME references multiple dimensions.	Error	The given logical table has a logical table source with a WHERE clause filter that references multiple dimensions. A WHERE clause with multiple dimensions is invalid.
[38126] 'Logical Table' '"Technology - WFA"."Fact WFA WO "' has name with leading or trailing space(s).	Error	Identifies an object with leading or trailing spaces in the object name. Repository objects can no longer have leading or trailing spaces in their names. Leading and trailing spaces in object names can cause query and reporting issues.
[38012] Logical column DIM_Start_Date.YEAR_QUARTER_NBR doesn't have a physical data type mapping, nor is it a derived column. [38001] Logical column DIM_Start_Date.YEAR_QUARTER_NBR has no physical data source mapping.	Error	Logical columns that aren't mapped to any logical table source are reported as consistency errors, because the logical table source mappings are invalid and would cause queries to fail. Both of the given validation rules are related to the same issue.
[39062] Initialization Block 'Authorization' uses Connection Pool '"My_DB". "My_CP"' which is used for report queries. This may impact query performance.	Warning	Indicates that the same connection pool is being used for both queries and for initialization blocks. This configuration isn't recommended. Instead, create a dedicated connection pool for initialization blocks. Otherwise, query performance might suffer, or user logins might stop responding if authorization initialization blocks can't run.
[39028] The features in Database 'MyDB' don't match the defaults. This can cause query problems.	Warning	Some database feature defaults were changed in this release of Oracle Analytics Server. Unless you've specific customizations to your feature set, it's recommended that you reset your database features to the new defaults.

Validation Rule Example	Type	Description
[39003] Missing functional dependency association for column: DIM_Offer_End_Date.CREATE_DT.	Warning	This warning indicates that the given column is only mapped to logical table sources that are disabled. The warning brings this issue to the repository developer's attention in case the default behavior isn't desired.
[39059] Logical dimension table MY_DIM has a source MY_DIM_DAILY at level Daily that joins to a higher level fact source MY_FACT_SUM.MTHLY_SUM	Warning	Even though this fact logical table source has an aggregate grain set in this dimension, no join was found that connects to any logical table source in this dimension (or a potentially invalid join was found). This means that either no join exists at all, or it does exist but is potentially invalid because it connects a higher-level fact source to a lower-level dimensional source. Such joins are potentially invalid because if followed, they might lead to double counting in query answers. For example, consider Select year, yearlySales. Even if a join exists between monthTable and yearlySales table on yearId, it shouldn't be used because such a join would overstate the results by a factor of 12 (the number of months in each year). If you get a 39059 warning after upgrading, verify that the join is as intended and doesn't result in incorrect double counting. If the join is as intended, then ignore the 39059 warning.
[39055] Fact table "HR"."FACT - HC Budget" isn't joined to tables in logical dimension "HR"."DIM - HR EmployeeDim". This can cause problems when extracting project(s).	Warning	This warning indicates that there's a physical join between the given fact and dimension sources, but there isn't a corresponding logical join between the fact table and the dimension table.
[39054] Fact table "Sales - STAR"."Fact - STAR Statistics" isn't joined to logical dimension table "Sales - STAR"."Dim - Plan". This can cause problems when extracting project(s).	Warning	This warning indicates that the aggregation content filter "Group by Level" in the logical table source of a fact table references logical dimension tables that aren't joined to that fact table. If that fact table is extracted in the extract/MUD process, the dimensions that aren't joined not be extracted. In this case, the aggregation content of the extracted logical table source wouldn't be the same as in the original logical table source.
[39057] There are physical tables mapped in Logical Table Source ""HR"."Dim - Schedule"."SCH_DEFN"" that aren't used in any column mappings or expressions.	Warning	This warning indicates that the given logical table source has irrelevant tables added that aren't used in any mapping. This situation shouldn't cause any errors.

Part II
Build Your Metadata Repository

This part details the steps and information required to build your metadata repository.

Chapters:

- Introduction to Building Your Metadata Repository
- Before You Begin
- Set Up and Use the Multiuser Development Environment
- Use a Source Control Management System for Repository Development
- Import Metadata and Working with Data Sources
- Work with ADF Data Sources
- Set Up Database Objects and Connection Pools
- Work with Physical Tables, Cubes, and Joins
- Work with Logical Tables, Joins, and Columns
- Work with Logical Dimensions
- Manage Logical Table Sources (Mappings)
- Create and Maintain the Presentation Layer
- Create and Persist Aggregates for Oracle BI Server Queries
- Apply Data Access Security to Repository Objects
- Complete Oracle BI Repository Setup
- Set Up Data Sources on Linux
- Manage Oracle BI Repository Files
- Use Expression Builder and Other Utilities
- Use Variables in the Oracle BI Repository
- Manage the Repository Lifecycle in a Multiuser Development Environment
- MUD Case Study: Eden Corporation
- Merge Rules
- Delete Unwanted Objects from the Repository
- Supported Data Types
- Exchange Metadata with Databases to Enhance Query Performance
- XML Schema Files for ADF Mapping Customizations
- Administration Tool Keyboard Shortcuts

3

Set Up and Use the Multiuser Development Environment

This chapter explains how to set up and use the multiuser development environment, including defining projects, setting up the multiuser development directory, checking out and publishing changes to projects, and merging metadata.
Multiuser development (MUD) provides a mechanism for concurrent development on overlapping code bases. The MUD environment manages subsets of metadata, in addition to multiple users, by providing a built-in versioning system for repository development.

See also the following resources:

- Manage the Repository Lifecycle in a Multiuser Development Environment
- Use a Source Control Management System for Repository Development

This chapter contains the following topics:

- About the Multiuser Development Environment
- Set Up Projects
- Set Up the Multiuser Development Directory
- Make Changes in a Multiuser Development Environment
- Publish Changes to Multiuser Development Repositories
- Branch in Multiuser Development
- View and Delete History for Multiuser Development
- Set Multiuser Development Options

About the Multiuser Development Environment

Multiuser development facilitates creating application metadata in enterprise-scale deployments.

Application metadata is stored in a centralized metadata Oracle BI repository (RPD) file. The Administration Tool is used to work with these repositories. You don't use a multiuser development environment with MDS XML-format repositories.

Master repository refers to the copy of a repository in the multiuser development directory

The following are examples of how you might use a multiuser development environment:

- Several developers work concurrently on subsets of the metadata and then merge these subsets back into a master repository without their work conflicting with other developers. For example, after completing an implementation of data warehousing at a company, an administrator might want to deploy Oracle Analytics Server to other functional areas.
- A single developer manages all development. For simplicity and performance, this developer might want to use the multiuser development environment to maintain metadata code in smaller chunks instead of in a large repository.

In both examples, an administrator creates projects in the repository file in the Administration Tool, then copies this repository file to a shared network directory, called the multiuser development directory. Developers are able to check out projects, make changes and then merge the changes into the master repository. When developers check out projects, using the Administration Tool, files are automatically copied and overwritten in the background. Your administrator must perform setup tasks. Developers must pay close attention to the Administration Tool messages that appear during check-out, merge, and publish procedures.

When developers check out projects, repository files aren't automatically copied or overwritten. The Administration Tool creates two new files when projects are checked out, one to hold the original project data, and one to hold the project changes.

For example, when a repository developer checks out project A from `master.rpd` in the `C:\multiuser` development directory, the Administration Tool extracts all metadata related to project A and prompts the developer for a new file name to save the data. When the developer chooses a new file name, for example, Mychanges.rpd, the Administration Tool creates two new files:

- A file called *MyChanges.rpd* that contains the changes made by the developer
- A file called *originalMyChanges.rpd* that contains the original project data

The Administration Tool determines the developer's changes by comparing the *Mychanges.rpd* with the *originalChanges.rpd*. The information about what has changed is required during the multiuser development merge process.

About the Multiuser Development Process

Multiuser development presupposes a clear understanding of customer technical and business objectives.

It also requires that you follow clearly defined development processes and adhere rigorously to those processes, including consistent merging and reconciliation practices.

The following procedure shows the general steps to follow when deploying a multiuser development environment. The first three steps are usually performed by an administrator, and the remaining steps are usually performed by one or more developers. See Creating Projects.

Oracle recommends merging changes as soon as possible and as often as possible. Frequent merges makes conflict resolution easier and simplifies the merge.

1. Define projects to organize voluminous metadata into manageable components. Consider these tips:
   - Use smaller RPDs to shorten and simplify development effort and unit testing.
   - Organize development resources by projects to spread workload and reduce inconsistencies and overwrites.
2. Set up a shared network directory to use as the multiuser development directory.
3. Copy the master repository to the multiuser development directory.
4. Extract one or more projects or the entire repository for local development.
5. Merge repository objects and resolve conflicts.

- Because metadata objects are often highly interrelated, several developers could be working on the same objects.
- You can perform regular subset refreshes to merge your local changes with the latest version of the master. When configuration conflicts occur during the merge process, developers are prompted for the correct process.

6. Publish changes to the network.
   - A final subset refresh (merge) is performed during the publishing step. Many developers can simultaneously work on the same objects, but only one can publish at a time. The repository is locked during the publishing step.
7. Use Logging and Backup features to identify points of erroneous or incorrect configuration.
   - The log file tracks multi-development activity, along with comments.
   - The master repository and developer repositories are automatically backed up for future reference and for use in manual rollback.

Set Up Projects

Projects are the central enabler of metadata management.

A project consists of a discretely-defined subset of the repository metadata, in the form of groups of logical stars with associated metadata. A project has the following characteristics:

- Is largely defined by logical fact tables in the applicable business model
- Automatically adds related logical dimension tables and other metadata during extract
- Can have one-to-many logical fact tables

For projects that are just beginning, the best practice is to begin with a repository containing all the necessary physical table and join definitions. In this repository, you create a logical fact table as a placeholder in the Business Model and Mapping layer and a subject area as a placeholder in the Presentation layer. As you add business model and subject area metadata, new projects based on individual subject areas and logical facts can be created.

Follow these guidelines when setting up projects:

- Only one person at a time can create projects in a master repository.
- Don't delete projects unless they're no longer under active development.
- Choose your project name carefully when creating a project. Don't rename projects.
- Use care when removing objects from projects to avoid problems with repository extract/check-out.

This section contains the following topics:

- About Projects
- Create Projects

About Projects

Projects can consist of Presentation layer subject areas and their associated business model logical facts, dimensions, groups, users, variables, and initialization blocks.

You can create projects so that developers can work on projects in their area of responsibility. The primary reason to create projects is to support multiuser development. During the development process, you can split up the work (metadata) between different teams within your company by extracting the metadata into projects so that each project group can access a different part of the metadata.

You might want to create projects for licensing reasons. Before releasing a new software version, you might want to ensure that only the metadata that's relevant to the licensed application is in a project and that everything is consistent and complete. You add only the fact tables that are relevant to the application. Project extractions are fact table-centric to ensure that project extracts are consistent, and to make licensing manageable.

Create Projects

A project can represent a subject area or a subset of logical fact tables related to the selected subject area. The Oracle BI Administration Tool automatically adds the related business model and Physical layer objects to the project.

You can use the same object in multiple projects. You can choose to group facts by business model or you can select a business model or a set of logical fact tables that are part of a business model to use in a project. You need to explicitly add Presentation layer objects to your project.

Although the project definition doesn't include Physical layer objects, these objects are determined and extracted through the project definition.

After you create projects, they become part of the metadata and are available to multiple developers who need to perform development tasks on the same master repository. When defined this way, projects become a consistent repository after a developer checks out the projects and saves them as a new repository file.

In your project, it's more common to select **Group Facts By** *Subject Area*.

You can include objects in your project that aren't referenced such as variables and initialization blocks that are directly referenced by other extracted objects. You can add the top node for each object type, for example, Variables, then selectively remove individual objects.

- If you're using initialization blocks for authentication, include any necessary initialization blocks.
- You can include repository variables or other objects that aren't yet referenced by other objects, but that you might want to use in future repository development.
- You can include users and application roles as part of your data access security settings.

If you don't see the set of subject areas you expect after the project is created, edit the project to explicitly add the subject areas you need.

1. In the Administration Tool, choose **File**, select **Open**, and then select **Offline**.
2. In the Open dialog, select the repository that you want to make available for multiuser development, then click **OK**. Provide the repository password, then click **OK** again.
3. Select **Manage**, then select **Projects**.

4. In the Project Manager dialog, in the right pane, right-click and then select **New Project**.
5. In the Project dialog, in **Name**, type a name for the project.
6. In **Group Facts By**, select *Business Model*, or *Subject Area*.
7. Perform one or more of the following steps to add fact tables to your project:
   - Under the **Group Facts By** area, select a subject area or business model, and then click **Add**.
   - Expand the subject areas or business models and select one or more logical fact tables, then click **Add**.
8. Optional: Click **Remove** to remove fact tables from the project.
9. Optional: Add any application roles, users, variables, initialization blocks, or lookup tables needed for the project.
10. Select the Presentation layer objects to include in your project and click **Add**.
11. Click **OK**.

Set Up the Multiuser Development Directory

Your administrator needs to perform these tasks to prepare for multiuser development.

To prepare for multiuser development:

- Identify or create a shared network directory dedicated to multiuser development projects.
- After creating all projects, copy the repository file with the projects to the multiuser development directory to use as your master repository for multiuser development.

After the administrator has identified the multiuser development directory and copied the repository file, developers must set up the Administration Tool to point to the multiuser development directory before they can check out projects.

This section contains the following topics:

- Identify the Multiuser Development Directory
- Copy the Master Repository to the Multiuser Development Directory
- Set Up a Pointer to the Multiuser Development Directory

Identify the Multiuser Development Directory

After defining all projects, the administrator must identify or create a shared network directory, called the multiuser development directory.

Developers can access, and then upload the master repository to the multiuser development directory.

Use the shared network directory only for multiuser development. The multiuser development directory contains copies of repositories that are maintained by multiple developers. You must use a Windows system for the multiuser development directory.

Developers create a pointer to the multiuser development directory when they set up the Administration Tool on their computers.

> **Important:**
>
> The administrator must set up a separate, shared network directory that's dedicated to multiuser development. If not set up and used as specified, critical repository files can be unintentionally overwritten and repository data can be lost.

Copy the Master Repository to the Multiuser Development Directory

After the multiuser development directory is identified, the administrator must copy the master repository file to the multiuser development directory.

Projects from this master repository are extracted and downloaded by the developers who make changes and then merge these changes back into the master repository.

After you copy the repository to the multiuser development network directory, notify developers that the multiuser development environment is ready.

Even after starting repository development, it might be necessary to make manual changes to the master repository from time to time. See *Manually Update the Master MUD Repository* in Troubleshoot Multiuser Development.

Set Up a Pointer to the Multiuser Development Directory

Before checking out projects, developers working in the multiuser development environment must set up their local copies of the Administration Tool to point to the multiuser development directory on the network.

The Administration Tool uses this location when the developer checks out and checks in objects in the multiuser development directory. Until the pointer is set up, the multiuser options aren't available in the Administration Tool

Initially, the network directory contains the master repositories. The repositories in this location are shared with other developers. Later, the network directory contains additional multiuser development history files, including historical subsets and repository versions. Don't manually delete any files in the multiuser development directory; these files are important and are used by the system.

When setting up the pointer, the developer can also complete the **Full Name** field. Although the **Full Name** field is optional, Oracle recommends completing this field to allow other developers to know who has locked the repository.

1. From the Administration Tool menu, choose **Tools**, and select **Options**.
2. In the Options dialog, click the Multiuser tab.
3. In the Multiuser tab, for **Multiuser development directory**, enter the full path to the network directory.
4. In the **Full Name** field, type your complete name, then click **OK**.

Make Changes in a Multiuser Development Environment

Each developer can publish (merge) changes into the master repository or discard the changes in a multiuser development environment.

During check-out, refresh, and publish, a copy of the master repository is temporarily copied to the developer's local repository directory, similar to the following location, *ORACLE_INSTANCE*\bifoundation\OracleBIServerComponent\coreapplication_obis*n*\repository by default.

This section contains the following topics:

- About Changing and Testing Metadata
- Make Changes to a Repository Using Projects
- Make Changes to an Entire Repository
- About Multiuser Development Menu Options

About Changing and Testing Metadata

Most types of changes that can be made to standard repository files are also supported for local repository files.

Developers can add new logical columns, add new logical tables, change table definitions, and change logical table sources. Developers might also work simultaneously on the same projects or entire repository locally. Oracle Analytics Server assumes the individual developer understands the implications that these changes might have on the master repository. For example, if a developer deletes an object in a local repository, this change is propagated to the master repository when local changes are merged without a warning prompt.

To ensure metadata integrity, don't remove a physical column unless there are no logical table source mappings to that physical column. Because of this, if you use a multiuser development environment, you can't delete a logical column and its associated physical column at the same time. Instead, you must first delete the logical column and perform a merge. Then, you can delete the physical column and perform another merge to safely remove the object.

In some cases, logical column types can change over the course of MUD development that results in unexpected logical column types. When this occurs, you can generate a list of logical columns and their types using the Generate Logical Column Type Document utility in the Administration Tool or `biservergentypexml`. Then use the Compare Logical Column Types utility for subsequent MUD versions to ensure that the logical column types match as expected. For example, you can generate a logical column type list for repository version 20, and use the Compare Logical Column Types utility to compare the list against repository version 30. See Generate a List of Logical Column Types and Compare Logical Column Types.

Changes to physical connection settings aren't propagated to the master repository upon merge and publish. This ensures that developers can apply settings for their local test data sources to perform unit testing of their model changes without impacting other developers.

In addition to physical connection settings, security settings and database feature table changes aren't retained in a multiuser development merge to prevent developers from overwriting passwords and other important objects in the master repository.

After making changes to a local repository, the developer uploads the repository and tests the edited metadata.

> **Note:**
>
> DSNs that are specified in the metadata must exist on the developer's workstation.

Make Changes to a Repository Using Projects

These topics provide information on making changes in a multiuser development environment using projects.

- About Repository Project Checkout
- Check Out Projects
- Use the extractprojects Utility
- Refreshing the Local Project Extract

About Repository Project Checkout

The Administration Tool performs the following tasks during checkout.

- In the developer's local repository directory, the Administration Tool makes a temporary copy of the master repository.

  > **Important:**
  >
  > If a repository with that name exists in this location, the developer is asked to confirm overwriting the existing repository. If the developer clicks Yes, the existing local repository is immediately overwritten in the background and after the new repository is saved, the temporary master repository file is automatically deleted.

- In the developer's local repository directory, the Administration Tool saves a local copy of the selected projects in a new repository such as *Metadata1.rpd*. The developer provides a name for the local copy. The developer makes metadata changes in this file. The number is incremented by 1 for each checkout for that session.
- In the developer's local repository directory, the Administration Tool saves a second local copy of the new repository, adding *original* as the prefix, for example, *originalMetadata1.rpd*.
- After the developer saves the new repository file, check out is complete. In the developer's local repository directory, the temporary copy of the master repository is automatically deleted.

> **Important:**
>
> When the developer selects and saves the projects to a local repository file, the Administration Tool doesn't place a lock on the projects in the master repository on the shared network drive. Therefore, nothing physically prevents others from working on the same project. To determine if a project has been checked out, you need to look in the log file in the multiuser development directory on the shared network drive.

Check Out Projects

Use this task to check out projects using the Oracle BI Administration Tool.

The Multiuser Development Checkout dialog doesn't display if there is only one repository in the multiuser development directory.

1. From the Administration Tool menu, choose **File** , select **Multiuser** , and then select **Checkout**.
2. If there is more than one repository in the multiuser development directory, then the Multiuser Development Checkout dialog is displayed. Select the appropriate repository, and click **OK**.
3. In the Extract from dialog, type the repository password, and click **OK**.
4. If there is more than one project in the master repository, then the Browse dialog is displayed. Select the projects that you want to be part of your project extract, and click **OK**.
5. In the Create new subset repository dialog, type a name for the new repository, for example, *Metadata1.rpd*, and click **Save**.

   A working project extract repository is saved on your local computer. The name is exactly as you specified and is opened in offline mode. A log file is also created.

   A second copy of the project extract repository is saved in the same location. The name of this version contains the word *original* added to the beginning of the name that you assigned to the repository extract. Don't change the original project extract repository. It's used during the multiuser development merge process, and when you want to compare your changes to the original projects.

Use the extractprojects Utility

You can use the Oracle BI Server extractprojects utility to cut projects from a given repository without the overhead of the MUD environment.

The `extractprojects` utility is available for Windows and Linux systems. You can use `extractprojects` only with binary repositories in the Oracle BI repository format.

The `extractprojects` utility generates an Oracle BI repository file that includes the set of projects that you specify. The utility doesn't perform the tasks that are performed when you check out projects using the Administration Tool, such as saving an original repository file or tracking the extract as a check-out in the MUD directory.

The `extractprojects` utility is located in the following directory:

BI_DOMAIN`/bitools/bin`

Syntax

The `extractprojects` utility takes the following parameters:

```
extractprojects -B base_repository_name -O output_repository_name {-I
input_project_name} [-P repository_password] [-L] [-E project_list_file_name]
```

The variables are as follows:

base_repository_name is the name and path of the repository from which you want to extract projects.

output_repository_name is the name and path of the repository generated by the utility.

input_project_name is the name of a project you want to extract. You can enter multiple projects. Precede each project entry with `-I`, for example, `-I project1 -I project2`. If the project name contains spaces, enclose it in double quotes, for example, *"project 1"*.

repository_password is the password for the repository from which you want to extract projects.
The *repository_password* argument is optional. If you don't provide the password argument, you're prompted to enter the password when you run the command. To minimize the risk of security breaches, Oracle recommends that you don't provide the password arguments in the command line or in scripts. The password argument is supported for backward compatibility only. For scripting purposes, you can pass the password through standard input.

`- L` enables logging. When logging is enabled, a log file with the name format, *ProjExtr.YYYYMMDD.HHMMSS.xml*, is created in the Oracle BI Server logging directory. For example:

```
ORACLE_INSTANCE/diagnostics/logs/OracleBIServerComponent/coreapplication_obisn/
ProjExtr.20100616.082904.xml
```

`-E` is an optional argument that lets you print a list of all projects contained in a repository into a file. Specify *project_list_file_name* after the option to specify the file name and location in which you want to store the project names. `-E` is only used with `-B` and `-P` and doesn' actually perform a project extract.

The `-U` and `-F` are visible in the syntax list, but are for internal use only.

Example

The following example extracts *project1* and *project2* from *my_repos.rpd* and creates a new repository called *extract_repos.rpd*.

```
extractprojects -B my_repos.rpd -O extract_repos.rpd -I project1 -I project2
Give password: my_rpd_password
```

> **Note:**
>
> Provide the full path names to your repository files, both the input file and the output file, if they're located in a different directory.

Refresh the Local Project Extract

Use the Refresh Subset option to update the local project extract with any changes that were made to the master repository.

The Refresh Subset option merges the latest changes with the local project extract. Use he Refresh Subset as an incremental step during development before publishing your final changes at the end of your development session.

As a best practice, you should refresh your local project extract frequently to enable resolving conflicts during merge. If too many changes are merged at one time, then making the appropriate merge decisions for conflicts is confusing and error-prone.

Make Changes to an Entire Repository

The preferred method for making changes to a repository in a multiuser development environment is to use projects.

You might encounter situations during which you want to make changes that involve you accessing the entire repository at one time.

Use this method only when necessary, because system performance is likely decrease, especially during merges of large repositories.

- In the Oracle BI Administration Tool, from the **File** menu, select **Multiuser**, and then select **Whole Rpd Checkout** to access the entire repository, including objects not assigned to projects.

About Multiuser Development Options

You can perform many tasks from the Multiuser menu.

After the local developer makes changes, tests the changes, and saves the repository locally, the local developer can perform the following tasks:

- **Compare with Original**. Compares the working extracted local repository to the original extracted repository. When you select this option, the Compare repositories dialog is displayed and lists all the changes that were made to the working extracted repository since you checked out the projects or the entire repository.
- **Refresh Subset**. Refreshes the local project extract with any changes that were made to the master repository. The changes from the master are merged with your local changes.

  If changes have been made to the master repository, then the old project extract file, *originalfilename.rpd*, is replaced with a new project extract file called *currentfilename.rpd*.
- **Publish to Network**. Publishes changes made to the local project extract or the entire repository to the master repository. A lock is acquired to lock the master repository during the publish step. Publishing the changes automatically performs a Refresh Subset operation to merge the local changes with any additional changes from the master. Then, the merged changes are published to the master repository, the lock is released, and the local repository is closed.
- **Undo Publishing**. Used when mandatory consistency checks are enforced during the publishing step, and errors occur. When you're notified of consistency check errors during publishing, you can choose to fix the errors immediately, as part of the publishing step. The master repository is locked during this process. If you need to release the lock on the

master and fix the changes later, then select **Undo Publishing** to release the lock and return to the latest subset extract repository.

- **Discard Local Changes**. Any time after check out and before publishing, you can discard your changes. If you select this option, there is no opportunity to change your mind. For example, no confirmation dialog is displayed. When you select this option, the working repository closes without giving you an opportunity to save your work.

Publish Changes to Multiuser Development Repositories

After changing and testing the metadata on a local computer, the developer must publish local changes to the master repository in the multiuser development directory.

The Oracle BI repository development process uses a three-way merge to manage concurrent development. Metadata merges are done first on local environments and then merged with the master repository. A three-way merge identifies local changes based on the following repository characteristics:

- The Master Oracle BI repository
- The Baseline Oracle BI repository or Master Oracle BI repository snapshot at time of project extraction
- The current locally developed and changed Oracle BI repository

Changes are managed by merge and reconciliation. Most of the merging process is automatic, and changes don't conflict. In case of any conflicting metadata sources, developers can locate and resolve them.

An administrator can also merge the changes from multiple repositories manually, or import objects from different repositories outside of a particular MUD environment.

Ensure that you merge changes frequently. The merge process is very complex and can become difficult if there are too many changes. See Merge Rules for how objects are merged during the merge process.

It's a best practice to refresh your subset repository regularly to identify conflicts early. Refreshing your subset performs a subset remerge with the latest version of the master, then leaves the repository open for you to continue making changes until you're ready to publish.

This section contains the following topics:

- About the Multiuser Development Process
- Publish to the Network
- Enforce Consistent Repositories When Publishing Changes

About the Multiuser Development Merge Process

The topic describes the multiuser development merge process.

The merge process involves the following files:

- Local (subset) repository, either original or current (refreshed). The local subset repository is one of the following:

  - If no subset refresh has been performed, contains the state of the projects or the entire repository as originally extracted. This repository name begins with *original*, for example, *originalDevelopment2.rpd*.
  - If a subset refresh has been performed, contains the state of the projects or the entire repository since the last merge that occurred during the subset refresh. The repository name begins with *current*, for example, *currentDevelopment2.rpd*.
- Modified local (subset) repository. Contains the extracted projects after being modified by the developer. This version is stored in the same location as the original or current version of the local subset repository.
- Latest master repository. The latest master is copied locally for the merge, then published to the multiuser development directory after the merge. Other developers could have made changes to file before this merge.

During the merge, the Administration Tool checks for added objects and if found, a warning message is displayed. The following list describes what happens during this step:

- Warning about added objects. A developer who checks out a project has the ability to modify that project in any way and check it back in. Deletions and modifications are ways in which the integrity of the project is maintained. However, adding objects might introduce objects into the repository that don't belong to any project. Therefore, all project-related objects are checked and if a new object is found, a warning message is displayed.

  > **Important:**
  >
  > You must add newly created metadata to the project definition in the master repository for it to be visible in future extracted versions of the project. For example, if a developer checks out a project, adds a new object, and checks it in, then the new object isn't visible in extracted versions of the project until it's explicitly added to the project definition. See Create Projects for instructions.

- Aggregation of related objects. In the warning message, only the parent object is reported. The Administration Tool aggregates all the objects to make the message more usable. For example, if a developer added a new business model, only the business model name is included in the warning message to the user. The names of the tables, columns, and dimensions aren't displayed to the user.

When a developer publishes changes to the network, the following actions occur:

- The master repository in the multiuser development directory is overwritten with the repository that contains the developer's changes.
- The *master_repository.lck* file is deleted. If another developer checks out the changed project from the master repository or checks out the entire repository, then the changes made by the first developer are exposed to the other developer.

How Are Multiuser Merges Different from Standard Repository Merges?

The multiuser development publishing process uses the same technology as the standard repository merge process with a few important differences.

In multiuser development merges, you shouldn't want to retain changes to security settings and data sources, to prevent developers from overwriting passwords and other important objects in the master repository. Changes to security settings and data source connections

aren't retained when you perform a MUD merge. In addition, inserts (created objects) are applied automatically.

See Merge Rules and Behavior for Multiuser Development Merges.

Publish to the Network

When the publishing process begins, the Oracle BI Administration Tool automatically copies the current version of the master repository from the multiuser development directory to the local repository directory on the developer's computer.

The published location is similar to *ORACLE_INSTANCE*\bifoundation\OracleBIServerComponent\coreapplication_obis*n*\repository. The publishing process also updates the log files in the local and multiuser development directories. This is necessary because the master repository in the multiuser development directory might have changed after the developer checked out the projects, or since the last subset refresh.

A lack of conflicts doesn't mean that there are no differences between the repositories. Instead, it means that there are no decisions that have to be explicitly made by the developer to publish changes. See How Are Multiuser Merges Different from Standard Repository Merges?.

1. In the Administration Tool, from the **File** menu, select **Multiuser**, and then select **Publish to Network**.
2. Click **Yes** if prompted to save changes.
3. In the Lock Information dialog, in the **Comment** field, type a description of the changes that you made, then click **OK**.
4. If there are any conflicts, then the Merge Repository Wizard opens and the Define Merge Strategy screen is displayed. Make merge decisions about whether to include or exclude objects by selecting **Current** or **Modified** from the **Decision** list. When you select an object in the decision table, the read-only text box below the decision table describes what changes were made to that object in the current repository. You can also click **View Change Statistics** to see a summary of changes. Click **Finish** when you're finished making merge decisions.

   A CSV file is created in the local directory that contains details of the merged changes.
5. After you confirm all the changes, click **Save**.

   The master repository in the multiuser development directory is overwritten with the copy of the repository containing the developer's changes.

Enforce Consistent Repositories When Publishing Changes

You can enforce a mandatory consistency check during the Publish to Network step by setting the Mandatory Consistency Check option to Yes in the multiuser development options file.

See Set Multiuser Development Options.

- If consistency errors occur, do one of the following:
  - Click **Yes** to fix the consistency check errors immediately; the master repository remains locked.

- Click **No** to cancel the publishing step, fix the consistency check errors later, and unlock the master repository.
- In the Oracle BI Administration Tool, from the **File** menu, select **Multiuser**, and then select **Undo Publishing** to release the lock on the master, or when fixing the changes is more complex than you anticipated.

Branch in Multiuser Development

Branching is a further refinement of the merging development process.

Branching can provide higher efficiencies over large development teams that have overlapping releases, but it requires significant administrative overhead.

This section contains the following topics:

- About Branching
- Use the Multi-Team, Multi-Release Model
- Synchronize Branches

About Branching

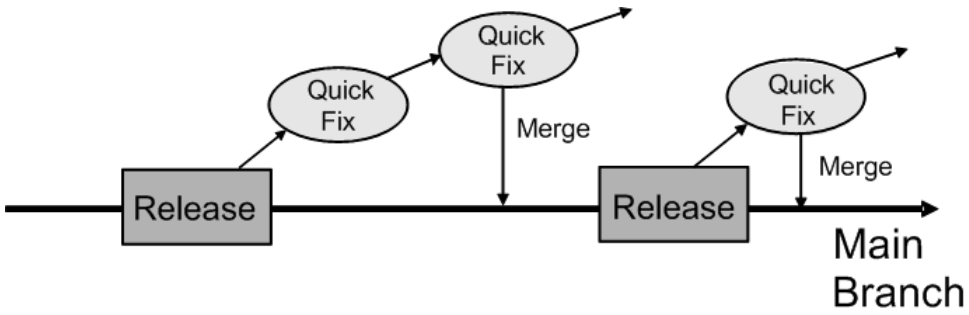
In branching, developers work on private branches to isolate their code from other developers and merge changes back to the main branch.

Different strategies can be followed, depending on the size of the development team.

In the Simple Development Model, all development occurs on a single main branch. This strategy has the following characteristics:

- Only for emergency fixes
- Checkouts might not be the most current code
- Carries a stability risk for the mainline branch

The figure shows the Simple Development Model.

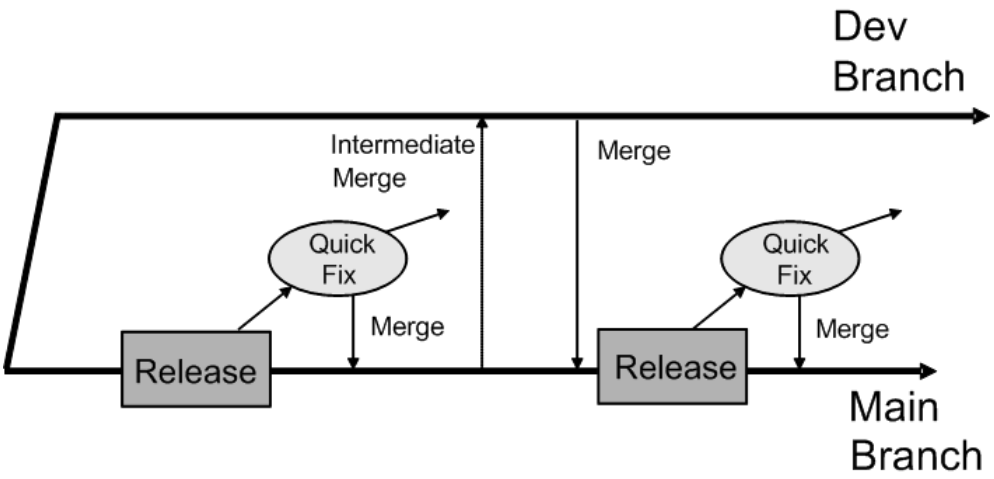


In the Small Team Development Model, development occurs on a single Dev branch, with a separate Main branch strictly for releases. This strategy has the following characteristics:

- The Mainline is the official release branch
- Development occurs on a separate branch

- Stable code is merged back to Main at key milestones
- Branches are synchronized periodically

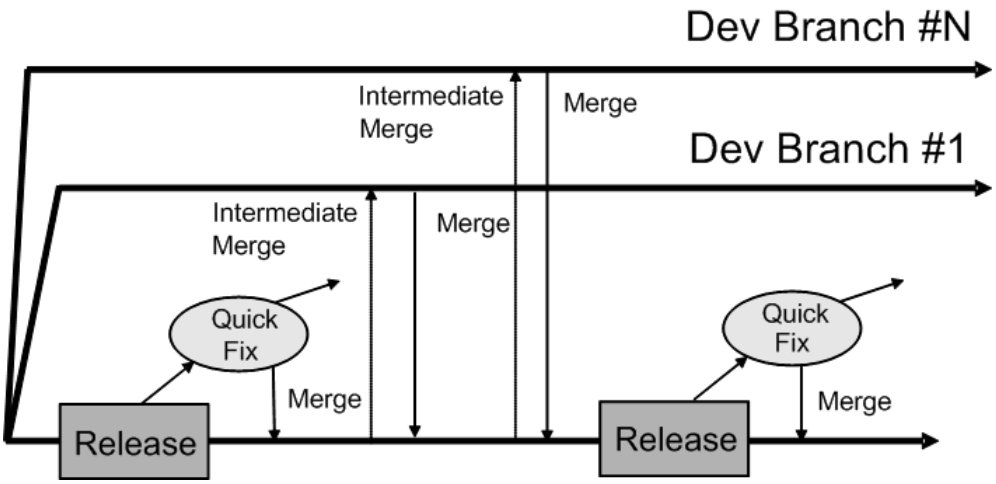
The figure shows the Small Team Development Model.



In the Multi-Team, Multi-Release Model, development occurs on multiple Dev branches, again with a separate Main branch strictly for releases. This strategy has the following characteristics:

- Supports more efficiency over disparate teams
- Development occurs on separate branches
- Stable code is merged back to Main at key milestones
- Branches are synchronized periodically

The figure shows the Multi-Team, Multi-Release Model.



Use the Multi-Team, Multi-Release Model

Using complex branching strategies requires attentive organization of repository files, as well as altering the Multiuser setting in the Oracle BI Administrative Tool.

The following provides an overview of the required steps.

1. Create a Main repository (Master Repository) and store it in the Master multiuser development directory.
   - Projects must be explicitly defined.
   - Branch developers shouldn't have access to the Master directory.
2. Create a subset of branch repositories by extracting from Main and storing them as the Team1 and Team2 multiuser development directories. The Main and Team RPDs must be stored and secured in separate directories on the network.
3. Developers must check out, develop, merge, and publish from their respective Team RPDs. Developers A1 through A3 and B1 through B3 should manage their metadata work and merge to their Team repository.
   - Teams 1 and 2 must maintain their own repositories and periodically synchronize from Main to Team branches.
   - The Team repositories must be merged back into and published in the Main repository.
4. One specific group, for example, release management, should manage all project definitions, perform merges, publish, and synchronize the Team RPDs back to Main.

Synchronize Branches

For large development teams, it's a good practice to perform periodic branch synchronization as Main changes, in order to ease the ultimate Team publishing step.

Use the Administration Tool to synchronize repositories in a three-way merge.

1. Publish all changes from your Team development branch and open the Oracle BI repository (RPD) in the Administration Tool. This the **current** repository.
2. Extract a fresh Branch subset from Main. This is the *modified* repository.
3. In the Administration Tool, select **File**, then select **Merge** and browse to the backup of the previous Branch subset. This is the original repository.
4. Resolve all issues and perform the merge.

   The repository named in the **Save merged repository as** field becomes the new branch development Oracle BI repository and is called the Original in future synchronizations.

View and Delete History for Multiuser Development

You can view and delete the development history of a multiuser development repository.

This section contains the following topics:

- View Multiuser Development History
- Delete Multiuser Development History

View Multiuser Development History

You can view the development history of a multiuser development repository.

In the Administration Tool, multiuser development history is only available when no repository is open and after the administrator sets up the shared network directory. This prevents the confusion that could occur if a user opened a history log that didn't match an open, unrelated repository.

1. Open the Administration Tool.
2. Without opening a repository, from the **File** menu, select **Multiuser** , and the select **History**.
3. In the Multiuser Development History dialog, select a repository.
4. In the Open Offline dialog, type the password for the repository.
5. In the Multi User History dialog, right-click a row and select an option. The table describes the options in the Multi User History dialog.

> **Tip:**
>
> To see details for all revisions, right-click in the background with no rows selected and select **View** , and select **Details**.

Delete Multiuser Development History

Only multiuser development administrators can delete history.

Administrators are defined in a special hidden option file in the multiuser development directory. See Set Multiuser Development Options.

You can delete the entire MUD history, or the oldest 1 to *n* versions. You can't delete versions in the middle of the range. For example, an administrator can't delete version 3 if there are still versions 1 and 2. If an administrator deletes the entire MUD history, newly assigned version numbers restart at version 1.

If an administrator deletes a MUD history version from which a developer has checked out a subset, and the developer is still working on it, the developer can't publish to the MUD directory. Deleting all MUD history prevents any developer who has currently checked out a subset from publishing it. Administrators should communicate with developers before the MUD history is cleared.

Set Multiuser Development Options

You can create a multiuser development option file to specify options for multiuser development. The option file is a text file, in standard Windows INI format.

The option file has the following properties and characteristics:

- The option file must be placed in the multiuser development directory. The file has the same name as the corresponding master repository, but with an .opt extension. For example, for `\\network\MUD\sales.rpd`, create an option file called `\\network\MUD\sales.opt`.

- The file should have the **Hidden** flag turned on.
- In general, the option file should be managed only by multiuser development administrators. To ensure this, you may want to change the sharing permissions for the file.

The following example shows a multiuser development option file:

```
[Options]
BuildNumber = Yes
Enforce Build Number = 11.1.1.7.0
Enforce MUD Protocol Version Number = 1
Prevent Rpd Upgrade = Yes
Admin = admin1;admin2
Mandatory Consistency Check = Yes
Equalize During Merge = Yes
```

Options that aren't explicitly set are turned off by default. To turn an option on, set its value to Yes. To turn an option off, either remove it from the option file, or set its value to No.

The table explains the options in the multiuser development option file.

Option	Description
BuildNumber	When set to Yes, the build version of the Administration Tool is displayed in the MUD history.
Enforce Build Number	When specified, ensures that only users with an exact match of the given version of the Administration Tool can perform MUD operations. Select the **Help** menu, then select **About** to view the Administration Tool version. Use this option in conjunction with Enforce MUD Protocol Version Number and Prevent Rpd Upgrade. Remove this line if you don't want to enforce Administration Tool version consistency for MUD operations.
Enforce MUD Protocol Version Number	When specified, ensures that only users with an exact match of the given MUD version can perform MUD operations. Select the **Help** menu, then select **About** to view the MUD version. Use this option in conjunction with Enforce Build Number and Prevent Rpd Upgrade. Remove this line if you don't want to enforce MUD version consistency for MUD operations.
Prevent Rpd Upgrade	When specified, ensures that only users with an exact match of the given repository version can perform MUD operations. Select the **Help** menu, then select **About** to view the repository version. Use this option in conjunction with Enforce Build Number and Enforce MUD Protocol Version Number. Remove this line if you don't want to enforce repository version consistency for MUD operations.
Admin	Lists multiuser development administrators. Administrators must be defined in the option file before they can delete MUD history. Administrators are defined by their `computer/network` login names. When multiple administrators exist, separate administrator names by semicolons, for example, *Admin=jsmith;mramirez;plafleur*

Option	Description
Mandatory Consistency Check	When set to *Yes*, the publish step performs a consistency check. Publishing can't proceed unless there aren't errors in the given repository.
Equalize During Merge	When set to Yes, the multiuser development merge process performs mandatory equalization during MUD merges. Setting this option to *Yes* affects the performance of the merge process.

4

Use a Source Control Management System for Repository Development

You can integrate the Administration Tool with third-party source control management systems for Oracle BI repository development.
The Administration Tool achieves this integration through the ability to save repository metadata as a set of XML documents in MDS XML format rather than as a single binary Oracle BI repository file (RPD). Using this integration, you can configure the Administration Tool to work with your own source control management system and save your repository output as MDS XML.

This chapter contains the following topics:

- About Using a Source Control Management System with the Administration Tool
- Set Up Your System for Repository Development Under Source Control Management
- Use Source Control Management in Day to Day Repository Development
- Use Source Control Management with MUD

About Using a Source Control Management System with the Administration Tool

You can integrate Administration Tool with a third-party source control management systems such as Subversion, Rational ClearCase, or Git during your repository development process.

In the Administration Tool:

- Convert your binary Oracle BI repository file to a set of MDS XML documents.

  You can save your repository in MDS XML format rather than using a single large binary repository file. In the MDS XML format, each repository object such as a connection pool, physical table, or business model is represented in its own XML file. You can manage the set of XML files that make up your repository in your source control management system.
- Set up a Source Control Management (SCM) configuration file.

  Use the SCM Configuration Editor in the Administration Tool to specify commands specific to your SCM system such as add file, delete, and check out and specify environment variables required by your SCM system.
- Designate your repository as under source control.

  The first time you open your MDS XML repository in the Administration Tool, you're prompted to specify whether the repository is a standalone MDS XML repository, or whether it's under source control. Choose **Use Source Control** to enable SCM integration for this repository in the Administration Tool.

About MDS XML

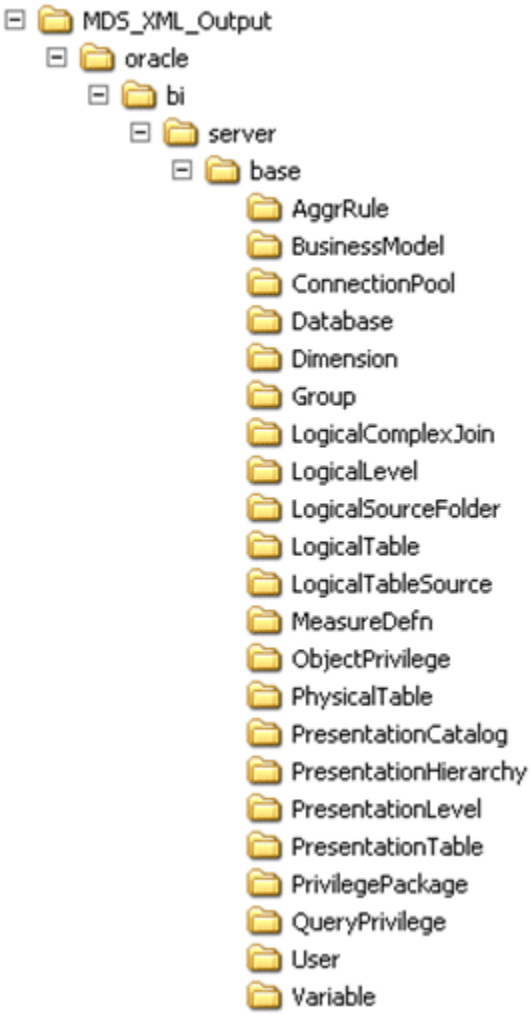
MDS XML format is used for repositories under source control.

MDS XML represents the Oracle BI Repository across a set of XML files rather than in a single file.

Each repository connection pool is stored in its own file, with an XML representation like the following:

```
<?xml version="1.0" encoding="UTF-8" ?>
<ConnectionPool mdsid="m80ca62c5-0bd5-0000-714b-e31d00000000"
name="SampleApp_Lite_Xml" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.oracle.com/obis/repository"
password="94F9321C85340FC48E4D9093AA941FF28844074B88D5AA6364E4815DEED7F9B
8792EF452219C2155DB68F61EE1555B4FA886F77E060E2E17F45AD8D18CAB2E4D3EFA15B75E
30D8B4BFA8C7B2D70552BD" timeout="4294967295" maxConnDiff="10" maxConn="10"
dataSource="VALUEOF(BI_EE_HOME)/sample/SampleAppFiles/Data" type="Default"
reqQualifedTableName="false" isSharedLogin="false"
isConcurrentQueriesInConnection="false" isCloseAfterEveryRequest="true"
xmlRefreshInterval="2147483647" outputType="xml" ignoreFirstLine="false"
bulkInsertBufferSize="0" transactionBoundary="0" xmlaUseSession="false"
multiThreaded="false" supportParams="false" isSiebelJDBSecured="false"
databaseRef="/oracle/bi/server/base/Database/Sample App Lite Data_80ca62c4
-0bcf-0000-714b-e31d00000000.xml#m80ca62c4-0bcf-0000-714b-e31d00000000"
>
<Description>
<![CDATA[
SampleAppLite connection pool to XML datasource. This connection pool points the
database to the location where physical XML files are stored. The location uses
the value of an RPD variable : BI_EE_HOME.
This variable needs to be correctly set in order for the server to connect to
the
files.
]]>
</Description>
</ConnectionPool>
```

The SampleAppLite repository generates MDS XML files in a structure like the following:



Note:

There is no one-to-one relationship between repository objects in the Administration Tool and the set of files produced as XML output. For example, physical columns appear as independent objects in the Administration Tool, but in MDS XML they're considered part of the Physical Table object.

See About the Oracle BI Server MDS XML API in *XML Schema Reference for Oracle Business Intelligence Enterprise Edition* for full information about the MDS XML schema representation of repository objects.

Set Up Your System for Repository Development Under Source Control Management

To set up your system for repository development under source control management, you must set up an SCM configuration file with commands specific to your SCM system, and generate an MDS XML repository and check it into your SCM system.

This section contains the following topics:

- Create an SCM Configuration File
- Create an MDS XML Repository and Check In Files to the SCM System

Create an SCM Configuration File

To integrate the Oracle BI Administration Tool with your source control management system (SCM), you must create an XML configuration file based on your specific SCM system.

The configuration file contains the SCM system commands for adding, deleting, checking out, and renaming files. The Administration Tool issues these commands to the SCM system when repository objects are created or updated, resulting in corresponding new or changed MDS XML files. The Administration Tool doesn't commit the changes to the SCM system. The repository developer must always check the files into the SCM system directly. The separate check-in in the SCM system facilitates viewing any conflicts or implementing merge decisions in the SCM environment rather than the Administration Tool environment.

If you create or edit an SCM configuration file while an MDS XML repository is open, you must ensure that **Use Source Control** is selected to enable the **New** or **Edit** buttons.

The default location for SCM configuration files is `Oracle_Home/user_projects/domains/bi/config/fmwconfig/biconfig/OBIS`.

You shouldn't store security-sensitive environment variables in the configuration file. If security-sensitive variables are required by your SCM system, to avoid the security risk, you can launch the Administration Tool from Windows Command Prompt with any security-sensitive variables already set.

1. Open the Administration Tool and select **Tools**, then select **Options**.
2. Select the Source Control tab.
3. Click **New** to create a new configuration file. The Specify new configuration file window is displayed.
4. Provide a file name using the XML file extension, and click **Save**.
5. Click **Load** in the SCM Configuration Editor.
6. Select a template file, and click **Open**.
7. In the SCM Configuration Editor, provide an optional description, then enter or edit commands for your system in the Commands tab.

   For longer commands, click the ellipsis button to enter commands in the Command Editor window.

Use the ${file}, ${filelist}, ${from}, and ${to} tokens to define the commands. You can also use the **List File** option in conjunction with the ${filelist} command to set the behavior. The tokens can be used as follows:

- `${file}` specifies that a command must be run sequentially, one file at a time. ${file} is required for the Add Folder and Add File commands.
- The behavior of `${filelist}` varies, depending on whether **List File** is selected:
  - `${filelist}` without **List File** selected causes the Administration Tool to group as many files as possible for the given command such as Pre-Delete, Delete, or Check-out, staying under the 32k character limit for launching a process. Processing is repeated until all files have been processed.
  - `${filelist}` with **List File** selected instructs the Administration Tool to create a temporary file that holds the file list. The file list format is one file name for each line, which is a typical format among SCM vendors. List File is valid for `Pre-Delete`, `Delete` or `Check-out` commands. It results in much faster operations and should always be selected for SCM systems that support it.

  You can use`${file}` or `${filelist}` for Pre-Delete, Delete, and Checkout. **List File** only works in conjunction with `${filelist}`.
- `${from}` and `${to}` are used to specify the original file name and new file name in Rename commands.

  Not all SCM systems support file rename operations natively. If this is the case, leave the **Rename** field blank rather than attempting to construct a rename operation by concatenating different commands. The Administration Tool performs the rename operation.

> **Note:**
>
> Some SCM systems don't include commands for working with folders. If this is the case, leave **Add Folder** blank. The Administration Tool always creates folders for you when needed.
>
> Even if your SCM system does include folder management commands, the Administration Tool doesn't remove folders. You must remove folders directly in the SCM system if necessary.

8. Select the Environment Variables tab, and then specify environment variables required by your SCM system.
9. Click **Test** in the Environment Variables tab to open the Test SCM Configuration window. Then, enter a command and click **Execute** to test a particular command. If the environment is correct, the correct output should appear after running the command.
10. Select the Post-save comment tab to enter text that appears after changes are saved in the Administration Tool. You can use the Post-save comment to remind developers to check-in there changes.
11. Click **OK** to save the configuration file, or click **Save As** to save a copy if you loaded and modified a template configuration file.

Create an MDS XML Repository and Check In Files to the SCM System

To integrate with an SCM system, you must convert your Oracle BI repository to MDS XML format.

Use one of the following options to create an MDS XML repository and check it into your source control system:

- Save an Existing Repository File in MDS XML Format
- Create a New Repository in MDS XML Format
- Link to Source Control Files to Convert Your Repository (Small Repositories Only)

Save an Existing Repository File in MDS XML Format

If you've an existing repository file, use these steps for initial import to convert it to MDS XML.

1. Open your existing Oracle BI repository file (RPD) in the Administration Tool in offline mode.
2. Select **File**, then select **Save As**, then select **MDS XML Documents**.
3. Select a root location for your MDS XML repository files, and then click **OK**.
4. Perform the steps in your source control management system to add and check in the files.

   Use the specialized commands for bulk file import, available for most SCM systems. These commands are optimized to deliver entire trees of files to source control in a very efficient way. For example, in Subversion, use the following command:

   ```
   svn import module_name -m "Initial import"
   ```

You can also use the `biserverxmlgen` utility with the `-M` and `-D` options to generate MDS XML from an existing Oracle BI repository. See Generating MDS XML from an Existing RPD Using a Command-Line Utility in the *XML Schema Reference for Oracle Business Intelligence Enterprise Edition*.

Create a New Repository in MDS XML Format

Use these steps to create a new repository in MDS XML format.

1. Open the Administration Tool and select **File**, then select **New Repository** to open the Create New Repository Wizard.
2. Select the **MDS XML Documents** option in the wizard. Complete the other wizard steps.
3. Perform the necessary steps in your source control management system to add and check in the files. For large repositories, use the specialized commands for bulk file import for your SCM system.

Don't create a new MDS XML-format repository, add objects, and then select **Link to Source Control**. This method doesn't work, and SCM commands aren't generated.

Link to Source Control Files to Convert Your Repository (Small Repositories Only)

For very small repositories, you can use the Link to Source Control files method to convert a binary Oracle BI repository file to MDS XML format.

Using the Link to Source Control Files method to initially import your repository is only recommended for very small repositories. This method is too slow for large repositories, tens of thousands of files, because the Administration Tool imports the files one at a time using the standard `add file` command, rather than using specialized commands for bulk file import.

The repeated invocation of the `add file` command might increase the chances of transient errors. If these occur, you might need to restart the process a few times before all files are successfully imported to source control.

See Create an SCM Configuration File.

1. Ensure that you've an SCM configuration file defined.
2. Create an empty root folder for the MDS XML repository.
3. Open your existing Oracle BI repository file in the Administration Tool in offline mode.
4. Select **File**, then select **Source Control**, then select **Link to Source Control Files**.
5. Select the root folder you created, and the appropriate SCM configuration file.

   If you need to change the configuration file later, select **Tools**, select **Options**, select **Source Control**, and then click **Edit** to change the configuration file.
6. Click **Save**. An MDS XML repository is created, and the necessary add file operations are performed in your source control system.
7. Commit the changes in your SCM system.

Use Source Control Management in Day to Day Repository Development

These topics describes typical scenarios that occur during day to day repository development.

This section contains the following topics:

- Update, Save, and Check In Changes for Repositories Under Source Control
- Handle Errors
- Test Repositories Under Source Control
- View the Source Control Log

Update, Save, and Check In Changes for Repositories Under Source Control

After your MDS XML repository is set up under source control, follow these steps to update, save, and check in changes to your repository.

1. Ensure that you've a local copy of your working MDS XML repository files that are under source control by issuing the appropriate commands in your SCM system. For example,

for Subversion, you can issue the command `svn info` as shown in the following example text:

```
C:\myProj\repos>svn info
Path: .
Working Copy Root Path: C:\myProj\repos
URL: file:///C:/SVN/myProj/trunk/sample1
Repository Root: file:///C:/SVN/myProj
Repository UUID: 6b995c92-3ec0-fa4b-9d58-c98e54f41792
Revision: 3
Node Kind: directory
Schedule: normal
Last Changed Author: joe_user
Last Changed Rev: 2
Last Changed Date: 2011-11-19 15:20:42 -0600 (Sat, 19 Nov 2011)
```

2. In the Administration Tool , from the **File** menu, select **Open**, then select **MDS XML**.

   Open the file in offline mode.

3. Select the root folder location for your MDS XML files and click **OK**.
4. If this is the first time you've opened this MDS XML repository in the Administration Tool, you're prompted to specify whether this repository is a standalone MDS XML repository, or whether it's under source control. Select **Use Source Control** and click **OK**.

   This choice is saved for this repository. To view the status of this repository at any time, select **Tools**, then select **Options**, then select the Source Control tab.
5. After you make changes to your repository, select **File**, then select **Save**, or click **Save** on the toolbar. The Administration Tool displays a list of changes.
6. Click **Yes** to run the commands in the SCM system.

   After accepting the changes, you can't cancel. Canceling the changes creates an inconsistent repository. You must continue the SCM commands process.

   When the Administration Tool issues the SCM commands, the commands rearranged into the most optimal order.
7. Check in the changes directly in your SCM system.

Handle Errors

Learn how to handle errors in the SCM system.

Sometimes errors, such as an expired label or network problem, occur when the Administration Tool delivers changes to the SCM system.

Saving to a binary Oracle BI repository file is the simplest option for transient problems like network errors, where you just need to try again later. Saving as MDS XML is required when some sort of work is required to fix the problem, such as merging conflicting changes.

1. In the Administration Tool, select **File**, then select **Save As** to save the repository to a temporary location in Oracle BI repository file format or MDS XML format. Close the Administration Tool.
2. Take action to resolve the issue. For example, refresh an expired label or test and view a failed network connection.

In the case of an expired label, you also need to merge the contents of the refreshed label with the temporary saved MDS XML repository. Use a third-party merge tool to do this.

For detailed information about the MDS XML representation of repository objects so that you can successfully make merge decisions, see *XML Schema Reference for Oracle Business Intelligence Enterprise Edition*.

3. Open the saved repository in the Administration Tool.
4. Select **File**, then select **Source Control**, then select **Link to Source Control**.
5. Click **Save** to save changes from the saved repository into the MDS XML files under source control.

Steps 4 and 5 of this procedure cause the Administration Tool to keep memory objects loaded from the saved Oracle BI repository file or MDS XML files, but to then consider them to belong to the source control MDS XML repository instead. When you click **Save**, the Administration Tool saves the memory objects to the source control repository.

Test Repositories Under Source Control

During the course of repository development, you need to perform testing in online mode to validate your repository.

You can only load an Oracle BI repository in RPD format into the Oracle BI Server to make it available for queries. Because of this, you must save your development MDS XML repository in Oracle BI repository format from time to time when you want to perform online testing.

See Make the Repository Available for Queries.

- In the Oracle BI Administration Tool, open your MDS XML repository in offline mode, select **Save As**, then select **Repository** to make the repository available for queries.

View the Source Control Log

The Source Control Log window shows the commands that the Administration Tool issues to your SCM system.

It also shows any post-save text you specified in the Post-save comment tab of the SCM Configuration Editor.

By default, the Source Control Log window appears when SCM commands are being run. Alternatively, you can select **File**, then select **Source Control**, then select **View Logs** to see the Source Control Log window.

You can choose the following options for this dialog:

- **Close when commands finish:** Causes the log window to close automatically when commands are complete, unless errors occur.
- **Only show dialog when errors occur:** Hides the window during SCM command processing unless errors occur. By default, the Source Control Log appears automatically when SCM commands are being run unless this option is selected.

The text displayed in the Source Control Log is persistent until you close the repository. This means that all SCM command output is available for view, regardless of whether the dialog is open during individual operations. While SCM commands are being run, the **Close** button is disabled until the SCM commands have finished or have stopped with an error, unless **Only show dialog when errors occur** has been selected.

The Source Control Log does have a 32K character limit. When the window buffer becomes full, then the oldest commands are removed from the Source Control Log display to make room for the latest command output. To see the full output, go to the Administration Tool log at:

ORACLE_INSTANCE/diagnostics/logs/OracleBIServerComponent/coreapplication_obisn/*user_name*_NQSAdminTool.log

Use Source Control Management with MUD

You can use source control management with your multiuser development environment.

For example, if you've an existing repository under multiuser development and you want to begin using source control management, you might follow the steps described in the following subsections:

- Put the MUD Master Repository and MUD Log File Under Source Control
- Check In New Versions of the MUD Master and MUD Log File to Source Control

Put the MUD Master Repository and MUD Log File Under Source Control

Use this procedure to put the MUD master repository and MUD log file under source control.

Run the `mhlconverter` command-line utility to convert your MUD log file (*.mhl) to an XML file.

See Save an Existing Repository File in MDS XML Format to convert your master MUD Oracle BI repository file to a set of MDS XML files on the file system.

1. At the command prompt, type `mhlconverter` with the input MHL file name and path, and the output XML file name and path. For example:

   ```
   mhlconverter -I C:\MUD\mud_repository.mhl -O C:\MUD\mud_repository.xml
   ```

2. Check the MDS XML files and XML-format MUD log file into your SCM system.

Check In New Versions of the MUD Master and MUD Log File to Source Control

After creating and checking in the initial version of the master MUD repository, you need to check in updated versions of the MUD master repository on an ongoing basis.

This section describes two different ways to perform this task.

- Manually Check In the Updated MUD Master Repository and Log File
- Use a Script to Check In the Updated MUD Master Repository and Log File

Manually Check In the Updated MUD Master Repository and Log File

Use these steps to manually check in changes to the master Oracle BI repository and log file that have occurred as part of the multiuser development process.

Consider using the automated check-in method described in Use a Script to Check In the Updated MUD Master Repository and Log File if you've a large repository. See Create an SCM Configuration File.

1. Open the latest copy of the master Oracle BI repository file in the Administration Tool.
2. Create or select the appropriate SCM configuration file.
3. Select **File**, then select **Source Control**, then select **Link to Source Control**. Select the directory that contains the MDS XML version of the master MUD repository.

   Using **Link to Source Control** isn't recommended for large repositories and might cause time-outs.
4. Click **Save** to save changes from the master MUD repository into the MDS XML files under source control. The Administration Tool determines which files to add, check out, modify, and delete and issues the commands to your SCM system.
5. Close the Administration Tool.
6. Follow these steps to update the MUD log file:
   a. In your SCM system, check out the XML-format MUD log file.
   b. Use the `mhlconverter` utility to overwrite the XML-format MUD log file with the latest changes from the `.mhl` version.
   c. Check in the latest XML-format MUD log file to your SCM system.
7. Check all changes into your SCM system.

It's recommended that you perform the steps in this section regularly to avoid having too many changes in a single transaction.

Use a Script to Check In the Updated MUD Master Repository and Log File

As an alternative to manually checking in changes, you can create a script to perform the check-in tasks and then schedule it to run at regular intervals.

See Compare Repositories Using comparerpd.

1. Identify the latest copy of the master Oracle BI repository file that you want to check into your SCM system.
2. Identify the last version of the master Oracle BI repository file that was checked into the SCM system. You can review the latest XML-format MUD log file under source control to determine this version.

   If you don't have the last checked-in version of the master repository in Oracle BI repository file format, you can use the `biserverxmlexec` utility with the -D option to read the latest MDS XML files checked into source control and re-create an Oracle BI repository file version.
3. Use the `comparerpd` utility with the `-M` option to compare the latest copy of the master Oracle BI repository file, the modified version, with the version that was last checked in, the original version. An MDS XML format diff is generated.

4. Create a script that does the following:
   a. Reads the MDS XML diff directory to identify which files are present.
   b. Issues commands in source control to check out the identified files or add new files.
   c. Copies the latest version of the files from the MDS XML diff directory to the source control directory.
   d. Reads the `oracle\bi\server\base\DeletedFiles.txt` file inside the MDS XML diff directory to determine which files to delete.
   e. Issues commands in source control to delete the appropriate files.
   f. Checks out the MDS XML-format MUD log file, runs the `mhlconverter` utility to convert the latest MHL-format log file to XML format, overwrites the existing MDS XML-format MUD log file with the new one, and checks it in.
   g. Performs all necessary check-in steps in the SCM system.

5

Import Metadata and Working with Data Sources

This chapter describes how to create a new Oracle BI repository, set up back-end data sources, and import metadata using the Import Metadata Wizard in the Administration Tool. It also describes how to use a standby database with Oracle Analytics Server.
This chapter contains the following topics:

- About Importing Metadata and Working with Data Sources
- Create an Oracle BI Repository
- Perform Data Source Preconfiguration Tasks
- Import Metadata from Relational Data Sources
- Import Metadata from Multidimensional Data Sources
- About Importing Metadata from XML Data Sources
- About Using a Standby Database

About Importing Metadata and Working with Data Sources

After creating an Oracle BI Repository file, you can import metadata from your data sources into the Physical layer of the repository.

In the Oracle BI Administration Tool, the Physical layer of the contains the data sources the Oracle BI Server uses to submit queries, and the relationships between physical databases and other data sources used to process multiple data source queries.

The metadata imported into an Oracle BI Repository must have an ODBC or native database connection to the underlying data source. You can also import metadata from software such as Microsoft Excel using an ODBC connection.

When you importing metadata from each data source, the structure of the data source is also imported into the Physical layer. You can display data from supported data sources on Oracle BI Server and other clients. You can't import metadata from unsupported data sources.

After you import metadata, properties in the associated database object and connection pool are set automatically. You can adjust database or connection pool settings, see Set Up Database Objects and Connection Pools.

Oracle recommends importing metadata rather than manually creating the physical layer to avoid errors.

Create an Oracle BI Repository

You can use the Create New Repository Wizard in the Administration Tool to create a new Oracle BI repository in either binary Oracle BI repository file (RPD) or MDS XML format.

If you've a repository, you can use the existing data source settings as a template to connect to different data sources. To use the existing data source settings and change the database type and connection pool information, see Set Up Database Objects and Create or Change Connection Pools.

1. In the Administration Tool, select **File**, then select **New Repository**.

   If an existing repository is open, you're prompted to save your changes, and the existing repository is closed.

2. Select **Binary** to create an Oracle BI repository file. To create a repository in MDS XML format, select **MDS XML Documents**.

3. For binary repositories, type a name for the repository. Keep the name to 156 characters or less to avoid problems with the metadata dictionary URL. An Oracle BI repository RPD file extension is automatically added if you don't explicitly specify it.

4. For **Location**, follow the steps appropriate for your repository type:

   - For binary repositories, select a location for the Oracle BI repository file. By default, new binary repositories are stored in the repository subdirectory, located at *ORACLE_INSTANCE*\bi\bifoundation\server.
   - For MDS XML format repositories, select a root folder location for the set of MDS XML files.

5. If you want to import metadata into the repository now, select **Yes** (the default) for **Import Metadata**. If you don't want to import metadata, select **No**.

6. Enter and confirm the password you want to use for this repository. The repository password must be eight characters, containing one or more numerals.

   You enter the same repository password that you used to open the repository in online or offline mode. Your password is used to encrypt the repository contents.

7. If you selected **Yes** for **Import Metadata**, click **Next**.

   You might need to set up your data sources before you importing metadata.

8. If you selected **No** for Import Metadata, click **Finish** to create an empty repository.

Perform Data Source Preconfiguration Tasks

You might need to perform configuration steps to access the data sources.

These configuration steps are sometimes required before you can import physical objects from your data sources into your repository file, or set up connection pools to your data sources.

For many data sources, you need to install client components. Client components are often installed on the computer hosting the Oracle BI Server for query access, and on the computer hosting the Administration Tool (if different) for offline operations such as

import. In some cases, you must install client components on the computer where the JavaHost process is located.

This section contains the following topics:

- Set Up ODBC Data Source Names (DSNs)
- Set Up Oracle Database Data Sources
- About Setting Up Oracle OLAP Data Sources
- Java Data Sources
- About Setting Up Oracle TimesTen In-Memory Database Data Sources
- About Setting Up Essbase Data Sources
- About Setting up Cloudera Impala Data Sources
- About Setting Up Apache Hive Data Sources
- About Setting Up Hyperion Financial Management Data Sources
- Set Up Oracle RPAS Data Sources
- Set Up Teradata Data Sources
- Enable NUMERIC Data Type Support for Oracle Database and TimesTen
- Configure SSO for Essbase, Hyperion Financial Management, or Hyperion Planning Data Sources

Set Up ODBC Data Source Names (DSNs)

Before you can import from a data source through an ODBC connection, or set up a connection pool to an ODBC data source, you must first create an ODBC Data Source Name (DSN) for that data source on the client computer.

You reference the DSN in the Import Metadata Wizard when you import metadata from the data source.

You can only use ODBC DSNs for import on Windows systems.

1. In Windows, locate and open the ODBC Data Source Administrator. The ODBC Data Source Administrator dialog appears.
2. In the ODBC Data Source Administrator dialog, click the System DSN tab, and then click **Add**.
3. From the Create New Data Source dialog, select the driver appropriate for your data source, and then click **Finish**.

   The remaining configuration steps are specific to the data source you want to configure. Refer to the documentation for your data source for more information.

ODBC DSNs on Windows systems are used for both initial import, and for access to the data source during query processing. On Linux systems, ODBC DSNs are only used for data access. See Set Up Data Sources on Linux.

See Set Up Teradata Data Sources.

Set Up Oracle Database Data Sources

When you import metadata from an Oracle Database data source or set up a connection pool, you can include the entire connect string for Data Source Name, or you can use the net service name defined in the tnsnames.ora file.

If you choose to enter only the net service name, you must set up a `tnsnames.ora` file in the following location within the Oracle Analytics Server environment, so that the Oracle BI Server can locate the entry:

BI_DOMAIN`/bidata/components/core/serviceinstances/ssi/oracledb`

You should always use the Oracle Call Interface (OCI) when importing metadata from or connecting to an Oracle Database. Before you can import schemas or set up a connection pool, you must add a TNS names entry to your `tnsnames.ora` file. See the Oracle Database documentation for more information.

This section contains the following topics:

- Oracle 12*c* Database In-Memory Data Sources
- Oracle 12*c* on Exadata Data Sources
- Advanced Oracle Database Features Supported by Oracle BI Server
- Oracle Database Fast Application Notification and Fast Connection Failover
- Additional Oracle Database Configuration for Client Installations
- Configure Oracle BI Server When Using a Firewall

See Enable NUMERIC Data Type Support for Oracle Database and TimesTen.

Oracle 12*c* Database In-Memory Data Sources

For all Oracle 12*c* Database In-Memory data sources, the Oracle BI Server creates tables in memory.

Oracle 12*c* Database In-Memory is a high-performance in-memory data manager. It uses In-Memory Column Store to store copies of tables and partitions in a special columnar format that exists in memory and provides for rapid scans. See the 12*c* Release 1 *Oracle Database Concepts Guide* and *Oracle Database Administrator's Guide* for more information.

Oracle 12*c* on Exadata Data Sources

For Oracle 12c Database on Exadata and Oracle 12c Database In-Memory on Exadata data sources, the Oracle BI Server creates tables in memory.

Oracle BI Server uses Exadata Hybrid Columnar Compression (EHCC) by default.

Oracle Exadata Database Machine is the optimal platform for running Oracle Database. Both Oracle 12*c* Database and Oracle 12*c* Database In-Memory run on the Oracle Exadata Database Machine. See the documentation included with the Exadata Database Machine for more information.

Advanced Oracle Database Features Supported by Oracle BI Server

The Oracle BI Server supports the compression, Exadata Hybrid Columnar Compression, and In-Memory features to take advantage of native Oracle Database functionality and significantly improve query time.

When you import metadata or specify a database type, the feature set for that database object is automatically populated with default values appropriate for the database type. The Oracle BI Server uses the SQL features with this data source. When a feature is marked as supported (checked) in the Features tab of the Database dialog, the Oracle BI Server pushes the function or calculation to the data source for improved performance. When a function or feature isn't supported in the data source, the calculation or processing is performed in the Oracle BI Server.

The following is information about Oracle Database features supported by Oracle BI Server:

- Compression

  Compression reduces the size of the database. Because compressed data is stored in fewer pages, queries need to read fewer pages from the disk, thereby improving the performance of I/O intensive workloads. Compression is used by default. If you create aggregates on your Oracle databases, then compression is applied to the aggregate tables by default.

  When you create a database object for any of the Oracle databases, the **COMPRESSION_SUPPORTED** feature is automatically applied to the object.

- Exadata Hybrid Columnar Compression (EHCC)

  Oracle's EHCC is optimized to use both database and storage capabilities on Exadata and enables the highest level of data compression to provide significant performance improvements. By default, Oracle 11*g* Database on Exadata, Oracle 12*c* Database on Exadata, and Oracle 12*c* Database In-Memory on Exadata use this type of compression.

  When you create a database object for any of the Oracle databases, the **EHCC_SUPPORTED** feature is automatically applied to the object.

  By default, compression is disabled for objects in the Oracle databases. To enable compression for an object, set the object's `PERF_PREFER_COMPRESSION` flag to on.

- In-Memory

  – In memory retrieval eliminates seek time when querying the data, which provides faster and more predictable performance than disk. The in memory feature creates tables in memory for Oracle 12*c* Database In-Memory and Oracle 12*c* Database In-Memory on Exadata. If you create aggregates on these databases, then the aggregates are created in memory.

  When you create a database object for any of the above mentioned Oracle databases, the **INMEMORY_SUPPORTED** feature is automatically applied to the object.

Oracle Database Fast Application Notification and Fast Connection Failover

If Fast Application Notification (FAN) events and Fast Connection Failover (FCF) are enabled on the Oracle Database, the Oracle Call Interface (OCI) uses the FAN events and enables FCF for the Oracle Database data sources.

Fast Application Notification (FAN) events and Fast Connection Failover (FCF) run in the background. When a query initiated by a user fails due to the unavailability of an Oracle

database, the query fails quickly and the user can then retry the query rather than wait for the database request to time out.

Additional Oracle Database Configuration for Client Installations

You must install the Oracle Database Client on the computer where you performed the client installation.

After installing the Oracle Database Client, create an environment variable called *ORACLE_HOME* and set it to the Oracle home for the Oracle Database Client. Create an environment variable called *TNS_ADMIN*, and set the variable to the `tnsnames.ora` file location of *`BI_DOMAIN`*`\config\fmwconfig\bienv\core`.

Configure Oracle BI Server When Using a Firewall

The presence of a firewall between the Oracle BI Server and the Oracle Database can result in very long query times.

You could experience long query times when using a simple `nqcmd` query that could take two to three minutes to return results, or when using Answers, you don't get a response after running or validating a SQL statement initiated in Presentation Services.

To improve query time, go to the `sqlnet.ora` file in *`BI_DOMAIN`*`\config\fmwconfig\bienv\core` and add the `BREAK_POLL_SKIP` and `DISABLE_OOB` parameters as follows:

```
BREAK_POLL_SKIP=10000
DISABLE_OOB=ON
```

You perform this configuration change only on the Oracle BI Server. You don't need to change configuration on the Oracle Database or on user client desktops.

DataDirect Drivers and Oracle Database

You must use ODBC DataDirect drivers to establish connections to ODBC data sources.

ODBC DataDirect drivers are also used by the Oracle Platform Security Services (OPSS) security store implementation to access credentials.

DataDirect ODBC framework, version 8.0.2, and Oracle Wire Protocol, version 8.0.0, support Oracle Database 12*c* connectivity, and are configured for data source name (DSN) and DNS-less connectivity without additional configuration.

The certified Oracle Database versions include:

- 12.2.1.2 or higher
- 11.2.0.4 or higher

You can find additional information about the DataDirect drivers in the Progress DataDirect documentation located in the following installation directories:

- *`mwhome`*`\bi\common\ODBC\Merant\7.1.6\help`
- *`mwhome`*`\bi\common\ODBC\Merant\8.0.0\help`
- *`mwhome`*`\bi\common\ODBC\Merant\8.0.2\help`

About Setting Up Oracle OLAP Data Sources

Before you import from an Oracle OLAP data source, ensure that the data source is a standard form Analytic Workspace.

You must install the Oracle Database Client on the computer where you performed the client installation before you can import from Oracle OLAP sources.

The `biadminservlet` Java process must be running to import from Oracle OLAP data sources, for both offline and online imports. You can use the Deployments option in Weblogic Console or Fusion Middleware Control to check the status of the `biadminservlet` Java process.

Use either the Administrator or Runtime client install option.

After installing the Oracle Database Client, create an environment variable called *ORACLE_HOME*, and set the variable to the Oracle home for the Oracle Database Client. Create an environment variable called *TNS_ADMIN*, and set the variable to the location of the `tnsnames.ora` file located in *`BI_DOMAIN`*`bidata/components/core/serviceinstances/ssi/oracledb`.

Java Data Sources

If you use the JDBC connection type, then the remote Java data sources must connect to Weblogic Server.

If you aren't using JDBC (Direct Driver) this configuration isn't required.

Before you can include JDBC and JNDI data sources in the repository, you must perform the required set up tasks.

You must configure JDBC in the Oracle WebLogic Server. For information about how to perform this configuration, see Using JDBC Drivers with WebLogic Server in the Oracle WebLogic Server documentation.

You must load data sources for importing into the repository. See Load Java Data Sources.

Load Java Data Sources

To make Java data sources available for import into the repository, you must first connect to the Java Datasource server to load the Java metadata.

1. In the Oracle BI Administration Tool, select **File**, and select **Load Java Datasources**.
2. In the Connect to Java Datasource Server dialog, enter the enter hostname, port, and credentials to access the server and load the Java metadata.
3. Click **OK**.

The Java metadata has been loaded from the server and is now available for import into the repository.

About Setting Up Oracle TimesTen In-Memory Database Data Sources

Oracle TimesTen In-Memory Database is a high-performance, in-memory data manager.

These preconfiguration instructions assume that you've already installed Oracle TimesTen, see Oracle Data Integrator for more information.

If you plan to create aggregates on your TimesTen source, you must also ensure that PL/SQL is enabled for the instance, and that the PL/SQL first connection attribute PLSQL is set to 1. You can enable PL/SQL at install time, or run the `ttmodinstall` utility to enable it post-install. See TimesTen In-Memory Database Reference for more information.

This section contains the following topics:

- Configure TimesTen Data Sources
- Improve Use of System Memory Resources with TimesTen Data Sources
- Configure Oracle BI Server to Access the TimesTen DLL on Windows

See Enable NUMERIC Data Type Support for Oracle Database and TimesTen.

Configure TimesTen Data Sources

You must configure TimesTen before you can use it as a data source.

1. On the computer where TimesTen has been installed, create a Data Manager DSN, as a system DSN.
2. Perform an initial connection to the data store to load the TimesTen database into memory, and then create users and grant privileges. The default user of the data store is the instance administrator, or in other words, the operating system user who installed the database.
3. On the computer running the Oracle BI Server, install the TimesTen Client.
4. On the computer where the TimesTen Client has been installed, create a Client DSN, as a system DSN.

If the TimesTen database is installed on the same computer as the TimesTen client, you can specify either the Data Manager DSN or the Client DSN in the Import Metadata Wizard.

After importing data from your TimesTen source, or when manually setting up a database object and connection pool, ensure that your database type and version are set correctly in the **Database** field of the General tab of the Database dialog. You must also ensure that the **Call interface** field in the General tab of the Connection Pool dialog is set correctly. See:

- Create a Database Object Manually in the Physical Layer
- Set Connection Pool Properties in the General Tab
- Oracle Exalytics In-Memory Machine for specific instructions on setting up TimesTen sources on the Oracle Exalytics Machine

Improve Use of System Memory Resources with TimesTen Data Sources

To improve the use of system memory resources, Oracle recommends that you increase the maximum number of connections for the TimesTen server.

To avoid lock timeouts, you might also want to adjust the LockWait interval for the connection as appropriate for your deployment. See `LockWait` in TimesTen In-Memory Database Reference Guide for more information.

1. In your TimesTen environment, open the `ttendaemon.options` file for editing. You can find this file at:

   install_dir\srv\info

2. Add the following line:

   -MaxConnsPerServer *number_of_connections*

   To determine *number_of_connections*, use the following formula: if there are M connections for each connection pool in the Oracle BI repository, N connection pools in the Oracle BI repository, and P Oracle BI Servers, then the total number of connections required is M * N * P.

3. Save and close the file.

4. In the ODBC DSN you're using to connect to the TimesTen server, set the Connections parameter to the same value you entered in Step 2:

   - On Windows, open the TimesTen ODBC Setup wizard from the Windows ODBC Data Source Administrator. The Connections parameter is located in the First Connection tab.
   - On Linux, open the `odbc.INI` file and add the *Connections* attribute to the TimesTen DSN entry, as follows:

     Connections=*number_of_connections*

5. Stop all processes connecting to TimesTen, such as the `ttisql` process and the Oracle BI Server.

6. Stop the TimesTen process.

7. After you've verified that the TimesTen process has been stopped, restart the TimesTen process.

Configure Oracle BI Server to Access the TimesTen DLL on Windows

If the user that starts Oracle BI Server doesn't have the path to the TimesTen DLL ($TIMESTEN_HOME\lib) in their operating system PATH variable, then you must add the TimesTen DLL path as a variable in the `obis.properties` file.

1. Open `obis.properties` for editing. You can find `obis.properties` at:

   BI_DOMAIN\config\fmwconfig\bienv\obis

2. Add the required TimesTen variable `TIMESTEN_DLL`, and also update the `LD_LIBRARY_PATH` variable, as shown in the following example.

   ```
   TIMESTEN_DLL=$TIMESTEN_HOME\lib\libttclient.so
   LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$TIMESTEN_HOME\lib
   ```

3. Save and close the file.

4. Restart OBIS1.

5. Repeat these steps on each computer that runs the Oracle BI Server process. If you're running multiple Oracle BI Server instances on the same computer, be sure to update the `ias-component` tag appropriately for each instance in `obis.properties`, for example, `ias-component id="coreapplication_obis1”`, and `ias-component id="coreapplication_obis2"`.

About Setting Up Essbase Data Sources

The Oracle BI Server uses the Essbase client libraries to connect to Essbase data sources.

The Essbase client libraries are installed by default. No additional configuration is required to enable Essbase data source access.

See Configure SSO for Essbase, Hyperion Financial Management, or Hyperion Planning Data Sources for configuration used for authentication using a shared token against Essbase installed with the EPM System Installer.

About Setting up Cloudera Impala Data Sources

These topics provide information about Windows ODBC drivers and Cloudera Impala Metadata.

Use the information in this section to set up Cloudera Impala data sources in the Oracle BI repository.

- Obtain Windows ODBC Driver for Cloudera
- Import Cloudera Impala Metadata Using the Windows ODBC Driver

Obtain Windows ODBC Driver for Cloudera

If you performed a client installation, then you don't have the Windows ODBC driver required for you to import Cloudera Impala metadata.

If you used the Installer to install the Oracle BI Administration Tool, then you don't have to perform this procedure.

1. Go to Cloudera's website.
2. Click the Downloads link and then click the Impala ODBC Drivers & Connectors link.
3. In the Download list, locate the required ODBC driver for your Administration Tool platform and click **Download Bits** to download the installer.
4. Run the ODBC driver installer to install the driver.

Import Cloudera Impala Metadata Using the Windows ODBC Driver

Cloudera Impala is a massively parallel processing (MPP) SQL query engine that runs natively in Apache Hadoop. Perform this procedure to import Cloudera Impala metadata into the Oracle BI repository.

To perform this procedure, you must have the required Windows ODBC driver. If you've a client installation of the Administration Tool, then you must follow the Obtain Windows ODBC Driver for Cloudera procedure to install the required Windows ODBC driver.

1. In Windows, locate and open the ODBC Data Source Administrator.
2. In the ODBC Data Source Administrator dialog, click the System DSN tab, and then click **Add**.

3. In the driver list, locate and select a Cloudera Impala driver. Click **Finish**.
4. In Cloudera ODBC Driver for Impala DSN Setup, enter the connection details for your Impala instance in these fields:
   - In the **Data Source Name** field, enter the data source name specified in the connection pool defined in the repository.
   - In the **Host** field, enter the fully qualified host name or the IP address.
   - In the **Port** field, enter the port number. The default is 21050.
   - In the **Database** field, specify the database. This value is usually *Default*.
5. If you're setting up a data source for Cloudera Impala driver, then click **Test**.
6. If you're setting up a data source for DataDirect Impala driver, then click **Test Connect**.
7. In the Administration Tool, select **File**, then select **Import Metadata**.
8. In the Import Metadata wizard, on the Select Data Source screen, confirm that ODBC 3.5 displays in the **Connection Type** field.
9. Select the Impala DSN, provide a user name and password, and click **Next**.
10. In the Select Metadata Types screen, click **Next** to accept the default values.
11. In the Select Metadata Objects screen, go to the Data source view list and select the Impala tables for import and click the **>** (Import selected) button to move them to the Repository view list.
12. Click **Finish**.
13. In the Physical Layer of the repository, double click the Impala database. The Database dialog appears.
14. In the **Database type** field, choose *Cloudera Impala*, and click **OK**.
15. Click **Save** to save the repository.
16. Optional: Model the newly imported data as necessary in the Business Model and Mapping layer and the Presentation layer.

About Setting Up Apache Hive Data Sources

These topics provide information about Windows ODBC drivers and Apache Hive.

This section contains the following topics:

- Obtain Windows ODBC Driver for Client Installation
- Limitations on the Use of Apache Hive

Obtain Windows ODBC Driver for Client Installation

If you've a client install of the Administration Tool, you don't have the Windows ODBC driver required for you to import Apache Hive metadata.

To obtain the Windows driver required to perform the import, log in to the My Oracle Support web site support.oracle.com and access DocID 1520733.1. The technical note associated with this DocID includes the required Windows driver, together with the instructions to install the driver and to perform the metadata import from the Hive data source.

Limitations on the Use of Apache Hive

These topics describes the limitations on the use of Hadoop and Hive with Oracle Analytics Server.

This section contains the following topics:

- Hive Limitation on Dates
- Hive Doesn't Support Count (Distinct M) Together with Group By M
- Hive Doesn't Support Differing Case Types
- Exception Thrown for Locate Function with an Out-of-Bounds Start Position Value
- Hive May Crash on Queries Using Substring
- Hive Doesn't Support Create Table
- Hive May Fail on Long Queries With Multiple AND and OR Clauses
- Queries with Subquery Expressions May Fail
- Hive Doesn't Support Distinct M and M in Same Select List

Hive Limitation on Dates

There are limitations with the DATE type with Hive data sources.

Hive supports the `Timestamp` data type. Use the `DATE` or `DATETIME` data type for timestamp columns in the repository's Physical layer.

Hive Doesn't Support Count (Distinct M) Together with Group By M

Learn the limitations of Hive data sources.

Queries similar to the following could cause Hive to crash.

- `SELECT M, COUNT(DISTINCT M) ... FROM ... GROUP BY M ...`

The situation occurs when the attribute in the `COUNT(DISTINCT...` definition is queried directly and if that attribute is also part of the table or foreign key or level key. Because `COUNT(DISTINCT X)` together with `GROUP BY X` always results in the count value of 1, a significant number of occurrences of this case are unlikely to happen.

To avoid this error when using `COUNT(DISTINCT...)` on a measure, don't include the exact attribute or any attribute in the same level.

Hive Doesn't Support Differing Case Types

Hive requires a strict check on types of the various parts of the Case statement.

This causes a presentation query such as the following to fail in Hive:

```
select supplierid, case supplierid when 10 then 'EQUAL TO TEN' when 20 then
'EQUAL TO TWENTY' else 'SOME OTHER VALUE' end as c2 from supplier order by c2
asc, 1 desc
```

The full error message in Hive for this query is:

```
FAILED: Error in semantic analysis: Line 2:32 Argument type mismatch '10':
The expressions after WHEN should have the same type with that after CASE:
"smallint" is expected but "int" is found
```

Exception Thrown for Locate Function with an Out-of-Bounds Start Position Value

Learn how to use the Locate function's syntax.

The full syntax of the Locate function is of the form:

```
LOCATE ( charexp1, charexp2, [, startpos] )
```

where `charexp1` is the string to search for within the string `charexp2`.

The optional parameter `startpos` is the character position within `charexp2` at which to begin the search.

If `startpos` has a value that's longer than the length of `charexp2`, such as in the following example:

```
select locate('c', 'abcde', 9) from employee
```

then Hive throws an exception instead of returning 0.

Hive May Crash on Queries Using Substring

Some queries that use the Substring function with a start position parameter value might cause Hive to crash.

The following might cause Hive to crash:

```
select substring(ProductName, 2) from Products
```

Hive Doesn't Support Create Table

As the Apache Hive ODBC driver doesn't support SQLTransact, which is used for creating tables, CREATE TABLE isn't supported by Hive.

Hive May Fail on Long Queries With Multiple AND and OR Clauses

The examples show conditions that could cause Hive data sources to fail.

The following `WHERE` clauses are examples of conditions that might cause queries to fail in Hive due to their excessive length:

Example 1

```
WHERE (Name = 'A' AND Id in (1))
   OR (Name = 'B' AND Id in (2))
   OR  .......
   OR (Name = 'H' AND Id in (8))
```

Example 2

```
WHERE (Id BETWEEN '01' AND '02')
   OR (Id BETWEEN '02' AND '03')
   OR  .......
   OR (Id BETWEEN '07' AND '08'))
```

Long queries could fail in Hive especially if the queries have conditions with multiple `OR` clauses each grouping together combinations of `AND` and `BETWEEN` sub-clauses as shown in the preceding examples.

Queries with Subquery Expressions May Fail

Queries with subquery expressions might fail in Hive.

If subquery expressions are used, the physical query that Oracle BI Server generates could include mixed data types in equality conditions. Because of Hive issues in equality operators, you could get an incorrect query result.

For example, for the following query:

```
select ReorderLevel from Product where ReorderLevel in
  (select AVG(DISTINCT ReorderLevel) from Product);
```

Oracle BI Server generates the following physical query that includes '`ReorderLevel = 15.0`' where `ReorderLevel` is of type `Int` and `15.0` is treated as `Float`:

```
Select T3120.ReorderLevel as c1 from Products T3120
 where (T3120.ReorderLevel = 15.0)
```

You can correct the mixed data types issue using the following command:

```
select ReorderLevel from Product where ReorderLevel in
  (select cast(AVG(DISTINCT ReorderLevel) as integer) from Product);
```

Hive Doesn't Support Distinct M and M in Same Select List

Learn about the limitations for using Select with Hive data sources.

Queries of the following form aren't supported by Hive:

- `SELECT DISTINCT M, M  ... FROM TABX`

About Setting Up Hyperion Financial Management Data Sources

Use these required steps to configure the Hyperion Financial Management data source.

Hyperion Financial Management 11.1.2.3.*x* or 11.1.2.4.*x* can use the ADM native driver or the ADM thin client driver. You can install and configure the ADM thin client driver on Linux operating system.

You can also use the Hyperion Financial Management 11.1.2.3.*x* and 11.1.2.4.*x* data sources with Oracle Analytics Server running in a Windows or Linux deployment.

Hyperion Financial Management ADM driver includes the ADM native driver and ADM thin client driver. For both Windows and Linux deployments, ensure that you perform the configuration using the Enterprise Performance Management Configurator.

- In the Windows and Linux configurations, provide the details for the Hyperion Shared Services Database to register with the Foundation server and the Hyperion Financial Management server.
- During configuration, make sure to enable DCOM configuration.
- If you're configuring for Windows, then in the DCOM User Details page, enter a domain user as the user for connecting to the Hyperion Financial Management

server. If you're configuring the ADM thin client driver for Linux, then you don't need to perform this step.

In addition, you must edit the `objih.properties` file on each system that's running the JavaHost process to include environment variables that are required by Hyperion Financial Management. The JavaHost process must be running to import from Hyperion Financial Management data sources, for both offline and online imports. If you've a client installation of the Administration Tool, then see Performing Additional Hyperion Configuration for Client Installations for JavaHost configuration steps.

> **Important:**
>
> You should always use forward slashes (/) instead of backslashes (\) when configuring the EPM paths in the `objih.properties` file.
>
> Forward slashes are required in the EPM paths on Windows. Backslashes don't work when configuring the EPM paths in the `objih.properties` file.

1. Locate the `objih.properties` at:

   ORACLE_HOME`/bi/modules/oracle.bi.cam.objih/env/objih.properties`

2. Open the `objih.properties` file for editing.
3. Append the following to the `OBIJH_ARGS` variable:

   ```
   DEPM_ORACLE_HOME=C:/Oracle/Middleware/EPMSystem11R1
   -DEPM_ORACLE_INSTANCE=C:/Oracle/Middleware/user_projects/epmsystem1
   -DHFM_ADM_TRACE=2
   ```

4. Add the following variables to the end of the `objih.properties` file:

   EPM_ORACLE_HOME`=C:/Oracle/Middleware/EPMSystem11R1`

   EPM_ORACLE_INSTANCE`=C:/Oracle/Middleware/user_projects/epmsystem1`

5. Locate the `loaders.xml` file in:

   ORACLE_HOME`/bi/bifoundation/javahost/config/loaders.xml`

6. In the `loaders.xml` file, locate `<!-- BI Server integration code -->`.
7. In the `<ClassPath>`, add the `fm-adm-driver.jar`, `fm-web-objectmodel.jar`, `epm_j2se.jar`, and `epm_hfm_web.jar` files using the format shown in the following:

   ```
   <ClassPath>
   {%EPM_ORACLE_HOME%}/common/hfm/11.1.2.0/lib/fm-adm-driver.jar;
   {%EPM_ORACLE_HOME%}/common/hfm/11.1.2.0/lib/fm-web-objectmodel.jar;
   {%EPM_ORACLE_HOME%}/common/jlib/11.1.2.0/epm_j2se.jar;
   {%EPM_ORACLE_HOME%}/common/jlib/11.1.2.0/epm_hfm_web.jar;
   </ClassPath>
   ```

8. Save and close the file.
9. Go to the *ORACLE_HOME*`/bi/bifoundation/javahost/lib/obisintegration/adm` directory and delete all jar files except for `admintegration.jar` and `admimport.jar`.
10. Restart OBIS1.
11. Repeat these steps on each computer that runs the JavaHost process.

Perform Additional Hyperion Configuration for Client Installations

If you install the Administration Tool using the Plus Client Installer, you must perform additional configuration before you can perform offline imports from Hyperion Financial Management data sources.

When importing from Hyperion Financial Management data sources in offline mode, the Administration Tool must point to the location of a running JavaHost.

The steps in this section are only required for client installations of the Administration Tool.

1. Close the Administration Tool.
2. On the same computer as the Administration Tool, open the local, use a text editor to open the `NQSConfig.INI` file located in:

   `BI_DOMAIN``\config\fmwconfig\biconfig\OBIS`
3. Locate the `JAVAHOST_HOSTNAME_OR_IP_ADDRESSES` parameter.
4. Update the `JAVAHOST_HOSTNAME_OR_IP_ADDRESSES` parameter to point to a running JavaHost, using a fully-qualified host name or IP address and port number. For example:

   ```
   JAVAHOST_HOSTNAME_OR_IP_ADDRESSES = "myhost.example.com:9810"
   ```

   In a full (non-client) Oracle Analytics Server installation, you can't manually edit the `JAVAHOST_HOSTNAME_OR_IP_ADDRESSES` setting because it's managed by Fusion Middleware Control.
5. Save and close the file.

Set Up Oracle RPAS Data Sources

Oracle BI Server can connect to Oracle RPAS (Retail Predictive Application Server) data sources through ODBC DSNs.

To set up Oracle RPAS data sources, you must first install the Oracle RPAS ODBC driver. During set up of the ODBC DSN, you must select the **SQLExtendedFetch** option, select DBMS from the **Authentication Method** list, and select No from the **Normalize Dimension Tables** list. See About Importing Metadata from Oracle RPAS Data Sources.

On Windows systems, you can connect to Oracle RPAS data sources for both initial import and for access to the data source during query processing. On Linux systems, you can only connect to Oracle RPAS data sources for data access.

Set Up Teradata Data Sources

You can use ODBC to access Teradata data sources.

See Set Up ODBC Data Source Names (DSNs).

After you've installed the latest Teradata ODBC driver and set up an ODBC DSN, you must add the lib directory for your Teradata data source to your Windows system Path environment variable. For example:

```
C:\Program Files\Teradata\Client\15.00\ODBC Driver for Teradata nt-x8664\Lib
```

You must edit `obis.properties` on each computer running the Oracle BI Server to include required Teradata variables.

1. Open `obis.properties` located in:

   BI_DOMAIN`\config\fmwconfig\bienv\obis`

2. In `PATH`, `LD_LIBRARY_PATH`, and `LIBPATH` enter the required variable information as shown in the following example.

   ```
   PATH=C:\Program Files\Teradata\Client\15.00\ODBC Driver for Teradatant-x8664\Lib;
   LD_LIBRARY_PATH=C:\Program Files\Teradata\Client\15.00\ODBC Driver forTeradata nt-
   x8664\Lib;
   LIBPATH=C:\Program Files\Teradata\Client\15.00\ODBC Driver for Teradatant-
   x8664\Lib;
   ```

   Note:

   If you use the default location when installing the Teradata client, then the `PATH` variable might exceed the 1024 character limit imposed by Windows. To avoid this issue, install the Teradata client in a directory with a shortened path name such as `C:\TD`, or use shortened 8.3 file names such as `C:\PROGRA~1\Teradata\Client\13.10\ODBCDR~1\Bin` instead of `C:\Program Files\Teradata\Client\13.10\ODBC Driver for Teradata\Bin`.

   To determine the correct 8.3 file names, run *dir /x* from the appropriate directory. For example:

   ```
   C:\>dir /x
    Volume in drive C has no label.
    Volume Serial Number is 0000-XXXX
    Directory of C:\
   08/25/2008  03:36 PM    <DIR>    DATAEX~1    DataExplorer
   04/20/2007  01:38 PM    <DIR>                dell
   08/28/2010  10:49 AM    <DIR>    DOCUME~1    Documents and Settings
   07/28/2008  04:50 PM    <DIR>    ECLIPS~2    EclipseWorkspace
   09/07/2007  11:50 AM    <DIR>                Ora92
   09/07/2007  11:50 AM    <DIR>                oracle
   05/21/2009  05:15 PM    <DIR>                OracleBI
   05/21/2009  05:12 PM    <DIR>    ORACLE~1    OracleBIData
   03/02/2011  04:51 PM    <DIR>    PROGRA~1    Program Files
   ```

3. Save and close the file.
4. Restart OBIS1.
5. Repeat these steps on each computer that runs the Oracle BI Server process. If you're running multiple Oracle BI Server instances on the same computer, be sure to update the ias-component tag appropriately for each instance in `obis.properties`, for example, `ias-component id="coreapplication_obis1"` and `ias-component id="coreapplication_obis2"`.

Avoid Spool Space Errors for Queries Against Teradata Data Sources

Some queries against Teradata might get a No more spool space error from the data source.

This error can occur for `DISTINCT` queries resulting from selecting **All Choices** in the Filters pane in Answers.

To avoid this error, you can ensure that the Oracle BI Server rewrites the query to use `GROUP BY` rather than `DISTINCT` for these queries by ensuring that the following conditions are met:

- There is only one dimension column in the projection list, and it's a target column rather than a combined expression.
- The original query from Answers is requesting `DISTINCT`, and doesn't include a `GROUP BY` clause
- The `FROM` table is a real physical table rather than an opaque view.
- The `FROM` table is an atomic table, not a derived table.
- The following ratio must be less than the threshold:

  (the distinct number of the projected column) / (number of rows of `FROM` table)

  Both values used in this ratio come from the repository metadata. To populate these values, click **Update Row Count** in the Administration Tool for both of the following objects:

  – The `FROM` physical table

  – The physical column for the projected column

  By default, the threshold for this ratio is 0.15. To change the threshold, create an environment variable on the Oracle BI Server computer called `SA_CHOICES_CNT_SPARSITY` and set it to the new threshold.

Enable NUMERIC Data Type Support for Oracle Database and TimesTen

You can enable NUMERIC data type support for Oracle Database and TimesTen data sources.

When NUMERIC data type support is enabled, NUMBER columns in Oracle Database and TimesTen data sources are treated as NUMERIC to provide greater precision. In addition, literals are instantiated as NUMERIC instead of DOUBLE for Oracle Database and TimesTen data sources.

See *Logical SQL Reference Guide for Oracle Business Intelligence Enterprise Edition*.

1. Set `ENABLE_NUMERIC_DATA_TYPE` to `YES` in `NQSConfig.INI` file located in *`BI_DOMAIN`*`/config/fmwconfig/biconfig/OBIS`.
2. Enable the NUMERIC_SUPPORTED database feature in the Physical layer database object. See SQL Features Supported by a Data Source for more about how to set database features.

The decimal/numeric data from other database types is mapped as DOUBLE when the `ENABLE_NUMERIC_DATA_TYPE` parameter is set to *YES*.

The data type of physical columns imported prior to changing the `ENABLE_NUMERIC_DATA_TYPE` setting remain unchanged. For existing DOUBLE physical columns, you must manually update the data type to NUMBER as needed.

Cast numeric data types to other number data types, and cast other number data types to numeric data types.

Numeric data type support isn't available when using the Oracle BI Server JDBC driver.

Your performance overhead could increase when numeric data types are enabled resulting from the higher number of bits for numeric data.

Configure Essbase to Use a Shared Logon

Shared logon is required and enabled by default for all Essbase connection pools.

You can't disable the Shared logon setting in the General tab of the Connection Pool Properties dialog.

Configure SSO for Essbase, Hyperion Financial Management, or Hyperion Planning Data Sources

Configure SSO and shared logon to use Hyperion Financial Management, or Hyperion Planning installed with the EPM System Installer as a data source.

If you use Hyperion Financial Management, or Hyperion Planning installed with the EPM System Installer as a data source for the Oracle BI Server, then use the SSO token option with shared logon. In this case, Oracle BI Server uses impersonation to connect to Hyperion Planning. The user details provided in the shared logon is used to connect to the data source, and the processing user is the impersonated user. The impersonated users should exist in the identity store used by Hyperion Financial Management or Hyperion Planning.

The user and the Enterprise Performance Management user must use the same identity store.

> **Note:**
>
> Essbase no longer supports CSS token based authentication. As a result, you must update the connection pools to use EssLoginAs authentication. EssLoginAS authentication provides reliable and better performance than CSS token based authentication, and provides the shared logon credentials of the Essbase administrator in the connection pool.

Import Metadata from Relational Data Sources

You can import metadata for supported relational data source types by selecting the appropriate connection type in the Import Metadata Wizard.

To import metadata, you must have all database connections set up on your local computer. You can import metadata in both offline and online modes.

See Import Metadata from Multidimensional Data Sources and Work with ADF Data Sources.

When you import physical tables, be careful to limit the import to only those tables that contain data that are likely to be used in the business models you create. You can use the Find feature to locate and select the tables that you want to import. Importing large numbers of extraneous tables and other objects adds unnecessary complexity and increases the size of the repository.

When you import metadata for most data sources, the default is to import tables, primary keys, and foreign keys. It's recommended that you import primary and foreign keys along with your tables so that the keys are automatically created in the Physical layer. If you don't import keys, you must create them manually, which can be a time-consuming process.

You can also import database views, aliases, synonyms, and system tables. Import these objects only if you want the Oracle BI Server to generate queries against them.

If you're importing metadata into an existing database in the Physical layer, then confirm that the **COUNT_STAR_SUPPORTED** option is selected in the Features tab of the Database properties dialog. If you import metadata without the **COUNT_STAR_SUPPORTED** option selected, then the **Update Row Count** option can't display in the right-click menu for the database's physical tables.

Other data source types are described in other sections:

- See Import Metadata from Multidimensional Data Sources for **Essbase**, **XMLA**, **Oracle OLAP**, **Hyperion ADM**. This section also describes importing from Oracle RPAS data sources over ODBC 3.5.
- See About Importing Metadata from XML Data Sources for **XML**.
- See Work with ADF Data Sources for **OracleADF_HTTP**.

If you want to import joins, select both **Keys** and **Foreign Keys**. If you want to import system tables, you must have the system privilege for your data source. To import from Customer Relationship Management (CRM) tables, select **Metadata from CRM tables**.

To search for a particular item, enter a keyword in the **Find** box and then click **Find Down** or **Find Up**.

1. In the Administration Tool, select **File**, then select **Import Metadata**.
2. In the Select Data Source screen, in the **Connection Type** field, select the type of connection appropriate for your data source, such as **ODBC 3.5**.

   Make sure to choose **OCI 10/11g** if your data source is an Oracle Database. Using OCI as your connection protocol to an Oracle Database ensures better performance and provides access to native database features that aren't available through ODBC.

For non-Oracle databases, it's recommended that you use ODBC 3.5 for importing schemas with International characters, such as Japanese table and column names.

The remaining fields and options on the Select Data Source screen vary according to the connection type you selected:

- For **ODBC 2.0** and **ODBC 3.5** data sources, in the **DSN** list, select a data source to import the schema. Then, provide a valid user name and password for the data source.
- For **OCI 10/11g** data sources, provide the name of the data source in the **Data Source Name** field, then provide a valid user name and password for the data source.

  For Oracle Database data sources, the data source name is either a full connect string or a net service name from the `tnsnames.ora` file. If you enter a net service name, you must ensure that you've set up a `tnsnames.ora` file within the Oracle Analytics Server environment, in:

  `BI_DOMAIN``/bidata/components/core/serviceinstances/ssi/oracledb`

When you've finished providing information in the Select Data Source screen, click **Next**. The Select Metadata Types screen appears.

If some objects couldn't be imported, a list of warning messages appears. In the dialog displaying the messages, you can perform the following actions:

- To search for specific terms, click **Find** and then **Find Again**.
- To copy the contents of the window so that you can paste the messages in another file, click **Copy**.

3. Select the objects types to import such as **Tables**, **Keys**, and **Foreign Keys**.
4. Click **Next**. The Select Metadata Objects screen appears.
5. Select the objects to import in the **Available** list and move them to the **Selected** list, using the **>** (Import selected) or **>>** (Import all) buttons.
6. Optional: Select **Show complete structure** to view all objects.

   Deselecting this option shows only the objects that are available for import.
7. Click **Finish**.

After you import metadata, you should check to ensure that your database and connection pool settings are correct. In rare cases, the Oracle BI Server can't determine the exact database type during import and instead assigns an approximate type to the database object. See Set Up Database Objects and Create or Change Connection Pools.

Visually inspect the imported data in the Physical layer such as physical columns and tables to ensure that the import completed successfully.

Import Metadata from Multidimensional Data Sources

You can import metadata from a multidimensional data source to the Physical layer of the Oracle BI repository.

Using multidimensional data sources enables the Oracle BI Server to connect to and extract data from a variety of sources.

During the import process, each cube in a multidimensional data source is created as a single physical cube table. The Oracle BI Server imports the cube metadata, including its

metrics, dimensions, and hierarchies. After importing the cubes, you need to verify that the physical cube columns have the correct aggregation rule, and that the hierarchy type is correct. See Work with Physical Hierarchy Objects.

> **Note:**
>
> Manually creating a physical schema from a multidimensional data source is labor-intensive and error prone. Therefore, it's strongly recommended that you use the import method.

Oracle recommends removing hierarchies and columns from the Physical layer if you aren't going to use the hierarchies and columns in the business model. Eliminating unnecessary objects in the Administration Tool could result in better performance.

If you're importing metadata into an existing database in the Physical layer, confirm that the **COUNT_STAR_SUPPORTED** option is selected on the Features tab in the Database properties dialog. If you import metadata without the **COUNT_STAR_SUPPORTED** option selected, the **Update Row Count** option doesn't display in the right-click menu for the database's physical tables.

See Multidimensional Connection Options.

1. In the Administration Tool, do one of the following:
   - Select **File**, then select **Import Metadata**.
   - From an existing database, right-click the connection pool in the Physical layer and select **Import Metadata**.
2. In Select Data Source, in the **Connection Type** field, select the type of connection appropriate for your data source, and click **Next**.
3. In **Select Metadata Types** (only Oracle RPAS data sources), select **Tables**, **Keys**, and **Foreign Keys** and then, click **Next**.
4. In **Select Metadata Objects**, from the **Available** list, select the objects to import using the Import **>**, or Import All **>>**.
5. Select **Import UDAs** if you want to import user-defined attributes (UDAs) from an Essbase data source.
6. Click **Finish**.

A list of warning messages display if some objects weren't imported. Resolve the issues as needed.

After you import metadata, you should verify that your database and connection pool settings are correct. In rare cases, the Oracle BI Server can't determine the exact database type during import and instead assigns an approximate type to the database object. See Set Up Database Objects and Create or Change Connection Pools.

Visually inspect the imported data in the Physical layer such as physical columns and hierarchical levels to confirm that the import completed successfully.

For Essbase data sources, all hierarchies are imported as Unbalanced by default. Review the **Hierarchy Type** property for each physical hierarchy and change the value if necessary. Supported hierarchy types for Essbase are **Unbalanced**, **Fully balanced**, and **Value**.

Multidimensional Data Source Connection Options

In the Oracle BI Administration Tool when importing multidimensional data sources into your repository, you can use these connection types in the Import Metadata wizard's Select Data Source page.

ODBC 3.5

The ODBC 3.5 connection type is used for Oracle RPAS data sources. Select the DSN entry and provide the user name and password for the selected data source. See Set Up ODBC Data Source Names (DSNs).

Essbase 9+

Use Essbase 9+ connection type for Essbase 9 or Essbase 11 data sources. Provide the host name of the computer where the Essbase Server is running in the Essbase Server field, then provide a valid user name and password for the data source. This information should be obtained from your data source administrator.

If the Essbase Server is running on a non-default port or in a cluster, include the port number in the Essbase Server field as `hostname:port_number`. See Work with Essbase Data Sources.

XMLA

Use the XMLA connection type for Microsoft Analysis Services and SAP/BW. Enter the URL of a data source from which to import the schema. You must specify the Provider Type such as Analysis Services 2000 or SAP/BW 3.5/7.0, and a valid user name and password for the data source.

You can use a new or existing Target Database.

Oracle OLAP

Provide the net service name in the Data Source Name field, and a valid user name and password for the data source. The data source name is the same as the entry you created in the `tnsnames.ora` file in the Oracle Analytics Server environment. You can also choose to enter a full connect string rather than the net service name.
Provide the URL of the `biadminservlet`. The servlet name is `services`, for example:

```
http://localhost:9704/biadminservlet/services
```

You must start the `biadminservlet` before you can use it. Check the status of the servlet in the Administration Console if you receive an import error. You can also check the Administration Server diagnostic log and the Domain log.

See Work with Oracle OLAP Data Sources.

You can use data sources from an Oracle Database data sources and the OLAP connection type. The data source can contain both relational tables and multidimensional tables. You should avoid putting multidimensional and relational tables in the same database object because you might need to specify different database feature sets for the different table types.

For example, Oracle OLAP queries fail if the database feature `GROUP_BY_GROUPING_SETS_SUPPORTED` is enabled. However, you might need to `GROUP_BY_GROUPING_SETS_SUPPORTED` enabled for Oracle Database relational tables.

You should create two separate database objects, one for relational tables, and one for multidimensional tables.

Hyperion ADM

Provide the URL for the Hyperion Financial Management or Hyperion Planning server.

For Hyperion Financial Management 11.1.2.1 and 11.1.2.2 using the ADM native driver, include the driver and application name (cube name), in the following format:

`adm:native:HsvADMDriver:`*`ip_or_host`*`:`*`application_name`*

For example:

```
adm:native:HsvADMDriver:192.0.2.254:UCFHFM
```

For Hyperion Financial Management 11.1.2.3 and 11.1.2.4 use the ADM thin client driver, and include the driver and application name (cube name) as follows:

`adm:thin:com.hyperion.ap.hsp.HspAdmDriver:`*`ip_or_host`*`:`*`port`*`:`*`application_name`*

For example:

```
adm:thin:com.hyperion.ap.hsp.HspAdmDriver:192.0.2.254:8300:UCFHP
```

For Hyperion Planning 11.1.2.4 or later, the installer doesn't deliver all of the required client driver `.jar` files. To ensure that you've the required .jar files, go to your instance of Hyperion, locate and copy the `adm.jar`, `ap.jar`, and `HspAdm.jar` files, and paste the files into *`MIDDLEWARE_HOME`*`\oracle_common\modules`.

For Hyperion Planning 11.1.2.4 or later using the ADM thin client driver, include the driver and application name (cube name), in the following format:

`adm:thin:com.oracle.hfm.HsvADMDriver:`*`server`*`:`*`application_name`*`?locale=en_US`

Select the provider type and enter a valid user name and password for your data source.

Before importing metadata, start the JavaHost process for both offline and online imports.

See Work with Hyperion Financial Management and Hyperion Planning Data Sources.

Review and complete the pre-configuration steps in About Setting Up Hyperion Financial Management Data Sources before importing.

About Importing Metadata from Oracle RPAS Data Sources

Learn about using the Oracle BI Administration to import metadata from Oracle RPAS.

When using the Administration Tool to import metadata from Oracle RPAS:

- Oracle RPAS schemas can only be imported on Windows.
- Before you import RPAS schemas, you must set the **Normalize Dimension Tables** field value in the ODBC DSN Setup page to *Yes* for the following reasons:
  - Setting this value to *Yes* uses an appropriate schema model (the snowflake schema) that creates joins correctly and enables drill down in the data.
  - Setting this value to *No* uses a star schema model that creates joins between all of the tables, causing an incorrect drill down. Many of the joins created in

the star schema more are unnecessary. You should remove the unnecessary joins manually.

See Set Up ODBC Data Source Names (DSNs).

- When you import RPAS schemas in the Administration Tool, you must import the data with joins. To do this, select the metadata types **Keys** and **Foreign Keys** in the Import Metadata Wizard.
- After you've imported RPAS schemas, you must change the **Normalize Dimension Tables** field value in the ODBC DSN Setup page back to *No*. You need to revert this setting back to *No* after import to enable the Oracle BI Server to correctly generate optimized SQL against the RPAS driver.

  If you don't change the **Normalize Dimension Tables** setting value to *No*, most queries fail with an error message similar to the following:

  ```
  [nQSError: 16001] ODBC error state: S0022 code: 0 message: [Oracle Retail][RPAS ODBC]Column:YEAR_LABEL not found..[nQSError: 16014] SQL statement preparation failed. Statement execute failed.
  ```

- If Oracle RPAS is the only data source, you must set the value of `NULL_VALUES_SORT_FIRST` to `ON` in the `NQSConfig.INI` file. See *Administering Oracle Analytics Server* for setting values in `NQSConfig.INI`.

After you import metadata from an Oracle RPAS data source, a database object for the schema is automatically created. Depending on your version of RPAS, you might need to adjust the data source definition in the Database property.

If RPAS is specified in the **data source definition Database** field and the version of RPAS is prior to 1.2.2, then the Oracle BI Server performs aggregate navigation when the SQL is generated and sent to the database. Because the table name used in the generated SQL is automatically generated, a mismatch between the generated SQL and the database table name could result. To enable the SQL to run, you must:

- Change the names of tables listed in the metadata so that the generated names are correct.
- Create tables in the database with the same names as the generated names.

If the database doesn't have tables with the same name or if you want to have the standard aggregate navigation, then you must change the **data source definition Database** field from RPAS to ODBC Basic. See Create a Database Object Manually in the Physical Layer.

About Importing Metadata from XML Data Sources

Learn how to import metadata from Extensible Markup Language (XML) documents.

This section contains the following topics:

- About Using XML as a Data Source
- Import Metadata from XML Data Sources Using the XML Gateway
- Import Metadata from XML Data Sources Using XML ODBC
- Examples of XML Documents

About Using XML as a Data Source

The Oracle BI Server supports the use of XML data as a data source for the Physical layer in the repository.

Depending on the method used to access XML data sources, a URL might represent a data source.

The following are data sources:

- A static XML file or HTML file that contains XML data islands on the Internet including intranet or extranet. For example:

  `tap://216.217.17.176/[DE0A48DE-1C3E-11D4-97C9-00105AA70303].XML`

- Dynamic XML generated from a server site. For example:

  `tap://www.aspserver.com/example.asp`

- An XML file or HTML file that contains XML data islands on a local or network drive. For example:

  `d:\xmldir\example.xml`

  `d:\htmldir\island.htm`

  You can also specify a directory path for local or network XML files, or you can use the asterisk (*) as a wildcard with the filenames. If you specify a directory path without a filename specification like `d:/xmldir`, all files with the XML suffix are imported. For example:

  `d:\xmldir\`

  `d:\xmldir\exam*.xml`

  `d:\htmldir\exam*.htm`

  `d:\htmldir\exam*.html`

- An HTML file that contains tables are wrapped in opening and closing `<table>` and `</table>` tags. The HTML file may reside on the Internet including intranet or extranet, or on a local or network drive, see About Using HTML Tables as a Data Source.

URLs can include repository or session variables, providing support for HTTP data sources that accept user IDs and passwords embedded in the URL. For example:

```
http://somewebserver/cgi.pl?userid=valueof(session_variable1)&password=
valueof(session_variable2)
```

This functionality also lets you create an XML data source with a location that's dynamically determined by some run-time parameters, see Use Variables in the Oracle BI Repository.

If the Oracle BI Server needs to access any non-local files, for example, network files or files on the Internet, you must run the Oracle BI Server using a valid user ID and password with sufficient network privileges to access these remote files.

Import Metadata from XML Data Sources Using the XML Gateway

When you use Oracle BI Server XML Gateway, the metadata import process flattens the XML document to a tabular form, and creates the XML file name using the stem of the table name. The second level element in the XML document is set as the row delimiter.

The stem is the filename without the suffix. All leaf nodes are imported as columns in the table. The hierarchical access path to leaf nodes is also imported.

The Oracle BI Server XML Gateway uses the metadata information contained in an XML schema. The XML schema is contained within the XML document, or is referenced within the root element of the XML document.

When a schema isn't available, all XML data is imported as text data. In building the repository, you can alter the data types of the columns in the Physical layer, overriding the data types for the corresponding columns defined in the schema. The gateway converts the incoming data to the type you specified in the Physical layer. You can also map the text data type to other data types in the Business Model and Mapping layer of the Administration Tool using the `CAST` operator.

The Oracle BI Server XML Gateway doesn't support:

- Resolution of external references contained in an XML document, other than a reference to an external XML schema, see Examples of XML Documents Generated by the Oracle BI Server XML Gateway.
- Element and attribute inheritance contained within the Microsoft XML schema.
- Element types of a mixed content model such as XML elements that contain a mixture of elements and CDATA such as `<p>hello <b>Joe</b>, how are you doing?</p>`.

If you're importing metadata into an existing database in the Physical layer, confirm that the `COUNT_STAR_SUPPORTED` option is selected in the Database properties dialog. If you import metadata without selecting the COUNT_STAR_SUPPORTED option, the **Update Row Count** option doesn't display in the right-click menu for the database's physical tables.

The Map to Logical Model and Publish to Warehouse screens are available only for ADF data sources.

URLs for the XML data source can include repository or session variables. If you browse for the XML data source, you can select a single file. For XML documents, you must specify the suffix `.xml` as part of the file name in the URL. If you don't specify the xml suffix, the documents are treated as HTML documents.

You can type an optional user name and password for connections to HTTP sites that employ the HTTP Basic Authentication security mode. The Oracle BI Server XML Gateway also supports Secure HTTP protocol and Integrated Windows Authentication (for Windows 2000), formerly called NTLM or Windows NT Challenge/Response authentication.

See Use Variables in the Oracle BI Repository

1. In the Administration Tool, do one of the following:
   a. Select **File**, then select **Import Metadata**.
   b. For an existing database and connection pool, right-click the connection pool in the Physical layer, and select **Import Metadata**.
2. In Import Metadata - Select Data Source, from the **Connection Type** list, select **XML**.

3. In **URL**, specify the XML data source URL.
4. In Select Data Source, click **Next**.
5. In Select Metadata Types, choose the options for the types of objects that you want to import, for example, **Tables**, **Keys**, and **Foreign Keys**.

   If you want to import joins, select both **Keys** and **Foreign Keys**. If you want to import system tables, you must have the system privilege for your data source.
6. Click **Next**. The Select Metadata Objects screen appears.
7. Select the objects you want to import in the **Available** list and move them to the **Selected** list, using the **>** (Import selected) or **>>** (Import all) buttons.
8. Click **Finish**.

After you import XML data, you must adjust connection pool settings. See Create or Change Connection Pools. You can do the following:

- In the Connection Pool dialog, type a name and optional description for the connection on the General tab.
- Click the XML tab to set additional connection properties, including the URL refresh interval and the length of time to wait for a URL to load before timing out.

  Because XML data sources are updated frequently and in real time, you can specify a refresh interval for Oracle BI Server XML Gateway data sources. The default timeout interval for queries (URL loading time-out) is 15 minutes. See About the Refresh Interval for XML Data Sources in *Administering Oracle Analytics Server*.

Examples of XML Documents Generated by the Oracle BI Server XML Gateway

These examples show sample XML documents and the corresponding columns that are generated by the Oracle BI Server XML Gateway.

XML Schema Contained in an External File

The following sample XML data document (*mytest.xml*) references an XML schema contained in an external file. The schema file is shown following the data document. The generated XML schema information available for import to the repository is shown at the end.

```
<?xml version="1.0"?>
<test xmlns="x-schema:mytest_sch.xml">

<row>
<p1>0</p1>
<p2 width="5">
      <p3>hi</p3>
      <p4>
         <p6>xx0</p6>
         <p7>yy0</p7>
      </p4>
      <p5>zz0</p5>
</p2>
</row>

<row>
```

```
<p1>1</p1>
<p2 width="6">
      <p3>how are you</p3>
      <p4>
         <p6>xx1</p6>
         <p7>yy1</p7>
      </p4>
      <p5>zz1</p5>
</p2>
</row>

<row>
<p1>a</p1>
<p2 width="7">
      <p3>hi</p3>
      <p4>
         <p6>xx2</p6>
         <p7>yy2</p7>
      </p4>
      <p5>zz2</p5>
</p2>
</row>

<row>
<p1>b</p1>
<p2 width="8">
      <p3>how are they</p3>
      <p4>
         <p6>xx3</p6>
         <p7>yy3</p7>
      </p4>
      <p5>zz2</p5>
</p2>
</row>
</test>
```

The corresponding schema file follows:

```
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
      xmlns:dt="urn:schemas-microsoft-com:datatypes">
      <ElementType name="test" content="eltOnly" order="many">
         <element type="row"/>
      </ElementType>
      <ElementType name="row" content="eltOnly" order="many">
      <element type="p1"/>
         <element type="p2"/>
      </ElementType>
      <ElementType name="p2" content="eltOnly" order="many">
         <AttributeType name="width" dt:type="int" />
         <attribute type="width" />
         <element type="p3"/>
         <element type="p4"/>
         <element type="p5"/>
      </ElementType>
      <ElementType name="p4" content="eltOnly" order="many">
         <element type="p6"/>
         <element type="p7"/>
      </ElementType>
      <ElementType name="p1" content="textOnly" dt:type="string"/>
      <ElementType name="p3" content="textOnly" dt:type="string"/>
      <ElementType name="p5" content="textOnly" dt:type="string"/>
```

```
      <ElementType name="p6" content="textOnly" dt:type="string"/>
      <ElementType name="p7" content="textOnly" dt:type="string"/>
</Schema>
```

The name of the table generated from the preceding XML data document (mytest.xml) would be `mytest` and the column names would be `p1`, `p3`, `p6`, `p7`, `p5`, and `width`.

In addition, to preserve the context in which each column occurs in the document and to distinguish between columns derived from XML elements with identical names but appearing in different contexts, a list of fully qualified column names is generated, based on the XPath proposal of the World Wide Web Consortium, as follows:

```
//test/row/p1
//test/row/p2/p3
//test/row/p2/p4/p6
//test/row/p2/p4/p7
//test/row/p2/p5
//test/row/p2@width
```

Nested Table Structures in an XML Document

The following example is a more complex example that demonstrates the use of nested table structures in an XML document. You can omit references to an external schema file, in which case all elements are treated as being of the `Varchar` character type.

```
===Invoice.xml===
<INVOICE>
   <CUSTOMER>
      <CUST_ID>1</CUST_ID>
      <FIRST_NAME>Nancy</FIRST_NAME>
      <LAST_NAME>Fuller</LAST_NAME>
      <ADDRESS>
         <ADD1>507 - 20th Ave. E.,</ADD1>
         <ADD2>Apt. 2A</ADD2>
         <CITY>Seattle</CITY>
         <STATE>WA</STATE>
         <ZIP>98122</ZIP>
      </ADDRESS>
      <PRODUCTS>
           <CATEGORY>
              <CATEGORY_ID>CAT1</CATEGORY_ID>
              <CATEGORY_NAME>NAME1</CATEGORY_NAME>
              <ITEMS>
                  <ITEM>
                     <ITEM_ID>1</ITEM_ID>
                     <NAME></NAME>
                     <PRICE>0.50</PRICE>
                     <QTY>2000</QTY>
                  </ITEM>
                  <ITEM>
                     <ITEM_ID>2</ITEM_ID>
                     <NAME>SPRITE</NAME>
                     <PRICE>0.30</PRICE>
                     <QTY></QTY>
                  </ITEM>
              </ITEMS>
         </CATEGORY>
           <CATEGORY>
              <CATEGORY_ID>CAT2</CATEGORY_ID>
              <CATEGORY_NAME>NAME2</CATEGORY_NAME>
```

```
            <ITEMS>
                <ITEM>
                   <ITEM_ID>11</ITEM_ID>
                   <NAME>ACOKE</NAME>
                   <PRICE>1.50</PRICE>
                   <QTY>3000</QTY>
                </ITEM>
                <ITEM>
                   <ITEM_ID>12</ITEM_ID>
                   <NAME>SOME SPRITE</NAME>
                   <PRICE>3.30</PRICE>
                   <QTY>2000</QTY>
                </ITEM>
            </ITEMS>
      </CATEGORY>
   </PRODUCTS>
</CUSTOMER>
<CUSTOMER>
   <CUST_ID>2</CUST_ID>
   <FIRST_NAME>Andrew</FIRST_NAME>
   <LAST_NAME>Carnegie</LAST_NAME>
   <ADDRESS>
      <ADD1>2955 Campus Dr.</ADD1>
      <ADD2>Ste. 300</ADD2>
      <CITY>San Mateo</CITY>
      <STATE>CA</STATE>
      <ZIP>94403</ZIP>
   </ADDRESS>
   <PRODUCTS>
        <CATEGORY>
           <CATEGORY_ID>CAT22</CATEGORY_ID>
           <CATEGORY_NAME>NAMEA1</CATEGORY_NAME>
           <ITEMS>
                <ITEM>
                   <ITEM_ID>122</ITEM_ID>
                   <NAME>DDDCOKE</NAME>
                   <PRICE>11.50</PRICE>
                   <QTY>2</QTY>
                </ITEM>
                <ITEM>
                   <ITEM_ID>22</ITEM_ID>
                   <NAME>PSPRITE</NAME>
                   <PRICE>9.30</PRICE>
                   <QTY>1978</QTY>
                </ITEM>
            </ITEMS>
      </CATEGORY>
        <CATEGORY>
           <CATEGORY_ID>CAT24</CATEGORY_ID>
           <CATEGORY_NAME>NAMEA2</CATEGORY_NAME>
           <ITEMS>
                <ITEM>
                   <ITEM_ID>19</ITEM_ID>
                   <NAME>SOME COKE</NAME>
                   <PRICE>1.58</PRICE>
                   <QTY>3</QTY>
                </ITEM>
                <ITEM>
                   <ITEM_ID>15</ITEM_ID>
                   <NAME>DIET SPRITE</NAME>
                   <PRICE>9.30</PRICE>
```

```
                    <QTY>12000</QTY>
                </ITEM>
            </ITEMS>
        </CATEGORY>
    </PRODUCTS>
</CUSTOMER>
<CUSTOMER>
    <CUST_ID>3</CUST_ID>
    <FIRST_NAME>Margaret</FIRST_NAME>
    <LAST_NAME>Leverling</LAST_NAME>
    <ADDRESS>
        <ADD1>722 Moss Bay Blvd.</ADD1>
        <ADD2> </ADD2>
        <CITY>Kirkland</CITY>
        <STATE>WA</STATE>
        <ZIP>98033</ZIP>
    </ADDRESS>
    <PRODUCTS>
         <CATEGORY>
            <CATEGORY_ID>CAT31</CATEGORY_ID>
            <CATEGORY_NAME>NAMEA3</CATEGORY_NAME>
            <ITEMS>
                <ITEM>
                    <ITEM_ID>13</ITEM_ID>
                    <NAME>COKE33</NAME>
                    <PRICE>30.50</PRICE>
                    <QTY>20033</QTY>
                </ITEM>
                <ITEM>
                    <ITEM_ID>23</ITEM_ID>
                    <NAME>SPRITE33</NAME>
                    <PRICE>0.38</PRICE>
                    <QTY>20099</QTY>
                </ITEM>
            </ITEMS>
        </CATEGORY>
          <CATEGORY>
            <CATEGORY_ID>CAT288</CATEGORY_ID>
            <CATEGORY_NAME>NAME H</CATEGORY_NAME>
            <ITEMS>
                <ITEM>
                    <ITEM_ID>19</ITEM_ID>
                    <NAME>COLA</NAME>
                    <PRICE>1.0</PRICE>
                    <QTY>3</QTY>
                </ITEM>
                <ITEM>
                    <ITEM_ID>18</ITEM_ID>
                    <NAME>MY SPRITE</NAME>
                    <PRICE>8.30</PRICE>
                    <QTY>123</QTY>
                </ITEM>
            </ITEMS>
        </CATEGORY>
    </PRODUCTS>
</CUSTOMER>
</INVOICE>
```

The generated XML schema consists of one table (`INVOICE`) with the following column names and their corresponding fully qualified names.

Column	Fully Qualified Name
ADD1	//INVOICE/CUSTOMER/ADDRESS/ADD1
ADD2	//INVOICE/CUSTOMER/ADDRESS/ADD2
CITY	//INVOICE/CUSTOMER/ADDRESS/CITY
STATE	//INVOICE/CUSTOMER/ADDRESS/STATE
ZIP	//INVOICE/CUSTOMER/ADDRESS/ZIP
CUST_ID	//INVOICE/CUSTOMER/CUST_ID
FIRST_NAME	//INVOICE/CUSTOMER/FIRST_NAME
LAST_NAME	//INVOICE/CUSTOMER/LAST_NAME
CATEGORY_ID	//INVOICE/CUSTOMER/PRODUCTS/CATEGORY/CATEGORY_ID
CATEGORY_NAME	//INVOICE/CUSTOMER/PRODUCTS/CATEGORY/CATEGORY_NAME
ITEM_ID	//INVOICE/CUSTOMER/PRODUCTS/CATEGORY/ITEMS/ITEM/ITEM_ID
NAME	//INVOICE/CUSTOMER/PRODUCTS/CATEGORY/ITEMS/ITEM/NAME
PRICE	//INVOICE/CUSTOMER/PRODUCTS/CATEGORY/ITEMS/ITEM/PRICE
QTY	//INVOICE/CUSTOMER/PRODUCTS/CATEGORY/ITEMS/ITEM/QTY

Only tags with values are extracted as columns. An XML query generates fully qualified tag names, to help ensure appropriate columns are retrieved.

The following shows the results of a sample query against the `INVOICE` table:

```
SELECT first_name, last_name, price, qty, name FROM invoice
-------------------------------------------------------------
FIRST_NAME  LAST_NAME         PRICE   QTY    NAME
-------------------------------------------------------------
Andrew      Carnegie           1.58     3    SOME COKE
Andrew      Carnegie          11.50     2    DDDCOKE
Andrew      Carnegie           9.30   12000  DIET SPRITE
Andrew      Carnegie           9.30    1978  PSPRITE
Margar      Leverling          0.38   20099  SPRITE33
Margar      Leverling          1.0       3   COLA
Margar      Leverling         30.50   20033  COKE33
Margar      Leverling          8.30    123   MY SPRITE
Nancy       Fuller             0.30          SPRITE
Nancy       Fuller             0.50    2000
Nancy       Fuller             1.50    3000  ACOKE
Nancy       Fuller             3.30    2000  SOME SPRITE
-------------------------------------------------------------
Row count: 12
```

About Using HTML Tables as a Data Source

The Oracle BI Server XML Gateway also supports the use of tables in HTML files as a data source. The HTML file can be identified as a URL pointing to a file on the internet, including intranet or extranet, or as a file on a local or network drive.

Even though tables, defined by the `<table>` and `</table>` tag pair, are native constructs of the HTML 4.0 specification, they're often used by Web designers as a general formatting device to achieve specific visual effects rather than as a data structure. The Oracle BI Server XML Gateway is currently the most effective in extracting tables that include specific column headers, defined by `<th>` and `</th>` tag pairs.

For tables that don't contain specific column headers, the Oracle BI Server XML Gateway employs some simple heuristics to make a best effort to determine the portions of an HTML file that appear to be genuine data tables.

The following is a sample HTML file with one table.

```
<html>
   <body>
      <table border=1 cellpadding=2 cellspacing=0>
         <tr>
            <th colspan=1>Transaction</th>
            <th colspan=2>Measurements</th>
         </tr>
         <tr>
            <th>Quality</th>
            <th>Count</th>
            <th>Percent</th>
         </tr>
         <tr>
            <td>Failed</td>
            <td>66,672</td>
            <td>4.1%</td>
         </tr>
         <tr>
            <td>Poor</td>
            <td>126,304</td>
            <td>7.7%</td>
         </tr>
         <tr>
            <td>Warning</td>
            <td>355,728</td>
            <td>21.6%</td>
         </tr>
         <tr>
            <td>OK</td>
            <td>1,095,056</td>
            <td>66.6%</td>
         </tr>
         <tr>
            <td colspan=1>Grand Total</td>
            <td>1,643,760</td>
            <td>100.0%</td>
         </tr>
      </table>
   </body>
</html>
```

The table name is derived from the HTML filename, and the column names are formed by concatenating the headings, defined by the <th> and </th> tag pairs, for the corresponding columns, separated by an underscore.

Assuming that our sample file is named `18.htm`, the table name would include `18_0`, because it's the first table in that HTML file, with the following column names and their corresponding fully qualified names:

Column	Fully Qualified Name
Transaction_Quality	\\18_0\Transaction_Quality
Measurements_Count	\\18_0\Measurements_Count

Column	Fully Qualified Name
Measurements_Percent	\\18_0\Measurements_Percent

If the table column headings appear in more than one row, the column names are formed by concatenating the corresponding field contents of those header rows.

For tables without any heading tag pairs, the Oracle BI Server XML Gateway assumes the field values, as delimited by the `<td>` and `</td>` tag pairs, in the first row to be the column names. The columns are named by the order in which they appear such as c0, c1, and c2.

See Import Metadata from XML Data Sources Using XML ODBC and Examples of XML Documents.

Import Metadata from XML Data Sources Using XML ODBC

Learn how to import metadata using ODBC.

Using the XML ODBC database type, you can access XML data sources through an ODBC interface. The data types of the XML elements representing physical columns in physical tables are derived from the data types of the XML elements as defined in the XML schema.

In the absence of a proper XML schema, the default data type of string is used. Data Type settings in the Physical layer don't override those defined in the XML data sources. When accessing XML data without XML schema, use the `CAST` operator to perform data type conversions in the Business Model and Mapping layer of the Administration Tool.

If you're importing metadata into an existing database in the Physical layer, confirm that the `COUNT_STAR_SUPPORTED` option is selected in the Features tab of the Database properties dialog. If you import metadata without selecting the `COUNT_STAR_SUPPORTED` option, the **Update Row Count** option doesn't display in the right-click menu for the database's physical tables.

When you import through the Oracle BI Server, the data source name (DSN) entries are on the Oracle BI Server computer, not on the local computer.

1. To access XML data sources through ODBC, you first need to license and install an XML ODBC driver.
2. Create ODBC DSNs that point to the XML data sources you want to access, making sure you select the XML ODBC database type.
3. In the Administration Tool, select **File**, then select **Import Metadata**.
4. In Select Data Source, from the **Connection Type** list, choose the connection type for your data source such as **ODBC 3.5**.
5. In the **DSN** list, select a data source to import the schema.
6. Type a valid user name and password for the data source, and click **Next**.
7. In Select Metadata Types, choose the types of objects to import such as **Tables**, **Keys**, **Synonyms**, and **Foreign Keys**, and click **Next**.

   Due to XML ODBC limitations, you must select the Synonyms option, or no tables are imported.
8. In Select Metadata Objects, choose objects to import from the **Available** list and move them to the **Selected** list, using **>** (Import selected) or **>>** (Import all).

9. Optional: Select **Show complete structure** to view all objects.

   Deselecting **Show complete structure** shows the objects that are available for import.

10. Click **Finish**.

Example of an XML ODBC Data Source

The example shows an XML ODBC data source in the Microsoft ADO persisted file format.

The example in this section shows an XML ODBC data source in the Microsoft ADO persisted file format. Both the data and the schema could be contained inside the same document.

XML ODBC Example

```
<xml xmlns:s='uuid:BDC6E3F0-6DA3-11d1-A2A3-00AA00C14882'
  xmlns:dt='uuid:C2F41010-65B3-11d1-A29F-00AA00C14882'
  xmlns:rs='urn:schemas-microsoft-com:rowset'
  xmlns:z='#RowsetSchema'>
<s:Schema id='RowsetSchema'>
  <s:ElementType name='row' content='eltOnly' rs:CommandTimeout='30'
  rs:updatable='true'>
    <s:AttributeType name='ShipperID' rs:number='1' rs:writeunknown='true'
    rs:basecatalog='Paint' rs:basetable='Shippers' rs:basecolumn='ShipperID'>
      <s:datatype dt:type='i2' dt:maxLength='2' rs:precision='5'
      rs:fixedlength='true' rs:benull='false'/>
    </s:AttributeType>
    <s:AttributeType name='CompanyName' rs:number='2' rs:writeunknown='true'
    rs:basecatalog='Paint' rs:basetable='Shippers' rs:basecolumn='CompanyName'>
      <s:datatype dt:type='string' rs:dbtype='str' dt:maxLength='40'
      rs:benull='false'/>
    </s:AttributeType>
    <s:AttributeType name='Phone' rs:number='3' rs:nullable='true'
    rs:writeunknown='true' rs:basecatalog='Paint' rs:basetable='Shippers'
    rs:basecolumn='Phone'>
      <s:datatype dt:type='string' rs:dbtype='str' dt:maxLength='24'
      rs:fixedlength='true'/>
    </s:AttributeType>
    <s:extends type='rs:rowbase'/>
  </s:ElementType>
</s:Schema>
<rs:data>
  <z:row ShipperID='1' CompanyName='Speedy Express' Phone='(503)
  555-9831          '/>
  <z:row ShipperID='2' CompanyName='United Package' Phone='(503)
  555-3199          '/>
  <z:row ShipperID='3' CompanyName='Federal Shipping' Phone='(503)
  555-9931          '/>
</rs:data>
</xml>
```

Examples of XML Documents

These examples of several different situations and explains how the Oracle BI Server XML access method handles those situations.

- The XML documents **83.xml** and **8_sch.xml** demonstrate the use of the same element declarations in different scope. For example, <p3> could appear within <p2> as well as within <p4>.

  Because the element <p3> in the preceding examples appears in two different scopes, each element is given a distinct column name by appending an index number to the second occurrence of the element during the import process. In this case, the second occurrence becomes p3_1. If <p3> occurs in additional contexts, they become p3_2, p3_3.

- The XML documents **83.xml** and **84.xml** (shown in demonstrate that multiple XML files can share the same schema (8_sch.xml).

83.xml

```
===83.xml===
<?xml version="1.0"?>
<test xmlns="x-schema:8_sch.xml">|
<row>
<p1>0</p1>
<p2 width="5" height="2">
   <p3>hi</p3>
   <p4>
      <p3>hi</p3>
      <p6>xx0</p6>
      <p7>yy0</p7>
   </p4>
   <p5>zz0</p5>
</p2>
</row>

<row>
<p1>1</p1>
<p2 width="6" height="3">
   <p3>how are you</p3>
   <p4>
      <p3>hi</p3>
      <p6>xx1</p6>
      <p7>yy1</p7>
   </p4>
   <p5>zz1</p5>
</p2>
</row>
</test>
```

8_sch.xml

```
===8_sch.xml===
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:dt="urn:schemas-microsoft-com:datatypes">
         <AttributeType name="height" dt:type="int" />
   <ElementType name="test" content="eltOnly" order="many">
      <AttributeType name="height" dt:type="int" />
      <element type="row"/>
   </ElementType>
   <ElementType name="row" content="eltOnly" order="many">
         <element type="p1"/>
      <element type="p2"/>
   </ElementType>
   <ElementType name="p2" content="eltOnly" order="many">
         <AttributeType name="width" dt:type="int" />
```

```
      <AttributeType name="height" dt:type="int" />
         <attribute type="width" />
      <attribute type="height" />
      <element type="p3"/>
      <element type="p4"/>
      <element type="p5"/>
   </ElementType>
   <ElementType name="p4" content="eltOnly" order="many">
      <element type="p3"/>
      <element type="p6"/>
      <element type="p7"/>
   </ElementType>
   <ElementType name="test0" content="eltOnly" order="many">
      <element type="row"/>
   </ElementType>
      <ElementType name="p1" content="textOnly" dt:type="string"/>
      <ElementType name="p3" content="textOnly" dt:type="string"/>
      <ElementType name="p5" content="textOnly" dt:type="string"/>
      <ElementType name="p6" content="textOnly" dt:type="string"/>
      <ElementType name="p7" content="textOnly" dt:type="string"/>
</Schema>
```

84.xml

```
===84.xml===
<?xml version="1.0"?>
<test0 xmlns="x-schema:8_sch.xml">
<row>
<p1>0</p1>
<p2 width="5" height="2">
   <p3>hi</p3>
   <p4>
      <p3>hi</p3>
      <p6>xx0</p6>
      <p7>yy0</p7>
   </p4>
   <p5>zz0</p5>
</p2>
</row>

<row>
<p1>1</p1>
<p2 width="6" height="3">
   <p3>how are you</p3>
   <p4>
      <p3>hi</p3>
      <p6>xx1</p6>
      <p7>yy1</p7>
   </p4>
   <p5>zz1</p5>
</p2>
</row>
</test0>
```

Island2.htm

```
===island2.htm===
<HTML>
   <HEAD>
<TITLE>HTML Document with Data Island</TITLE>
</HEAD>
```

```
   <BODY>
<p>This is an example of an XML data island in I.E. 5</p>
   <XML ID="12345">
   test>
      <row>
         <field1>00</field1>
         <field2>01</field2>
   </row>
      <row>
         <field1>10</field1>
         <field2>11</field2>
   </row>
      <row>
         <field1>20</field1>
         <field2>21</field2>
      </row>
   </test>
</XML>
<p>End of first example.</p>
<XML ID="12346">
   <test>
      <row>
         <field11>00</field11>
         <field12>01</field12>
      </row>
      <row>
         <field11>10</field11>
         <field12>11</field12>
      </row>
      <row>
         <field11>20</field11>
         <field12>21</field12>
      </row>
   </test>
</XML>
<p>End of second example.</p>
</BODY>
</HTML>
```

About Using a Standby Database

You should use a standby database for its high availability and failover functions, and as a backup for the primary database.

You schedule frequent and regular replication jobs from the primary database to a secondary database in a standby database configuration. Configure short intervals in the replication to enable writing to the primary database and facilitate reading from the secondary database without causing any synchronization or data integrity problems.

Because a standby database is essentially a read-only database, you can use the standby database as a business intelligence query server, relieving the workload of the primary database and improving query performance.

The following topics explain how to use a standby database:

- Configure a Standby Database
- Create the Database Object for the Standby Database Configuration
- Create Connection Pools for the Standby Database Configuration

- Update Write-Back Scripts in a Standby Database Configuration
- Set Up Usage Tracking in a Standby Database Configuration
- Set Up Event Polling in a Standby Database Configuration
- Set Up Oracle BI Scheduler in a Standby Database Configuration

Configure a Standby Database

In a standby database configuration, you've two databases: a primary database that handles all write operations and is the source of truth for data integrity, and a secondary database that's exposed as a read-only source.

When you use a standby database configuration, all write operations are off-loaded to the primary database, and read operations are sent to the standby database.

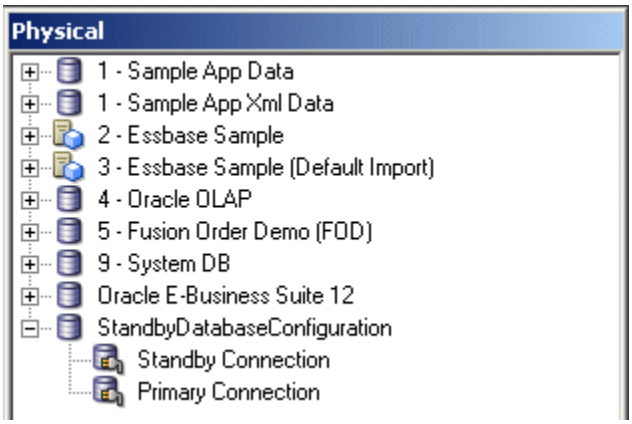
Write operations that need to be routed to the primary source may include the following:

- Oracle BI Scheduler job and instance data
- Temporary tables for performance enhancements
- Writeback scripts for aggregate persistence
- Usage tracking data, if usage tracking has been enabled
- Event polling table data, if event polling tables are being used

The following list provides an overview of how to configure the Oracle BI Server to use a standby database.

1. Create a single database object for the standby database configuration, with temporary table creation disabled.
2. Configure two connection pools for the database object:
    - A read-only connection pool that points to the standby database
    - A second connection pool that points to the primary database for write operations
3. Update any connection scripts that write to the database so that they explicitly specify the primary database connection pool.
4. If usage tracking has been enabled, update the usage tracking configuration to use the primary connection.
5. If event polling tables are being used, update the event polling database configuration to use the primary connection.
6. Ensure that Oracle BI Scheduler isn't configured to use any standby sources.

Even though there are two separate physical data sources for the standby database configuration, you create only one database object in the Physical layer. The image shows the database object and connection pools for the standby database configuration in the Physical layer.



Create the Database Object for the Standby Database Configuration

Use the Administration Tool to create a database object in the repository for the standby database configuration.

When you create the database object, make sure that the persist connection pool isn't assigned, to prevent the Oracle BI Server from creating temporary tables in the standby database.

1. In the Administration Tool, right-click the Physical layer and select **New Database** to create a database object.
2. In **Name**, provide a name for the database.
3. From the **Database Type** list, select the type of database.
4. In the **Persist connection pool** field, verify that the value is *not assigned*.

Create Connection Pools for the Standby Database Configuration

After you've created a database object in the repository for the standby database configuration, use the Administration Tool to create two connection pools, one that points to the standby database, and another that points to the primary database.

Because the standby connection pool is used for the majority of connections, make sure that the standby connection pool is listed first. Connection pools are used in the order listed, until the maximum number of connections is achieved. Ensure that the maximum number of connections is set in accordance with the standby database tuning. See Create or Change Connection Pools.

1. In the Administration Tool, in the Physical layer, right-click the database object for the standby database configuration and select **New Object**, then select **Connection Pool**.
2. Provide a name for the connection pool, and ensure that the call interface is appropriate for the standby database type.
3. Provide the **Data source name** for the standby database.
4. Enter a user name and password for the standby database.
5. Click **OK**.
6. In the Administration Tool, in the Physical layer, right-click the database object for the standby database configuration and select **New Object**, then select **Connection Pool**.

7. Provide a name for the connection pool, and ensure that the call interface is appropriate for the primary database type.
8. Provide the **Data source name** for the primary database.
9. Enter a user name and password for the primary database.
10. Click **OK**.

Update Write-Back Scripts in a Standby Database Configuration

If you use scripts that write to the database such as scripts for aggregate persistence, you must update the scripts to explicitly refer to the primary connection pool.

Information written through the primary connection is automatically transferred to the standby database through the regularly scheduled replication between the primary and secondary databases. The information is available through the standby connection pool.

The following example shows a write-back script for aggregate persistence that explicitly specifies the primary connection pool:

```
create aggregates sc_rev_qty_yr_cat for "DimSnowflakeSales"."SalesFacts"
("Revenue", "QtySold") at levels ("DimSnowflakeSales"."Time"."Year",
"DimSnowflakeSales"."Product"."Category") using connection pool
"StandbyDemo"."Primary Connection" in "StandbyDemo"."My_Schema"
```

Set Up Usage Tracking in a Standby Database Configuration

The Oracle BI Server supports the collection of usage tracking data.

When usage tracking is enabled, the Oracle BI Server collects usage tracking data for each query and writes statistics to a usage tracking log file or inserts them directly to a database table.

If you want to enable usage tracking on a standby database configuration using direct insertion, you must create the table used to store the usage tracking data such as `S_NQ_ACCT` on the primary database. Then, import the table into the physical layer of the repository using the Administration Tool.

You must ensure that the database object for the usage tracking table is configured with both the standby connection pool and the primary connection pool. Then, ensure that the `CONNECTION_POOL` parameter for usage tracking points to the primary database. For example, in `NQSConfig.ini`:

```
CONNECTION_POOL = "StandbyDatabaseConfiguration"."Primary Connection";
```

See *Administering Oracle Analytics Server*.

Set Up Event Polling in a Standby Database Configuration

You can use an Oracle BI Server event polling table (event table) as a way to notify the Oracle BI Server that one or more physical tables have been updated.

The event table is a physical table that resides on a database available to the Oracle BI Server. It's normally exposed only in the Physical layer of the Administration Tool, where it's identified in the Physical Table dialog as an Oracle BI Server event table.

The Oracle BI Server requires write access to the event polling table. Because of this, if you're using event polling in a standby database configuration, you must ensure that the database object for the event table only references the primary connection pool.

See Cache Event Processing with an Event Polling Table in *Administering Oracle Analytics Server* for full information about event polling, including how to set up, activate, and populate event tables.

Set Up Oracle BI Scheduler in a Standby Database Configuration

Oracle BI Scheduler is an extensible application and server that manages and schedules jobs, both scripted and unscripted.

Oracle BI Scheduler is an extensible application and server that manages and schedules jobs, both scripted and unscripted. To use Oracle BI Scheduler in a standby database configuration, you must ensure that the database object for Oracle BI Scheduler only references the primary connection pool.

See Configuration Tasks for Oracle BI Scheduler in *Integrator's Guide for Oracle Business Intelligence Enterprise Edition* for full information about setting up and using Oracle BI Scheduler.

6

Work with ADF Data Sources

This chapter describes how to set up Oracle ADF Business Components, and how to import metadata from ADF data sources.
Connecting to ADF data sources enables users to query data from any application that's built using the ADF Framework. For example, because Oracle CRM applications are developed using the ADF Framework, users can report on CRM data using an ADF data source that implements the required ADF Application Programming Interface (API).

By using the ADF components as a data source to the Oracle BI Server, users can quickly integrate operational reporting with any application that's built on top of the ADF Framework.

This chapter contains the following topics:

- What Are ADF Business Components?
- About Importing ADF Business Components
- About Specifying a SQL Bypass Database
- Set Up ADF Data Sources
- Import Metadata from ADF Data Sources
- Configure SSL in Oracle WebLogic Server
- Enable the Ability to Pass Custom Parameters to the ADF Application
- Propagate Labels and Tooltips from ADF Data Sources

What Are ADF Business Components?

Oracle Application Development Framework (Oracle ADF) is an object-relational framework used to create J2EE business services and expose underlying database objects.

This framework provides an abstraction layer that enables application developers to build applications quickly and efficiently.

When you use Oracle ADF to build service-oriented Java EE applications, you implement your core business logic as one or more business services. These back-end services provide clients with a way to query, insert, update, and delete business data as required, while enforcing appropriate business rules. ADF Business Components are prebuilt application objects that provide a ready-to-use implementation of Java EE design patterns and best practices.

The ADF model is represented through the ADF Business Component constructs called Entity Objects and View Objects, usually constructed and defined during design time:

- Entity Objects

  Entity objects are ADF framework components that represent a row in a database table and simplify modifying its data. Entity object enable encapsulating domain business logic for those rows to ensure your business policies and rules are consistently validated.

- View Objects

View objects are ADF framework components that encapsulate a SQL query and simplify working with its results. In addition to read-only view objects, there are entity-based view objects that support updatable rows. The view object queries just the data needed for the client-facing task at hand, then cooperates with one or more entity objects in your business domain layer to automatically validate and save changes made to its view rows. An entity-based view object encapsulates a SQL query. You can link an entity object into master/detail hierarchies using view links. You can use entity objects in the data model of your application modules.

Applications built using ADF obtain their data by querying the defined View Objects using the ADF APIs.

The ADF model also includes an application module, which is the transactional component that UI clients use to work with application data. It defines an updatable data model along with top-level procedures and functions, called service methods, related to a logical unit of work related to an end-user task.

The application module serves as a container for multiple View Objects and Entity Objects, and also contains configuration related to the JDBC data source.

About Operational Reporting with ADF Business Components

You can use integrated ADF Business Components to generate reports on data within your applications.

For example, you can generate reports based on expense data entered into an expense application. You would import the expense application metadata into the Oracle BI Repository using the Administration Tool, map the data from the Physical layer to the Business Model and Mapping layer, and then map the data to the Presentation layer. After you restart the Oracle BI Server and reload the metadata into Oracle BI Presentation Services, you can log in to Answers and drag and drop the columns to generate a report using the expense application data. You could select columns to view a report of your expenses grouped by a specific category such as airline travel expenses.

About Importing ADF Business Components

During import, the required physical tables and complex joins are automatically created.

The ViewObject and ViewLink instances are imported into Oracle Analytics Server. During query processing, the definitions retrieved from these instances are used to create the CompositeVO (view objects) in Oracle Application Development Framework (ADF).

These complex joins are dummy joins and aren't run in Oracle Analytics Server. Instead, the dummy joins denote ViewLink instances that connect pairs of View Objects in the ADF model. The physical table and complex join names correspond to the fully qualified ViewObject and ViewLink instance names. This convention allows arbitrary nesting of ApplicationModules in the ADF model.

The name of the automatically generated joins uses a naming convention similar to *ViewObjectName1_ViewObjectName2*, for example, *AppModuleAM.AP_VO1_AppModuleAM_BU_VO1*. The ViewLink instance name appears in the **ViewLink Name** field of the Complex Join dialog.

The complex joins are created automatically if a ViewLink instance is available. Complex joins aren't created for ViewLink definitions. You must create joins using ViewLink definitions manually. To manually create a join using ViewLink definitions, specify the ViewLink definition name in the **ViewLink Name** field of the Complex Join dialog.

The **External Expression** field in the Complex Join dialog for ADF data sources is populated with the join condition defined in the view link.

If custom properties are defined on the ApplicationModule, joins between view objects in different ApplicationModules are created on import from ADF. The format for the property name and value are as follow:

- The property name format is BI_VIEW_LINK_*property_name*
- The property value format is *source_view_object_instance_name*, *ViewLink_definition_name*, *destination_view_object_instance_name*

Be sure to use the fully qualified view object instance names for the source and destination view objects, as well as the fully qualified package name for the ViewLink definition.

About Specifying a SQL Bypass Database

The Oracle BI Server can automatically create composite View Objects at runtime enabling an ad-hoc BI query to reference multiple View Objects in the ADF layer.

For improved performance, a SQL bypass query is generated that incorporates the projection columns, filters, and joins required by the BI query.

The SQL Bypass feature directly queries the database so that aggregations and other transformations are pushed down where possible, reducing the amount of data streamed and worked on in Oracle Analytics Server. When using a SQL Bypass database, the system gets the view object query from the ADF data source and then wraps it with the aggregations in the Logical SQL query. The query, including the aggregations, is then run in the database. Because the database computes the aggregation and fewer rows are streamed back to Oracle Analytics Server, using a SQL Bypass database can result in significant performance gains.

Multiple View Objects are modeled as separate BI physical tables and are connected with dummy complex joins. These joins only represent the ViewLinks in the ADF model and aren't run by the Oracle BI Server.

You can specify the name of the SQL Bypass database in the connection pool for the ADF data source. The SQL Bypass database must be a physical database in the Physical layer of the repository. The database object for the SQL Bypass database must have a valid connection pool, with connection information that points to the same database that's being used by the JDBC Data source defined in the Oracle WebLogic Server that runs the ADF application.

The SQL Bypass database doesn't need to have any tables under it. After a valid database name is supplied, the SQL Bypass feature is enabled for all queries against that ADF database.

Set Up ADF Data Sources

These topics explain how to configure your ADF Business Components.

This section contains the following topics:

- Create a WebLogic Domain for ADF Business Components
- Deploy OBIEEBroker as a Shared Library in Oracle WebLogic Server
- Deploy the Application EAR File to Oracle WebLogic Server from JDeveloper
- Set Up a JDBC Data Source in the WebLogic Server
- Set the Logging Level for the Deployed Application in Oracle WebLogic Server

Create a WebLogic Domain for ADF Business Components

To configure your ADF Business Components for use with Oracle Analytics Server, you need to create a WebLogic Domain for your ADF Business Components that supports WebLogic Server, Oracle Application Core (Webapp), and Oracle JRF.

1. Start the WebLogic Configuration Wizard.

   For example, on Windows, run *MW_HOME*\wlserver\common\bin\config.cmd.
2. Select **Create a new WebLogic domain** and click **Next**.
3. On the Select Domain Source screen, ensure that **Basic WebLogic Server Domain**, **Oracle JRF**, and **Oracle Application Core (Webapp)** are selected.
4. Follow the remaining steps in the wizard, providing values appropriate for your environment.
5. Click **Create** on the Configuration Summary screen to create the domain.

You can start and stop the Oracle WebLogic Server for this domain using command-line scripts in the domain directory. For example, on Windows, use the following:

- *BI_DOMAIN*\bin\startWebLogic.cmd
- *BI_DOMAIN*\bin\stopWebLogic.cmd

Deploy OBIEEBroker as a Shared Library in Oracle WebLogic Server

To configure your ADF Business Components for use with Oracle Analytics, you need to install OBIEEBroker, making its physical file or directory known to Oracle WebLogic Server, and start it.

This process deploys the OBIEEBroker library as a shared library in Oracle WebLogic Server.

After the library has been installed and started, other deployed modules can reference the library. The OBIEEBroker shared library is installed by default.

1. Ensure that Oracle WebLogic Server is running. If it isn't running, start it. For example, on Windows, run *BI_DOMAIN*\bin\startWebLogic.cmd.
2. Open the WebLogic Server Administration Console. For example, if your Oracle WebLogic Server is running locally on port 7001, go to `http://localhost:7001/console`.
3. Log in to the WebLogic Server Administration Console with the credentials you created when you set up your WebLogic domain.
4. In the Change Center, click **Lock & Edit**.
5. On the Home Page, in the left pane, click **Deployments**.

6. In the right pane, click **Install**.
7. Using the Install Application Assistant, locate the `OBIEEBroker EAR` file in:

   ORACLE_HOME`\bi\bifoundation\javahost\lib\obisintegration\adf\oracle.bi.integration.adf.ear`
8. Click **Next**.
9. Select **Install this deployment as a library** and click **Next**.
10. Select the servers and/or clusters to which you want to deploy the OBIEEBroker library.

    Make sure to select all servers and clusters to which modules or applications that reference the library are deployed.
11. Click **Next**.
12. Optional: Update settings about the deployment.
13. Click **Next**, then click **Finish** to complete the installation.
14. In the Change Center, click **Activate Changes**.

Deploy the Application EAR File to Oracle WebLogic Server from JDeveloper

To configure your ADF Business Components for use with Oracle Analytics Server, you need to deploy the application EAR file to Oracle WebLogic Server from JDeveloper.

Before beginning this procedure, ensure that the following conditions are true:

- You've an ADF Model project that contains `ApplicationModules` and view objects that are exposed to Oracle Analytics Server.
- You've deployed `OBIEEBroker` as a shared library in Oracle WebLogic Server. See Deploy OBIEEBroker as a Shared Library in Oracle WebLogic Server.
- Oracle WebLogic Server is running.

1. Start JDeveloper, on Windows, run *MW_HOME*`\jdeveloper\jdev\bin\jdev.exe`.
2. Select **File**, then select **Open** to open the project that contains your ADF Business Components in JDeveloper. If prompted, allow JDeveloper to migrate the project to the latest version.
3. Create a new Application Module configuration, as follows:
   a. In the Model project, double-click the application module, then click the Configurations tab for that application module.
   b. Create a new configuration with the following characteristics:
      - Select **JDBC DataSource** for **Connection Type**.
      - Keep the default **DataSource Name**, for example, `java:comp/env/jdbc/ApplicationDBDS`.

      When you set up the JDBC data source in Oracle WebLogic Server in a later step, you use part of this DataSource Name as the JNDI name required by Oracle WebLogic Server. The JNDI name is the DataSource Name without the `java:comp/env` context prefix, for example, `jdbc/ApplicationDBDS`.
4. Create a Business Component Archive deployment profile, as follows:

a. In the Projects window, right-click the Model project and choose **New**.

b. Select **Deployment Profiles** under **General** in the left pane, then choose **Business Components Archive** in the right pane and click **OK**.

c. Provide a name for the deployment profile, for example, *MyApplication_Archive*, and click **OK**.

d. On the Deployment page, click **OK**.

5. In the Projects window, right-click the Model project and select **Deploy***your_deployment_profile_name* from **Deploy**, or use the deployment wizard by selecting **Deploy to File**.

   After the project has been deployed, two jar files are created in the deploy directory for the Model project, for example, `MyApplication_Archive_Common.jar` and `MyApplication_Archive_MiddleTier.jar`.

6. Create a new web project for the application, as follows:

   a. Right-click the global application and select **New Project**.

   b. Select **Projects** from the left pane, then select **Web Project** from the right pane.

   c. Provide a project name, for example, *OBIEEBroker*.

   d. Click **Next** until you reach the **Web Project Profile** page.

   e. Modify the **Java EE Context Root** to a name that better represents your application, for example, *MyApplication*.

      This value determines the URL that you use to connect to the application from Oracle Analytics Server, for example, `http://localhost:7001/MyApplication/obieebroker`.

7. Edit the Profile Dependencies of the WAR deployment, as follows:

   a. Right-click the Web Project you just created, for example, **OBIEEBroker**, and select **Project Properties**.

   b. From the left pane, select **Deployment**. Then, open the WAR File deployment profile on the right pane.

   c. Select **Profile Dependencies** from the left pane, and then, on the right pane, select the Common and Middle Tier deployment profiles of your Model project.

8. Expand the Web Project and open `web.xml`. Then, go to the source view of the file.

9. In the `web.xml` source, replace the content within the `<web-app>` element with the following:

```
<context-param>
  <description>This holds the Principals (CSV) that a valid end user should
have (at least one) in order to query the ADF layer from BI.</description>
  <param-name>oracle.bi.integration.approle.whitelist</param-name>
  <param-value>Application_Roles_List</param-value>
</context-param>

<filter>
  <filter-name>ServletADFFilter</filter-name>
  <filter-class>oracle.adf.share.http.ServletADFFilter</filter-class>
</filter>

<filter-mapping>
```

```
  <filter-name>ServletADFFilter</filter-name>
  <servlet-name>OBIEEBroker</servlet-name>
  <dispatcher>FORWARD</dispatcher>
  <dispatcher>REQUEST</dispatcher>
</filter-mapping>

<servlet>
  <servlet-name>OBIEEBroker</servlet-name>
  <servlet-class>oracle.bi.integration.adf.v11g.obieebroker.OBIEEBroker
  </servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>OBIEEBroker</servlet-name>
  <url-pattern>/obieebroker</url-pattern>
</servlet-mapping>
```

Following this step ensures that the OBIEEBroker servlet is used to access your application from Oracle Analytics Server.

For *application_roles_list*, provide a list of application roles in CSV form. For example:

```
<param-value>FBI_TRANSACTION_ANALYSIS_GENERIC_DUTY, OBIA_ANALYSIS_GENERIC_DUTY,
OBIA_EXTRACT_TRANSFORM_LOAD_DUTY, FUSION_APPS_BI_APPID</param-value>
```

If you provide a list of application roles, a user's application role is checked before access is allowed to the application. This runtime check requires the following grant to be present in the *domain_name*`/config/fmwconfig/system-jazn-data.xml` file for the WebLogic domain:

```
<grant>
  <grantee>
    <codesource>
      <url>file:${domain.home}/servers/${weblogic.Name}/tmp/
      _WL_user/oracle.bi.integration.adf/-</url>
    </codesource>
  </grantee>
  <permissions>
    <permission>
      <class>oracle.security.jps.JpsPermission</class>
      <name>IdentityAssertion</name>
      <actions>execute</actions>
    </permission>
    <permission>
      <class>oracle.security.jps.JpsPermission</class>
      <name>AppSecurityContext.setApplicationID.obi</name>
    </permission>
  </permissions>
</grant>
```

If you don't want application roles to be checked by the OBIEEBroker servlet, use DISABLE_WHITELIST_ROLE_CHECK as the value for the <context-param> in web.xml. For example:

```
<param-value>DISABLE_BI_WHITELIST_ROLE_CHECK</param-value>
```

10. Create an EAR deployment profile for the application, as follows:
    a. Right-click the global application and select **Application Properties**.
    b. From the left pane, select **Deployment**, then click **New** on the right pane to create a new deployment profile.

c. For **Archive Type**, select **EAR File**. Then, provide a name for the deployment profile, for example, MyApplication.

   The deployment profile name is used as the name displayed in the list of deployments in Oracle WebLogic Server.

d. From the left pane, select **Application Assembly**. Then, on the right pane, select the **webapp** deployment profile of your Web Project.

   Following this step ensures that the WAR file from your Web Project is included in the EAR file.

11. Under **Application Resources**, select **Descriptors**, select **META-INF** , and then select `weblogic-application.xml`.
12. On the left, select the Libraries tab.
13. Create two new Shared Library References, as follows:
    - Create the first Shared Library Reference with the following characteristics:
      - **Library Name:** `oracle.bi.integration.adf`
      - **Implementation Version:** `11.1.1.2.0`
    - Create the second Shared Library Reference with the following characteristics:
      - **Library Name:** `oracle.applcore.model`
      - **Implementation Version:** `11.1.1.0.0`

    These two Shared Library References create the following entries in the `weblogic-application.xml` file for the application:

```
<library-ref>
  <library-name>oracle.bi.integration.adf</library-name>
  <implementation-version>11.1.1.2.0</implementation-version>
</library-ref>
<library-ref>
  <library-name>oracle.applcore.model</library-name>
  <implementation-version>11.1.1.0.0</implementation-version>
</library-ref>
```

14. Right-click and select **Deploy** *EAR_deployment_profile_name* to deploy the EAR file to Oracle WebLogic Server by the global application.
15. From the dialog that appears, select **Deploy to Application Server**, and then follow the instructions in the wizard.
16. To verify that the application has been deployed, log in to the WebLogic Server Administration Console and click **Deployments** under **Your Deployed Resources**. Verify that your application appears in the list, for example, `obieebroker_`*`app_name`*.

Set Up a JDBC Data Source in the WebLogic Server

You must configure the ADF Business Components that you plan to use with Oracle Analytics Server.

To configure your ADF Business Components for use with Oracle Analytics Server, you must set up a JDBC data source in Oracle WebLogic Server for your application.

1. Ensure that Oracle WebLogic Server is running. If it isn't running, start it. For example, on Windows, run *`BI_DOMAIN`*`\bin\startWebLogic.cmd`.

2. Open the WebLogic Server Administration Console. For example, if your Oracle WebLogic Server is running locally on port 7001, go to `http://localhost:7001/console`.
3. Log in to the WebLogic Server Administration Console with the credentials you created when you set up your WebLogic domain.
4. On the Home Page, select **JDBC**, then select **Data Sources**.
5. Click **New**.
6. Provide information for your data source. For **Name** and **JNDI Name**, provide the DataSource Name you specified in the Application Module configuration for the application, without the `java:comp/env` context prefix, for example, `jdbc/ApplicationDBDS`. In addition, make sure to select the target on which you want to deploy the data source before exiting the wizard.
7. Click **Finish** when you're done providing JDBC data source settings.

Set the Logging Level for the Deployed Application in Oracle WebLogic Server

The *server_name*`-diagnostic.log` file for the server where your application is deployed contains information about your deployed application.

You can find this file in the server-specific directory within your domain. For example, on Windows, the log file for the AdminServer is located in:

BI_DOMAIN`\servers\AdminServer\logs`.

Log levels include:

- *SEVERE*
- *WARNING*
- *INFO*
- *CONFIG*
- *FINE*
- *FINER*
- *FINEST*

1. Open the Oracle WebLogic Server file `logging.xml` for editing, located in:

   BI_DOMAIN`\servers\`*server_name*

2. Within the `<loggers>` element, add the following child elements:

   ```
   <logger name="oracle.bi.integration.adf" level="LOG_LEVEL"/>
   <logger name="oracle.bi.integration.adf.v11g.obieebroker" level="LOG_LEVEL"/>
   ```

3. Save and close the file.
4. Restart Oracle WebLogic Server.

Import Metadata from ADF Data Sources

There are different ways to import metadata from ADF data sources.

Before you can import metadata from ADF sources, you must complete the steps in Set Up ADF Data Sources

This section contains the following topics:

- Perform an Initial Import from ADF Data Sources
- Use Incremental Import to Propagate Flex Object Changes
- Automatically Map Flex Object Changes to the Logical Model
- Automatically Map Flex Object Changes Using the biserverextender Utility

Perform an Initial Import from ADF Data Sources

You can use the Import Metadata Wizard to perform an initial import from the Oracle Application Development Framework (ADF) data sources.

In the Import Metadata wizard, you can search for a specific item by typing a keyword in **Find**.

Use **Show complete structure** to view all objects, including those that have already been imported. Deselecting this option shows only the objects that are available for import. When this option is selected, objects that have already been imported appear grayed out.

If this import is creating a new connection to the data source, when you move the items from the **Data source view** to the **Repository View** list, the Connection Pool dialog opens to show the values that you provided in Select Data Source page of the Import Metadata Wizard. You can provide the name of a **SQL Bypass Database** field.

- See Automatically Map Flex Object Changes to the Logical Model.
- See Create a new Application Module configuration, in Deploy the Application EAR File to Oracle WebLogic Server from JDeveloper.
- See About Specifying a SQL Bypass Database.

1. In the Administration Tool, do one of the following:
   - From the **File** menu, select **Import Metadata**.
   - If you've an existing ADF data source and connection pool, right-click the connection pool in the Physical layer, and select **Import Metadata**.
2. In the Import Metadata Select Data Source page, from the for **Connection Type** list, choose **OracleADF_HTTP**.

   When you've finished providing information The Select Metadata Objects screen appears.
3. Under Connection Pool, select **New Connection**, or select **Existing Connection** and click **Browse** to locate and select an existing connection pool.

   If you're using an **Existing Connection**, the values in **Data Source**, **AppModule Definition**, **AppModule Config**, or **URL**, and the **User Name** and **Password** fields are populated from the connection pool definition.
4. In the **Data Source** field leave the field blank to use the default JDBC data source, or type a JDBC data source name such as `jdbc/nWindORA05` to use a different data source.

5. In **AppModule Definition**, type the fully qualified Java package name of the root application module to use for the connection such as `oracle.apps.fii.receivables.model.RootAppModule`, or `snowflakesales.SnowflakeSalesApp`.
6. In **URL**, type the URL to the Oracle Analytics Server broker servlet using the following format:

   `http://host:port/APP_DEPLOYMENT_NAME/obieebroker`

   The URL is case-sensitive, for example:

   `http://localhost:7001/MyApp/obieebroker`
7. In **User Name** and **Password**, provide a valid user name and password for the Oracle ADF application.

   You must set up the user name and password in the Oracle WebLogic Server security realm.
8. In the **Data source view**, select the objects to import and move them to the **Repository View**.
9. In Select Data Source, click **Next**.
10. Click **Finish**.
11. Expand the database object for the ADF data source in the Physical layer to validate that your import was successful, right-click a physical table, and click **View Data**.

Use Incremental Import to Propagate Flex Object Changes

If you make changes to flexfields in your ADF applications, then you can use the Import Metadata Wizard in the Oracle BI Administration Tool to incrementally import the changes to the Physical layer of the Oracle BI repository.

The Import Metadata Wizard includes a synchronization feature for ADF data sources that enables you to import only the changes made to objects. Synchronization detects the changed objects, including new joined dimensions (KFF) and new attributes (DFF and EFF) to enable adding the objects automatically, without the need to search for the changed object.

The synchronization feature detects the following:

- Changes in columns
- Additions or deletions of tables and columns
- Additions of keys and foreign keys
- Newly joined tables

  New tables that are joined to any existing table are only imported when you select the option **Automatically include any missing joined objects** on the Select Metadata Objects screen.

Because data is imported incrementally, modifications to properties of attributes are detected and propagated. For example, if an attribute changes its data type, that change is propagated to the physical layer objects.

After import, the ADF data is modeled as shown in the table.

ADF Metadata	Imported BI Metadata
Root Application Module	Database
View Objects	Physical Tables
View Object Attribute	Physical Column
View Object Key	Physical Key
View Links	Physical Joins

If you're importing metadata into an existing database in the Physical layer, then confirm that the COUNT_STAR_SUPPORTED option is selected in the Features tab of the Database properties dialog. If you import metadata without the **COUNT_STAR_SUPPORTED** option selected, the **Update Row Count** option doesn't display in the right-click menu for the database's physical tables.

See Automatically Map Flex Object Changes to the Logical Model.

1. In the Administration Tool, in the Physical layer, right-click the connection pool for your ADF OLTP source and select **Import Metadata**.
2. Click **Synchronize** to locate and automatically select all recent changes for import.
3. Review the selected metadata to locate the new attributes.
4. Click **Finish** to close the wizard, or click **Next** to continue to Map to Logical Model.

Automatically Map Flex Object Changes to the Logical Model

After importing changes to flexfields in your ADF application, you can use the Map to Logical Model screen of the Import Metadata Wizard in the Oracle BI Administration Tool to automatically propagate the changes to the Business Model and Mapping layer and Presentation layer.

You can override the default mapping behavior during by renaming logical tables, splitting a view object into multiple tables, and combining multiple view objects into a single logical table.

See Customize the Mapping Behavior.

You can keep the default behavior, or customize the behavior for your needs. For example, you might want to rename tables and columns in the Business Model and Mapping layer, map to an existing logical table, or map a logical column to multiple source columns. The Column Mapping grid shows alias columns as well as regular columns, so that you can handle customized mappings that include alias columns. The Table Mapping grid enables a single physical table to map to multiple logical tables, and the reverse.

The Table Mapping grid includes a **VO Type** column. Options include **Normal**, **ETL Only**, and **Query Only**. ETL Only view objects exist only to extend the ETL mappings, and aren't used for queries. Logical table sources that reference imported view objects of this type are marked as disabled in the Business Model and Mapping layer. Query Only view objects are only used for queries, and aren't passed to the BI Extender for extension into the data warehouse.

The Table Mapping grid also includes a **Hierarchy** column to use with hierarchies.

Select **Create Logical Joins** if the imported tables are being mapped to a new business model that's created during the Map to Logical Model step. If the required logical joins in place, don't select the **Create Logical Joins** option to avoid creating erroneous multiple logical joins.

See Use Incremental Import to Propagate Flex Object Changes.

1. In the Administration Tool, in the Physical layer, right-click **Properties**.
2. In Properties, select the Connection Pool tab, `ADF OLTP` source and select **Import Metadata**.
3. Complete the fields in Select Metadata Objects, and click **Next**.
4. In Map to Logical Model, review the Table Mapping and Column Mapping grids display the results of a default drag-and-drop.
5. Optional: In the **VO Type**, select the option to use.
6. Optional: In the **Hierarchy** column, select this option for objects in hierarchies.
7. Optional: Select **Create Logical Joins** when the imported logical joins don't already exist.
8. Click **Finish** to close the wizard.

Customize the Mapping Behavior

When setting up automatic mapping to the Logical Model, you can create a set of XML files that specify custom requirements for the mappings displayed in the Map to Logical Model screen.

The Administration Tool reads the XML files and then automatically maps the KFF, DFF, and EFF segments according to the specified logic. Each XML file has a top-level element with an `appName` attribute that specifies the application to which the file applies.

You must create your XML files according to the logic in the XML schema files `app_segment_rule.xsd` and `mapping_rules.xsd`. You can find these files in:

ORACLE_HOME`\bi\bifoundation\javahost\lib\obisintegration\biextender`

All XML files in this directory with the prefix mapping_rules and app_segment_rules are parsed by the Administration Tool for ADF data sources.

You can use the existing `app_segment_rules_*.xml` and `mapping_rules_*.xml` in this directory as examples.

See XML Schema Files for ADF Mapping Customizations.

Manually Map Flex Object Changes to the Logical Model

You can drag and drop the physical objects to the Business Model and Mapping layer and Presentation layer and skip the logical mapping step in the Import Metadata Wizard.

The Oracle BI Administration Tool supports incremental drag-and-drop for ADF data sources, which enables physical database and schema objects to be dragged and dropped into an existing business model, resulting in updates made only for the incremental changes.

The current logic includes data source-specific default rules that can enable, for example, logical dimensions and hierarchies to be automatically created.

Automatically Map Flex Object Changes Using the biserverextender Utility

You can use the biserverextender utility to import flex object changes from your ADF sources and map them to the Business Model and Mapping layer and Presentation layer.

Because this feature doesn't require the Administration Tool, it's especially useful when you want to map flex object changes on Linux systems where the Administration Tool isn't available.

To use the `biserverextender` utility, you must first create an XML parameter file that contains the connection pool for an existing ADF data source. The `biserverextender` utility retrieves the existing ADF connection pool name from the parameter file, synchronizes the ADF data source, updates the deployed objects in the source, and then maps physical metadata to the Business Model and Mapping and Presentation layers based on the default rule files in the following directory:

ORACLE_HOME`/bi/bifoundation/javahost/lib/obisintegration/biextender`

See Customize the Mapping Behavior for information about rule files.

Syntax

```
biserverextender -R base_repository_name [-P repository_password]
-O output_repository_name -I input_XML_file [-S]
```

Where:

`-R` *`base_repository_name`* is the name and path of the repository into which you want to import and map flex object changes.

`-P` *`repository_password`* is the Oracle BI repository password for the base repository.

The *repository_password* argument is optional. If you don't provide the password argument, you're prompted to enter the password when you run the command. To minimize the risk of security breaches, Oracle recommends that you don't provide password arguments from the command line or in scripts. The password argument is supported for backward compatibility only. For scripting purposes, you can pass the password through standard input.

`-O` *`output_repository_name`* is the name and path of the repository generated by the utility.

`-I` *`input_XML_file`* is the name and path of an input XML parameter file that contains the fully-qualified name of a connection pool for an ADF data source.

`-S` is optional. If `-S` isn't specified, only the changes from the ADF source's DFF, KFF, and EFF objects are synchronized to the Oracle BI Repository. If `-S` is specified, Oracle BI Administration Tool reimports all of the DFF, KFF, and EFF objects from the ADF source based on the ADF source's database properties, and re-synchronizes the Oracle BI Repository.

`-S` also incorporates the following changes in the `app_segment_rules.xml` rules file:

- New mapping rules segments

- New alias table creation
- New *ADF VO To Be Exposed* subject area or presentation table

Example

```
biserverextender -R /scratch/my_repos.rpd -O /scratch/my_repos_modelled.rpd
-I /scratch/ADFSource.xml -S
Give password: password
```

Sample XML Parameter File

```
<BIExtenderParameters>
 <ConnectionDetails>
  <ConnectionPool>
   <ConnectionPoolName>"oracle.apps.fscm.model.analytics.applicationModule.Fscm
   TopModelAM_FscmTopModelAMLocal"."Connection Pool"</ConnectionPoolName>
  </ConnectionPool>
 </ConnectionDetails>
</BIExtenderParameters>
```

Configure SSL in Oracle WebLogic Server

You can configure one-way and two-way SSL in Oracle WebLogic Server.

This section contains the following topics:

- Configure One-Way SSL in Oracle WebLogic Server
- Configure Two-Way SSL in Oracle WebLogic Server

Configure One-Way SSL in Oracle WebLogic Server

One-way SSL is required to properly secure the communication between Oracle Analytics Server and Oracle WebLogic Server.

1. From the Oracle WebLogic Server Administration Console home page, click **Servers** under the **Environment** heading.
2. In the Servers table, select the name of the server you want to manage.
3. On the General tab in the Configuration tab, select **SSL Listen Port Enabled**.
4. Use the Administration Tool to update the appropriate connection pool object in the Physical layer to use `https://` instead of http://.
5. Update the port number to use the SSL port number, *7002*, by default.

Configure Two-Way SSL in Oracle WebLogic Server

You can set up two-way SSL to secure the communication between the Oracle BI Server and Oracle WebLogic Server.

Perform queries against ADF using your Oracle BI Server client of choice such as nqcmd. The Oracle BI Server should communicate with the ADF Oracle WebLogic Server using mutual SSL / client certificates.

See *Managing Security for Oracle Analytics Server*.

In the Oracle WebLogic Server Administration Console modify the ADF Oracle WebLogic Server to accept SSL connections and to perform mutual SSL.

If you generate a client certificate file, the `cacert.pem` file is stored in:

```
ORACLE_HOME/user_projects/domains/bifoundation_domain/config/fmwconfig/
biinstances/coreapplication/ssl
```

Your trust keystore might use a location similar to the following:

```
/scratch/user_name/view_storage/user_name_fmw/fmwtools/mw_home/wlserver_10.3/
server/lib
```

1. Optional: (Optional) Create client certificates in the Oracle BI Server, if they don't already exist.
2. Log in to the Oracle WebLogic Server Administration Console and click **Servers** under the **Environment** heading, then click the server name.
3. In the Change Center, click **Lock & Edit** to enable configuration changes.
4. In the **General** tab, select **SSL Listen Port Enabled**, record the **SSL Listen Port** number, and then click **Save**.
5. Select the SSL tab, then select **Advanced**.
6. For **Two Way Client Cert Behavior**, select **Client Certs Requested and Enforced**, and then click **Save**.
7. Select the **Keystores** tab and record the location and file name for the Trust Keystore.
8. Click **Activate Changes**.
9. On the Oracle BI Server computer, find the CA file for the client certificate verify that the Certificate Authority (CA) for the Oracle BI Server client certificate is trusted by the ADF Oracle WebLogic Server.
10. Copy the `cert.pem` file to a known location.
11. On the ADF Oracle WebLogic Server computer, open a command window and go to the location of the trust keystore.
12. Copy the client CA file, for example, *cacert.pem* to the trust keystore location.
13. Use the following command in the JDK keytool utility to import the client CA into the trust keystore for the ADF server, making it a trusted CA:

    ```
    keytool -import -file client_CA_file -keystore
    keystore_file -keystorepass keystore_password
    ```

    For example:

    ```
    /scratch/my_name/view_storage/my_name_fmw/jdk6/bin/keytool -import -file
    ~/Downloads/SSL/cacert.pem -keystore DemoTrust.jks -keystorepass
    DemoTrustKeyStorePassPhrase
    ```

14. In the Administration Tool, in the Physical layer, open the first ADF connection pool object and select the Miscellaneous tab to update the Physical layer of the Oracle BI repository.
15. Update the **URL** field to use the *https* protocol and the SSL port, and then click **OK**.
16. Repeat the previous two steps for each additional ADF connection pool object.

17. Save the repository and restart the Oracle BI Server.
18. Configure the Oracle BI Server ODBC DSN to use SSL.

    For example, on Windows do the following:

    a. Open the ODBC Data Source Administrator and select the **System DSN** tab.
    b. Double-click the *DSN* for the Oracle BI Server.

       The DSN should start with *coreapplication_OH*.
    c. Select **Use SSL**.
    d. Click **Next**, click **Next** again, and then click **Finish**.

Enable the Ability to Pass Custom Parameters to the ADF Application

Some ADF applications have custom properties defined on the ApplicationModule such as EFFECTIVE_DATE or TREE_VERSION.

You can include these custom properties in your application queries, and the Oracle BI Server passes them to the Oracle ADF application.

You can't use this feature to pass any custom property to your Oracle ADF application. Only certain custom properties, like `EFFECTIVE_DATE` and `TREE_VERSION`, are supported.

1. Open your repository in the Administration Tool.
2. Select **Manage**, and then select **Variables**.
3. Select **Action**, select **New**, select **Repository**, and then select **Variable**.
4. For **Name**, enter `ADF_PARAM_LIST`. Don't enter the name of the custom property as the name of the variable.
5. Ensure that the **Type** is **Static**.
6. For **Default Initializer**, enter the name or names of the custom properties as a character string. If you've multiple custom properties, include them as a comma delimited list. For example:

   ```
   'PARAM_EFFECTIVE_DATE'
   ```

   ```
   'PARAM_EFFECTIVE_DATE, ApplicationIdBind, KeyFlexfieldCodeBind'
   ```
7. Click **OK**.
8. Save and close the repository.

After you register the custom properties as a repository variable, you can include these variables in queries. For example:

```
set variable PARAM_EFFECTIVE_DATE=2001-01-01 : SELECT c1 FROM t1;
```

or

```
set variable ApplicationIdBind = '0', KeyFlexfieldCodeBind = 'KFF1' :
select_physical ApplicationID, KeyFlexfieldCode, DataSecurityObjectName,
SegmentLabelCode from adfdb..."AppModule.KFFHierFilterVO1";
```

When you're including a custom property of type `PARAM_EFFECTIVE_DATE`, the date format for the property value must use the format, *yyyy-mm-dd*.

Propagate Labels and Tooltips from ADF Data Sources

You can propagate user interface hints such as labels and tooltips, from ADF data sources to display when users work with analyses.

When translated labels and tooltips, based on user locale, are maintained within an ADF data source, you can query the data source to access this translated data. You use the Administration Tool to configure presentation columns to use when creating analyses.

This section contains the following topics:

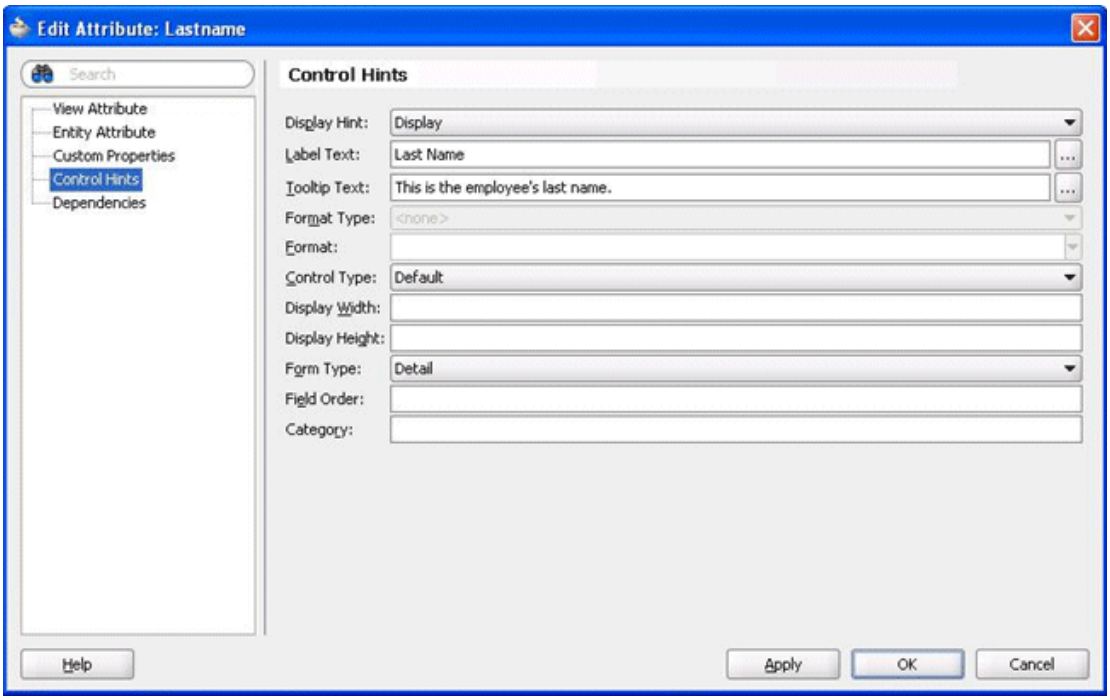
- What are Labels and Tooltips?
- About the Session Variable Naming Scheme for UI Hints
- About Determining the Physical Column for a Presentation Column
- About Initializing Session Variables Automatically for Propagating UI Hints
- Use UI Hints from an Oracle ADF Data Source When Creating Analyses
- Use XML Code in Initialization Blocks to Query UI Hints

What Are Labels and Tooltips?

The propagation of UI hints enables a presentation column in the Oracle BI Administration Tool to use a label and tooltip as its Custom display name and Description respectively.

A label is the text that's used in prompts or table headers that precedes the value of a data item. A tooltip is the text that's displayed when a user hovers over the item. Each attribute of a view object has an associated label and tooltip. A view object is the Oracle ADF component that enables a developer to work easily with SQL query results.

The image shows the Label Text and Tooltip Text options in the Edit Attribute dialog in JDeveloper.



About the Session Variable Naming Scheme for UI Hints

Learn about the session variable naming scheme.

Session variable names are generated by the Oracle Analytics Server broker servlet in Oracle WebLogic Server in the following format:

`ADF_UI Hint Type_Database Name_View Object Name_Attribute's Name`

Where:

UI Hint Type is a `LABEL` or *TOOLTIP*.

Database Name is the value for the `database` attribute of the `ADFQuery` element in the XML query. Special characters such single quotes ('), double quotes (''), and spaces are replaced by the underscore character.

View Object Name is the name attribute of the view object. Oracle ADF prohibits special characters and spaces in the name.

Attribute's Name is the name of the attribute for the session variable. Oracle ADF prohibits special characters and spaces in the name.

Every character in the session variable name is uppercase. The XML query example, in Use XML Code in Initialization Blocks to Query UI Hints, generates four session variables with the following names:

ADF_LABEL_MY_ORCLADF_EMPLOYEESVIEW_FIRSTNAME

ADF_TOOLTIP_MY_ORCLADF_EMPLOYEESVIEW_FIRSTNAME

ADF_LABEL_MY_ORCLADF_EMPLOYEESVIEW_LASTNAME

ADF_TOOLTIP_MY_ORCLADF_EMPLOYEESVIEW_LASTNAME

About Determining the Physical Column for a Presentation Column

Each presentation column must map to a physical column as required by the naming scheme for session variables.

When you choose to **Generate ADF Label** or **Generate ADF Tooltip** for a presentation layer object, the physical column is located using the following rules:

- Examine the presentation column and determine its logical column. If the logical column is derived from an existing logical column, then the physical column can't be found.
- If the default aggregation rule for the logical column isn't None or Sum, then the physical column can't be found. It doesn't make sense semantically to use the ADF UI hints for aggregation rules other than Sum.
- A logical column can be mapped to physical columns by multiple logical table sources. Only logical table sources that aren't disabled are searched.
- Don't search logical table sources that map the logical column using non-trivial expressions, that is, anything more than a physical column name. If no logical table sources are searched, then the physical column can't be found.
- From the remaining ordered list of logical table sources, examine the physical column that's mapped by the first logical table source. A physical column must map to a view object attribute. The physical column must exist as part of a physical database of type Oracle ADF 12c.
  - If this condition is satisfied, then the physical column for obtaining UI hints is found.
  - If this condition isn't satisfied, then continue to examine the physical column that's mapped by the next logical table source until the physical column that's mapped to a view object attribute is found.

    If all logical table source are searched without satisfying the condition, then the physical column can't be found.

If the physical column for obtaining UI hints is found using these rules, then the custom display name or description is populated with a session variable that has a name based on a predetermined naming scheme. See About the Session Variable Naming Scheme for UI Hints.

If the physical column for obtaining UI hints isn't found using these rules, then the **Generate ADF Label** and **Generate ADF Tooltip** options are shown as disabled in the right-click menu.

As an alternative to using the physical column found using these rules, you can use XML code in an initialization block to initialize your own session variables with ADF UI hints. You must then enter these session variable names in the **Custom display name** and **Custom description** fields manually. See Use XML Code in Initialization Blocks to Query UI Hints.

About Initializing Session Variables Automatically for Propagating UI Hints

Learn when session variables are created.

If the **Generate ADF Label** and **Generate ADF Tooltip** options were used to successfully generate the session variable names for UI hints from Oracle ADF, then the session variables are created and initialized when Oracle BI Presentation Services queries them during the session.

The variables aren't created and initialized during the session logon stage for performance reasons. Variables are created, using Allow deferred processing, and initialized when the variables are needed by a specific query in a session.

When Oracle BI Presentation Services queries the custom display names and custom descriptions through ODBC, the Oracle BI Server checks to determine if the associated session variables have been created. If the variables weren't created, the Oracle BI Server dynamically generates the appropriate XML query to retrieve the UI hints from the Oracle ADF data source. The Oracle BI Server uses the UI hints to create and initialize the session variables. To optimize performance, the Oracle BI Server queries UI hints for each view object. If the Oracle BI Server needs the UI hints of a view object's attributes, then the UI hints for all the attributes under the view object are queried and propagated through session variables.

Use UI Hints from an Oracle ADF Data Source When Creating Analyses

You can use UI hints from an Oracle ADF data source when creating analyses.

Before you can perform this task, the following prerequisites must be met:

- UI hints must have been configured in the Oracle ADF data source.
- A working repository must have been configured for the Oracle ADF data source in the Administration Tool.
- Right-click the column in the Presentation layer and select **Externalize Display Names**, select **Generate ADF Label**, select **Externalize Descriptions**, and then select **Generate ADF Tooltip** to generate tooltip strings for all of the columns.

Use XML Code in Initialization Blocks to Query UI Hints

You can use specialized XML code in place of SQL statements in initialization blocks to query the data source for UI hints, within a single repository and subject area.

See About the Session Variable Naming Scheme for UI Hints.

After configuring the initialization blocks, you must manually enter the session variable names in the **Custom display name** and **Custom description** text fields for the appropriate presentation column.

Follow the procedure in the example in Use UI Hints from an Oracle ADF Data Source When Creating Analyses, but replace the first step with the following ones:

Create session initialization blocks in the Administration Tool, see Create Session Variables.

1. In the Session Variable Initialization Block Data Source dialog, enter the **Initialization** string.
2. In the Session Variable Initialization Block dialog, from the Variable Target list, select **Row-wise initialization**.
3. Click **Test** to test the query against the Oracle ADF data source.
4. Configure a custom display name and write a description in presentation columns.

5. Right-click a physical table, select **Query Related Objects**, select **Presentation**, and then select **Presentation Table**.
6. In Query Related Objects, select the required presentation table and click **Go To**.
7. Expand the presentation table to view the presentation columns.
8. Double-click the presentation column to display the Presentation Column dialog.
9. Select **Custom display name** and enter a value similar to the following:

   ```
   VALUEOF(NQ_SESSION.ADF_LABEL_MY_ORCLADF_EMPLOYEESVIEW_LASTNAME)
   ```

10. Select **Custom description** and enter a value similar to the following:

    ```
    VALUEOF(NQ_SESSION.ADF_TOOLTIP_MY_ORCLADF_EMPLOYEESVIEW_LASTNAME)
    ```

11. Click **OK**.
12. Save the changes in the repository and restart the Oracle BI Server.

ADFQuery Element Reference

Use the ADFQuery element and its mode, database, and locale attributes in your XML code.

The element requires zero or more child elements. The following is the syntax of the element:

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<ADFQuery mode="mode" database="database_name"
locale="VALUEOF(NQ_SESSION.WEBLANGUAGE)">
     <ViewObject><![CDATA[view_object_name]]></ViewObject>
     <Attribute>
     <ViewObject><![CDATA[attribute_view_object_name]]></ViewObject>
     <Name><![CDATA[attribute_name]]></Name>
     </Attribute>
</ADFQuery>
```

Where:

`mode` specifies what you want to query:

- *label* for querying attributes' label
- *tooltip* for querying attributes' tooltip
- *ui_hints* for querying attributes' label and tooltip

`database_name` specifies the name of the physical database object in the Administration Tool, which contains the physical columns that correspond to the attributes in the Oracle ADF data source.

view_object_name specifies the name of the view object to obtain the UI hints of all attributes in it.

attribute_view_object_name specifies the name of the view object that contains the attribute.

attribute_name specifies the name of the attribute that belongs to the associated view object to obtain the UI hints of this attribute.

Querying Labels for All View Objects

Don't include child elements in the ADFQuery element when querying the UI hints of all attributes in all View Objects. For example, to query the labels of all attributes in all View Objects under the *My_orclADF* physical database object, use the following XML code:

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<ADFQuery mode="label" database="My_orclADF"
locale="VALUEOF(NQ_SESSION.WEBLANGUAGE)">
</ADFQuery>
```

Querying Tooltips for Specific View Objects

The ADFQuery element can contain zero or more child elements named ViewObject if UI hints of all attributes in specific View Objects are queried. Each ViewObject element has a text content that contains the View Object's name. The ViewObject element is used to specify the View Objects from which the UI hints of all attributes are queried. For example, to query the tooltips of all attributes in the View Object that's named EmployeesView and CustomersView under the My_orclADF physical database object, use the following XML code:

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<ADFQuery mode="tooltip" database="My_orclADF"
locale="VALUEOF(NQ_SESSION.WEBLANGUAGE)">
   <ViewObject><![CDATA[EmployeesView]]></ViewObject>
   <ViewObject><![CDATA[CustomersView]]></ViewObject>
</ADFQuery>
```

Querying UI Hints for Specific Attributes

The ADFQuery element can contain zero or more child elements named Attribute. Each Attribute element has two required child elements named `ViewObject` and `Name`. The Attribute element is used to specify the attributes from which the UI hints are queried. The `ViewObject` child element has a text content that contains the View Object's name. This element specifies the View Object that the attribute belongs to. The `Name` child element has a text content which contains the attribute's name. For example, to query the labels and tooltips of the attributes named *Firstname* and *Lastname* in the EmployeesView View Object under the *My_orclADF* physical database object, use the following XML code:

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<ADFQuery mode="ui_hints" database="My_orclADF"
locale="VALUEOF(NQ_SESSION.WEBLANGUAGE)">
   <Attribute>
      <ViewObject><![CDATA[EmployeesView]]></ViewObject>
      <Name><![CDATA[Firstname]]></Name>
   </Attribute>
   <Attribute>
      <ViewObject><![CDATA[EmployeesView]]></ViewObject>
      <Name><![CDATA[Lastname]]></Name>
   </Attribute>
</ADFQuery>
```

7

Set Up Database Objects and Connection Pools

This chapter describes the properties of the database and connection pool objects in the Physical layer.
Properties for database objects and connection pools are set automatically when you import metadata from your data sources. You might want to adjust database or connection pool settings, or create a database object or connection pool manually.

This chapter contains the following sections:

- Set Up Database Objects
- About Connection Pools
- Create or Change Connection Pools
- Set Up Persist Connection Pools
- List Connection Pool Command
- Update Connection Pool Command
- Use the BIServerT2PProvisioner.jar Utility to Change Connection Pool Passwords

Set Up Database Objects

Importing metadata from a data source automatically creates a database object for the schema, but you may need to adjust or view the database properties.

The following sections provide information about how to create, edit, or view properties for database objects in the Physical layer:

- About Database Types in the Physical Layer
- Create a Database Object Manually in the Physical Layer
- SQL Features Supported by a Data Source
- View Database Properties
- Review Supported Database Features

About Database Types in the Physical Layer

If you import the physical schema into the Physical layer, the Administration tool usually assigns database type automatically.

The following list contains additional information about automatic assignment of database types:

- Relational data sources

  During the import process, ODBC drivers provide the Oracle BI Server with the database type. If the server can't determine the database type, an approximate ODBC type is

assigned to the database object. Replace the ODBC type with the closest matching entry from the **Database** list.

- Multidimensional data sources

  Microsoft Analysis Services are the only supported XMLA-compliant data sources currently available. After you import metadata from a multidimensional data source, check the database object and update the appropriate database type and version if necessary.

Create a Database Object Manually in the Physical Layer

When you create a database object manually, you must also manually set up an associated connection pool.

For multidimensional data sources, if you create the physical schema in the Physical layer of the repository, you need to create one database in the physical layer for each cube, or set of cubes, that are in the same catalog (database) in the data source. A physical database can have more than one cube. The cubes must belong to the same catalog in the data source. To learn the properties to specify and their values when creating a database, see Database General Properties Reference.

> **Important:**
>
> Oracle strongly recommends importing your physical schema.

1. In the Administration Tool, in the Physical layer without any objects selected, right-click and select **New Database**.
2. In the Database dialog on the **General** tab, type a **Name** for the database.
3. In Data source definition, from the **Database Type** list, select *Database* as the value.
4. Optional: Select **CRM metadata tables** only for relational data sources and legacy Siebel Systems sources.
5. Optional: Select **Virtual Private Database** to identify the physical database source as a virtual private database (VPD).

   Always select **Virtual Private Database** for Essbase, Hyperion Financial Management, and Hyperion Planning data sources that are configured for SSO in the corresponding connection pool.
6. Optional: Select **Allow populate queries** by default to give users the ability to populate the database.
7. Optional: Select **Allow direct database requests by default** to allow users to run queries.

Database General Properties Reference

Review the database properties in the table to learn which properties to configure and when you can or should specify values.

Option	Description
Data source definition: Database	The database type for your database.
CRM metadata tables	When selected, indicates that the definition of physical tables and columns for Siebel CRM tables was derived from the Siebel metadata dictionary.
Data source definition: Virtual Private Database	Identifies the physical database source as a virtual private database (VPD). When a VPD is used, returned data results are contingent on the user's authorization credentials. Therefore, it's important to identify these sources. These data results affect the validity of the query result set that's used with caching. Always select this option for Essbase, Hyperion Financial Management, and Hyperion Planning data sources that are configured for **SSO** in the corresponding connection pool. If you select this option, you also should select the **Security Sensitive** option in the Session Variable dialog.
Persist connection pool	To use a persistent connection pool, you must set up a temporary table first.
Allow populate queries by default	When selected, allows everyone to run `POPULATE SQL`. If you want most, but not all, users to be able to run `POPULATE SQL`, select this option and then limit queries for specific users or groups.
Allow direct database requests by default	When selected, allows all users to run physical queries. The Oracle BI Server sends unprocessed, user-entered, physical SQL directly to an underlying database. The returned results set can be rendered in Oracle BI Server, and then charted, rendered in a dashboard, and treated as an Oracle Analytics Server request. If you want most, but not all, users to be able to run physical queries, select this option and then limit queries for specific users or groups.

When to Allow Direct Database Requests by Default

The property, `Allow direct database requests by default`, provides the ability for users to run physical queries.

If configured incorrectly, it can expose sensitive data to an unintended audience.

Use the following recommended guidelines when setting this database property:

- The Oracle BI Server should be configured to accept connection requests only from a computer on which the Oracle BI Server, Oracle BI Presentation Services, or Oracle BI Scheduler are running. This restriction should be established at the TCP/IP level using the Oracle BI Server IP address. This allows only a TCP/IP connection from the IP address of Oracle BI Server.
- To prevent users from running `nqcmd`, a utility that runs SQL scripts, by logging in remotely to this computer, you should disallow access by the following to the computer on which you installed Oracle BI Presentation Services:

  - TELNET
  - Remote shells
  - Remote desktops
  - Teleconferencing software such as Windows NetMeeting

  If necessary, you might want to make an exception for users with administrator permissions.

- Only users with administrator permissions should be allowed to perform the following tasks:
  - TELNET into the Oracle BI Server and Oracle BI Presentation Services computers to perform tasks such as running `nqcmd` for cache seeding.
  - Access the advanced SQL page of Answers to create requests.
- Set up group/user-based permissions on Oracle BI Presentation Services to control access to editing, preconfigured to allow access by Oracle BI Presentation Services administrators, and processing, preconfigured to not allow access by anyone, direct database requests.

SQL Features Supported by a Data Source

When you import metadata or specify a database type in the General tab of the Database dialog, the set of SQL features for that database object is automatically populated with default values appropriate for the database type.

The Oracle BI Server uses the supported SQL features with the specified data source.

When a feature is marked as supported, checked in the Default column on the Features tab of the Database dialog, the Oracle BI Server pushes the function or calculation down to the data source for improved performance. When a function or feature isn't supported in the data source, the calculation or processing is performed in the Oracle BI Server.

The supported features list uses the defaults defined in the `DBFeatures.defaults` file, located in *`ORACLE_HOME`*`/bi/bifoundation/server/bin`. You shouldn't modify this file. You can review the `DBFeatures.defaults` file to compare the features supported by different data source types.

You can tailor the query features for a data source such as when upgrading to a new version of a data source to see if the updated feature is reflected in the Oracle BI Server defaults. When the supported feature isn't shown in the Features tab, you can update the settings in the Features tab to reflect the actual features supported by the new version of the data source. If a data source supports a particular feature such as left outer join queries but you want to prohibit the Oracle BI Server from sending such queries to a particular data source, you can change this default setting in the Features tab. If you've federated data sources that run functions differently, to ensure that query results are consistent, you can disable the appropriate functions on the Features tab so that the calculations are performed in a consistent manner in the Oracle BI Server.

Important:

If you enable SQL features that the data source doesn't support, your query may return errors and unexpected results. If you disable supported SQL features, the server could issue less efficient SQL to the data source.

In most cases, you should keep the default values. If you do change the defaults to mark a feature as supported in the Features tab, make sure that the feature is actually supported by the data source.

Note:

Don't change the `OPTIMIZE_MDX_FILTER_QUALIFICATION` value.

See Review Supported Database Features.
The table lists the options available on the Features tab of the Database dialog.

Option	Description
Feature	The name of the database feature, such as `COUNT_DISTINCT_SUPPORTED`.
Value	Shows the current value for the given feature. Selected indicates that the feature is supported in the data source, and that the function or feature should be performed in the data source rather than in the Oracle BI Server. Some features show a default value in the **Value** column rather than selected/not selected, such as 10 for `MAX_ENTRIES_PER_IN_LIST`. It's strongly recommended that you keep the default selections and default values.
Default	Shows the default value for the given feature. The defaults listed in this column are specified in the file `DBFeatures.defaults`.
Find	Searches for a feature in the list.
Find Again	This option becomes available after you click **Find**. It lets you perform multiple searches for the same string.
Query DBMS	Use Query DBMS only when you're installing and querying a data source that has no set of feature defaults in the Oracle BI Server. Query DBMS enables querying the type of data source for Feature table entries so that you can find out which SQL features are supported. You can then change the entries that appear in the Features tab based on your query results. **Query DBMS** isn't available if you're using an XML or a multidimensional data source. The Query DBMS feature results aren't always an accurate reflection of the SQL features actually supported by the data source. When using this feature, you should verify that the list of supported features in the Features tab matches the actual features supported by your data source. Refer to the documentation for your data source for details.
Reset to defaults	This button restores the default values for this data source type from the `DBFeatures.defaults` file.

View Database Properties

You can extend the Physical layer metadata for some data sources.

For example, for Oracle ADF data sources, you can view custom database properties that are passed to the Administration Tool from Oracle ADF BI view objects. These properties aren't usually edited.

The table shows examples of custom properties.

Category	Key Name	Value	Description
FscmTopModelAM.AccountBIAM	BIObject_FLEX_TREE_VS_COST_CENTER_LABEL_VI	Dim - Cost Center	FLEX_TREE_VS_COST_CENTER_LABEL_VI view object needs to map to the Dim - Cost Center logical dimension.
FscmTopModelAM.AccountBIAM	BIFlexfieldViewUsage	FLEX_BI_AcctKff_VI	FLEX_BI_AcctKff_VI is the CCID view object for FscmTopModelAM.AccountBIAM.
FscmTopModelAM.AccountBIAM	EnforceCustomDataType_FscmTopModelAM.AccountBIAM	"Segment 1":"VARCHAR"; "Segment ID":"DOUBLE"	For FscmTopModelAM.AccountBIAM view objects, the data type of some physical columns, the values are overridden with values passed in the property.

Review Supported Database Features

In the Oracle BI Administration Tool, you can review the features supported by databases and data sources. You can use Database Features when trying to troubleshoot a query or other operation that isn't working as expected.

Features are the SQL expressions, statements, function, operations, and other features that you can run against the database such as a query that uses an `ISDESCENDANT` statement, operations such as `ADD` or `SQRT` (square root) operations are supported. If a check displays in the **Value** or **Default** columns, the feature is supported. For specific information about the **Value** or **Default** columns, see SQL Features Supported by a Data Source.

1. Open the Oracle BI Administration Tool.
2. From the **File** menu, select **Online Mode** or **Offline Mode**.
3. In the Open Repository dialog, select a repository, and click **Open**.
4. In the Physical column, right-click a database or data source, and select **Properties**.
5. In Database Properties, click the Features tab to review the supported features for the specific database or data source.

About Connection Pools

The connection pool is an object in the Physical layer that describes access to the data source.

The connection pool contains information about the connection between the Oracle BI Server and that data source.

The Physical layer in the BI Server contains at least one connection pool for each database. When you create the Physical layer by importing a schema for a data source, the connection pool is created automatically. You can configure multiple connection pools for a database. Connection pools allow multiple concurrent data source requests (queries) to share a single database connection, reducing the overhead of connecting to a database. It's recommended that you create a dedicated connection pool for initialization blocks. See About Connection Pools for Initialization Blocks.

For each connection pool, you must specify the maximum number of concurrent connections allowed. After this limit is reached, the connection request waits until a connection becomes available.

Increasing the allowed number of concurrent connections can potentially increase the load on the underlying database accessed by the connection pool. Test and consult with your DBA to make sure the data source can handle the number of connections specified in the connection pool. Also, if the data sources have a charge back system based on the number of connections, you might want to limit the number of concurrent connections to keep the charge-back costs down.

In addition to the potential load and costs associated with the database resources, the Oracle BI Server allocates shared memory for each connection upon server startup. This raises the number of connections and increases Oracle BI Server memory usage.

About Connection Pools for Initialization Blocks

You should create a dedicated connection pool for initialization blocks.

Don't use this connection pool for queries.

You should isolate the connections pools for different types of initialization blocks. By isolating the connection pools, you can ensure that authentication and login-specific initialization blocks don't slow down the login process. The following types of initialization blocks should have separate connection pools:

- All initialization blocks that set session variables.
- All initialization blocks that set repository variables. Run initialization blocks that set repository variables using credentials with administrator privileges.

  Be aware of the number of these initialization blocks, their scheduled refresh rate, and when they're scheduled to run. It would take an extreme case for this scenario to affect resources. For example, refresh rates set in minutes, greater than 15 initialization blocks that refresh concurrently, and a situation in which either of these scenarios could occur during prime user access time frames.

You should design initialization blocks to set the maximum number of Oracle BI Server variables for each block. For example, if you've five variables, it's more efficient and less resource intensive to construct a single initialization block containing all five variables. When using one initialization block, the values are resolved with one call to the back end tables using the initialization string. Constructing five initialization blocks, one for each variable, would result in five calls to the back end tables for assignment.

If an initialization block fails for a particular connection pool during Oracle BI Server start-up, no more initialization blocks using that connection pool are processed. Instead, the connection pool is blocklisted and subsequent initialization blocks for that connection pool are

skipped. This behavior ensures that the Oracle BI Server starts in a timely manner, even when a connection pool has a large number of associated initialization blocks or variables.

If this occurs, a message similar to the following appears in the server log:

```
[OracleBIServerComponent] [ERROR:1] [43143] Blacklisted connection pool
name_of_connection_pool
```

If you see this error, check the initialization blocks for the given connection pool to ensure they're correct.

Create or Change Connection Pools

If you didn't import physical schemas, you must create a database object before you create a connection pool.

Database objects and connection pools are created automatically when you import physical schemas, for both relational and multidimensional data sources.

You create or change a connection pool in the Physical layer of the Administration Tool.

To modify more than one connection pool, use the List Connection Pool Command and the Update Connection Pool Command.

If you've already defined an existing database and connection pool, you can right-click the connection pool in the Physical layer and select **Import Metadata** to import metadata for this data source. The Import Metadata Wizard appears with the information on the Select Data Source screen pre-filled. See Import Metadata and Working with Data Sources.

To automate connection pool changes for use in a process such as production migration, consider using the XML API. See About the Oracle BI Server XML API in *Managing Security for Oracle Analytics Server*.

Note: Oracle Analytics doesn't support the variables :user and :password in data source connection credentials.

1. In the Physical layer of the Administration Tool, right-click a database, select **New Object**, and then select **Connection Pool**.
2. Specify or adjust the properties as needed, then click **OK**.

Set Connection Pool Properties in the General Tab

You can learn about the properties in the General tab of the Connection Pool dialog.

The properties listed in the General tab vary according to the data source type. For example, XMLA data sources have a connection pool property for **URL**, while relational and XML data sources have the option **Require fully qualified table names**.

- In the Connection Pool dialog, click the General tab, and then complete the fields.

Common Connection Pool Properties in the General Tab

The topic describes connection pool properties in the General tab that are common among most data source types.

The table describes the properties in the General tab of the Connection Pool dialog that are common for different data source types.

Property	Description
Name	The name for the connection pool. The name is assigned automatically for connection pools created on import.
Permissions	Use this option to assign permissions for individual users or application roles to access the connection pool. For example, you can set up a privileged group of users to have its own connection pool. These permissions aren't intended for use as data access security. For example, connection pool permissions don't protect cache entries. See Apply Data Access Security to Repository Objects.
Call interface	Identifies the application programming interface (API) to access the data source. You can access some databases using native APIs, using ODBC, or with APIs and ODBC together. Java data sources are accessed using JDBC/JNDI. If the call interface is XML, the XML tab is displayed for you to update the applicable properties.
Maximum connections	The maximum number of connections allowed for this connection pool. The default is 10. You can determined the value by the database make and model and the configuration of the hardware for the computer on which the database runs, as well as the number of concurrent users who require access. For Microsoft Analysis Services data sources, you might encounter 503 Service Not Available errors if the Max Connections setting in the connection pool (default 10) is greater than the `XMLA MaxThreadsPerClient` setting configured in Analysis Services (default 4). To avoid these errors, increase the `MaxThreadsPerClient` setting in the `msmdpump.ini` file, or reduce the Max Connections setting in the repository connection pool. See Improve Use of System Memory Resources with TimesTen Data Sources. For deployments with Oracle Analytics Interactive Dashboards pages, consider estimating this value at 10% to 20% of the number of simultaneous users multiplied by the number of requests on a dashboard. You can adjust the number based on usage. Define the total number of all connections in the repository to less than 800. To estimate the maximum connections needed for a connection pool dedicated to an initialization block, you might use the number of users concurrently logged on during initialization block processing.

Property	Description
Require fully qualified table names	Select this option if the database or database configuration requires fully qualified table names. This option isn't available for some data source types. When this option is selected, all requests sent from the connection pool use fully qualified names to query the underlying database. The fully qualified names are based on the physical object names in the repository. If you're querying the same tables from which the Physical layer metadata was imported, you can safely select this option. If you've migrated your repository from one physical database to another physical database that has different database and schema names, the fully qualified names are invalid in the newly migrated database. In this case, if you don't select this option, the queries succeed against the new database objects. For some data sources, fully qualified names are a safer because they guarantee that the queries are directed to the desired tables in the desired database. For example, if the RDBMS supports a master database concept, a query against a table named Customer first looks for that table in the master database, and then looks for it in the specified database. If the table named Customer exists in the master database, that table is queried, not the table named Customer in the specified database. It's sometimes necessary to select this option when you're using an Oracle Database, and you're accessing the database with a user that isn't the owner of the schema containing the tables. When the Oracle Database interprets table names in SQL, it assumes that the user that made the query is the owner if the table name isn't fully qualified in the query. This can result in an incorrect qualified name. For example, if the user SAMPLE creates a table called CUSTOMER, the fully qualified table name is SAMPLE.CUSTOMER. When the SAMPLE user references the CUSTOMER table in a query, the Oracle Database assumes the fully qualified table name is SAMPLE.CUSTOMER, and the access is successful. However, if the JANEDOE user references the CUSTOMER table in a query, the Oracle Database assumes the fully qualified table name is JANEDOE.CUSTOMER, and a Table or view not found error can result. To enable access for JANEDOE, you must select **Require fully qualified table names** in the connection pool so that the Oracle BI Server specifies SAMPLE.CUSTOMER in all queries.
Data source name	The name of the data source to which you want this connection pool to connect and send physical queries. The value you enter in this field depends on the selected call interface: • If the call interface is OCI, enter a full connect string or a net service name from the tnsnames.ora file you set up within the Oracle Analytics Server environment, in *BI_DOMAIN*`/bidata/components/core/serviceinstances/ssi/oracledb`. • If you're using a native interface for a different database, enter the name of the database for that system. • If the call interface is ODBC, the data source name field displays a list containing all the User and System DSNs defined for ODBC on the local computer. Select the correct one for the data source to which you want connect.

Property	Description
Shared logon	When selected, all users whose queries use the connection pool to access the underlying database use the same user name and password. If this option is selected, then all connections to the database that use the connection pool use the user name and password specified in the connection pool, even if the user has specified a database user name and password in the DSN or in user configuration. If this option isn't selected, connections through the connection pool use the database user ID and password specified in the DSN or in the user profile. When you use Oracle Call Interface (OCI) to connect to the database, you can deselect the Shared logon property of a connection pool for the Oracle database. OCI gets the credentials of the user in the repository. The Shared logon option is enabled by default in Essbase connection pools. You can't disable Shared logon for Essbase connection pools..
Enable connection pooling	When selected, allows a single database connection to remain open for the specified time for use by future query requests. Connection pooling saves the overhead of opening and closing a new connection for every query. If you don't select this option, each query sent to the database opens a new connection.
Timeout	Specify the amount of time and in what increment such as minutes that a connection to the data source remains open after a request completes. During this time, new requests use this connection rather than open a new one, up to the number specified for the maximum connections. The time is reset after each completed connection request. If you're using an ADF data source and the call interface is `OracleADF_HTTP` and the query mode is *SQLBypass*, then **Timeout** specifies the maximum processing time before the connection is canceled.
Use multithreaded connections	When this option is selected, the Oracle BI Server terminates idle physical queries (threads). When not selected, one thread is tied to one database connection, number of threads = maximum connections. Even if threads are idle, they consume memory. The parameter `DB_GATEWAY_THREAD_RANGE` in the Server section of `NQSConfig.ini` establishes when the Oracle BI Server terminates idle threads. The lower number in the range is the number of threads that are kept open before the Oracle BI Server takes action. If the number of open threads exceeds the low point in the range, the Oracle BI Server terminates idle threads. For example, if `DB_GATEWAY_THREAD_RANGE` is set to 40-200 and 75 threads are open, the Oracle BI Server terminates any idle threads.
Parameters supported	If this option isn't selected, and the database features table supports parameters, special code runs that allows the Oracle BI Server to push filters (or calculations) with parameters to the database. The Oracle BI Server does this by simulating parameter support within the gateway/adapter layer by sending extra SQLPrepare calls to the database.

Property	Description
Isolation level	For ODBC gateways only. The value sets the transaction isolation level on each connection to the back-end database. The isolation level setting controls the default transaction locking behavior for all statements issued by a connection. You can only set one at a time. It remains set for that connection until it's explicitly changed. The following options are available: **Dirty read**. Implements dirty read, isolation level 0 locking. This is the least restrictive isolation level. When this option is set, it's possible to read uncommitted or dirty data, change values in the data, and have rows appear or disappear in the data set before the end of the transaction. Dirty data is data to clean before running a query to obtain correct results, for example, duplicate records, records with inconsistent naming conventions, or records with incompatible data types. **Committed read**. Specifies that shared locks are held while the data is read to avoid dirty reads. You can change the data before the end of the transaction, resulting in non-repeatable reads or phantom data. **Repeatable read**. Places locks on all data that's used in a query, preventing other users from updating the data. You can insert new phantom rows into the data set by another user and are included in later reads in the current transaction. **Serializable**. Places a range lock on the data set, preventing other users from updating or inserting rows into the data set until the transaction is complete. This is the most restrictive of the four isolation levels. Because concurrency is lower, use this option only if necessary.

Multidimensional Connection Pool Properties in the General Tab

Learn how to use the connection pool properties.

The table describes the properties in the General tab of the Connection Pool dialog that are specific to multidimensional data sources. Some properties only appear for certain types of multidimensional data sources.

- URL

  This property is only displayed for XMLA data sources. Specify the URL to connect to the XMLA provider. This URL points to the XMLA virtual directory of the computer hosting the cube. This virtual directory must be associated with `msxisapi.dll`, part of the Microsoft XML for Analysis SDK installation. For example, the URL might look like the following:

  ```
  http://SDCDL360i101/xmla/msxisap.dll
  ```

- Essbase Server

  This property is only displayed for Essbase data sources. Specify the host name of the computer where the Essbase Server is running.

  You can import metadata from an Essbase cluster, but you must still specify an individual Essbase Server host name and port number in the **Essbase Server** field.

  If the Essbase Server is running on a non-default port, or if it's part of an Essbase Cluster, you must include the port number in the **Essbase Server** field, in the format *`hostname:port`*.

- SSO

This property is only displayed for Essbase, Hyperion Financial Management, and Hyperion Planning data sources.

For Essbase, select this option if you want Essbase to be able to enforce security policies that provide different cube access or member-level access to different users. If you select this option then you must also select the **Shared logon** option.

Don't select this option if all users are expected to have the same access to the Essbase cube. In this case, all the users have the same access to the cube based on the shared credentials specified in the connection pool. If you don't select this option then you must also select the **Shared logon** option.

For Hyperion Financial Management or Hyperion Planning, select this option and be sure that the **Shared logon** option is *unchecked* to authenticate against Hyperion Financial Management or Hyperion Planning using a shared token, rather than using a set of shared credentials in the connection pool.

If you select this option, you should also select **Virtual Private Database** in the corresponding database object to protect cache entries.

For Essbase, Hyperion Financial Management, and Hyperion Planning data sources installed with the EPM System Installer, preconfiguration is required before you select this option. See Configure SSO for Essbase, Hyperion Financial Management, or Hyperion Planning Data Sources.

- Shared logon

  This property is only displayed for Essbase, Hyperion Financial Management, and Hyperion Planning data sources.

  For all Essbase data sources, it's required that you select this option. See Configure Essbase to Use a Shared Logon.

  For Hyperion Financial Management or Hyperion Planning, you set this option based on how you set the **SSO** property.

  – If you checked the **SSO** property, then don't check this option. Not checking this option causes authentication against Hyperion Financial Management or Hyperion Planning using a shared token, rather than using a set of shared credentials in the connection pool.

  – If you didn't check the **SSO** property, then check this option to enable the Oracle BI Server to use the same shared logon credentials to connect to the data source for all Oracle BI users. All users share the same access to the data source.

- Data Source

  Specify the vendor-specific information used to connect to the multidimensional data source. Consult your multidimensional data source administrator for setup instructions because specifications can change. For example, if you use *v 1.0* of the XML for Analysis SDK, then use the value `Provider-MSOLAP;Data Source-local`. If you use *v 1.1*, use the value, `Local Analysis Server`.

- Catalog

  Specify the list of catalogs available, if you imported data from your data source. The cube tables correspond to the catalog you use in the connection pool.

- Use session

  This property is only displayed for XMLA data sources. An option that controls whether queries go through a common session. Consult your multidimensional data source administrator to determine whether this option is enabled. Default is Off, not selected.

Set Connection Pool Properties in the Connection Scripts Tab

You can create connection scripts and set the scripts to run before the connection is established, before a query is run, after a query is run, or after the connection is disconnected.

For example, you can create a connection script that, on connect, inserts the name of the user and the connection time into a table.

This topic describes the properties in the Connection Scripts tab of the Connection Pool dialog. The Connection Scripts tab is available for ODBC, OCI, Oracle OLAP, and ADF data sources.

Connection scripts can contain any commands accepted by the database, such as a command to turn on quoted identifiers. This enables mainframe environments to maintain security in one central location.

Because the connection script is sent directly to the data source, the script should use native SQL or another language understood by the data source, not Oracle BI Server Logical SQL.

- In the Connection Pool dialog, click the Connection Scripts tab, and then complete the fields using the information in the following table.

  To enter a new connection script, click **New** next to the appropriate script type. Then, enter or paste the SQL statements for the script and click **OK**.

  You can edit existing scripts by clicking the ellipsis button to launch the Physical SQL window. Use the Up Arrow and Down Arrow buttons to reorder existing scripts.

  Click **Delete** to remove a script.

The table describes the properties in the Connection Scripts tab of the Connection Pool dialog.

Property	Description
Execute on connect	Contains SQL queries that are run before the connection is established.
Execute before query	Contains SQL queries that are run before the query is run.
Execute after query	Contains SQL queries that are run after the query is run.
Execute on disconnect	Contains SQL queries that are run after the connection is closed.

Set Connection Pool Properties in the XML Tab

Use the Connection Pool Properties in the XML tab to set properties for XML data sources.

The XML tab in the Connection Pool dialog provides the same functionality as the XML tab of the Physical Table dialog. The properties in the XML tab of the Physical Table dialog override the corresponding settings in the Connection Pool dialog.

- In the Connection Pool dialog, click the XML tab, and then complete the fields using the information in the table that follows.

The table describes the properties in the XML tab of the Connection Pool dialog.

Property	Description
Connection method: Search script	This property is only displayed for XML Server data sources. Click **Browse** to select the search script file you want to use to locate the XML Server data source.
Connection properties: URL refresh interval	This property is used for XML data sources and isn't available for XML Server data sources. The refresh interval is analogous to setting cache persistence for database tables. The URL refresh interval is the time interval after which the XML data source is queried again directly rather than using results in cache. The default setting is infinite, meaning the XML data source is never refreshed. If you specified a URL to access the data source, set the URL refresh interval. • Select a value from the list (Infinite, Days, Hours, Minutes or Seconds). • Specify a whole number as the numeric portion of the interval.
Connection properties: URL loading time-out	The timeout interval for queries. The default is 15 minutes. If you specified a URL to access the data source, set the URL loading time-out as follows: • Select a value from the list (Infinite, Days, Hours, Minutes or Seconds). • Specify a whole number as the numeric portion of the interval.
Connection properties: Maximum connections	The maximum number of connections. The default is 10.
Query input supplements: Header file/Trailer file	This property is only displayed for XML Server data sources. Click **Browse** to locate the header and trailer files.
Query output format	For XML data sources, choose only **XML**. Other output formats are available for XML Server data sources.

Search Script Example

Use this search script example to create a search script for XML data source.

Example 7-1 Search Script

```
<ConnectionPool name="Connection Pool" parentName="&amp;quot;Stock Quotes&amp;quot;"
parentId="3023:3037" parentUid="80000557-0bcf-0000-714b-e31d00000000" id="3029:3046"
uid="8000055b-0bd5-0000-714b-e31d00000000" password="E3130008E1C4CAD47041E4AE68B048E6
7C2E35213306F12832914CBE7A9DD95561D771DED06484112B1FC6F27B6D0D58" timeout="4294967295"
maxConnDiff="10" maxConn="32" dataSource="http://www.host.net/stockquote.asmx"
type="Default" reqQualifedTableName="false" isSharedLogin="true"
isConcurrentQueriesInConnection="false" isCloseAfterEveryRequest="true"
xmlRefreshInterval="4294967295" scriptPath="java.exe -Dhttp.proxyHost=www-
proxy.us.oracle.com -Dhttp.proxyPort=80 -classpath
\analytics\server\Query\Execution\DbGateway\DbGatewayXML\Test\stocktick_webservices -
jar
\analytics\server\Query\Execution\DbGateway\DbGatewayXML\Test\stocktick_webservices\XML
ServiceAdapter.jar" outputType="xml" gwDelim="," ignoreFirstLine="false"
bulkInsertBufferSize="0" transactionBoundary="0" xmlaUseSession="false"
xmlHeaderPath="C:\orahome_2015\biclient\oraclebi\orainst\config\OracleBIServerComponent
\coreapplication\NQSQueryHeader.xml"
trailerPath="C:\orahome_2015\biclient\oraclebi\orainst\config\OracleBIServerComponent\c
oreapplication\NQSQueryTrailer.xml" supportParams="false" isSiebelJDBSecured="false">
</ConnectionPool>
```

Set Connection Pool Properties in the Write Back Tab

Use the Write Back tab to set write back properties for ODBC, OCI, Oracle OLAP, and ADF data sources.

- In the Connection Pool dialog, click the Write Back tab, and then complete the fields using the information in the table.

See About Setting the Buffer Size and Transaction Boundary.

The table describes the properties in the Write Back tab of the Connection Pool dialog.

Property	Description
Temporary table: Prefix	When the Oracle BI Server creates a temporary table, these are the first two characters in the temporary table name. The default value is `TT`.
Temporary table: Owner	Table owner name used to qualify a temporary table name in a SQL statement, for example to create the table `owner.tablename`. If left blank, the user name specified in the writeable connection pool is used to qualify the table name. Set the **Shared logon** field on the General tab.
Temporary table: Database name	Database where the temporary table is created. This property applies only to IBM OS/390 because IBM OS/390 requires database name qualifier as part of the `CREATE TABLE` statement. If left blank, OS/390 defaults the target database to a system database for which the users may not have Create Table privileges.
Temporary table: Tablespace name	Tablespace where the temporary table is created. This property applies to OS/390 only as OS/390 requires tablespace name qualifier as part of the `CREATE TABLE` statement. If left blank, OS/390 defaults the target database to a system database for which the users may not have Create Table privileges.
Bulk insert: Buffer size (KB)	Used for limiting the number of bytes each time data is inserted in a database table. For optimum performance, consider setting this parameter to 128.
Bulk insert: Transaction boundary	Controls the batch size for an insert in a database table. For optimum performance, consider setting this parameter to 1000.

Property	Description
Unicode database type	Select this option when working with columns of an explicit Unicode data type, such as `NCHAR`, in a Unicode database. This makes sure that the binding is correct and that data is inserted correctly. Different database vendors provide different character data types and different levels of Unicode support. Use the following general guidelines to determine when to set this option: • On a database where `CHAR` data type supports Unicode and there isn't a separate `NCHAR` data type, don't select this option. • On a database where `NCHAR` data type is available, it's recommended to select this option. • On a database where `CHAR` and `NCHAR` data type are configured to support Unicode, selecting this option is optional. Unicode and non-Unicode data types can't coexist in a single non-Unicode database. For example, mixing the `CHAR` and `NCHAR` data types in a single non-Unicode database environment isn't supported.

Connection Pool Properties in the Miscellaneous Tab

Use the Miscellaneous tab of the Connection Pool dialog to set application properties for ADF, JDBC, and JNDI data sources.

To set application properties, see Specify Application Properties for JDBC (Direct Driver) or JDBC (JNDI) Data Sources.

The table describes the properties in the Miscellaneous tab of the Connection Pool dialog.

Property	Description
AppModule Definition	The fully qualified Java package name of the Root Application Module to which you want to connect, such as `oracle.apps.fii.receivables.model.RootAppModule`.
AppModule Config	Determines which application configuration is used in the connection, such as `RootAppModuleShared`.

URL	The URL to the Oracle Analytics Server broker servlet, in the format:
	`http://host:port/APP_DEPLOYMENT_NAME/obieebroker`
	For example:
	`http://localhost:7001/SnowflakeSalesApp/obieebroker`
	The URL is case-sensitive.

Property	Description
SQL Bypass Database	(Optional) The name of the SQL Bypass database. The SQL Bypass database must be a physical database in the Physical layer of the repository. The database object for the SQL Bypass database must have a valid connection pool, with connection information that points to the same database that's being used by the JDBC Data source defined in the Oracle WebLogic Server. The SQL Bypass database doesn't need to have any tables under it. After a valid database name is supplied, the SQL Bypass feature is enabled for all queries. The SQL Bypass feature directly queries the database so that aggregations and other transformations are pushed down where possible, reducing the amount of data streamed and worked on in Oracle Analytics Server. See About Specifying a SQL Bypass Database.

Specify Application Properties for JDBC (Direct Driver) or JDBC (JNDI) Data Sources

Use the steps to set application properties for JDBC (Direct Driver) or JDBC (JNDI) data sources.

1. In the Oracle BI Administration Tool, double-click the physical database to set application properties for JDBC (Direct Driver) or JDBC (JNDI) data sources.
2. In Properties, click the Connection Pools tab.
3. Select the Connection and click Edit to open the Connection Pool dialog.
4. In the Connection Pool dialog, click the Miscellaneous tab.
5. Complete the fields using the following information:
   - **Required Cartridge Version** defaults to 12.1.
   - **Use SQL Over HTTP** for JDBC (JNDI) call interface, only. If you're using Oracle BI Cloud Service, set this field to *false* to use HTTP to communicate between networks. For example, set this field to *false* if the Oracle BI Server and the data source you're accessing reside on different Oracle clouds.
   - **Javads Server URL** for JDBC (Direct Driver) call interface, only. The field is populated with the hostname and port that was specified in the Connect to Java Datasource Server dialog. The **Javads Server URL** is the URL for the Java Datasource server that supplies the Java metadata into the Physical layer.
   - **Driver Class** for JDBC (Direct Driver) call interface, only. Specify the driver to connect to the database. You must select a driver that's deployed in Oracle WebLogic Server.

     By default the Oracle JDBC driver, `oracle.jdbc.OracleDriver`, is available in Oracle WebLogic Server.

EXECUTE PHYSICAL DATABASE

Use EXECUTE PHYSICAL DATABASE statement to send physical SQL to the Oracle BI Server to connect to data sources.

The `EXECUTE PHYSICAL DATABASE` statement enables processing physical queries from the client without knowing the connection pool information.

Syntax

`EXECUTE PHYSICAL DATABASE` *`DatabaseName/*add a valid SQL statement`*

Set Up Persist Connection Pools

A persist connection pool is a database property used for specific types of queries such as queries used to support Marketing.

In some queries, all of the logical query can't be sent to the transactional database because that database might not support all of the functions in the query. This issue might be solved by temporarily constructing a physical table in the database and rewriting the Oracle BI Server query to reference the new temporary physical table.

You can use the persist connection pool in the following situations:

- Populate stored procedures. Use to rewrite the Logical SQL result set to a managed table. Typically used by Oracle's Siebel Marketing Server to write segmentation cache result sets.
- Perform a generalized subquery. Stores a nonfunction subquery in a temporary table, and then rewrites the original subquery result against this table. Reduces data movement between the Oracle BI Server and the database, supports unlimited IN list values, and might result in improved performance.

  In these situations, the user issuing the Logical SQL query must have been granted the Populate privilege on the target database.

The persist connection pool functionality designates a connection pool with write-back capabilities for processing this type of query. You can assign one connection pool in a single database as a persist connection pool. If this functionality is enabled, the user name specified in the connection pool must have the privileges to create DDL (Data Definition Language) and DML (Data Manipulation Language) in the database.

See Set Connection Pool Properties in the Write Back Tab.

1. In the Physical layer of the Administration Tool, double-click the database object for which you want to assign a persist connection pool.
2. In the Database dialog, click the General tab.
3. In the **Persist connection pool** area, click **Clear**.

   The database name is replaced by `not assigned` in the **Persist connection pool** field.
4. If there are multiple connection pools, in the Browse dialog, select the appropriate connection pool, and then click **OK**.

   The selected connection pool name appears in the **Persist connection pool** field.
5. Optional: (Optional) click the Connection Pools tab to set write-back properties.

6. In the connection pool list, double-click the connection pool.
7. In the Connection Pool dialog, click the Write Back tab.
8. Click **OK**, then click **OK** again to save the persist connection pool.

Remove the Persist Connection Pool Property

Use these steps to remove the Persist Connection Pool property.

1. In the Physical layer of the Oracle BI Administration Tool, double-click the database object that contains the persist connection pool you want to remove.
2. In the Database dialog, click the General tab
3. In the **Persist connection pool** area, click **Clear**.

   The database name is replaced by `not assigned` in the **Persist connection pool** field.
4. Click **OK**.

About Setting the Buffer Size and Transaction Boundary

If each row size in a result set is 1 KB and the buffer size is 20 KB, then the maximum array size is 20 KB.

If there are 120 rows, there are 6 batches with each batch size limited to 20 rows.

If you set **Transaction boundary** to 3, the server commits twice. The first time, the server commits after row 60 (3 * 20). The second time, the server commits after row 120. If there is a failure when the server commits, the server only rolls back the current transaction. For example, if there are two commits and the first commit succeeds but the second commit fails, the server only rolls back the second commit.

For optimum performance, consider setting the buffer size to 128 and the transaction boundary to 1000.

List Connection Pool Command

Use the `listConnectionpool` command to create a list of connection pools in JSON format for a specific service instance.

Use the `listConnectionpool` command and the `updateConnectionpool` utility when you need to update more than one connection pool.

You run the utility through a launcher script, `datamodel.sh` on Linux and `datamodel.cmd` on Windows.

If the domain is installed in default folder then the location of the launcher script looks like the following:

Oracle_Home`/user_projects/domains/`*Domain_Name*`/bitools/bin/datamodel.sh` or `datamodel.cmd` on Windows.

If the client install doesn't have domain names, the launcher script location is as follows:

Oracle_Home`\bi\bitools\bin\datamodel.cmd`

See What You Need to Know Before Using the Command.

Syntax

The `listConnectionpool` command takes the following parameters:

```
listConnectionpool -SI <service_instance> -U <cred_username> [-P
<cred_password>] [-S <hostname>] [-N <port_number>] [-V <true/false>] [-O
<outputFile.json>] [-SSL] [-H]
```

Where

`SI` specifies the name of the service instance.

`U` specifies a valid user's name to be used for authentication.

`P` specifies the password corresponding to the user's name that you specified for `U`. If you don't supply the password, then you're prompted for the password when the command is run. Oracle recommends that you include a password in the command only if you're using automated scripting to run the command.

`S` specifies the host name. Only include this option when you're running the command from a client installation.

`N` specifies the port number. Only include this option when you're running the command from a client installation.

`V` specifies whether to include repository variables used in the connection pool. The default is `false`.

`O` specifies the output file name with the `.json` suffix.

`SSL` specifies to use SSL to connect to the Oracle WebLogic Server to run the command. Only include this option when you're running the command from a client installation.

`H` displays the usage information and exits the command. Use `-H` or run `.sh` without any parameters to display the help content.

Example

```
datamodel.sh listConnectionpool -SI bi -U weblogic -P password -S
server1.example.com -N 7777  -SSL -V true -O output.json
```

Update Connection Pool Command

Use the update `updateConnectionpool` command to upload a modified JSON file containing updated connection pool values to a specific server instance.

Use the `updateConnectionpool` command and the `listConnectionpool` utility when you need to update more than one connection pool.

Use the `listConnectionpool` command to create a JSON file containing a list of connection pools for a specific service instance. Modify the connection pool information in this file and then upload it to the service instance using the `updateConnectionpool` command. You must not modify the `uid` and `connPool` values in the file. See List Connection Pool Command.

You run the utility through a launcher script, `datamodel.sh` on Linux and `datamodel.cmd` on Windows.

If the domain is installed in default folder then the location of the launcher script looks like the following:

Oracle_Home`/user_projects/domains/`*Domain_Name*`/bitools/bin/datamodel.sh` or `datamodel.cmd` on Windows

If the client install doesn't have domain names, the launcher script location is as follows:

Oracle_Home`\bi\bitools\bin\datamodel.cmd`

See What You Need to Know Before Using the Command.

Syntax

The `updateConnectionpool` command takes the following parameters:

```
updateConnectionpool -C <connectionpoolList.json> -SI <service_instance> -U
<cred_username> [-P <cred_password>] [-S <hostname>] [-N <port_number>] [-SSL] [-
H]
```

Where

`C` specifies the name of the modified JSON file that you want to upload. This file must not contain modified `uid` and `connPool` values.

`SI` specifies the name of the service instance.

`U` specifies a valid user's name to use for authentication.

`P` specifies the password corresponding to the user's name that you specified for `U`. If you don't supply the password, you're prompted for the password when the command is run. For security purposes, Oracle recommends that you include a password in the command only if you're using automated scripting to run the command.

`S` specifies the host name. Only include this option when you're running the command from a client installation.

`N` specifies the port number. Only include this option when you're running the command from a client installation.

`SSL` specifies to use SSL to connect to the Oracle WebLogic Server to run the command. Only include this option when you're running the command from a client installation.

`H` displays the usage information and exits the command. Use `-H` or run `.sh` without any parameters to display the help content.

Example

```
datamodel.sh updateConnectionpool -C connpool.json -SI bi -U weblogic -
P password -S server1.example.com -N 7777 -SSL
```

Use the BIServerT2PProvisioner.jar Utility to Change Connection Pool Passwords

When moving your Oracle BI repository from one environment to another, you often need to change connection pool information for data sources, because the connection information in one environment is typically different from the connection information in another environments.

Although you can use `BIServerT2PProvisioner.jar` to update connection pool passwords, Oracle's preferred method is the `updateConnectionpool` command. See Update Connection Pool Command.

Connection pool passwords are encrypted and stored inside the encrypted repository file. Because of this, encrypt plain-text passwords before using with an Oracle BI repository.

You can use the `BIServerT2PProvisioner.jar` utility to programmatically change and encrypt connection pool passwords in a repository. The utility only works with repositories in Oracle BI repository file (RPD) format; you can't use the utility with MDS XML-format repositories. In addition, the utility requires JDK 1.6.

To use the `BIServerT2PProvisioner.jar` utility to change connection pool passwords:

The location of `BIServerT2PProvisioner.jar` is:

Oracle_Home`/bi/bifoundation/server`

Oracle doesn't recommend leaving clear-text passwords available on the system. Instead, delete the input password file completely, or encrypt it to prevent seeing the password.

1. Run `BIServerT2PProvisioner.jar` using the `-generate` option to generate a template file where you can input the new passwords, as follows:

   ```
   java -jar ORACLE_HOME/bifoundation/server/bin/BIServerT2PProvisioner.jar -generate
   repository_name -output password_file
   ```

   Where:

   repository_name is the name and path of the Oracle BI Repository that contains the connection pools for which you want to change passwords.

   password_file is the name and path of the output password text file. This file contains the connection pool names from the specified repository.

   Then, enter the repository password when prompted.

   For example:

   ```
   java -jar BIServerT2PProvisioner.jar -generate original.rpd -output
   inputpasswords.txt
   Enter the repository password: My_Password
   ```

2. Edit the password file to replace <*Change Password*> with the updated password for each connection pool. A sample password file might appear as follows:

   ```
   "SQLDB_UsageTracking"."UTCP" = <Change Password>
   "SQLDB_Data"."Db Authentication Pool" = <Change Password>
   ```

> **Tip:**
>
> Only edit the text to the right of the equals sign. If you change the text to the left of the equals sign, the syntax for the connection pool names is incorrect.

Save and close the password file when your edits are complete.

3. Run `BIServerT2PProvisioner.jar` again with the `-passwords` option, as follows:

```
java -jar BIServerT2PProvisioner.jar -passwords password_file
-input input_repository -output output_repository
```

Where:

password_file is the name and path of the text file that specifies the connection pools and their corresponding changed passwords.

input_repository is the name and path of the Oracle BI repository where you want to apply the changed passwords.

output_repository is the name and path of the output repository that contains the updated passwords.

Then, enter the repository password when prompted.

For example:

```
java -jar BIServerT2PProvisioner.jar -passwords inputpasswords.txt -input
original.rpd -output updated.rpd
Enter the repository password: My_Password
```

8

Work with Physical Tables, Cubes, and Joins

Learn how to work with objects in the Physical layer of the Oracle BI repository, and describes Oracle Essbase, Hyperion Financial Management, and Oracle OLAP representations in the Physical layer. It also explains other Physical layer concepts like opaque views, hints, row counts, physical layer folders, and how to use the Physical Diagram.
This chapter contains the following topics:

- About Working with the Physical Layer
- Work with the Physical Diagram
- Create Physical Layer Folders
- Work with Physical Tables
- Work with Multidimensional Sources in the Physical Layer
- Work with Essbase Data Sources
- Work with Hyperion Financial Management and Hyperion Planning Data Sources
- Work with Oracle OLAP Data Sources
- Work with Physical Foreign Keys and Joins
- Deploy Opaque Views
- Use Hints in SQL Statements
- Display and Update Row Counts for Physical Tables and Columns

About Working with the Physical Layer

The Physical layer of the Oracle BI repository contains objects that represent physical data constructs from back-end data sources.

The Physical layer defines the objects and relationships available to the Oracle BI Server for writing physical queries. This layer encapsulates data source dependencies to enable portability and federation.

Each data source of the repository model typically has its own discrete physical model in the Physical layer. The top-level object in the Physical layer is a database, and the type of database determines which features and rules apply to that physical model. For example, a relational database such as Oracle 12c has relational objects such as physical tables and joins. In contrast, a multidimensional source such as Essbase has cube tables and physical hierarchies. Therefore, some sections of this chapter apply to only certain database types.

Physical tables, cubes, joins, and other objects in the Physical layer are typically created automatically when you import metadata from the data sources. After these objects have been imported, you can perform tasks such as create additional join paths that aren't in the data source, create alias tables for physical tables that need to serve in different roles, and adjust properties of physical hierarchies from multidimensional data sources.

Work with the Physical Diagram

In the Oracle BI Administration Tool, you can open the Physical Diagram view to access a graphical model of tables and joins.

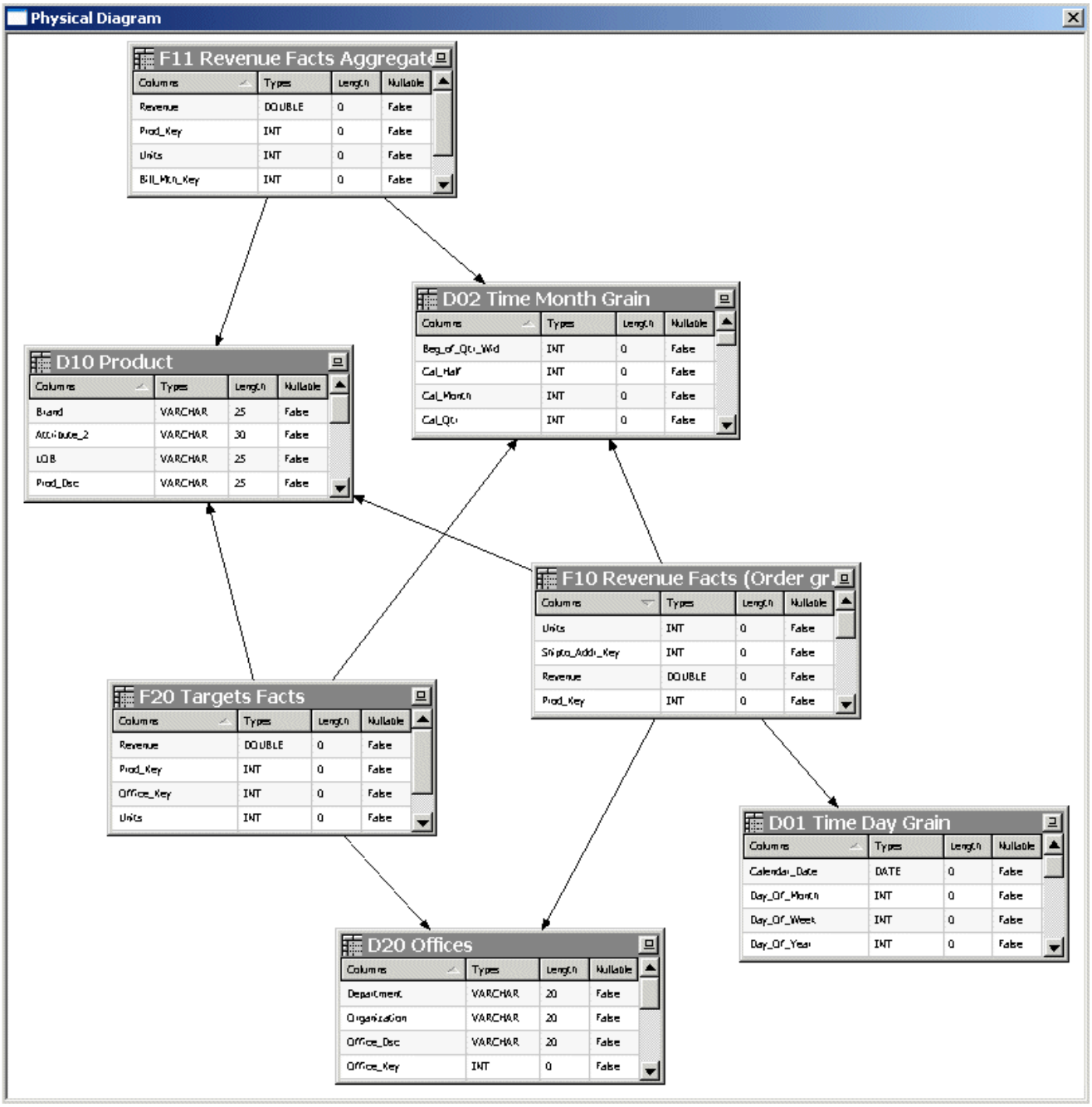
The Physical Diagram is typically used with relational and XML sources rather than multidimensional sources. Although the Physical Diagram view for a multidimensional source does display a denormalized table representation of a cube table, the primary means of working with a multidimensional physical model is by working in the physical tree using dimensions, hierarchies, and columns.

The Physical Diagram displays only physical tables and joins. It doesn't display other Physical layer objects, such as connection pools, physical hierarchies, or levels.

To access the Physical Diagram, right-click an object in the Physical layer tree view such as a physical database or table, and select **Physical Diagram**. Then, select one of the following options:

- **Selected Object(s) Only.** Displays only the selected objects. Joins appear only if they exist between the objects that you select.
- **Object(s) and Direct Joins.** Displays the selected objects and any tables that join to the objects that you select.
- **Object(s) and All Joins.** Displays the selected objects, and each object that's related directly or indirectly to the selected object through some join path. If all the objects in a schema are related, then using this option diagrams every table, even if you only select one table.

The image shows the Physical Diagram.



You can also open the Physical Diagram by selecting one or more objects in the tree view and clicking the **Physical Diagram** button on the toolbar:

Only the objects you selected appear. Joins appear only if they exist between the selected objects. Joins are represented by a line with an arrow at the *one* end of the join.

To help you better understand the model's logical-to-physical mappings, you can view the physical objects that are associated with a particular logical object by selecting one or more

business models, logical tables, or logical table sources in the Business Model and Mapping layer tree view and clicking the **Physical Diagram** button on the toolbar. Only physical objects that are related to the objects you selected appear. You can view the same information by right-clicking a logical object and selecting **Objects and Direct Join(s) within Business Model** from the Physical Diagram submenu. You can also choose one of the other Physical Diagram display options.

To add tables to the Physical Diagram, leave the Physical Diagram window open and right-click the table or tables you want to add. Then, select **Physical Diagram** and choose one of the display options.

Additional options are available in the right-click menu for the graphical tables and joins displayed in the Physical Diagram. For example, you can delete objects or view their properties, or you can add related objects using the right-click options **Add Direct Joins**, **Add Tables Joined to Whole Selection**, and **Add All Joins**. You can also select **Find in Tree View** to locate a particular object in the Physical layer tree view in the right pane, or check out objects in online mode.

You can also right-click an object in the Physical Diagram view and select **Hide** to hide particular objects in the diagram. The hide effect is temporary and doesn't persist.

Use the **Print** and **Print Preview** options on the File menu to manage printing options for the Physical Diagram. You can also use the **Print** option on the toolbar.

See Define Physical Joins with the Physical Diagram.

Create Physical Layer Folders

Use folders to organize the contents of the Physical layer.

This section contains the following topics:

- Create Physical Layer Catalogs and Schemas
- Use a Variable to Specify the Name of a Catalog or Schema
- Set Up Display Folders in the Physical Layer

Create Physical Layer Catalogs and Schemas

Physical layer catalogs are optional ways to group different schemas.

A catalog contains all the schemas (metadata) for a physical database object. A schema contains only the metadata information for a particular user or application.

- You must create a physical database object before you can create a physical catalog object or a physical schema object.
- After you implement a certain type of grouping, you can't change it later. For example, if you decide to implement database then schema then table, you can't add a catalog afterward.

Create Catalogs

In the Physical layer of a large repository, administrators can create physical catalogs that contain one or more physical schemas.

1. In the Physical layer of the Administration Tool, right-click a physical database and select **New Object**, then select **Physical Catalog**.
2. In the Physical Catalog dialog, type a name for the catalog.
3. Type a description for the catalog and click **OK**.

Create Schemas

The schema object contains tables and columns for a physical schema.

Schema objects are optional in the Physical layer of the Oracle BI Administration Tool.

1. Open the repository in the Administration Tool.
2. In the Physical layer right-click a physical database or physical catalog, select **New Object**, and then select **Physical Schema**.
3. In Physical Schema, in **Name**, type a name for the schema.
4. Type a description for the schema, and click **OK**.

Use a Variable to Specify the Name of a Catalog or Schema

You can use a variable to specify the names of catalog and schema objects.

For example, you've data for multiple clients and you structured the data source so that data for each client was in a separate catalog. You would initialize a session variable named Client, for example, that could be used to set the name for the catalog object dynamically when a user signs on to the Oracle BI Server.

You specify the session variable to use in the Dynamic Name tab of the Physical Catalog or Physical Schema dialog. The Dynamic Name tab is active when at least one session variable is defined.

The name of the variable is displayed in the dynamic name field, and the **Select** button toggles to the **Clear** button.

1. In the **Name** column of the Dynamic Name tab, click the name of the session variable that you want to use.
2. To select the highlighted variable, click **Select**.
3. Optional: In the **Default Initializer** column, update the variable value.

To remove assignment for a session variable in the Dynamic Name tab:

- Click **Clear** to remove the assignment for the variable as the dynamic name.

  The value **not assigned** is displayed in the dynamic name field, and the **Clear** button toggles to the **Select** button.

To sort column entries in the Dynamic Name tab:

- Click the **Name** or **Default Initializer** column heading to sort the entries in a column. Clicking a column heading toggles the order of the entries in that column between ascending and descending order, according to the column type.

Set Up Display Folders in the Physical Layer

You can create display folders to organize table objects in the Physical layer. They have no effect on query processing.

After you create a display folder, the selected tables appear in the folder as a shortcut and in the Physical layer tree as an object. You can hide the objects so that you only view the shortcuts in the display folder. Deleting a table in a display folder deletes only the shortcut to that object. When you delete a column in a display folder, however, the column is actually deleted. See the information about the Repository tab of the Options dialog in Set Administration Tool Options.

1. In the Physical layer of the Administration Tool, right-click a physical database and select **New Object**, then select **Physical Display Folder**.
2. In the Physical Display Folder dialog, type a name for the folder.
3. To add tables to the display folder, click **Add**. Then, in the Browse dialog, select the fact or physical tables you want to add to the folder and click **Select**.
4. Click **OK**.

Work with Physical Tables

Learn about the different things you can do with physical table objects in the Physical layer of the Oracle BI repository.

Both physical tables from relational data sources and physical cube tables from multidimensional data sources use the table type, Physical Table.

Many of the tasks described in this section apply to relational and multidimensional data sources. See Work with Multidimensional Sources in the Physical Layer.

This section contains the following topics:

- About Tables in the Physical Layer
- About Physical Alias Tables
- Create and Manage Physical Tables and Physical Cube Tables
- Create and Manage Columns and Keys for Relational and Cube Tables
- View Data in Physical Tables or Columns

About Tables in the Physical Layer

A physical table is an object in the Physical layer of the Oracle BI repository that corresponds to a table in a data source.

Metadata for physical tables is usually imported from the data source. This metadata enables the Oracle BI Server to access the data source tables with SQL requests.

When you delete a physical table, all dependent objects are deleted, for example, columns, keys, and foreign keys. When you delete a physical cube table, hierarchies are also deleted. The deletion fails if an alias exists on the physical table.

In addition to importing data source tables into the Physical layer, you can create virtual physical tables in the Physical layer, using values in the **Table Type** field in the

Physical Table dialog. Creating virtual tables can provide the Oracle BI Server and the underlying data sources with the proper metadata to perform some advanced query requests.

You can store a virtual physical table as a stored procedure or a `SELECT` statement. A virtual physical table created from a `SELECT` statement is also called an opaque view. You can define an opaque view, and deploy it in the data source to create a deployed view, see Deploy Opaque Views.

Use the **Table Type** list in the General tab of the Physical Table dialog to specify the physical table object type. The following table describes the available object types.

Table Type	Description
Physical Table	Specifies that the physical table object represents a data source table. This table type is used for both relational physical tables and multidimensional cube tables.

Stored Proc	Specifies that the physical table object is a stored procedure. When you select this option, you type the stored procedure in the text box. Requests for this table call the stored procedure.
	For stored procedures that are data source-specific, select **Use database specific SQL**. When you select this option, the **Database** column displays supported data sources by brand, with Default as the root. You can enter data source-specific initialization strings by selecting the database type on the left and entering the corresponding string on the right. The initialization string for the Default option is run when the queried database type doesn't have a corresponding database-specific string defined.
	Stored procedures within an Oracle Database might not return result sets. You can't initiate stored procedures from within Oracle Analytics Server. You need to rewrite the procedure as an Oracle function, use the Oracle function in a `SELECT` statement in the Administration Tool initialization block, and associate the Oracle function with the appropriate Oracle BI Server session variables in the Session Variables dialog.
	The following example shows a SQL initialization string using the `GET_ROLES` function that's associated with the `USER`, `GROUP`, and `DISPLAYNAME` variables. The function takes a user Id as a parameter and returns a semicolon-delimited list of group names:
	`SELECT user_id, get_roles(user_id), first_name || ' ' || last_name`
	`FROM csx_security_table`
	`WHERE user_id = ':USER' and password = ':PASSWORD'`

Table Type	Description
Select	Specifies that the physical table object is a `SELECT` statement. When you select this option, you type the `SELECT` statement in the text field, and you also need to manually create the table columns. The column names must match the ones specified in the `SELECT` statement. Column aliases are required for advanced SQL functions, such as aggregates and `CASE` statements. Requests for this table run the `SELECT` statement. For `SELECT` statements that are data source-specific, select **Use database specific SQL**. When you select this option, the **Database** column displays supported data sources by brand, with Default as the root. You can enter data source-specific initialization strings by selecting the database type on the left and entering the corresponding string on the right. The initialization string for the Default option is run when the queried database type doesn't have a corresponding database-specific string defined. If you're using Physical SQL to deploy an opaque view, then you must use the `VALUELISTOF` function. This type of table is also called an opaque view.

About Physical Alias Tables

An alias table (alias) is a physical table that references a different physical table as its source, called the original table.

Alias tables are an important part of designing a Physical layer because they enable you to reuse an existing table more than once, without having to import it several times.

There are two main reasons to create an alias table:

- To set up multiple tables, each with different keys, names, or joins, when a single data source table needs to serve in different semantic roles. Setting up alias tables in this case is a way to avoid triangular or circular joins.

  For example, an order date and a shipping date in a fact table may both point to the same column in the time dimension data source table, but alias the dimension table so that each role is presented as a separately labeled alias table with a single join. These separate roles carry over into the business model, so that *Order Date* and *Ship Date* are part of two different logical dimensions. If a single logical query contains both columns, the physical query uses aliases in the SQL statement so that it can include both of them.

  You can also use aliases to enable a data source table to play the role of both a fact table, and a dimension table that joins to another fact table, often called a *fan trap*.

- To include best practice naming conventions for physical table names. For example, you can prefix the alias table name with the table type such as fact, dimension, or bridge, and not change the original physical table names. Some organizations create alias tables for all physical tables to enforce best practice naming conventions. In this case, all mappings and joins are based on the alias tables rather than the original tables.

Alias table names appear in physical SQL queries. Using alias tables to provide meaningful table names can make SQL queries referencing those tables easier to read. For example:

```
WITH
SAWITH0 AS (select sum(T835.Dollars) as c1
from
     FactsRevT835/*AllRevenue(Billed Time Join)*/)
select distinct 0 as c1,
     D1.c1 as c2
from
     SAWITH0 D1
order by c1
```

In this query, the meaningful alias table name *A11 Revenue (Billed Time Join)* has been applied to the terse original physical table name *FACTSREV*. In this case, the alias table name provides information about which role the table was playing each time it appears in SQL queries.

Alias tables can have cache properties that differ from their original tables. To set different cache properties for an alias table, select the option **Override Source Table Caching Properties** in the Physical Table dialog for the alias table. In alias tables, you can't add, delete, or modify columns. Because columns are automatically synchronized, no manual intervention is required.

Synchronization ensures that the original tables and their related alias tables have the same column definitions. For example, if you delete a column in the original table, the column is automatically removed from the alias table.

You can't delete an original table unless you delete all its alias tables first. Alternatively, you can select the original table and all its alias tables and delete them at the same time.

You can change the original table of an alias table, if the new original table is a superset of the current original table. However, this could result in an inconsistent repository if changing the original table deletes columns that are being used. If you attempt to do this, a warning message appears to let you know that this could cause a problem and lets you cancel the action. Running a consistency check identifies orphaned aliases.

When you edit a physical table or column in online mode, you must check out all alias tables and columns. Alias tables and original tables never share columns. Aliases and original tables have distinctly different columns that alias each other. The behavior of online checkout uses the following conventions:

- If an original table or column is checked out, all its alias tables and columns are checked out.
- If an alias table or column is checked out, its original table and column are checked out.
- The checkout option is available for online repositories (if not read-only) and for all original and alias tables and columns.

Alias tables inherit some properties from their original tables. A property that's *proxied* is a value that's always the same as the original table. The proxied properties are the ones that are dimmed in the alias table dialog. If the original table changes its value for that particular property, the same change is applied on the alias table.

The following is a list of the properties that are proxied:

- Cacheable, the inherited property is overridden
- Cache never expires and Cache persistence time, the inherited properties are overridden
- Row Count

- Last Updated
- Table Type
- External Db Specifications

The following is a list of the properties that aren't proxied:

- Name
- Description
- Display Folder Containers
- Foreign Keys
- Columns
- Table Keys
- Complex Joins
- Source Connection Pool
- Polling Frequency
- All XML attributes

Create and Manage Physical Tables and Physical Cube Tables

Use the General tab of the Physical Table dialog to create or edit physical tables and physical cube tables in the Physical layer of the Administration Tool.

This section contains the following topics:

- Create Physical Tables
- Create Alias Tables
- Set Physical Table Properties for XML Data Sources

Create Physical Tables

You can create or edit the general properties for a table, including both relational physical tables and physical cube tables.

In the properties for physical tables, you can view the name-value pairs used for data sources as a generic mechanism for extending the Physical layer metadata. These values are passed from the data source, but you can edit the values as needed. See View Physical Column Properties.

There are additional configuration settings that affect the behavior of the query cache. See Configure Query Caching.

Review Physical Table Properties before you create the physical table to understand the configuration options.

> **Important:**
>
> Oracle strongly recommends that you import cube tables, not create them manually.

1. In the Administration Tool, expand a database in the Physical layer, right-click the database schema, and select **New Physical Table**.
2. In the Physical Table dialog, specify a **Name** for the table.
3. From the **Table Type** list, select Physical Table.
4. Optional: Select **Use Dynamic Name** when using a non-multidimensional data source.
5. In Browse, select the value to use for the table.
6. Select **Cacheable** when the table isn't accessed in real time.
7. If you selected **Cacheable**, select the persistence time frame for the table.
8. Optional: In **Hint,** specify a value.
9. Optional: In **Description**, provide brief information about the table, and click **OK**.

Physical Table Properties

Review this table before creating physical tables, stored procedure tables, or selection tables.

Property	Description
Name	Indicates the physical table name.
Table Type	Indicates the physical table type. The valid values are *Physical Table*, *Stored Proc* (stored procedure), or *Select*.
Use Dynamic Name	Select **Use Dynamic Name** to use a session variable to specify the physical table name, similar to catalog and schema objects. This option is available for non-multidimensional data source tables when you select **Physical Table**. You can choose **Use Dynamic Name** if you've a multi-tenancy implementation, and you want to define a separate physical table name for each customer. Another example would be to select between primary and shadow tables that are valid at different times in the ETL cycle. In both cases, you can assign session variables to dynamically select the appropriate table.
Default Initialization String / Use database specific SQL	When creating a physical table for non-multidimensional data source tables (not alias tables), this option is available in the Physical Table dialog if you choose the **Stored Proc** or **Select** table types. For multidimensional data source tables, this option is available in the Physical Table dialog if you choose the **Select** table type. When you select **Default Initialization String / Use database specific SQL** option, you can specify the data source and type the SQL statements.
Cacheable	Select **Cacheable** to include the table in the Oracle BI Server query cache. Typically, select this option for tables that don't need to be accessed in real time. When you select this option, the Cache persistence time settings become active.
Cache never expires	When you select **Cache never expires**, cache entries don't automatically expire. This option is useful when a table is important to a large number of queries that users might run. For example, if most queries have a reference to an account object, keeping it cached indefinitely could actually improve performance rather than compromise it. Selecting **Cache never expires** doesn't mean that an entry always remains in the cache. Other invalidation techniques, such as manual purging, LRU (Least Recently Used) replacement, metadata changes, or use of the cache polling table can result in entries being removed from the cache.

Property	Description
Cache persistence time	Specifies the time period in which the table entries are persisted in the query cache. Setting a cache persistence time is useful for OLTP data sources and other data sources that are updated frequently. For example, you could set this option to refresh the underlying physical tables daily for a particular dashboard. If a query references multiple physical tables with different persistence times, the cache entry for the query exists for the shortest persistence time set for any of the tables referenced in the query. This makes sure that no subsequent query gets a cache hit from an expired cache entry. See Troubleshoot Problems with Event Polling Tables in *Administering Oracle Analytics Server*.
External name	Applies to physical cube tables from multidimensional data sources. The external name is the physical name that's used when referencing the cube table in physical SQL queries. This value must reflect the external name defined in the data source.
Display Column	For Essbase data sources only, see Work with Essbase Data Sources.
Hint	Available only for some data sources, see Use Hints in SQL Statements.

Create Alias Tables

You can also create aliases on opaque views and stored procedures.

The following table describes properties that are specific to alias tables. See Create Physical Tables.

Property	Description
Source Table	Applies to alias tables. Click **Select** to choose the original physical table from which to create an alias table.
Override Source Table Caching Properties	Applies to alias tables. Click this field to enable the cacheable properties. You can select or clear the appropriate cacheable options.

- In the Oracle BI Administration Tool, with a repository open, right-click an existing physical table and select **New Object**, then select **Alias** to create an alias table.

Set Physical Table Properties for XML Data Sources

Use the XML tab to set or edit properties for an XML data source.

The XML tab of the Physical Table dialog provides the same functionality as the XML tab of the Connection Pool dialog. However, setting properties in the Physical Table dialog overrides the corresponding settings in the Connection Pool dialog. See Set Connection Pool Properties in the XML Tab.

Create and Manage Columns and Keys for Relational and Cube Tables

Each physical table and physical cube table in the Physical layer of the Oracle BI Administration Tool has one or more physical columns.

You can use the Columns, Keys, and Foreign Keys tabs in the Physical Table dialog to view, create new, and edit existing columns, keys, and foreign keys that are associated with the table.

This section contains the following topics:

- Create and Edit a Column in a Physical Table
- Specify a Primary Key for a Physical Table
- Delete Physical Columns for All Data Sources
- View Physical Column Properties

Create and Edit a Column in a Physical Table

An imported column's properties are set automatically. After import, you can modify the column's property, including its type and whether null values are allowed for the column.

Creating, modifying, or deleting a column in an original physical table also creates, modifies, or deletes the same column on all its alias tables.

The following list contains information about nullable and data type values for columns imported into the Physical layer.

- `Nullable` indicates whether null values are allowed for the column. If null values can exist in the underlying table, you need to select this option. This allows null values to be returned to the user, which is expected with certain functions and with outer joins. It's generally safe to change a non-nullable value to a nullable value in a physical column.
- `Type` indicates the data type of the column. Use caution when changing the data type. Setting the values to data types that are incorrect in the underlying data source might cause unexpected results. If there are any data type mismatches, correct them in the repository or reimport the columns that have mismatched data types.

  If you reimport columns, you also need to remap any logical column sources that reference the remapped columns. The data type of a logical column in the business model must match the data type of its physical column source. The Oracle BI Server passes these logical column data types to client applications.

  `Longvarchar` and `longvarbinary` data types are supported for writing complete Logical SQL statements into usage tracking tables for debugging purposes. They aren't supported for general-purpose queries, and can't be displayed in Oracle BI Server. Use direct SQL utilities to access columns with these data types.

Except when stated otherwise, the characteristics and behavior of a physical cube column are the same as for other physical columns.

For XML data sources, this field stores and displays the unqualified name of a column (attribute) in an XML document.

A new physical cube column is created as a measure by default. See Work with Multidimensional Sources in the Physical Layer.

1. In the Administration Tool, in the Physical layer, right-click a physical table and select **New Object**, then select **Physical Column** to create a column.
2. Right-click a physical cube table, select **New Object**, and then select **Physical Cube Column** to create a physical cube column for a multidimensional data source.
3. Double-click the physical column object in the Physical layer to edit an existing physical column.

4. In the Physical Column dialog, type a name for the physical column.
5. In the **Type** field, select a data type for the physical column.
6. If applicable, specify the length of the data type.

   When using multidimensional data sources, if you select `VARCHAR`, you must type a value in the **Length** field.
7. Select the **Nullable** option if the column is allowed to have null values.
8. In the **External Name** field, type an external name.
   - Required if the same name such as `STATE` is used in multiple hierarchies.
   - Optional for XML documents. The **External Name** field stores and displays the fully qualified name of a column (attribute).
9. In multidimensional data sources when the physical cube column is a measure, from the **Aggregation role** list, select the appropriate value.
10. Click **OK**.

Specify a Primary Key for a Physical Table

Use the Physical Key dialog to specify the column or columns that define the primary key of the physical table.

1. In the Physical layer of the Administration Tool, right-click a physical table and select **Properties**.
2. In the Physical Table dialog, click the Keys tab.
3. In the Keys tab, click **New**.
4. In the Physical Key dialog, type a name for the key.
5. Select the column that defines the primary key of the physical table.
6. Optional: Type a description for the key.
7. Click **OK**.

Delete Physical Columns for All Data Sources

Learn what happens when you delete a physical column.

When you delete a physical column, the following occurs:

- **Multidimensional data sources.** If you delete property or key columns from a level, the association is deleted and the column changes to a measure under the parent cube table.
- **Alias tables.** Deleting a column in an original physical table deletes the same column on all its alias tables.

View Physical Column Properties

The Properties tab for physical columns displays name-value pairs that are used for some data sources as a generic mechanism for extending the Physical layer metadata.

The values are passed up from the data source, but you can edit the values if needed.

View Data in Physical Tables or Columns

You can view the data in a physical table or an individual physical column by right-clicking the object and choosing **View Data**.

In online editing mode, you must check in changes before you can use this option.

View Data doesn't work in online mode if you set the user name and password for connection pools to `:USER` and `:PASSWORD`. In offline mode, the Set values for variables dialog appears so that you can populate `:USER` and `:PASSWORD` as part of the viewing process.

View Data isn't available for physical cube tables or columns. See View Members in Physical Cube Tables.

Because the View Data feature issues a row count, it isn't available for data sources that don't support row counts. See Display and Update Row Counts for Physical Tables and Columns.

Work with Multidimensional Sources in the Physical Layer

Learn about physical cube tables, dimensions, and hierarchies from multidimensional data sources.

This section contains the following topics:

- About Physical Cube Tables
- About Measures in Multidimensional Data Sources
- About Working with Physical Dimensions and Physical Hierarchies
- View Members in Physical Cube Tables

About Physical Cube Tables

Each cube from a multidimensional data source is structured as a physical cube table, a type of physical table.

A cube has all the elements of a table such as physical cube columns and keys (optional) and foreign keys (optional). A cub also has specific metadata such as hierarchies and levels.

When you import the physical schema, the Oracle BI Server imports the metadata for the cube, including its metrics, hierarchies, and levels. Expanding the hierarchy object in the Physical layer reveals the levels in the hierarchy.

Each multidimensional catalog in the data source can contain multiple physical cubes. You can import the metadata for one or more of these cubes into the Oracle BI Repository. Although you can create a cube table manually, Oracle recommends importing the metadata for cube tables and their components.

If you create cubes manually, you must build each cube one hierarchy at a time and test each one before building another. For example, create the time hierarchy and a measure and test it. When the cube is correct, create the geography hierarchy and test it. Creating and testing manually created cubes helps ensure that you've set up each cube correctly, and helps you identify any setup errors.

About Measures in Multidimensional Data Sources

You need to select the aggregation rule for a physical cube column carefully to make sure the measures are correct.

Setting it correctly might improve performance.

Always verify aggregation rules after importing cube metadata. Typically, aggregation rules are assigned correctly when you import cube metadata. However, if a measure is a calculated measure, the aggregation rule is reported as None. Therefore, you must examine the aggregation rule for all measures after importing a cube to verify that the aggregation rule has been assigned correctly.

For all measures assigned *None* as the aggregation rule value, contact the multidimensional data source administrator to verify that the value is accurate. If you need to change the aggregation rule, you can change it in the Physical Cube Column dialog.

Use the following guidelines to assign the correct aggregation rule:

- If the generated physical queries to the database should send an aggregation function, such as `SUM(revenue)`, then set that function as the aggregation rule. With this setting, the Oracle BI Server typically sends the aggregation to the database in the query, but might also perform aggregations itself in certain situations.
- If the data for this measure shouldn't be aggregated in the query or by the Oracle BI Server, use the External Aggregation rule. It's important to choose this setting when the measure uses a more complex calculation inside the data source than the Oracle BI Server can replicate with a simple aggregation rule such as calculations for ratios, consolidations and allocations. This option is also useful when the cube persists a full set of pre-aggregated results.

About Externally Aggregated Measures

In a multidimensional data source, some cubes contain very complex, multi-level based measures.

If you assign an aggregation rule of External Aggregation, the Oracle BI Server bypasses its internal aggregation mechanisms and uses the pre-aggregated measures. When imported, these measures are assigned an aggregate value of None.

The following are some guidelines for working with pre-aggregated measures:

- External aggregation only applies to multidimensional data sources such as Essbase, Hyperion Financial Management, and Microsoft Analysis Services that support these complex calculations.
- You can't assign external aggregation to measures from non-multidimensional data sources. If the required aggregation rule is supported by the Oracle BI Server and can be mapped to a relational data source, then it isn't complex and doesn't require external aggregation.
- There is only one aggregation rule for a logical measure. Therefore, a single logical column can't federate a non-complex aggregation rule for a mapping to a non-multidimensional source, with a complex aggregation rule for a mapping to a

multidimensional source. Instead, you need to create one logical measure for each source, and create a third logical measure that derives from the first two.

- You can mix non-complex measures from non-multidimensional data sources with non-complex measures from multidimensional data sources if they're aggregated through the Oracle BI Server.

About Working with Physical Dimensions and Physical Hierarchies

Most dimensions and hierarchies are imported into the Physical layer from multidimensional data sources, rather than created manually.

If a particular hierarchy isn't imported, any columns associated with that hierarchy are also not imported. If users need access to columns that aren't imported, first add these columns to the Physical layer by manually creating them and associate them with a level in a hierarchy.

Each level in a hierarchy has a level key. The first cube column associated with (added to) the level of a hierarchy is the level key. This must match with the data source definition of the cube. The icon for the column that you select first changes to the key icon after it's associated with the level of a hierarchy.

Oracle Analytics Server supports unbalanced hierarchies for all multidimensional data sources. In general, you can configure unbalanced hierarchies in the Physical layer by changing the hierarchy type.

You can view and edit properties for physical dimensions and hierarchies by double-clicking physical dimension and physical hierarchy objects in the Physical layer of the Answers. You can also view and edit these objects from the Dimensions and Hierarchies tabs of the Cube Table dialog.

This section contains the following topics:

- Work with Physical Dimension Objects
- Work with Physical Hierarchy Objects

Work with Physical Dimension Objects

In the Physical Dimension dialog, you can view and edit the name and description of the dimension.

You can also add, remove, or edit hierarchies for that dimension, and add, remove, or edit columns that represent dimension properties.

Work with Physical Hierarchy Objects

When you select columns to add to a hierarchy, it's recommended that you select them in hierarchical order, starting with the highest level.

If you select multiple columns and bring them into the hierarchy at the same time, the order of the selected group of columns remains the same. After adding columns to the hierarchy, you can change the order of the columns in the Browse dialog.

In the Physical Hierarchy dialog, you can view and edit the name and description of the hierarchy, along with the properties described in the table. For level-based hierarchies, you can add, remove, edit, or reorder levels. For value-based hierarchies, click the Column tab to add, remove, or edit columns. To specify a key column, double-click a column name.

In the Physical Level dialog, you can view and edit the name, external name, and description of the level. You can also add, remove, or edit columns for that level. To designate a column as a level key, double-click a column name.

Always review the hierarchy type after import to ensure that it's set appropriately. The way this parameter is set upon import depends on the data source. For example, all Essbase hierarchies are initially set to Unbalanced. Review the hierarchy type for each hierarchy and change it as appropriate.

Typically, you always need to manually set the hierarchy type for parent-child (value) hierarchies, except for Hyperion Financial Management hierarchies, which are always set to Value by default upon import. Review the hierarchy type and change the type to Value as appropriate. Parent-child (value) hierarchies are those in which a business transaction, or a cube refresh, can change the number of levels.

For parent-child hierarchies, you must manually set the physical hierarchy type to Value before dragging metadata to the Business Model and Mapping layer. The hierarchy type in the Business Model and Mapping layer is set automatically based on the physical hierarchy setting. For all other types, you can determine the hierarchy type later, without needing to rebuild the logical model.

You must also ensure that the corresponding logical dimension properties are correct for queries to work. See Work with Logical Dimensions.

Property	Description
External Name	The physical name that's used when referencing the hierarchy in physical MDX queries. This value must reflect the external name defined in the data source.
Dimension Name	(Dimension Unique Name) Dimension to which the hierarchy belongs.
Dimension Type	Identifies whether this hierarchy belongs to a time dimension, measure dimension, or other type of dimension.

Property	Description
Hierarchy Type	Identifies the type of hierarchy, as follows: • Fully balanced: A level-based hierarchy with no unbalanced or skip characteristics. Corresponds to a level-based hierarchy in the Business Model and Mapping layer. • Unbalanced: Also called ragged. A hierarchy where the leaves (members with no children) don't necessarily have the same depth. Corresponds to a level-based hierarchy with the Ragged option selected in the Business Model and Mapping layer. • Ragged balanced: Also called skip. A hierarchy where there are members that don't have a value for a particular ancestor level. Corresponds to a level-based hierarchy with the Skipped Levels option selected in the Business Model and Mapping layer. • Network: This hierarchy type isn't used. • Value: Also called parent-child. A hierarchy of members that all have the same type. This contrasts with level-based hierarchies, where members of the same type occur only at a single level of the hierarchy. Corresponds to a parent-child hierarchy in the Business Model and Mapping layer. For level-based hierarchies with both unbalanced and skip-level characteristics, choose either Unbalanced or Ragged balanced as the physical hierarchy type. Then, ensure that both Ragged and Skipped Levels are selected for the corresponding logical dimension in the Business Model and Mapping layer.
Default member type ALL	This option isn't used.
Use unqualified member name for better performance	Select this option when member names, including aliases are unique in a given hierarchy so that the Oracle BI Server can take advantage of specific MDX syntax to optimize performance.

Adding or Removing Cube Columns in a Hierarchy

After importing a hierarchy, you may need to add or remove a column.

If you remove a cube column from a hierarchy, it's deleted from the hierarchy but remains in the cube table and is available for selection to add to other levels.

1. In the Physical layer of the Administration Tool, double-click the physical hierarchy to add or remove a cube column.
2. For level-based hierarchies, double-click the level for which you want to add or remove columns. Then, in the Physical Level dialog, you can add, remove, or edit columns. When you're finished, click **OK** in the Physical Level dialog.
3. For value-based hierarchies, click the Columns tab. You can add, remove, or edit columns, and designate member key and parent key columns.
4. Click **OK** in the Hierarchy dialog.

View Members in Physical Cube Tables

You can view members of hierarchies or levels in the Physical layer of repositories.

Viewing the list of members by level in the hierarchy can help you determine if the connection pool is set up properly. You might want to reduce the time it takes to return data or the size of

the returned data by specifying a starting point (Starting from option) and the number of rows you want returned (Show option).

1. Open the Administration Tool in online mode.
2. In the Physical layer, right-click a hierarchy or level.
3. Select **View Members**.

   A window opens showing the number of members in the hierarchy and a list of the levels. You might need to enlarge the window and the columns to view all the returned data.
4. Click **Query** to display results.
5. When finished, click **Close**.

Work with Essbase Data Sources

Learn how Essbase data is modeled by default in the Physical layer of the Oracle BI repository, and describes the tasks you can perform to model the data in different ways.

This section contains the following topics:

- About Using Essbase Data Sources
- Work with Essbase Alias Tables
- Model User-Defined Attributes
- Associate Member Attributes to Dimensions and Levels
- Model Alternate Hierarchies
- Model Measure Hierarchies
- Improve Performance by Using Unqualified Member Names

About Using Essbase Data Sources

When you import metadata from Essbase data sources, the cube metadata is mapped to the Physical layer in a way that supports the Oracle BI logical model.

Metadata that applies to all members of the dimension such as aliases are modeled as dimension properties by default. Level-based properties such as outline sort or memnor information are mapped as separate physical cube columns in the dimension.

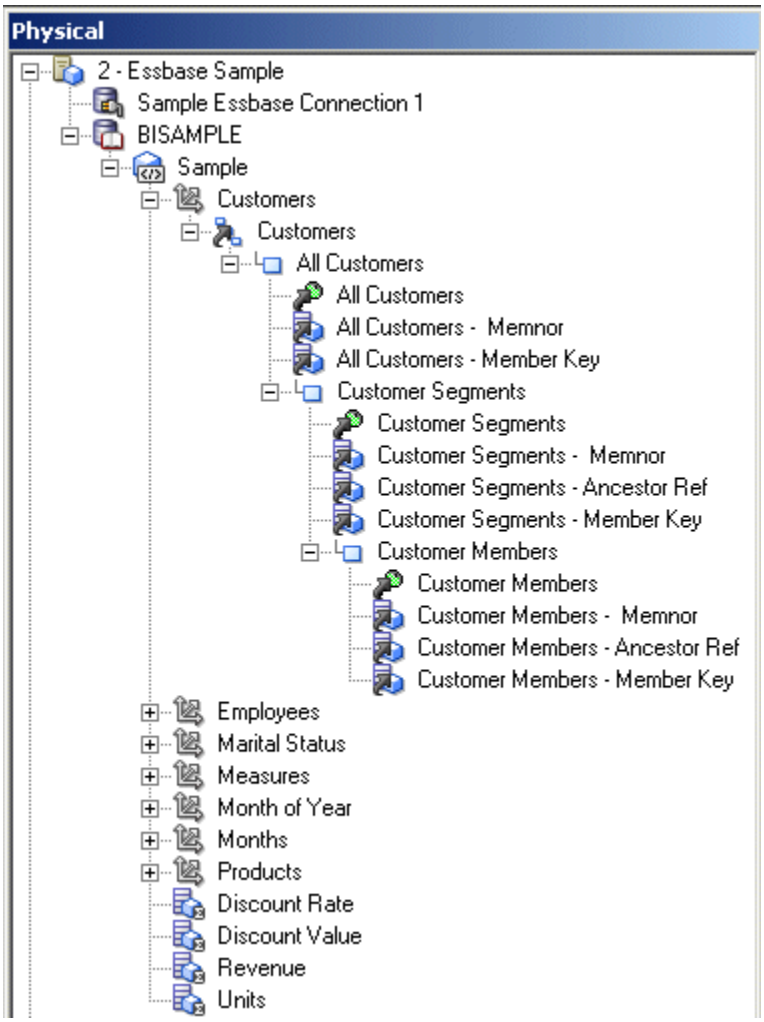
The following physical column types are used for Essbase metadata:

- **Member Alias**: Indicates an Alias column.
- **UDA**: Indicates the column's a User Defined Attribute (UDA).
- **Outline Sort**: Indicates the column's of memnor type, used for outline sorts in the logical layer. Imported at the lowest level of each dimension.
- **Attribute**: Indicates the column's of attribute type, for attribute dimensions.
- **Other**: The type's different than those listed, or unknown.
- **Ancestor Reference**: References the ancestor of a dimension.
- **Member Key**: Indicates the column's a member key.

- **Leaf**: Indicates that the column's the lowest member of the hierarchy.
- **Root**: Indicates that the column's the root member of the hierarchy.
- **Parent Reference**: References the parent of a dimension.

The column types **Outline Sort**, **Ancestor Reference**, **Member Key**, **Leaf**, **Root**, and **Parent Reference** are used internally by the system and shouldn't be changed.

The image shows Essbase data that's been imported into the Physical layer.



There are different options in the Physical layer that let you control how you want to model certain types of metadata. Choose the option that best meets the needs of the user base. For example, many types of Essbase metadata are modeled as dimension properties by default in the Physical layer.

You can choose to flatten the Essbase metadata in the Physical layer for ease of use with the attribute-style reporting supported in previous releases of Oracle BI.

The following list summarizes some of these modeling options:

- Aliases. Aliases are modeled as dimension properties by default, but you can also choose to flatten them using the **Create Columns for Alias Table** feature, see Work with Essbase Alias Tables.

- UDAs. UDAs are modeled as dimension properties by default, but you can also choose to flatten them using the **Create Columns for UDA** feature, see Model User-Defined Attributes.
- Alternate Hierarchies. Alternate hierarchies are modeled as separate hierarchies by default, but you can choose to view them in as a single hierarchy using the **Convert to single hierarchy view** feature, see Model Alternate Hierarchies.
- **Measure Hierarchies** By default, measures are imported as a single measure column that represents all the measures, but you can also choose to view each measure as an individual column using the **Convert measure dimension to flat measures** feature, see Configure Oracle Analytics Server to Use DataDirect.

Using Essbase data sources with Oracle BI repository includes:

- Substitution variables. Essbase substitution variables are automatically retrieved and populated into corresponding Oracle BI Server repository variables. Depending on the scope of the Essbase variable, the naming convention for the Oracle BI Server variable is as follows:

  Server instance scope: *server_name*:*var_name*

  Application scope: *server_name*:*app_name*:*var_name*

  Cube scope: *server_name*:*app_name*:*cube_name*:*var_name*

  A single initialization block is also created in the repository for the Essbase variables. Set the appropriate refresh interval in the initialization block to reflect anticipated update cycles for Essbase variables.
- Essbase Generations. Essbase Generations are mapped to physical level objects.
- Time series functions. The Oracle BI Server time series functions `AGO`, `TODATE`, and `PERIODROLLING` are sent to Essbase to take advantage of the native capabilities of the Essbase server.
- Database functions. You can use the database SQL functions `EVALUATE` and `EVALUATE_AGGREGATE` to leverage functions specific to Essbase data sources.

  The `EVALUATE_PREDICATE` isn't supported for use with Essbase data sources.
- Gen 1 levels. By default, Gen 1 levels are included when you drag and drop an Essbase cube or dimension from the Physical layer to the Business Model and Mapping layer. However, because Gen 1 levels aren't usually needed for analysis, you can choose to exclude Gen 1 levels when you drag and drop Essbase objects to the business model. To do this, select **Skip Gen 1 levels in Essbase drag and drop actions** in the General tab of the Options dialog, see Set Administration Tool Options.
- Hierarchy types. For Essbase data sources, all hierarchies are imported as Unbalanced by default. Review the Hierarchy Type property for each physical hierarchy and change the value if necessary. Supported hierarchy types for Essbase are *Unbalanced*, *Fully balanced*, and *Value*.

About Incremental Import

You can import Essbase metadata incrementally by performing an initial import and then performing another import.

You might want to use an incremental import when information in the data source has changed, or when the first import only included a subset of the metadata.

- When you re-import metadata that already exists in the Physical layer, a message appears, warning you that the Physical objects are overwritten in the import operation.
- If you delete data in the source, re-importing the metadata doesn't automatically delete the same data in the Physical layer. You must manually delete the corresponding Physical objects.
- If you rename an object in the source, the renamed object is imported as a new object. The old object and the new (renamed) object are both displayed in the Physical layer.
- Customizations in the Physical layer data such as creating an alias column to use for display are retained after an incremental import. If you want to revert to the default imported view, you must delete the existing Physical layer objects, and re-import the metadata.

Work with Essbase Alias Tables

Essbase cubes support aliases which are alternate names for members or shared members. Members might have separate aliases for each user language to enable users to view member names in their own language.

For example, the member name might be a product code (100), with a default alias for the product name (Cola) and an additional alias for the long name (Cherry Cola).

Aliases are stored in alias tables that map a specific set of alias names to member names. A default alias table exists for each cube.

This section contains the following topics:

- Determine the Value to Use for Display
- Explicitly Define Columns for Each Alias

Determine the Value to Use for Display

When you import metadata from Essbase into the Oracle BI repository, the Essbase cube table object in the Physical layer has a property that determines which value to display for members.

The values are for the member name, the default alias name, or some other alias name. By default, the columns display the default alias name.

1. In the Physical layer of the Administration Tool, double-click an Essbase cube table.
2. In the General tab of the Cube Table dialog, choose the appropriate value for **Display Column**, select one of the following:
   - **Member Names**.
   - **Alias** and choose an alias table name from the list.
   - **Variable** and choose a variable that contains a valid display column name.
3. Click **OK**.

Explicitly Define Columns for Each Alias

Aliases are modeled as dimension properties in the Physical layer after import.

If you want to work with more than one alias, such as when you want to flatten attributes for reporting purposes or externalize strings for translation, you can explicitly define columns for each alias. You can define alias columns at the cube, dimension, or hierarchy level.

1. In the Administration Tool, in the Physical layer, right-click the cube table, physical dimension, or physical hierarchy for which you want to define alias columns.
2. Select **Create Columns for Alias Table**. Then, from the sub-list, select the alias table for which you want to create columns.
3. Click **Create**.
4. Drag the new alias columns to the appropriate location in the Business Model and Mapping layer.

To externalize strings for translation based on the alias columns, see Localize Oracle Analytics Server in *Administering Oracle Analytics Server*.

Model User-Defined Attributes

Essbase supports the concept of user-defined attributes (UDAs). A UDA is any arbitrary textual string that you can associate with any member from a dimension.

A member can have multiple strings associated to it.

You can choose whether to import UDAs in the Import Metadata Wizard. If you choose to import UDAs, then by default, each UDA is modeled as a dimension property in the Physical layer of the repository.

You can also choose to model each UDA as a separate physical column. To do this, perform one of the following tasks:

- To model all UDAs, do one of the following:
  - Right-click the cube table and select **Create columns for UDA**. All UDAs in the cube are modeled as separate physical columns.
  - Right-click the dimension object and select **Create columns for UDA**, then select **All UDAs**. All UDAs in the dimension are modeled as separate physical columns.
  - Right-click the dimension object and select **Create columns for UDA**, then select the specific UDA you want to model. The selected UDA is modeled as a separate physical column for each level.

Associate Member Attributes to Dimensions and Levels

Member attributes aren't automatically associated to corresponding dimensions and levels during the import process.

- Open the Oracle BI Administration Tool, drag and drop the columns from the attribute dimension in the Physical layer to the appropriate logical tables in the Business Model and Mapping layer.

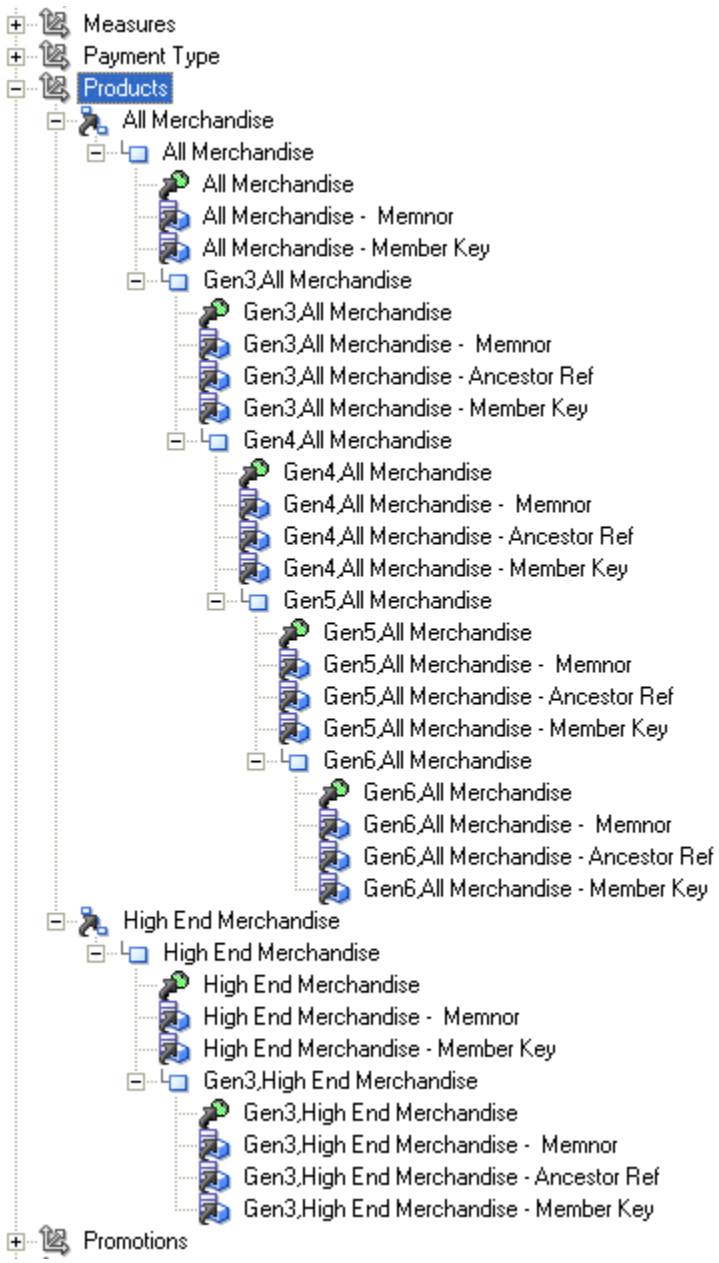
Model Alternate Hierarchies

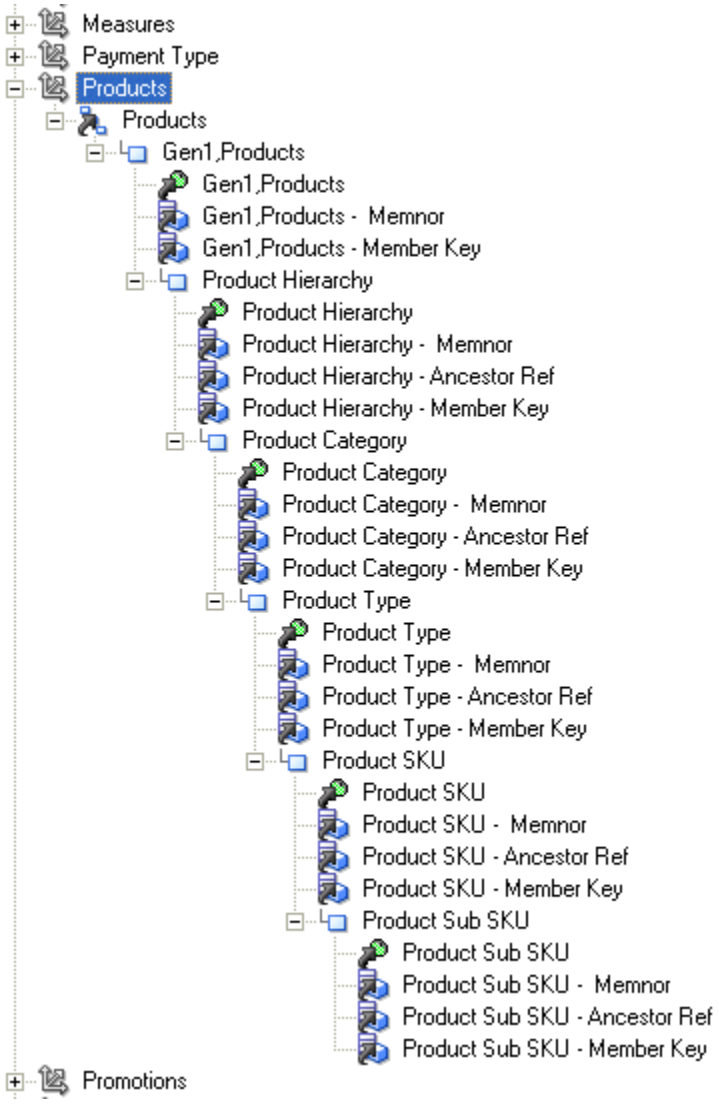
Alternate hierarchies are modeled as separate hierarchies in the Physical layer.

You can choose to view them as separate hierarchies, called the multi-hierarchy view, or as a single hierarchy.

To view alternate hierarchies as a single hierarchy, right-click the dimension object containing the alternate hierarchies and select **Convert to single hierarchy view**. To return to the multi-hierarchy view, right-click the dimension object again and select **Convert to multi-hierarchy view**.

For example, the image shows the multi-hierarchy view for an alternate hierarchy.





Model Measure Hierarchies

Measures are imported as measure hierarchies. The cube contains a single measure column that represents all the measures.

You can choose to flatten the measure hierarchy to view each measure as an individual column.

- In the Administration Tool, right-click the cube object, and select **Convert measure dimension to flat measures** to view an individual column.

Improve Performance by Using Unqualified Member Names

When member names (including aliases) are unique in a given hierarchy, the Oracle BI Server can take advantage of specific MDX syntax to optimize performance.

The import process can't verify that member names are unique for a given hierarchy. You must confirm member name uniqueness. Query errors result when a hierarchy is specified as having unique members when it doesn't.

- Do one of the following:
  - From the Essbase outline, update each offending member variable by adding a prefix or suffix to make the member name unique, update SQL queries, and reload the data and members in the Essbase outline.
  - In the Hierarchy dialog, select **Use unqualified member name for better performance** to enable this capability.

Work with Hyperion Financial Management and Hyperion Planning Data Sources

Learn about importing and querying metadata from Hyperion Financial Management and Planning data sources.

This topic contains the following sections:

- Import Metadata from Hyperion Financial Management Data Sources
- Import Metadata from Hyperion Planning Data Sources
- About Query Support for Hyperion Financial Management and Hyperion Planning Data Sources

Import Metadata from Hyperion Financial Management Data Sources

When you import metadata from Hyperion Financial Management data sources, both measures and dimensions are imported into the Physical layer.

The Hyperion Financial Management hypercube model is exposed in the Physical layer in the following ways:

- Hyperion Financial Management has one measure, called Value. The Value measure is modeled as a single fact column in the Physical layer. The column is called Value.
- The Value measure column uses the DOUBLE data type.
- The `Value` measure has three base properties: `CellText`, `CurrencyType`, and `Attribute`. These properties are all represented as additional fact columns.
- The `Attribute` property has additional properties such as `IsReadOnly`. These properties are exposed as additional columns.

All Hyperion Financial Management dimensions are modeled as parent-child hierarchies in the Physical layer. Alternate hierarchies and unbalanced hierarchies are supported.

Shared members from Hyperion Financial Management or Hyperion Planning data sources aren’t supported. The Oracle BI Server can't support shared members on Hyperion Financial

Management sources because the shared member key value returned by HFM doesn't always match the member key value for the parent member. If the Oracle BI Server ignores the prefix for the shared member key, the result negates the uniqueness of the member key making it impossible for the Oracle BI Server to differentiate instances of the shared members and the parent. For example, if Hyperion Financial Management returns the *ConsolGroup3.EasternUSA* value for a shared member of the parent key value, *DomesticEntities.EasternUSA*, and the Oracle BI Server removes the *ConsolGroup3* prefix to try to match the group members, the instances all have the key, *EasternUSA*. Modify the data model modified in the Hyperion Financial Management layer to ensure unique member keys on the Hyperion Financial Management server.

Dimension member properties are exposed as columns such as Name, Description, and ShortName. An additional column called Sort Order is also displayed for each dimension. This column contains custom sort information retrieved from the Hyperion Financial Management data source.

Each Hyperion Financial Management dimension has a corresponding Point of View (POV) value that provides customized information for different users. This POV value is exposed as the **Default Member** in the Hierarchies tab of the Dimension dialog. Although the **Default Member** field is populated upon import, you might need to update the default values according to your user needs.

> **Note:**
>
> Don't select the **Default member type ALL** option for Hyperion Financial Management hierarchies.

Import Metadata from Hyperion Planning Data Sources

When you import data from Hyperion Planning data sources, both measures and dimensions are imported into the Physical layer.

Hyperion Planning Server version 11.1.2.4 or above is required for importing and querying metadata from Hyperion Planning Data Sources.

The Hyperion Planning model is exposed in the Physical layer in the following ways:

- The imported Hyperion Planning data source contains multiple Value columns to support measures of different data types.
- Some measures specific to Hyperion Financial Management aren't imported for Hyperion Planning data sources.
- The Attribute property has additional sub-properties, such as IsReadOnly. These properties are also exposed as additional columns.

All Hyperion Planning dimensions are modeled as parent-child hierarchies in the Physical layer. Shared members, alternate hierarchies, and unbalanced hierarchies are supported.

Dimension member properties are exposed as columns such as Name, Description, and ShortName. An additional column called Sort Order is also displayed for each dimension. This column contains custom sort information retrieved from the Hyperion Planning data source.

Each Hyperion Planning dimension has a corresponding Point of View (POV) value that provides customized information for different users. This POV value is exposed as the **Default Member** in the Hierarchies tab of the Dimension dialog. The Default Member field is populated upon import. You might need to update the default values according to the user needs.

> **Note:**
>
> Don't select the **Default member type ALL** option for Hyperion Planning hierarchies.

About Query Support for Hyperion Financial Management and Hyperion Planning Data Sources

Both member queries (dimensional browsing) and data queries (measure analysis) are supported for Hyperion Financial Management and Hyperion Planning data sources.

Use the `EVALUATE_PREDICATE` Logical SQL function to access these functions specific to Hyperion Financial Management and Hyperion Planning:

- `PeriodOffset` used to access prior or future periods through an offset.
- `SuppressDerived`, `SuppressInvalidIntersection`, `SuppressNoAccess`, `SuppressZero`, `SuppressError` NA suppression functions specific to Hyperion Financial Management.
- `Base` to return the leaf members below a given ancestor member.
- `CommonChildren`.
- User-defined functions.

Oracle Analytics Server supports `PERF_PREFER_SUPPRESS_EMPTY_TUPLES` for inserting the `SuppressMissing` function into the Hyperion Financial Management or Hyperion Planning data query to suppress missing cells. The `PERF_PREFER_SUPPRESS_EMPTY_TUPLES` function controls whether empty tuples with empty cell values are eliminated. `PERF_PREFER_SUPPRESS_EMPTY_TUPLES` doesn't change the null suppression behavior on the final result set.

There is no native support for time series functions. Time series functions are only supported through data modeling.

Work with Oracle OLAP Data Sources

Oracle Database has an OLAP Option that provides an embedded, full-featured online analytical processing server.

The OLAP Option is used in the following roles:

- A summary management solution to SQL-based business intelligence tools and applications.
- A calculation engine that provides SQL-based business intelligence tools with rich analytic content.

- A full-featured multidimensional server, servicing dimensionally oriented business intelligence tools and applications.

Oracle Analytics Server supports Oracle OLAP as a data source. When you import metadata from an Oracle OLAP source, the Oracle OLAP objects appear in the Physical layer of the Administration Tool. This section provides information about the Oracle OLAP objects in the Physical layer.

This section contains the following topics:

- About Importing Metadata from Oracle OLAP Data Sources
- Work with Oracle OLAP Analytic Workspace (AW) Objects
- Work with Oracle OLAP Dimensions, Hierarchies, and Levels
- Work with Oracle OLAP Cubes and Columns

About Importing Metadata from Oracle OLAP Data Sources

Learn how to use the Oracle BI Administration Tool to import metadata from Oracle OLAP.

When using the Oracle BI Administration Tool:

- For Oracle OLAP cubes with multi-language metadata, only the default language is imported.
- Only dimensions that contain at least one hierarchy are imported.
- Multiple hierarchies in a single query aren't supported. If a query includes columns from multiple hierarchies in a given dimension, the Oracle BI Server returns an error.
- The default aggregation rule in the Business Model and Mapping layer for Oracle OLAP measures is External Aggregation. The External Aggregation rule means that the Oracle BI Server isn't aware of the underlying aggregation rule for the specific measure and doesn't compute it internally. Instead, the Oracle BI Server always sends the query to the underlying multidimensional data source for aggregation.

  In some cases, you may want to set the aggregation rule for a measure to something other than External Aggregation. For example, you can have federated multiple data sources, or you might want to perform higher-level aggregation along dimension attributes that aren't represented by a level in Oracle OLAP. In both of these cases, you can change the default aggregation rule to match the rule in the underlying data source or sources. The aggregation is performed in the Oracle OLAP data source where possible.

Work with Oracle OLAP Analytic Workspace (AW) Objects

You can view Oracle OLAP Analytic Workspace (AW) objects in the Physical layer of the Oracle BI Administration Tool.

These objects correspond to the analytic workspace object in the Oracle OLAP metadata, and are similar to physical catalog and physical schema objects. Analytic workspaces are containers for storing related cubes. You create dimensions, cubes, and other dimensional objects within the context of an analytic workspace.

Oracle OLAP Analytic Workspace objects have properties for **Name**, **Description**, and **Dynamic Name**. You can use the Dynamic Name tab to provide a variable that specifies the name of the Analytic Workspace object. You must define at least one session variable to make the Dynamic Name tab inactive. See Use a Variable to Specify the Name of a Catalog or Schema.

Work with Oracle OLAP Dimensions, Hierarchies, and Levels

Oracle OLAP dimensions are lists of unique values that identify and categorize data.

They form the edges of a cube, and thus of the measures within the cube. In a report, the dimension values or their descriptive attributes provide labels for the rows and columns.

There are three types of Oracle OLAP dimensions:

- **Level-based dimensions** Members of level-based dimensions naturally group into levels based on their type such as month and year. Most dimensions are level-based.
- **Value-based dimensions** These dimensions have parent-child relationships among their members, but the members are all the same type such as Employee or Account, so these relationships don't form meaningful levels.
- **List or flat dimensions** These dimensions have no levels or hierarchies.

  Oracle Analytics Server doesn't support dimensions that have no hierarchies (flat dimensions). Importing flat dimensions from an Oracle OLAP data source results in an error. If you've flat dimensions, replace the flat dimensions with single-level hierarchies in the data source before you attempt to import.

You can edit the name and description of the dimension and the following dimension properties:

- *Time* Indicates that this dimension is a time dimension.
- *Ragged* Indicates that this dimension contains a hierarchy that has at least one member with a different base, creating a ragged base level for the hierarchy
- *Skipped levels* Indicates that this dimension contains a hierarchy that has at least one member whose parents are more than one level above it, creating a hole in the hierarchy. An example of a skip-level hierarchy is City-State-Country, where at least one city has a country as its parent, for example, Washington D.C. in the United States.
- *External Name* The physical name that's used when referencing the dimension in physical SQL queries. This value must reflect the external name defined in the data source.
- *Cache properties.* Select **Cacheable** to include this dimension in the Oracle BI Server query cache. To specify that cache entries don't expire, select **Cache never expires**. Alternatively, you can select **Cache persistence time** and enter a value to specify how long entries should persist in the query cache. If a query references multiple physical objects with different persistence times, the cache entry for the query exists for the shortest persistence time set for any of the tables referenced in the query. This makes sure that no subsequent query gets a cache hit from an expired cache entry.

The Columns and Hierarchies tabs of the Oracle OLAP Dimension dialog list the dimension members and hierarchies that belong to the dimension. In the Columns tab, you can add or remove columns, and edit particular columns. In the Hierarchies tab, you can add, remove, or edit hierarchies. You can also use the type (key) button to select the default hierarchy for the dimension.

Dimensions can contain one or more hierarchies. Most hierarchies are level-based and consist of one or more levels of aggregation. Members roll up into the next higher level in a many-to-one relationship, and these members roll up into the next higher level, and then to the top level. Ragged and skip-level hierarchies are also supported.

Dimensions can contain value-based hierarchies which are parent-child hierarchies that don't support levels. For example, an employee dimension might have a parent-child relationship that identifies each employee's supervisor. The levels that group together first-, second-, and third-level supervisors might not prove meaningful for analysis.

For value-based hierarchies, the *Nullable* option is selected by default for the root member physical cube column. You must select the *Nullable* option for the root member for value-based hierarchies to work correctly.

Multiple hierarchies for a dimension typically share the base-level dimension members and branch into separate hierarchies. They can share the top level if they use all the same base members and use the same aggregation operators. Otherwise, they need different top levels to store different aggregate values.

Use the Oracle OLAP Hierarchy dialog to view and edit the name, external name, and description of the hierarchy. For level-based hierarchies, you can add, remove, edit, or reorder levels. For value-based hierarchies, you can add, remove, or edit columns.

1. In the Business Model and Mapping column, right-click an OLAP table, select **Query Related Objects**, and select **Oracle OLAP Hierarchy**.
2. Double-click a column name to designate a column as a level key,

Work with Oracle OLAP Cubes and Columns

Oracle OLAP cubes are informational objects that identify measures with the exact same dimensions and thus are candidates for being processed together at all stages: data loading, aggregation, storage, and querying.

Cubes define the shape of the business measures and are defined by a set of ordered dimensions. The dimensions form the edges of a cube, and the measures are the cells in the body of the cube.

Oracle OLAP cubes have properties similar to other cubes. You can view, edit the name and description of the cube, and update the following cube properties:

- **External Name.** The physical name that's used when referencing the cube in physical SQL queries. This value must reflect the external name defined in the data source.
- **Density and Materialization.** For Oracle OLAP cubes that are sparse and fully materialized, specify values for these properties to optimize queries. If you set the **Density** option to **Sparse** and the **Materialization** option to **Fully Materialized**, the Oracle BI Server generates a loop clause to skip empty cells. If you leave the **Density** option blank, the Oracle BI Server assumes the data is sparse.

  If you set these values, make sure that you set them to reflect the actual properties of the data source. Don't specify that the data is sparse and fully materialized unless this is true for the data source.

  You don't need to set these values for Oracle OLAP 11*g* cubes. For these objects, optimization happens automatically.

- **Cache properties.** Select **Cacheable** to include the cube in the Oracle BI Server query cache.

  To specify that cache entries don't expire, select **Cache never expires** or select **Cache persistence time** and enter a value to specify how long entries should persist in the query cache.

  If a query references multiple physical objects with different persistence times, the cache entry for the query exists for the shortest persistence time set for any of the tables referenced in the query. This makes sure that no subsequent query gets a cache hit from an expired cache entry.

The Columns tab on the Oracle OLAP Cube dialog lists the columns that belong to the cube. You can add or remove columns, and edit particular columns.

You can define measures, calculated measures, attributes, or level keys as Oracle OLAP columns. The Oracle OLAP columns have the same properties as other physical columns. See Create and Edit a Column in a Physical Table.

Work with Physical Foreign Keys and Joins

You can create physical foreign keys and complex joins using either the Physical Diagram, or the Joins Manager.

However, you don't create joins for multidimensional data sources.

This section contains the following topics:

- About Physical Joins
- Define Physical Joins with the Physical Diagram
- Define Physical Joins with the Joins Manager

About Physical Joins

When you import keys in a physical schema, the primary key-foreign key joins are automatically defined.

You must explicitly define any other joins in each data source or between data sources to express relationships between tables in the Physical layer.

You don't have to use imported key and foreign key joins in metadata. Joins that are defined to enforce referential integrity constraints can result in specifying incorrect joins in queries. For example, joins between a multipurpose lookup table and several other tables can result in unnecessary or invalid circular joins in the SQL queries issued by the Oracle BI Server.

This section contains the following topics:

- About Primary Key and Foreign Key Relationships
- About Complex Joins
- About Multi-Database Joins
- About Fragmented Data

About Primary Key and Foreign Key Relationships

A primary key and foreign key relationship defines a one-to-many relationship between two tables.

A foreign key is a column or a set of columns in one table that references the primary key columns in another table. The primary key is defined as a column or set of columns where each value is unique and identifies a single row of the table.

There are two cases where multiple foreign key columns in a table point to the same table:

- When the primary key of the foreign table is concatenated, meaning that it consists of a set of columns. This is a single join between two tables that happens to use multiple columns.
- When you've created an alias to the foreign table, because the foreign table needs to serve in different roles. Each foreign key joins to a primary key in a role-playing alias; see About Physical Alias Tables.

You can specify primary key and foreign keys in the Physical Diagram, or by using the Keys and Foreign Keys tabs of the Physical Table dialog, see Define Physical Joins with the Physical Diagram and Create and Manage Columns and Keys for Relational and Cube Tables.

About Complex Joins

In the Physical layer of the repository, complex joins are joins over non-foreign key and primary key columns. In other words, physical complex joins are joins that use an expression rather than key column relationships.

When you create a complex join in the Physical layer, you specify the expression for the join.

For most data sources, foreign key joins are preferred for performance reasons. Complex joins usually don't perform as well because they don't use key column relationships to form the join. The exception is ADF data sources, which use physical complex joins exclusively to denote ViewLink instances that connect pairs of View Objects in the ADF model.

About Multi-Database Joins

A multi-database join is defined as a table under one metadata database object that joins to a table under a different metadata database object.

You must specify multi-database joins to combine the data from different databases. Use the Physical Diagram to specify multi-database joins, see Define Physical Joins with the Physical Diagram.

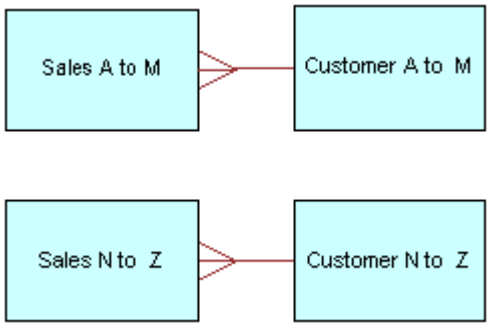
You can create multi-database joins between tables in most types of databases. You can't create multi-database joins to tables in Oracle OLAP data sources.

While the Oracle BI Server has several strategies for optimizing the performance of multi-database joins, these joins are significantly slower than joins between tables within the same database. As a result of the negative performance impact, you should avoid using multi-database joins whenever possible.

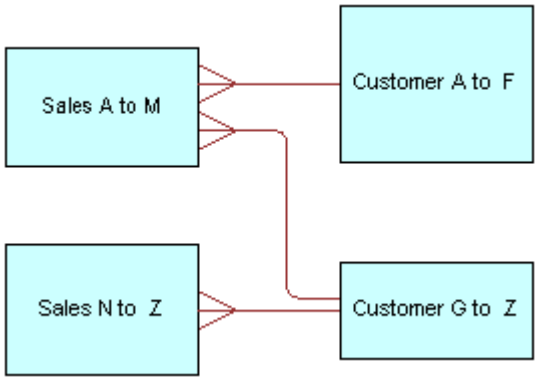
About Fragmented Data

Fragmented data is data from a single domain that's split between multiple tables.

For example, a data source might store sales data for customers with last names beginning with the letter A through M in one table and last names from N through Z in another table. With fragmented tables, you need to define all of the join conditions between each fragment and all the related tables. The figure shows the physical joins with a fragmented sales table and a fragmented customer table where the data are fragmented the same way (A through M and N through Z).



You could have a fragmented fact table and a fragmented dimension table with fragments across different values. You create the joins to the fragmented table and define a one-to-many join, as shown in the Customer A to F and from Customer G to Z to Sales A to M example.



Note:

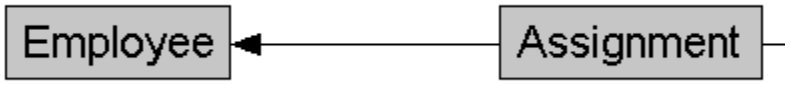
Avoid adding join conditions where they aren't necessary, for example, between Sales A to M and Customer N to Z. Extra join conditions can cause performance degradations.

Define Physical Joins with the Physical Diagram

You can define foreign keys and complex joins between tables, whether or not the tables are in the same data source.

When you use the Physical Diagram to create joins, the Administration Tool determines what type of join to create based on the selected object types and the join expression.

In the Physical Diagram, the join is represented by a line between the two selected tables, with an arrow at the *one* end of the join. The image shows a join in the Physical Diagram.



The physical foreign key joins are the default join type. The object type might change to a complex join after you define the join and click **OK**.

If you don't want the Administration Tool to automatically determine what type of join to create, use the Joins manager to explicitly create the join. See Define Physical Joins with the Joins Manager.

See Use Hints in SQL Statements and Specify a Driving Table. The driving table option isn't available for selection because the Oracle BI Server implements driving tables only in the Business Model and Mapping layer.

1. In the Physical layer of the Administration Tool, right-click a table, from Physical Diagram, select an option.
2. From the Diagram menu, click **New Join**.
3. In the Physical Diagram, select the first table in the join, the table representing many in the one-to-many join.
4. Move the cursor to the table to which you want to join, the table representing one in the one-to-many join, and select the second table.
5. Select the joining columns from the left and the right tables.
6. Optional: For complex joins, you can set the cardinality for each side of the join, for example, *N*, *0,1*, *1*, or *Unknown*.

   To set the cardinality to unknown, you only need to select *Unknown* for one side of the join. For example, choosing unknown-to-1 is equivalent to unknown-to-unknown and appears as such the next time you open the dialog for this join.
7. If appropriate, specify a database hint.
8. If you're creating a complex join for ADF ViewObject or ViewLink instances, specify the ViewLink instance name or the ViewLink definition name in the **ViewLink Name** field.
9. To open Expression Builder, click the button to the right of the Expression pane. The expression displays in the Expression pane.

The default join expression for ViewObject or ViewLink instances is arbitrary and has no meaning.

10. Click **OK** to apply the selections.

Define Physical Joins with the Joins Manager

You can use the Joins Manager to view join relationships and to create physical foreign key joins and complex joins.

See Use Hints in SQL Statements.

1. In the Administration Tool toolbar, select **Manage**, then select **Joins**.
2. In the Joins Manager dialog, perform one of the following tasks:
   - From **Action** , select **New** , and then select **Complex Join**.
   - From **Action**, select **New**, select **Physical Foreign Key**, and then click **Browse** and double-click a table.
3. In the Complex Join or Physical Foreign Key dialog, type a name for the join.
4. Click the **Browse** button for the **Table** field on the left side of the dialog, and locate the table that the foreign key references.
5. Select the columns in the left table that the key references.
6. Select the columns in the right table that comprise the foreign key columns.
7. Optional: For complex joins, you can set the cardinality for each side of the join, for example, *N*, *0,1*, *1*, or *Unknown*.

   To set the cardinality to unknown, you only need to select **Unknown** for one side of the join. For example, choosing unknown-to-1 is equivalent to unknown-to-unknown and appears as such the next time you open the dialog for this join.
8. If appropriate, specify a database hint.
9. If you're creating a complex join for ADF ViewObject or ViewLink instances, specify the ViewLink instance name or the ViewLink definition name in the **ViewLink Name** field.
10. Click the button to the right of the Expression pane to open the Expression Builder.

    The default join expression for ViewObject or ViewLink instances is arbitrary and has no meaning.
11. Click **OK** to save.

Deploy Opaque Views

An opaque view is a Physical layer table that consists of a SELECT statement.

When you need a new table, you must create a physical table or a materialized view. Use an opaque view only if there is no other solution.

See Exchange Metadata with Databases to Enhance Query Performance.

This section contains the following topics:

- About Deploying Opaque Views
- Deploy Opaque View Objects

- Undeploy a Deployed View
- When to Delete Opaque Views or Deployed Views
- When to Redeploy Opaque Views

About Deploying Opaque Views

You deploy an opaque view in the data source using the Deploy Views utility.

In the repository, opaque views appear as view tables in the data source, but the view doesn't actually exist until you deploy it. Data sources such as XLS and non-relational data sources don't support opaque views and can't run the view deployment utility.

After deploying an opaque view, it's called a deployed view. Opaque views can be used without deploying them, but the Oracle BI Server has to generate a more complex query when an opaque view is encountered.

To verify that opaque views are supported by a data source, check whether the `CREATE_VIEW_SUPPORTED` SQL feature is selected in the Database dialog, in the Features tab. See SQL Features Supported by a Data Source.

Deploy Opaque View Objects

In offline mode, the Deploy Views utility is available when importing from data sources with ODBC data sources.

Oracle Native (client) drivers are also supported in the offline mode for deploying views. In online mode, view deployment is available for supported data sources using Import through server, the settings on the client are ignored.

Use the Create View SELECT Statement

The SQL statement for deploying opaque views in the Physical layer of the repository is available for supported data sources.

To determine which data sources support opaque views, contact your system administrator or consult your data source documentation.

Use only repository variables in the definition. The system generates an error if the view definition contains a session variable.

Syntax

`CREATE VIEW` *`view_name`* `AS` *`select_statement`*,

Where:

- *select_statement* is the user-entered SQL statement in the opaque view object. If the SQL statement is invalid, the create view statement fails during view deployment.
- *view_name* is one of the two following formats: `schema.viewname`, or `viewname`. The connection pool settings determine if the schema name is added.

If you want your SELECT statement to reference a row-wise initialization variable, then you must use the `VALUELISTOF` function. For example, to get the customers assigned to the user names in the variable `LIST_OF_USERS`, use the following syntax:

- `RW.CUSTOMERS.USER_NAME in (VALUELISTOF(NQ_SESSION.LIST_OF_USERS))`
- To filter by only specific values in the list, then use `ValueNameof` as show in the below example. The first value is 0, not 1.

  ```
  RW.CUSTOMERS.USER_NAME in '(ValueNameOf(0,NQ_SESSION.LIST_OF_USERS))
  ```

For opaque view objects, the right-click menu contains the **Deploy View(s)** option. When you select **Deploy View(s)**, the Create View SQL statement runs and attempts to create the deployed view objects. The following list describes how to initiate view deployment and the results of each method:

- Right-click a single opaque view object. When you select **Deploy View(s)**, the Create View SQL statement runs and attempts to create a deployed view for the object.
- Right-click several objects. If at least one of the selected objects is an opaque view object, the right-click menu contains the **Deploy View(s)** option. When you select **Deploy View(s)**, the Create View SQL statement runs and attempts to create the deployed views for any qualifying objects.
- Right-click a physical schema or physical catalog. If any opaque view object exists in the schema or catalog, the right-click menu contains the **Deploy View(s)** option. When you select **Deploy View(s)**, the Create View SQL statements for all qualifying objects run and attempt to create deployed views for the qualifying objects contained in the selected schema or catalog.

During deployment, names are assigned to the views. If you change the preassigned name, the new name must use alphanumeric characters with a maximum length of 18 characters. If these guidelines aren't followed, the object name is automatically transformed to a valid name using the following Name Transform algorithm:

- All non-alphanumeric characters are removed.
- If there are 16 or more characters after Step 1, the first 16 characters are kept.
- Two digits starting from 00 to 99 are appended to the name to make the name unique in the corresponding context.

After the deployment process completes, the following occurs:

- Views that have been successfully and unsuccessfully deployed appear in a list.
- For unsuccessful deployments, a brief reason appears in the list.
- If deployment is successful, the object type of the opaque view changes from Select to None and the deployed view is treated as a regular table.

  If you change the type back to Select, the associated opaque views are dropped from the data source, or an error message appears. See When to Delete Opaque Views or Deployed Views.
- In the Administration Tool, the view icon changes to the deployed view icon for successfully deployed views.

1. In the Physical layer of the Administration Tool, right-click the opaque view that you want to deploy.
2. In the right-click menu, select **Deploy View(s)**.
3. (Optional) In the View Deployment - Deploy View(s) dialog, in the **New Table Name** column, change the new deployed view names.

If the change doesn't conform to the naming rules, a new name is assigned and the dialog appears again so that you can accept or change it. This action repeats until all names pass validation.

If you don't want to deploy one or more of the views, clear the appropriate rows.

4. In the Select Connection Pool dialog, choose a connection pool, and click **Select**.
5. In the View Deployment Messages dialog, search for views using **Find** and **Find Again**, or copy the contents.
6. When you're finished, click **OK**.

Undeploy a Deployed View

Running the Undeploy Views utility on a deployed view deletes the view and converts the view table back to an opaque view with its original `SELECT` statement.

1. In the Physical layer of the Administration Tool, right-click a physical database, catalog, schema, or table.

   If a deployed view exists that's related to the selected object, the right-click menu contains the **Undeploy View(s)** option.
2. Select **Undeploy View(s)**.

   A list of views to be undeployed appears.
3. If you don't want to undeploy one or more of the views, clear the appropriate rows.
4. In the View Deployment - Undeploy View(s) dialog, click **OK** to remove the views.

   A message appears if the undeployment was successful.
5. In the View Deployment Messages dialog, search for undeployed views using **Find** and **Find Again**, or copy the contents.
6. When you're finished, click **OK**.

When to Delete Opaque Views or Deployed Views

Use these guidelines to remove opaque or deployed view objects in the repository.

- Removing an undeployed opaque view in the repository

  If the opaque view hasn't been deployed, you can delete it from the repository.
- Removing a deployed view

  When you deploy an opaque view, a view table is created physically in both the data source and the repository. Therefore, you must undeploy the view before deleting it. You use the Undeploy Views utility in the Administration Tool. This utility removes the opaque view from the back-end data source, changes the Table Type from None to Select, and restores the `SELECT` statement of the object in the Physical layer of repository.

> **Important:**
>
> Don't manually delete the view table in the data source. If this table is deleted, then the Oracle BI Server can't query the view object. When you undeploy the view, it's removed automatically from the data source.

When to Redeploy Opaque Views

After removing an opaque view, you can redeploy the same opaque view.

The Administration Tool doesn't distinguish between a first-time deployment and a redeployment. Make sure that you remove a deployed view before deploying the opaque view again. The deploy operation fails and the data source returns error messages if you don't remove the deployed view before deploying the opaque view again.

Use Hints in SQL Statements

Hints are instructions that you place within a SQL statement that tell the data source query optimizer the most efficient way to run the statement.

Hints override the optimizer's processing plan, so you can use hints to improve performance by forcing the optimizer to use a more efficient plan. Hints are only supported for Oracle Database data sources.

Using the Administration Tool, you can add hints to a repository, in both online and offline modes, to optimize the performance of queries. When you add a hint to the repository, you associate it with Physical layer objects. When the object associated with the hint is queried, the Oracle BI Server inserts the hint into the SQL statement.

The table shows the physical objects with which you can associate hints. It also shows the Administration Tool dialog that corresponds to the physical object. Each of these dialogs contains a **Hint** field, into which you can type a hint to add it to the repository.

Database Object	Dialog
Complex join	Complex Join
Physical foreign key	Physical Foreign Key
Physical table	Physical Table - General tab

Hints are only supported when the **Table Type** is set to **Physical Table**. For other table types, the hint text is ignored. For physical tables with a table type of **Select**, you can provide the hint text as part of the SQL statement entered in the **Default Initialization String** field.

How to Use Oracle Hints

Review these topics about using Oracle hints with the Oracle BI Server.

See Oracle hints in the SQL reference guide for the version of the Oracle Database that you use.

This section contains the following topics:

- About the Index Hint

- About the Leading Hint

About the Index Hint

The Index hint explains how the optimizer scans a specified index rather than a table.

If queries against the `ORDER_ITEMS` table are slow, you can review the processing plan of the query optimizer. If the `FAST_INDEX` wasn't used, you can create an Index hint to force the optimizer to scan the `FAST_INDEX` rather than the `ORDER_ITEMS` table. The syntax for the Index hint is as follows:

`index(`*`table_name`*`,`*`index_name`*`)`

To add this hint to the repository, open the Physical Table dialog in the Administration Tool, and type the following text in the **Hint** field:

```
index(ORDER_ITEMS, FAST_INDEX)
```

About the Leading Hint

The Leading hint forces the optimizer to build the join order of a query with a specified table.

The syntax for the Leading hint is `leading(`*`table_name`*`)`. If you were creating a foreign key join between the Products table and the Sales Fact table and wanted to force the optimizer to begin the join with the Products table, you would go to the Physical Foreign Key dialog in the Administration Tool and type the following text in the **Hint** field:

```
leading(Products)
```

About Performance Considerations for Hints

Hints that are well researched and planned can result in significantly better query performance.

However, hints can also negatively affect performance if they result in a suboptimal processing plan.

Follow these guidelines to create hints to optimize query performance:

- Only add hints to a repository after you've tried to improve performance in the following ways such as:
  - Adding physical indexes or other physical changes to the Oracle Database.
  - Makin modeling changes within the server.
- Avoid creating hints for physical table and join objects that are queried often. If you drop or rename a physical object that's associated with a hint, you must also alter the hints accordingly.

Create Hints

You can add hints to the repository using the Oracle BI Administration Tool.

Although hints are identified using SQL comment markers (/* or --), don't type SQL comment markers when you type the text of the hint. The Oracle BI Server inserts the comment markers when the hint is run.

1. In the Administration Tool, go to one of the following dialogs:
   - Physical Table—General tab
   - Physical Foreign Key
   - Complex Join
2. Type the text of the hint in the **Hint** field and click **OK**.

   For a description of available Oracle hints and hint syntax, see SQL reference for the version of the Oracle Database that you use.

Display and Update Row Counts for Physical Tables and Columns

When you request row counts, the Administration Tool retrieves the number of rows from the data source for all, or selected tables and columns (distinct values are retrieved for columns), and stores those values in the repository.

The amount of time required to count the number of rows depends upon the number of row counts retrieved.

When updating all row counts, the Updating Row Counts window appears while row counts are retrieved and stored. If you click **Cancel**, the retrieve process stops after the in-process table and its columns have been retrieved. Row counts include all tables and columns for which values were retrieved before the cancel operation.

Updating all row counts for a large repository might take a long time to complete. You might want to update only selected table and column counts.

Row counts aren't available for the following:

- Stored Procedure object types
- XML data sources and XML Server data sources
- Multidimensional data sources
- Data sources that don't support the `COUNTDISTINCT` function, such as Microsoft Access and Microsoft Excel, or data sources for which `COUNT_STAR_SUPPORTED` has been disabled in the database features table
- In online mode, Update Row Count doesn't work with connection pools in which the session variables *:USER* and *:PASSWORD* are set as the user name and password.

  In offline mode, the Set values for variables dialog appears so that you can populate the session variables *:USER* and *:PASSWORD*.

- In online mode, after importing or manually creating a physical table or column, the Oracle BI Server doesn't recognize the new objects until you check them in. **Update Row Count** isn't available in the menu until you check in the objects.
- To update the row count in the Administration Tool, do one of the following:
  - In the Physical layer, select **Tools**, then select **Update All Row Counts**.
  - In the Physical layer, right-click a single table, column, or select multiple objects, right-click and select **Update Row Count**.

  If the repository is open in online mode, and the Check Out Objects window opens, click **Yes** to check out the objects

  Any row counts that have changed since the last update are refreshed.

Display Row Counts in the Physical Layer

Use these steps to display the row count in the Physical layer.

1. In the Administration Tool, select **Tools**, then select **Options**.
2. In the Options dialog, on the General tab, select **Show row count in physical view**, and click **OK**.

9

Work with Logical Tables, Joins, and Columns

This chapter explains how to work with objects in the Business Model and Mapping layer of the Oracle BI repository, such as logical tables, joins, and columns. It also explains other Business Model and Mapping layer concepts like display folders, bridge tables, the Business Model Diagram, and how to enable write back on columns.
This chapter contains the following sections:

- About Working with the Business Model and Mapping Layer
- Create the Business Model and Mapping Layer
- About Working with the Business Model Diagram
- Create and Manage Logical Tables
- Define Logical Joins
- Create and Manage Logical Columns
- Enable Write Back On Columns
- Set Up Display Folders in the Business Model and Mapping Layer
- Model Bridge Tables
- Model Binary Large Object (BLOB) Data and Character Large Object (CLOB) Data

About Working with the Business Model and Mapping Layer

The Business Model and Mapping layer of the Oracle BI repository defines the business, or logical, model of the data and specifies the mapping between the business model and the Physical layer schemas.

Business models are always dimensional, unlike objects in the Physical layer, which reflect the organization of the data sources. The Business Model and Mapping layer can contain one or more business models. Each business model contains logical tables, columns, and joins.

Even though similar terminology is used for logical table and physical table objects, such as the concept of keys, logical tables and joins in the Business Model and Mapping layer have their own set of rules that differ from those of relational models. For example, logical fact tables aren't required to have keys, and logical joins can represent many possible physical joins.

Logical tables, joins, mappings, and other objects in the Business Model and Mapping layer are typically created automatically when you drag and drop objects from the Physical layer to a particular business model. After these objects have been created, you can perform tasks like creating additional logical joins, performing calculations and transformations on columns, and adding and removing keys from dimension and fact tables.

Create the Business Model and Mapping Layer

After creating all of the elements of the Physical layer, you can drag tables or columns from the Physical layer to a business model in the Business Model and Mapping layer to create logical objects in the metadata.

This section contains the following topics:

- Create Business Models
- Automatically Create Business Model Objects
- Automatically Create Business Model Objects for Multidimensional Data Sources
- Duplicate a Business Model and Subject Area

Create Business Models

The Business Model and Mapping layer of the Administration Tool can contain one or more business models.

A business model contains the business model definitions and the mappings from logical to physical tables for the business model.

When you work in a repository in offline mode, remember to save your repository from time to time. You can save a repository in offline mode even though the business models may be inconsistent.

1. In the Administration Tool, right-click in the Business Model and Mapping layer below existing objects.
2. Select the option **New Business Model** from the shortcut menu.
3. Specify a name for the business model.
4. New business models are disabled by default. If you want to make the corresponding Presentation layer available for queries, deselect **Disabled**.

   The business model should be consistent before you deselect this option.
5. (Optional) Type a description of the business model.
6. Click **OK**.

After you create a business model, you can create business model objects by dragging and dropping objects from the Physical layer.

Automatically Create Business Model Objects

To automatically map objects in the Business Model and Mapping layer to sources in the Physical layer, you can drag and drop Physical layer objects to a particular business model in the logical layer.

When you drag a physical table to the Business Model and Mapping layer, a corresponding logical table is created. For each physical column in the table, a corresponding logical column is created. If you drag multiple tables at once, a logical join is created for each physical join, but only the first time the tables are dragged onto a new business model.

Automatically Create Business Model Objects for Multidimensional Data Sources

Setting up objects in the Business Model and Mapping layer for multidimensional data sources is similar to setting up logical layer objects for a relational data source.

When creating the business model layer, you can drag and drop the Physical layer cube to the logical layer. A fully configured and consistent business model is created automatically that retains all metrics, attributes and dimensions.

For Essbase data sources, Oracle recommends creating a separate business model for each Essbase cube by dragging each cube individually to the Business Model and Mapping layer.

Duplicate a Business Model and Subject Area

Learn how to make copies of a selected business model and assign new names to the copy.

You can select a business model and its corresponding subject area, make a copy, and assign new names to the duplicate objects. Aliases aren't copied.

1. In the Administration Tool with the repository open, right-click a business model, and select **Duplicate with Subject Area**.
2. In the Copy Business Model and Subject Area dialog, select the business model and corresponding subject area you want to copy.
3. Specify new names for the business model and subject area in the appropriate name fields, and then click **OK**.

The copied business model appears in the Business Model and Mapping layer, and the copied subject area appears in the Presentation layer.

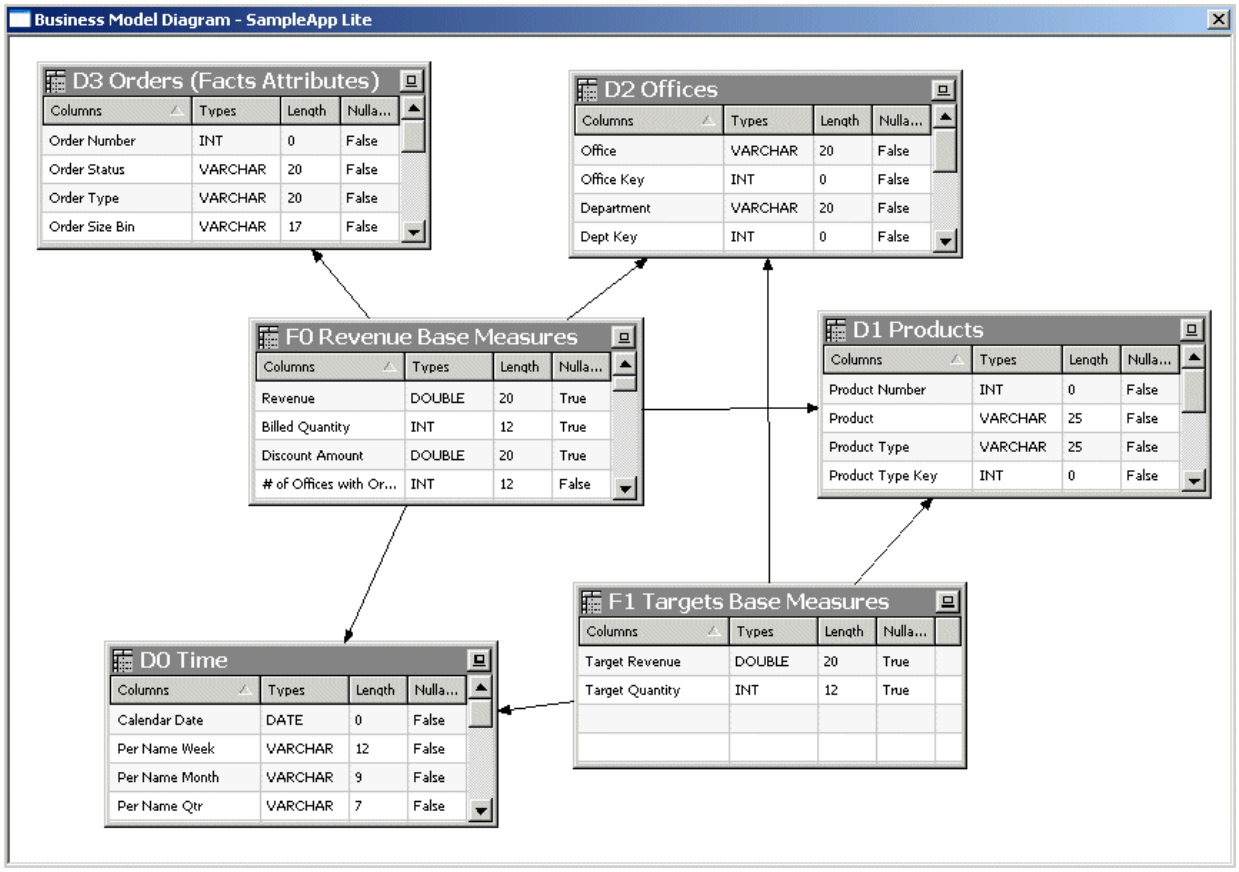
About Working with the Business Model Diagram

Open the Business Model Diagram to see a graphical model of logical tables and joins.

The Business Model Diagram displays only logical tables and joins. It doesn't display other Business Model and Mapping layer objects, such as business models, dimensions, or hierarchies. Joins are represented by a line with an arrow at the *one* end of the join.

To access the Business Model Diagram, right-click an object in the Business Model and Mapping layer such as a dimension or fact table, and select **Business Model Diagram**. Then, select one of the following options:

- **Whole Diagram**. Displays all logical tables and joins in the business model.
- **Selected Tables Only**. Displays only the selected logical tables. Logical joins appear only if they exist between the objects that you selected. This option is only available when you select one or more logical tables.
- **Selected Tables and Direct Joins**. Displays the selected logical tables and any logical tables that join to the tables that you selected. This option is only available when you select one or more logical tables.
- **Selected Fact Tables and Dimensions**. Displays the selected logical tables and their associated logical dimensions. This option is only available when your selection includes at least one fact table.



To add more tables to the Business Model Diagram, leave the Business Model Diagram window open and then right-click the table or tables you want to add. Then, select **Business Model Diagram** and choose one of the display options.

Other options are available in the right-click menu for the graphical tables and joins displayed in the Business Model Diagram. For example, you can delete objects or view their properties, or you can include additional related objects using the right-click options **Add Direct Joins**, **Add Tables Joined to Whole Selection**, and **Add All Joins**. You can also select **Find in Tree View** to locate a particular object in the Business Model and Mapping layer view in the middle pane, or check out objects in online mode.

You can also right-click an object in the **Business Model Diagram** view and select **Hide** to hide particular objects in the diagram. The hide effect is temporary and doesn't persist.

Use the **Print** and **Print Preview** options on the **File** menu to manage printing options for the Business Model Diagram. You can also use the **Print** option on the toolbar.

See Define Logical Joins with the Business Model Diagram to learn about defining logical joins.

Create and Manage Logical Tables

Logical tables exist in the Business Model and Mapping layer.

The logical schema defined in each business model must contain at least two logical tables, and you must define relationships between them.

Each logical table is associated with one or more logical columns and one or more logical table sources. You can add a new logical table source, edit or delete an existing table source, create or change mappings to the table source, or define when to use logical tables sources. See Create Logical Table Sources.

You can change the logical table name, reorder the logical table sources, and configure the logical keys, both primary and foreign

This section contains the following topics:

- Create Logical Tables
- Specify a Primary Key in a Logical Table
- Review Foreign Keys for a Logical Table

Create Logical Tables

Dragging and dropping physical tables from the Physical layer to the Business Model and Mapping layer is the recommended method for creating logical tables. If a table doesn't exist in your physical schema, you can create the logical table manually.

If you drag and drop physical tables from the Physical layer to the Business Model and Mapping layer, the columns in the table are also to the logical table along with key and foreign key relationships. Logical keys and joins are created that mirror the keys and joins in the Physical layer.

After creating a logical table using the menu option method, you must create all keys and joins manually.

After adding objects to the Business Model and Mapping layer, you can modify the objects in the logical table without affecting the objects in the Physical layer.

If you create new tables or drag additional tables from the Physical layer to the Business Model and Mapping layer, you must create the logical mappings between the new or newly dragged tables and the previously dragged tables.

See Define Logical Joins with the Joins Manager and Define Logical Joins with the Business Model Diagram.

A lookup table stores multilingual data corresponding to rows in the base tables. See Localize Oracle Analytics Server in *Administering Oracle Analytics Server*.

1. In the Administration Tool, to create a logical table, do one of the following:
   - (Recommended method) Select one or more table objects in the Physical layer, then drag and drop the table objects to a business model in the Business Model and Mapping layer.
   - (Manual method) In the Business Model and Mapping layer, right-click the business model, select **New Object** , and then select **Logical Table**.
2. For manually created tables, right-click the table, and select **Properties**.
3. For manually created tables, in the Logical Table General tab, in **Name**, type a name for the logical table.
4. Optional: Select **Lookup table** when you intend to use the table as a lookup table.

5. Optional: In **Description**, type an explanation of the table's use.
6. Click **OK**.

Enable Data Driven Fragment Selection in Logical Table Sources

You can improve the performance of fragmented logical table sources by enabling the data driven fragment selection option in the

Data driven fragment selection is disabled by default.

1. In the Administration Tool, from the Business Model and Mapping column, right-click a model that uses fragmented logical table sources and select **Query Related Objects** , and then select **Logical Table Source**.
2. In Query Related Objects, select a logical table source, and click **Edit**.
3. In Logical Table Sources, in the Content tab, click **Enable Data Driven Fragment Selection**, and click **OK**.

Specify a Primary Key in a Logical Table

After creating tables in the Business Model and Mapping layer, you specify a primary key for each dimension table.

Logical dimension tables must have a logical primary key. Logical keys can be composed of one or more logical columns.

> **Note:**
>
> Oracle recommends that you don't specify logical keys for logical fact tables.

1. In the Business Model and Mapping layer of the Administration Tool, double-click a table.
2. In the Logical Table dialog, select the Keys tab and then click **New**.
3. In the Logical Key dialog, type a name for the key and select the column that defines the key of the logical table.
4. Click **OK**.

Review Foreign Keys for a Logical Table

Oracle recommends that you don't use foreign key joins in logical tables.

You must enable the **Allow logical foreign key join creation** option in the Options dialog to create joins with foreign keys.

See Create Logical Foreign Key Joins with the Joins Manager.

Define Logical Joins

Relationships between logical tables are expressed by logical joins.

Logical joins are conceptual, rather than physical, joins. Logical joins don't join to specific keys or columns. A single logical join can correspond to many possible physical joins.

A key property of a logical join is cardinality. Cardinality expresses how rows in one table are related to rows in the table to which it's joined. A one-to-many cardinality means that for every row in the first logical dimension table, there are 0, 1, or many rows in the second logical table. The Administration Tool considers a table to be a logical fact table if it's at the Many end of all logical joins that connect it to other logical tables.

Specifying the logical table joins is required so that the Oracle BI Server can have the necessary metadata to translate a logical request against the business model to SQL queries against the physical data sources. The logical join information provides the Oracle BI Server with the many-to-one relationships between the logical tables. This logical join information is used when the Oracle BI Server generates queries against the underlying databases.

You don't need to create logical joins in the Business Model and Mapping layer if both of the following statements are true:

- You create the logical tables by simultaneously dragging and dropping all required physical tables to the Business Model and Mapping layer.
- The logical joins are the same as the joins in the Physical layer.

You might need to create some logical joins in the Business Model and Mapping layer, because you can't drag and drop all physical tables simultaneously except in very simple models.

You can create logical joins using either the Joins Manager or the Business Model Diagram. When you create a complex join in the Physical layer, you can specify expressions and the specific columns on which to create the join. When you create a logical join in the Business Model and Mapping layer, you can't specify expressions or columns on which to create the join. The existence of a join in the Physical layer doesn't require a matching join in the Business Model and Mapping layer.

It's recommended that you don't have foreign keys for logical tables. However, for backward compatibility, you can create logical foreign key joins using the Joins Manager if you select **Allow logical foreign key join creation** in the Options dialog. Compose a logical key for a fact table using the key columns that join to the attribute tables. You might need logical foreign key joins if the Oracle BI Server is used as an ODBC data source for certain third-party query and reporting tools.

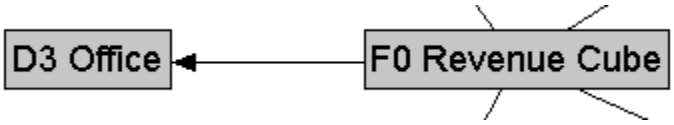
This section contains the following topics:

- Define Logical Joins with the Business Model Diagram
- Define Logical Joins with the Joins Manager
- Specify a Driving Table
- Factors That Determine Join Trimming
- Identify Physical Tables That Map to Logical Objects

Define Logical Joins with the Business Model Diagram

The Business Model Diagram shows logical tables and any defined joins between them.

In the Business Model Diagram, the join is represented by a line between the two selected tables, with an arrow at the *one* end of the join. The image shows a join in the Business Model Diagram.



You can use the Business Model Diagram to define logical joins between tables. See Specify a Driving Table.

This driving table option is useful for optimizing the manner in which the Oracle BI Server processes multi-database inner joins when one table is very small and the other table is very large. Don't select a driving table unless multi-database joins are going to occur.

> **Important:**
>
> Use extreme caution in deciding whether to specify a driving table. Driving tables are used for query optimization only under rare circumstances and when the driving table is extremely small (fewer than 1000 rows). Choosing a driving table incorrectly can lead to severe performance degradation.

1. In the Administration Tool, right-click a business model and select **Business Model Diagram**, then select **Whole Diagram**.
2. From the Diagram menu, click the **New Join** button on the Administration Tool toolbar.
3. In the Business Model Diagram, click to select the first table in the join, the table representing many in the one-to-many join.
4. Move the cursor to the table to use for the join, the table representing one in the one-to-many join, and then click the second table to select it.
5. Optional: To specify a driving table for the key, select a table from the **Driving table** list, and an applicable cardinality.
6. Select the join type from the **Type** list, or keep the default value.
7. Set the **Cardinality** for each side of the join, or keep the default values.
8. Click **OK**.

Define Logical Joins with the Joins Manager

You can use the Joins Manager to view logical join relationships and to create logical joins.

You can also use the Joins Manager to create logical foreign key joins if you select **Allow logical foreign key join creation** in the Options dialog, although this isn't recommended.

This section contains the following topics:

- Create Logical Joins with the Joins Manager
- Create Logical Foreign Key Joins with the Joins Manager

Create Logical Joins with the Joins Manager

Logical joins are recommended over logical foreign key joins in the Business Model and Mapping layer.

Use the driving option for optimizing the manner in which the Oracle BI Server processes multi-database inner joins when one table is very small and the other table is very large. Don't select a driving table unless multi-database joins are going to occur.

> **Note:**
>
> Use extreme caution in deciding whether to specify a driving table. Driving tables are used for query optimization only under rare circumstances and when the driving table is extremely small, that is, less than 1000 rows. Choosing a driving table incorrectly can lead to severe performance degradation.

See Specify a Driving Table.

1. In the Administration Tool, select **Manage**, then select **Joins**.
2. In the Joins Manager, select **Action**, select **New** , and then select **Logical Join**.
3. In the Logical Join dialog, type a name for the logical join.
4. In the **Table** lists on the left and right side of the dialog, select the tables that the logical join references.
5. Optional: To specify a driving table for the key, select a table from the **Driving** list, and an applicable cardinality.
6. Select the join type from the **Type** list, or keep the default value.
7. Set the **Cardinality** for each side of the join, or keep the default values.
8. Click **OK**.

Create Logical Foreign Key Joins with the Joins Manager

You might need logical foreign key joins if you plan to use the Oracle BI Server as an ODBC data source for certain third-party query and reporting tools.

You shouldn't create logical foreign keys. See Specify a Driving Table.

The driving table option is useful for optimizing the manner in which the Oracle BI Server processes multi-database inner joins when one table is very small and the other table is very large. Don't select a driving table unless multi-database joins are going to occur.

> **Important:**
>
> Use extreme caution in deciding whether to specify a driving table. Driving tables are used for query optimization only under rare circumstances and when the driving table is extremely small, that is, less than 1000 rows. Choosing a driving table incorrectly can lead to severe performance degradation.

1. In the Administration Tool, select **Tools**, then select **Options**.
2. In the General tab of the Options dialog, select **Allow logical foreign key join creation**.
3. Click **OK**.
4. Select **Manage**, then select **Joins** to display the Joins Manager.
5. Select **Action**, select **New**, and then select **Logical Foreign Key**.
6. In the Browse dialog, double-click a table.
7. In the Logical Foreign Key dialog, type a name for the foreign key.
8. In the **Table** list on the left side of the dialog, select the table that the foreign key references.
9. Select the columns in the left table that the foreign key references.
10. Select the columns in the right table that make up the foreign key columns.
11. (Optional) To specify a driving table for the key, select a table from the **Driving** list, and an applicable cardinality.
12. Select the join type from the **Type** list, or keep the default value.
13. Set the **Cardinality** for each side of the join, or keep the default values.
14. In Expression Builder, type an expression for the join.
15. Click **OK** to save your work.

Specify a Driving Table

Driving tables are useful for optimizing how the Oracle BI Server processes cross-database joins when one table is very small and the other table is very large.

Specifying driving tables leads to query optimization only when the number of rows being selected from the driving table is much smaller than the number of rows in the table to which it's being joined.

> **Important:**
>
> To avoid problems, only specify driving tables when the driving table is extremely small - less than 1000 rows.

You can specify a driving table for logical joins from the Logical Joins window. When you specify a driving table, the Oracle BI Server uses the driving table if the query plan determines that the table's use can optimize query processing. The small table (the driving table) is scanned, and parameterized queries are issued to the large table to select matching rows. The other tables, including other driving tables, are then joined together.

> **Important:**
>
> If large numbers of rows are being selected from the driving table, specifying a driving table could lead to significant performance degradation or, if the `MAX_QUERIES_PER_DRIVE_JOIN` limit is exceeded, the query terminates.

Use driving tables with inner joins, and for outer joins when the driving table is the left table for a left outer join, or the right table for a right outer join. Driving tables aren't used for full outer joins. See Define Logical Joins for instructions on specifying a driving table.

There are two entries in the database features table that control and tune driving table performance.

- `MAX_PARAMETERS_PER_DRIVE_JOIN`

  This is a performance tuning parameter. The larger its value, the fewer parameterized queries are generated. Values that are too large can result in parameterized queries that fail due to back-end database limitations. Setting the value to 0 (zero) turns off drive table joins.

- `MAX_QUERIES_PER_DRIVE_JOIN`

  This is used to prevent runaway drive table joins. If the number of parameterized queries exceeds its value, the query is terminated and an error message is returned to the user.

Factors That Determine Join Trimming

When determining which joins Oracle BI Server can trim from a physical query, the Oracle BI Server considers the factors described in this section.

The following join trimming rules are enforced for tables within a logical table source:

- Join Outerness (Inner, Left Outer, Right Outer, or Full Outer).
- Join Cardinality, {0...1, 1, N, Unknown} to {0...1, 1, N, Unknown}; for example, 0...1 to N represents a zero or one-to-many join. There are nine join cardinality combinations excluding those with Unknown cardinality on at least one side of the join.
- Whether the logical table source contains a WHERE clause filter.
- Whether the physical join is a complex join or a foreign key join.

For the Oracle BI Server to trim a join, meeting the following criteria is required.

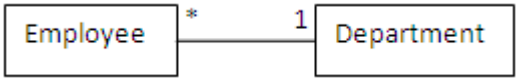
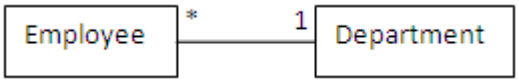
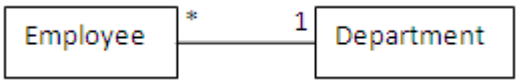
- No references to the trimmed table can exists anywhere in the query such as in the projected list of columns or in the WHERE clause.
- The trimmed table mustn't cause the cardinality of the result set to change. If removing a join could potentially change the number of rows selected, then the Oracle BI Server doesn't trim it.

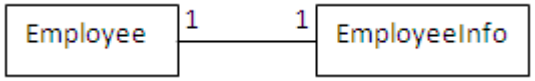
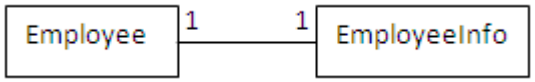
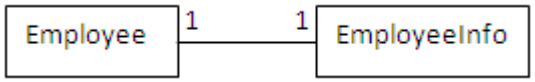
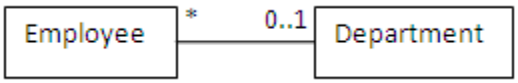
  A join is considered to have the potential to change the number of rows in the result set if any of the following conditions are true. If any of these conditions are true, then the join isn't trimmed from the query:

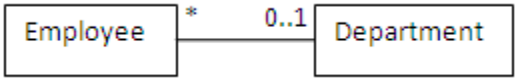
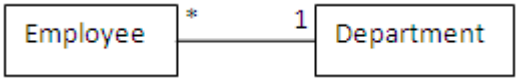
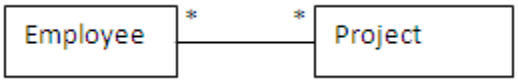
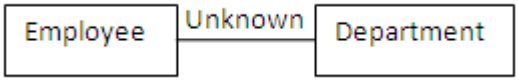
  - The join is a full outer join, only inner joins, left outer joins, and right outer joins are candidates for trimming
  - The join cardinality is unknown on either side

- The table to trim is on the many side of a join, in other words, the detail table is never trimmed in a master-detail relationship
- The table to trim has a 0..1 cardinality and the join is an inner join. 0..1 cardinality implies that a possible matching row in the table. A join with 0..1 cardinality on one side is effectively like a filter. Oracle BI Server can't trim the table without changing the number of rows selected.
- The table to trim is on the left side of a left outer join or on the right side of a right outer join, the row-preserving table is never trimmed. There is an exception to this rule for queries that select only attributes in which a DISTINCT clause is added to the query. Because of the DISTINCT clause, trimming the row-preserving table doesn't affect the number of rows returned from the null-supplying table. In the special case of distinct queries on attributes, you can trim the row-preserving table from an outer join.

The following table provides examples of when the Oracle BI Server can trim joins from the query.

Scenario	Result
 Employee INNER JOIN Department	Oracle BI Server can trim Department because it's on the one side of an inner join. Oracle BI Server can't trim Employee because it's on the many side of an inner join.
 Employee LEFT OUTER JOIN Department	Oracle BI Server can trim Department because it's on the one side of the join and it's on the right side of a LEFT OUTER JOIN, the null supplying table. Oracle BI Server can't trim Employee because it's on the many side, and because it's on the left side of a LEFT OUTER JOIN, the row preserving table.
 Employee RIGHT OUTER JOIN Department	Oracle BI Server can't trim Department because it's on the right side of a RIGHT OUTER JOIN, the row preserving table. Oracle BI Server can't trim Employee because it's on the many side of the join.

Scenario	Result
 Employee INNER JOIN EmployeeInfo	Oracle BI Server can trim either side because both tables are on the one side of an inner join.
 Employee LEFT OUTER JOIN EmployeeInfo	Oracle BI Server can trim EmployeeInfo since it's on the one side of the join, and it's on the right side of a LEFT OUTER JOIN, the null supplying table. Oracle BI Server can't trim Employee because it's on the left side of a LEFT OUTER JOIN, the row preserving table.
 Employee RIGHT OUTER JOIN EmployeeInfo	Oracle BI Server can trim EmployeeInfo because it's on the right side of a RIGHT OUTER JOIN, the row preserving table. You can trim Employee because it's on the "one" side of the join, and it's on the left side of a RIGHT OUTER JOIN, the null supplying table.
 Employee INNER JOIN Department	Oracle BI Server can trim Department because it's on the 0..1 side of an inner join. Oracle BI Server can trim Employee because it's on the many side of an inner join.

Scenario	Result
 Employee LEFT OUTER JOIN Department	Oracle BI Server can trim Department because it's on the 0..1 side of an outer join, and it's on the right side of a LEFT OUTER JOIN, the null supplying table. The Oracle BI Server allows trimming the null supplying table on the 0..1 side of an outer join, because in this case, trimming Department from the query wouldn't change the number of rows selected from the Employee table. Oracle BI Server can trim Employee since it's on the many side of an outer join.
 Employee FULL OUTER JOIN Department	Oracle BI Server can't trim either side because the join is a FULL OUTER JOIN.
 Employee MANY TO MANY Project	Oracle BI Server can't trim either side because the join is many-to-many.
 Employee UNKNOWN Department	Oracle BI Server can't trim either side because the join has unknown cardinality.

Identify Physical Tables That Map to Logical Objects

The Physical Diagram shows the physical tables that map to the selected logical object and the physical joins between each table.

One of the joins options, **Object(s) and Direct Joins within Business Model**, is unique to the logical layer. It creates a physical diagram of the tables that meet both of the following conditions:

- Tables in the selected objects and tables that join directly
- Tables that are mapped, exist in logical table sources in the business model, in the business model

1. In the Administration Tool Business Model and Mapping layer, right-click a business model, logical table, or logical table source.
2. Select **Physical Diagram** and then one of the joins options.
3. Click and drag any object to more clearly view the relationship lines such as one-to-many.

Create and Manage Logical Columns

Many logical columns are automatically created by dragging tables from the Physical layer to the Business Model and Mapping layer.

Other logical columns, especially ones that involve calculations based on other logical columns, can be created later.

Logical columns are displayed in a tree structure expanded out from the logical table to which they belong. If the column is a primary key column or participates in a primary key, the column is displayed with a key icon. If the column has an aggregation rule, it's displayed with a ruler icon. You can also reorder logical columns in the Business Model and Mapping layer.

This section contains the following topics:

- Create Logical Columns
- Base the Sort for a Logical Column on a Different Column
- Enable Double Column Support by Assigning a Descriptor ID Column
- Create Derived Columns
- Set Default Levels of Aggregation for Measure Columns
- Associate an Attribute with a Logical Level in Dimension Tables
- Move or Copy Logical Columns

Create Logical Columns

Use this procedure to create logical columns in the Business Model and Mapping layer.

1. In the Business Model and Mapping layer, right-click a logical table.
2. From the shortcut menu, select **New Object**, then select **Logical Column**.
3. In the General tab, type a name for the logical column.

   The name of the business model and the associated logical table appear in the **Belongs to Table** field.
4. Select **Writeable** to enable write back for this column.
5. Click **OK**.

Base the Sort for a Logical Column on a Different Column

For a logical column, you can specify a different column on which to base the sort order.

Change the sort order of a column when you don't want to order the values alphabetically (lexical order).

In a lexical order sort, numbers are ordered by their alphabetic spelling and not divided into a separate group.

For example, if you sorted on month (using a column such as `MONTH_NAME`), the results return February, January, March in their lexicographical sort order. You might want sort months in chronological order. Your table needs to have a month key such as `MONTH_KEY`with values of 1 (January), 2 (February), 3 (March) to achieve the chronological sort order. You set the Sort order column field for the `MONTH_NAME` column to the `MONTH_KEY` and then, a request to order by `MONTH_NAME` would return January, February, and March.

The sort column is automatically defined for Essbase data sources when business models are created by dragging and dropping cubes from the Physical layer.

1. In the Logical Column dialog, in the General tab, click **Set** next to the **Sort order column** field.
2. In the Browse dialog, select a column.
3. To view the column details, click **View** to open the Logical Column dialog for that column, and then click **Cancel**.

   You can make some changes in this dialog. If you make changes, click **OK** to accept the changes instead of **Cancel**.
4. In the Browse dialog, click **OK**.

Enable Double Column Support by Assigning a Descriptor ID Column

When multilingual columns are based on a lookup function, it's common to specify the non-translated lookup key column as the descriptor ID column of the translated column.

Assigning a descriptor ID column to a logical column enables double column support. You can use double column support to defining language-independent filters. For example, in Answers, users see the display column, but the query filters on the hidden descriptor ID column.

See Support Multilingual Data.

Double column support provides a mechanism for associating two columns. One column provides the display and description values such as the description of an item. The second column provides a descriptor ID or code column. For example, you can use the actual column to provide the project list, and hide the ID column associated with the first column, as in Clinic and Clinic ID. Only the Clinic description is displayed to the user. Using the double column approach helps satisfy the uniqueness requirements of Essbase. In the Clinic example, you would add an association to a column that contains the clinic ID using the steps in the procedure.

1. Open the repository in the Oracle BI Administration Tool.

2. In the Business Model and Mapping layer, expand the business model, and expand the table contain the column to update.
3. Right-click the column, select **Query Related Objects**, select **Business Model and Mapping**, and then select **Logical Column**.
4. In the **Logical Column(s) related to** *the selected column* dialog, click the column to associate with the selected column, and click **Edit**.
5. In the Logical Column dialog, next to the **Descriptor ID column** field, click **Set**
6. In the Browse dialog, select a column to use as the Descriptor ID, and click **OK**.

Create Derived Columns

Some columns are derived from other logical columns as a way to apply post-aggregation calculations to measures.

You specify the derived column expression in the Column Source tab of the Logical Column dialog.

You can also create a set of derived columns using the Calculation Wizard. See Use the Calculation Wizard.

If the parameter `PREVENT_DIVIDE_BY_ZERO` is set to `YES` in `NQSConfig.INI`, the Oracle BI Server prevents errors in divide-by-zero situations, even for Answers column calculations. The Oracle BI Server creates a divide-by-zero prevention expression using `nullif()` or a similar function when it writes the physical SQL. Because of this, you don't have to use `CASE` statements to avoid divide-by-zero errors, as long as `PREVENT_DIVIDE_BY_ZERO` is set to `YES` (the default value).

You can also apply calculations pre-aggregation. See Define Physical to Logical Table Source Mappings and Creating Calculated Items.

To optimize performance, don't define aggregations in Expression Builder. Instead, use the Aggregation tab of the Logical Column dialog. See Set Default Levels of Aggregation for Measure Columns.

1. In the Logical Column dialog, select the Column Source tab.
2. Select the option **Derived from existing columns using an expression**.
3. Click the **Expression Builder** button to open Expression Builder.
4. In the Expression Builder - Derived logical column dialog, specify the expression from which the logical column should be derived.
5. Click **OK**.

You can display data from multilingual database schemas by using Expression Builder to create a lookup function. See Support Multilingual Data in *Administering Oracle Analytics Server*.

Configure Logical Columns for Multicurrency Support

You can configure logical columns so that users can select the currency in which they prefer to view currency columns in analyses and dashboards.

You can set up this feature so that all users see the same static list of currency options, or you can provide a dynamic list of currency options that changes based on a Logical SQL statement you specify.

When you use session variables in an expression for Presentation Services, you must preface their names with `NQ_SESSION`. Edit any logical columns that display currency values to use the appropriate conversion factor using the `PREFERRED_CURRENCY` session variable.

See Create Session Variables and Create Initialization Blocks.

The following logical column expression uses the value of the `NQ_SESSION.PREFERRED_CURRENCY` variable to switch between different currency columns. The currency columns are expected to have the appropriate converted values.

```
INDEXCOL( CASE VALUEOF(NQ_SESSION.PREFERRED_CURRENCY) WHEN 'gc1' THEN 0
WHEN 'gc2' THEN 1 WHEN 'orgc' THEN 2 WHEN 'lc1' THEN 3 ELSE 4 END,
"Paint"."Sales Facts"."USDCurrency",
"Paint"."Sales Facts"."DEMCurrency" ,
"Paint"."Sales Facts"."EuroCurrency" ,
"Paint"."Sales Facts"."JapCurrency" ,
"Paint"."Sales Facts"."USDCurrency" )
```

1. Create a session variable named *PREFERRED_CURRENCY*, along with an initialization block to use in the variable.

   Select **Enable any user to set the value** when you create the session variable.

2. In the Business Model and Mapping layer, double-click the appropriate logical column, select the **Column Source** tab, and create a derived expression that uses the *PREFERRED_CURRENCY* variable.

3. Optional: To provide a dynamic list of currency options, create a table in your data source that provides the entries you want to display for the user-preferred currency. This table must include the following columns:

   - The first column contains the values used to set the session variable `PREFERRED_CURRENCY`. Each value in this column is a string that uniquely identifies the currency (for example, gc2).
   - The second column contains currency tags from the *currencies.xml* file. The `displayMessage` values for each tag are used to populate the Currency box and currency prompts, for example, `int:euro-1`. The *currencies.xml* file is located in *`ORACLE_HOME`*`\bi\bifoundation\web\display`.
   - You can provide a third column that contains the values used to set the presentation variable `currency.userPreference`. Each value in this column is a string that identifies the currency, for example, Global Currency 2. If you omit this column, then the values for the `displayMessage` attributes for the corresponding currency tags in the `currencies.xml` file are used.

   Sample user-preferred currency entries:

   - UserPreference: orgc1, CurrencyTag: loc:en-BZ, UserPreferenceName: Org currency
   - UserPreference: gc2, CurrencyTag: int:euro-1, UserPreferenceName: Global currency 2
   - UserPreference: lc1, CurrencyTag: int:DEM, UserPreferenceName: Ledger currency
   - UserPreference: gc1, CurrencyTag: int:USD, UserPreferenceName: Global Currency 1

Additional configuration is required in Presentation Services to enable this feature. For full information about the Oracle BI Presentation Services configuration, see Define User-Preferred Currency Options in *Administering Oracle Analytics Server*.

Set Default Levels of Aggregation for Measure Columns

You need to specify aggregation rules for mapped logical columns that are measures.

Only perform aggregation on measure columns, with the possible exception of the aggregation `COUNT` and `COUNTDISTINCT`. Measure columns should exist only in logical fact tables.

You can select different aggregation rules for different dimensions that are associated with this logical column. For example, if someone queries the aggregate column along with one dimension, you may want to use one type of aggregation rule, whereas with another dimension, you may want to use a different aggregation rule.

When the default aggregation rule is Count Distinct, you can specify an override aggregation expression for specific logical table sources. For example, you may want to specify override aggregation expressions when you're querying different aggregate table sources that already contain some level of aggregation. If you don't specify any override, then the default rule prevails.

You can choose the **EVALUATE_AGGR** aggregation rule to enable queries to call custom functions in the data source. See Define Aggregation Rules for Multidimensional Data Sources.

By default, data is considered sparse. However, you might have a logical table source with dense data. A logical table source is considered to have dense data if it has a row for every combination of its associated dimension levels. When setting up aggregate rules for a measure column, you can specify that data is dense only if all the logical table sources to which it's mapped are dense.

See Set Up Dimension-Specific Aggregate Rules for Logical Columns.

For measures in which the additivity is the same in all dimensions, select one of the aggregate functions from the **Default Aggregation Rule** list. The function you select is always applied when a user or an application requests the column in a query, unless an override aggregation expression has been specified. When you select **Count Distinct** as the default aggregation rule, you can specify an override aggregation expression for specific logical table sources. Choose this option when you've more than one logical table source mapped to a logical column and you want to apply a different aggregation rule to each source.

1. In the Business Model and Mapping layer, double-click a logical column.
2. In the Logical Column dialog, click the Aggregation tab.
3. In the Aggregation tab, choose one of the following options:
   - Select one of the aggregate functions from the **Default Aggregation Rule** list.
     a. Click the **Add** button to select logical table sources for which you want to specify individual aggregation rules.
     b. In the Browse dialog, select the logical table source you want to add, and click **OK**.
     c. In the **Formula** list for that logical table source, select the aggregation rule you want to use.

- Select **Based on dimensions** if your measure has different additivity for different dimensions, for semi-additive measures.
  a. Click the **Add** button to select additional dimensions for which you want to specify aggregation rules.
  b. In the Browse dialog, select the dimension you want to add, and then click **OK**.
  c. In the **Formula** list for that dimension, select the aggregation rule you want to use, or click the **Expression Builder** button to build the aggregation rule using Expression Builder.
  d. The **Data is dense** option appears when you select **Based on dimensions**. Select this option only if all the logical table sources to which this column is mapped are dense.

     Selecting **Data is dense** indicates that all sources to which this column is mapped have a row for every combination of dimension levels that they represent. Selecting this option when any table source that's used by this column doesn't contain dense data returns incorrect results.

4. Click **OK**.

Set Up Dimension-Specific Aggregate Rules for Logical Columns

The majority of measures have the same aggregation rule for each dimension. Some measures can have different aggregation rules for different dimensions.

For example, bank could calculate account balances averages over a specific time, but calculated averages on individual accounts with a simple summation for a period. You can configure dimension-specific aggregation rules. You can specify one aggregation rule for a given dimension and specify other rules to apply to other dimensions.

You need to configure dimensions in the Business Model and Mapping layer to set up dimension-specific aggregation. See Manage Logical Table Sources (Mappings).

After selecting rules for specified dimensions, set the aggregation rule for any remaining dimensions by using the dimension labeled Other.

When calculating the measure, aggregation rules are applied in the order (top to bottom) established in the dialog. If you've multiple dimensions, use **Up** or **Down** to change the order in which the dimension-specific rules are performed.

1. In the Business Model and Mapping layer, double-click a logical column.
2. In the Logical Column dialog, click the Aggregation tab.
3. In the Aggregation tab, select **Based on dimensions**.
4. In the Browse dialog, select a dimension over which you want to aggregate, and then click **OK**.
5. In the Aggregation tab, from the **Formula** list, select a rule.
6. Optional: If you need to create more complex formulas, click the **Expression Builder** button to open Expression Builder.
7. Click **OK**.

Specify Dimension-Specific Aggregation Rules for Multiple Logical Columns

You can specify aggregation rules for multiple logical fact columns using the steps in this task.

When calculating the measure, aggregation rules are applied in the order (top to bottom) established in the dialog.

Select a minimum of two columns to enable the Set Aggregation menu item. Set Aggregation isn't enabled if one or more of the columns are derived columns.

1. In the Business Model and Mapping layer, select multiple logical fact columns.
2. Right-click and select **Set Aggregation**.
3. In the Aggregation dialog, select **All columns the same** or select **Clear** and select specific columns.
4. In the Aggregation tab, select **Based on dimensions**.
5. In the Browse dialog, select a dimension over which you want to perform aggregation, and then click **OK**.
6. After setting up the rule for a dimension, specify aggregation rules for any other dimensions in the entry labeled Other.
7. Click the **Expression Builder** button to the right of the **Formula** column.
8. In the Expression Builder - Aggregate dialog, from the **Formula** list, select the aggregation to perform over the dimension.
9. To change the order in which the dimension-specific rules are performed, click **Up** or **Down**, and then click **OK**.

Define Aggregation Rules for Multidimensional Data Sources

Learn the best practices for defining aggregation rules for logical measures sourced from Essbase, Oracle OLAP, and other multidimensional data sources, like Microsoft Analysis Services.

When you import Essbase and some other multidimensional cubes into the Physical layer, Oracle BI Server can't read the aggregation rules set within the data source. As a result of the default behavior, the measures are imported automatically with the default aggregation rule of **External Aggregation**.

External Aggregation means that the Oracle BI Server:

- isn't aware of the underlying aggregation rule for the specific measure.
- can't compute the measure.
- always ships the query to the underlying multidimensional data source for aggregation.

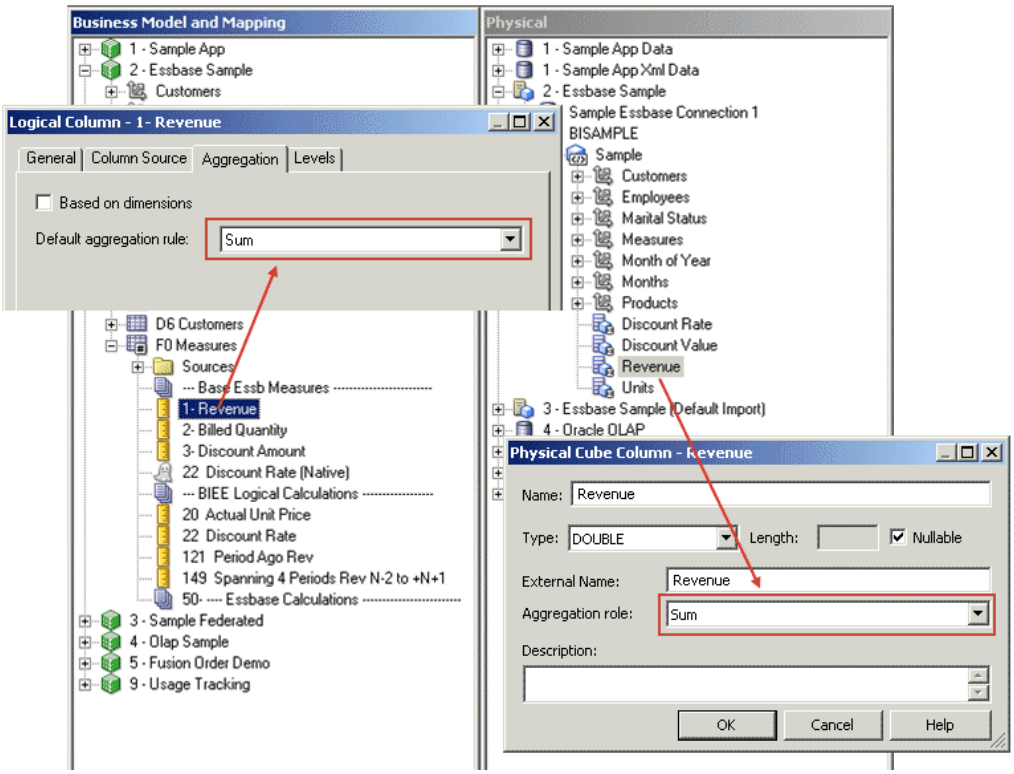
Because the underlying data sources are extremely efficient, pushing the aggregation rules down to the data source ensures that the Oracle BI Server returns the results without adding any additional overhead in processing. Oracle recommends updating the aggregation rule for each measure, both in the Physical layer and Business Model and Mapping layer, with the corresponding aggregation rule defined in the data source. Updating the aggregation rule for each measure ensures that the Oracle BI Server can do additional computations when needed. There is no query performance impact, since the Oracle BI Server still pushes down optimized queries wherever possible.

If the Oracle BI Server needs to do additional aggregation for a particular query, and the aggregation rule is set to the default of External Aggregation, the server returns the following error:

```
An external aggregate is found in an outer query block.
```

This error occurs because the Oracle BI Server can't read the aggregation rule in the underlying data source. To ensure that correct results are returned for these queries, you should change the aggregation rules set in the Oracle BI Repository to match the aggregation rules set in the underlying data source.

You must ensure that the aggregation rule defined matches the rule in the underlying data source. Also, you must set the appropriate aggregation rule in both the Physical layer and Business Model and Mapping layer, as shown in the following image.



For custom aggregations or aggregations which don't have a corresponding function within the Oracle BI Server, it's recommended to leave the aggregation as External Aggregation for both the physical measure column and its corresponding logical measure column.

For Oracle OLAP data sources, you don't explicitly set Physical layer aggregation rules for Oracle OLAP columns. Because of this, you only need to set the aggregation rule for Oracle OLAP columns in the Business Model and Mapping layer.

In addition, if a query requests an aggregate that doesn't exist in the Oracle OLAP data source, and the aggregation rule is set to External Aggregation, then the Oracle BI Server returns an error. To avoid this error, make sure to explicitly set the aggregation rule for the Oracle OLAP column in the Business Model and Mapping layer.

If you don't explicitly set the aggregation rule for Oracle OLAP columns to something other than External Aggregation, requests from Oracle BI Presentation Services custom groups fail, because custom groups always request aggregates that don't exist in the data source.

Associate an Attribute with a Logical Level in Dimension Tables

You can associate attributes with a logical level.

You can associate measures with levels from multiple dimensions and aggregate to the levels specified. A measure is associated to a level is called a level-based measure. A level-based measure is computed at that grain, even when the query context has a lower grain. For example, if yearlySales is associated to year level, it's computed at the yearly level in the following query: `Select month, yearlySales`.

Dimensions appear in the Dimensions list. If this attribute is associated with a logical level, the level appears in the Levels list.

Another way to associate a measure with a level in a dimension is to expand the dimension tree in the Business Model and Mapping layer, and then use drag-and-drop to drop the column on the target level. See Level-Based Measure Calculations.

1. In the Business Model and Mapping layer of the Administration Tool, double-click a logical column to associate a measure with a logical level in a dimension.
2. In the Logical Column dialog, click the Levels tab.
3. In the Levels tab, click the row containing the logical dimension to associate with a logical level.
4. From the **Logical Level** list, select the level.
5. Click **OK**.

Move or Copy Logical Columns

Dragging and dropping a logical column from one table to another moves the logical column.

If a column with the same name already exists, the new column is renamed, for example, *mycolumn#1*.

You can also choose the option **Prompt when moving logical columns** in the Options dialog to cause the Sources for moved columns dialog to be displayed when you drag and drop a logical column. This dialog gives you options about the drag and drop behavior.

See Set Administration Tool Options to read about the **Prompt when moving logical columns** option.

After completing this procedure, the column that you move or copy is associated with the logical source. The action list values are as follows:

- If you select **Ignore**, no logical source is added in the Sources folder of the destination table.
- If you select **Create new**, a copy of the logical source associated with the logical column is created in the Sources folder of the destination table.
- If you select **Use existing**, you must select a logical source from the Sources folder of the destination table.

1. In the Business Model and Mapping layer, drag and drop a logical column to a different logical table.

2. In the Sources for moved columns dialog, select from the **Action** list.

Enable Write Back On Columns

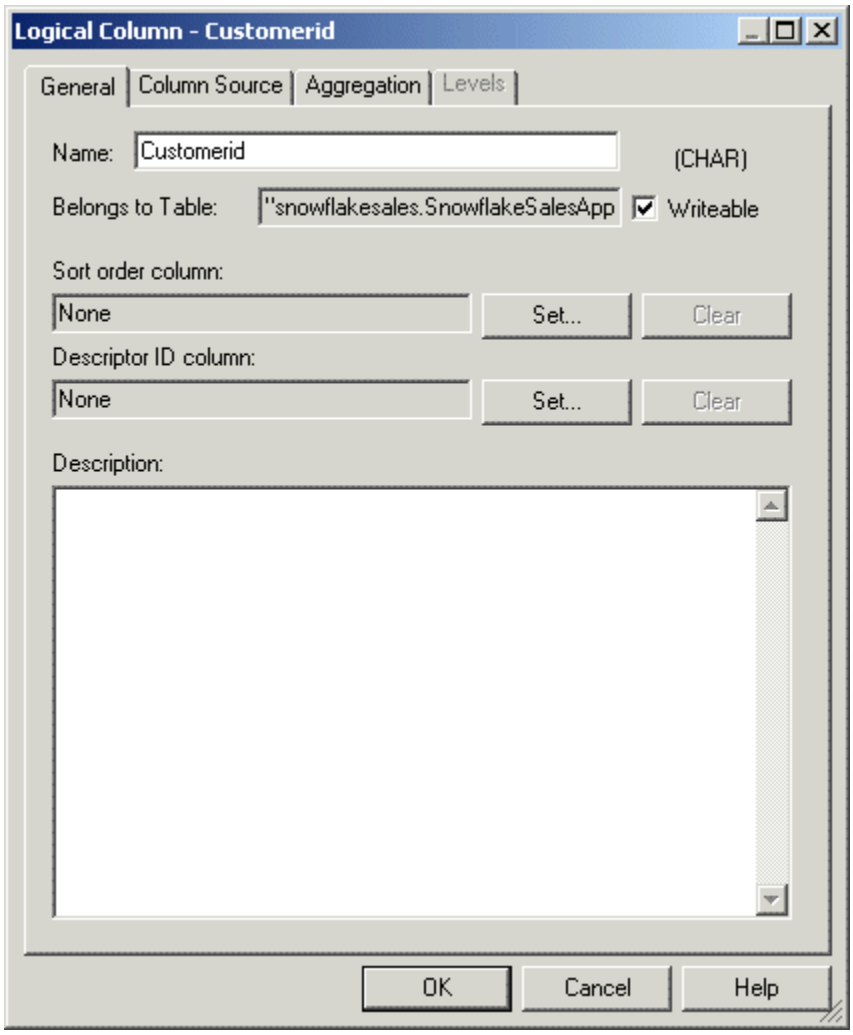
You can configure individual logical columns so that users in Oracle BI Presentation Services can update column data and write the changes back to the data source.

To enable write back on a particular column, you must select the **Writeable** option for the logical column, and enable the **Read/Write** permission for the corresponding presentation column. You must also disable caching on the corresponding physical table.

You must perform additional tasks to enable write back in Oracle BI Presentation Services. See Configuring for Write Back in Analyses and Dashboards in *Administering Oracle Analytics Server* for full information.

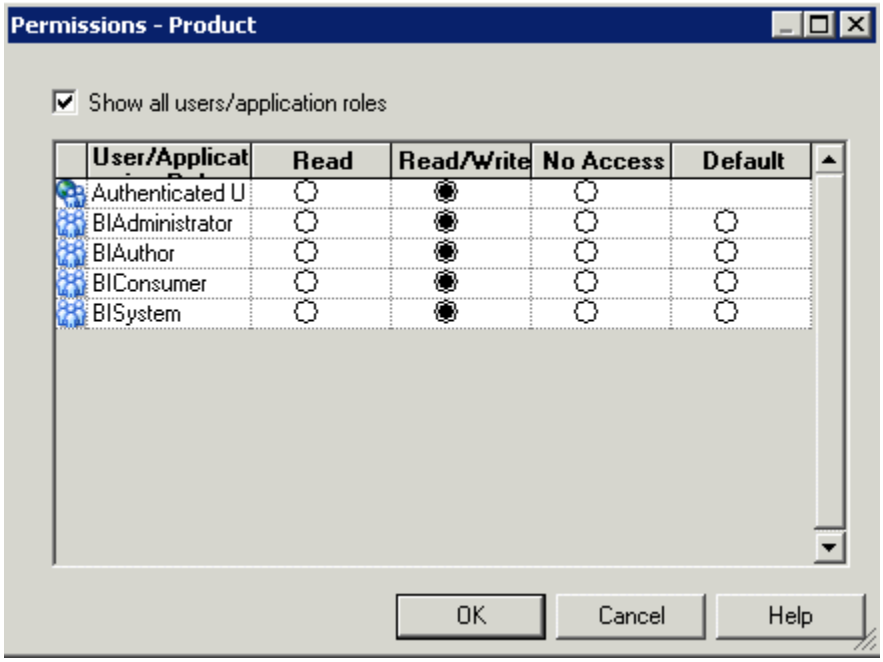
1. In the Administration Tool, in the Physical layer, double-click the physical table that contains the column for which you want to enable write back.
2. On the General tab of the Physical Table dialog, ensure that **Cacheable** isn't selected. Deselecting this option ensures that Oracle BI Presentation Services users can see updates immediately.
3. In the Business Model and Mapping layer, double-click the corresponding logical column.

   The image shows the Logical Column dialog.



4. In the Logical Column dialog, select **Writeable**, then click **OK**.
5. In the Presentation layer, double-click the column that corresponds to the logical column for which you enabled write back. The Presentation Column dialog opens.
6. Click **Permissions**.
7. Select the **Read/Write** permission for the appropriate users and application roles.

   The image shows the Permissions dialog with the Read/Write permission selected..



8. Click **OK** in the Permissions dialog.
9. Click **OK** in the Presentation Column dialog.

Set Up Display Folders in the Business Model and Mapping Layer

You can create display folders to organize objects in the Business Model and Mapping layer. Display folders have no effect on query processing.

After you create a display folder, the selected tables and dimensions appear in the folder as a shortcut and in the business model tree as the object. You can hide the objects so that you only view the shortcuts in the display folder. See the information about the Repository tab of the Options dialog in Set Administration Tool Options about hiding these objects.

Deleting a table in a display folder deletes only the shortcut to that object. When you delete a column in a display folder, however, the column is actually deleted.

1. In the Business Model and Mapping layer of the Administration Tool, right-click a business model and select **New Object**, then select **Logical Display Folder**.
2. In the Logical Display Folder dialog, in the Tables tab, type a name for the folder.
3. To add tables to the display folder, click **Add**. In the Browse dialog, select the fact or dimension tables you want to add to the folder and click **Select**.

   Alternatively, you can drag one or more logical tables to the display folder after you close the dialog.
4. To add dimensions to the display folder, click the Dimensions tab and click **Add**. In the Browse dialog, select the dimensions that you want to add to the folder and click **Select**.

   Alternatively, you can drag one or more dimensions to the display folder after you close the dialog.

5. Click **OK**.

Model Bridge Tables

A bridge table enables you to resolve many-to-many relationships between tables.

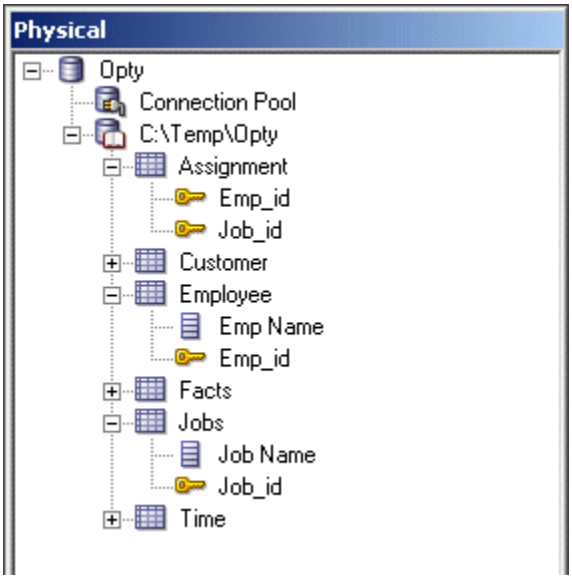
For example, you might hold information about employees in an Employees table, and information about the jobs they do in a Jobs table. However, an organization's employees can have multiple jobs, and the same job can be performed by multiple employees. This situation would result in a many-to-many relationship between the Employees table and the Jobs table.

To resolve the many-to-many relationship, you can create a bridge table or intermediate table called Assignments. Each row in the Assignments table is unique, representing one employee doing one job. If an employee has several jobs, there are several rows in the Assignments table for that employee. If a job is done by several employees, there are several rows in the Assignments table for that job. The primary key of the Assignments table is a composite key, made up of a column containing the employee ID and a column containing the job ID.

By acting as a bridge table between the Job and Employee tables, the Assignments table enables you to resolve the many-to-many relationship between Employees and Jobs into:

- A one-to-many relationship between Employees and Assignments
- A one-to-many relationship between Assignments and Jobs

The image shows a Physical layer view of the example bridge and associated dimension tables.



You should include Weight Factor as an additional column in the bridge table, and to calculating during ETL for efficient query processing.

The following sections explain how to model bridge tables in the Physical and Business Model and Mapping layers:

- Create Joins in the Physical Layer for Bridge and Associated Dimension Tables
- Model the Associated Dimension Tables in Separate Dimensions
- Model the Associated Dimension Tables in a Single Dimension

Create Joins in the Physical Layer for Bridge and Associated Dimension Tables

To model bridge tables in the Physical layer, create joins between the bridge table and the associated dimension tables.

1. In the Administration Tool, in the Physical layer, select the fact, bridge, and associated dimension tables.
2. Right-click the objects, select **Physical Diagram**, and then choose **Selected Object(s) Only**.
3. Click **New Join**, select the bridge table, and then select one of the dimension tables.
4. Click **OK** in the Physical Foreign Key dialog.
5. Repeat steps 2 and 3 for the other associated dimension table.
6. Ensure that one of the associated dimension tables is joined to the fact table.

Model the Associated Dimension Tables in a Single Dimension

In the Business Model and Mapping layer, you can choose to model the two dimension tables associated with a bridge table in a single dimension, or in two separate dimensions.

To model the associated dimension tables in one dimension, create a second logical table source that maps to the bridge table and the other dimension table, and then add columns from the other dimension table. Don't drag the bridge table and the associated dimension table that isn't joined to the fact table. For the example described in the previous sections, you'd drag all objects except for the Assignment and Employee tables.

Providing two separate logical table sources makes queries more efficient, because it ensures that queries against a single dimension table don't involve the bridge table.

It's a good practice to use the bridge table name as the name of the source.

1. Drag objects from the Physical layer to the Business Model and Mapping layer.
2. In the Business Model and Mapping layer, right-click the dimension table that's joined to the fact table, and select **New Object**, then select **Logical Table Source**.
3. In the Logical Table Source dialog, provide a name for the new bridge table source.
4. Click the **Add** button in the upper right corner of the Logical Table Source dialog.
5. Select the bridge table from the **Name** list, and then click **Select**.
6. Click **Add**, select the associated dimension table that isn't joined to the fact table, and then click **Select**.
7. In the Logical Table Source dialog, click **OK**.

8. Drag columns from the dimension table that isn't joined to the fact table, Employees in this example, from the Physical layer to the logical table source that you just created.

You can create dimensions based on your logical tables, including the logical table with the bridge table source.

Model the Associated Dimension Tables in Separate Dimensions

As an alternative to modeling the two dimension tables associated with a bridge table in a single dimension, you can choose to model them in separate dimensions.

Create a logical join between the fact table and the dimension table that isn't physically joined to the fact table, and then modify the logical table source for that same dimension table to add the other table mappings.

1. Drag objects from the Physical layer to the Business Model and Mapping layer.

   Because you want to model the dimension tables in separate dimensions, drag both of the dimension tables associated with the bridge table. You don't need to drag and drop the bridge table object.
2. In the Business Model and Mapping layer, select the fact table and the two dimension tables that are associated with the bridge table.
3. Right-click the objects and select **Business Model Diagram**, and then choose **Selected Tables Only**.
4. With the Business Model Diagram displayed, click **New Join** on the toolbar.
5. Select the fact table, and then select the dimension table not currently joined to the fact table.
6. Click **OK** in the Logical Join dialog.
7. Double-click the logical table source for the logical table for which you created the logical join.
8. In the Logical Table Source dialog, click **Add**.
9. Select the bridge table from the **Name** list, and click **Select**.
10. Click the **Add** button again and select the other associated dimension table and then click **Select**.
11. In the Logical Table Source dialog, click **OK**.

You can create dimensions based on your logical tables, including both logical tables associated with the bridge table.

Model Binary Large Object (BLOB) Data and Character Large Object (CLOB) Data

Learn how to model binary large object (BLOB) data and character large object (CLOB) data in the Oracle BI repository.

CLOB data is a large plain text document in any character set. The supported BLOB image types are: GIF, PNG, TIFF, JPEG, and BMP. BLOB formats not supported are: PDF, audio, or video.

The default data type for BLOB columns after the import is LongVarBinary, while for CLOB columns it's LongVarChar. The column for the BLOB or CLOB can't exceed the the Oracle BI Server `MaxFieldSize` limit of 32 KB.

When configuring the physical joins create a physical join between the tables using the primary key when the primary key is used as a foreign key in the other table.

1. Import the physical table containing the BLOB or CLOB data from the data source using the Import Metadata Wizard.
2. After import, open the Physical Column dialog for the BLOB or CLOB column, and change the **Length** field.
3. Configure physical joins.
4. Drag the BLOB or CLOB column to the Business Model and Mapping layer to generate a logical column.
5. Configure a physical lookup for the logical column to ensure that the Oracle BI Server doesn't generate a group by or order by on the logical column.
6. In the Logical Column dialog on the General tab, configure the **Descriptor ID column** to ensure that Presentation Services uses the correct column when generating filters.
7. Configure the **Sort order column**, configure the sort order column to ensure that the Oracle BI Server orders column as expected.
8. Save the changes.

10

Work with Logical Dimensions

This chapter explains how to work with logical dimension objects in the Business Model and Mapping layer of the Oracle BI repository.
This chapter contains the following topics:

- About Working with Logical Dimensions
- Create and Manage Dimensions with Level-Based Hierarchies
- Create and Manage Dimensions with Parent-Child Hierarchies
- Model Time Series Data

About Working with Logical Dimensions

In the Business Model and Mapping layer, a dimension object represents a hierarchical organization of logical columns (attributes).

You can associate one or more logical dimension tables with one dimension object.

Common dimensions include time periods, products, markets, customers, suppliers, promotion conditions, raw materials, manufacturing plants, transportation methods, media types, and time of day. Dimensions exist in the Business Model and Mapping (logical) layer and in the Presentation layer.

In each dimension, you organize logical columns into the structure of the hierarchy. The structure represents the organization rules and reporting needs required by your business and provides the metadata the Oracle BI Server uses to drill into and across dimensions to get detailed views of the data.

There are two types of logical dimensions:

- Dimensions with level-based hierarchies (structure hierarchies)

  In level-based hierarchies, members are of several types, and members of the same type occur only at a single level.

- Dimensions with parent-child hierarchies (value hierarchies)

  In parent-child hierarchies, members all have the same type.

Oracle Analytics Server also supports a special type of level-based dimension, called a time dimension, that provides special functionality for modeling time series data.

Because dimensions for multidimensional data sources are defined in the source, you don't create dimension level keys. A dimension is specific to a particular multidimensional data source. You can't create and manipulate a dimension individually. Each cube in the data source should have at least one dimension and one measure in the Business Model and Mapping layer.

You can expose logical dimensions to Answers users by creating presentation hierarchy objects that are based on particular logical dimensions. Creating hierarchies in the Presentation layer enables users to create hierarchy-based queries, see Work with Presentation Hierarchies and Levels.

You can also expose dimension hierarchies by adding one or more columns from each hierarchy level to a subject area in the Presentation layer. Answers supports drill-down on these hierarchical columns.

Create and Manage Dimensions with Level-Based Hierarchies

Each business model can have one or more dimensions, each dimension can have one or more logical levels, and each logical level has one or more attributes (columns) associated with it.

The following sections explain how to create dimensions:

- About Level-Based Hierarchies
- Manually Create Dimensions, Levels, and Keys with Level-Based Hierarchies
- Rules for Automatically Created Dimensions with Level-Based Hierarchies
- Automatically Create Dimensions with Level-Based Hierarchies
- Populate Logical Level Counts Automatically

About Level-Based Hierarchies

A dimension contains two or more logical levels. When creating logical levels, you should create a Grand Total level and then create child levels, working down to the lowest level.

The following are the parts of a dimension:

- Grand Total level

  The Grand Total level represents the sum of all totals for a dimension. Each dimension can have just one Grand Total level. The Grand Total level doesn't contain dimensional attributes and doesn't have a level key. You can associate measures with a Grand Total level. The aggregation level for those measures is the grand total for the dimension. The Grand Total level can exist without any columns.

- Level

  Levels must have at least one column. You don't need to explicitly associate all of the columns from a table with logical levels. Any column that you don't associate with a logical level is automatically associated with the lowest level in the dimension that corresponds to that dimension table. You must associate all logical columns in the same dimension table with the same dimension.

  You can have an unlimited number of levels in a dimension. When using extremely complex SQL queries, a few levels in a dimension can impact query performance.

- Hierarchy

  Each dimension contains one or more hierarchies. All hierarchies must have a common leaf level. For example, a time dimension might contain a fiscal hierarchy and a calendar hierarchy, with a common leaf level of Day. In this example, Day has two named parent levels, Fiscal Year and Calendar Year that are both children of the All root level.

In the Business Model and Mapping layer, logical hierarchies aren't defined as independent metadata objects, unlike hierarchies in the Presentation layer. Logical hierarchies exist implicitly through the relationships between levels.

You can define intermediate levels in your hierarchies to avoid having very large numbers of members at one level. For example, if you're creating a Product dimension for an automotive company that tracks data on 500 different car models, you might want to create some finer-grained hierarchical levels such as Minivans, Subcompacts, and Midsize Sedans. You could improve query performance and make reports and diagrams easier to read and navigate.

- Level keys

  Each logical level, except the Grand Total level, must have one or more attributes that compose a level key. The level key defines the unique elements in each logical level. You must associate the dimension table logical key with the lowest level of a dimension.

  A logical level can have multiple level keys. When a logical level has multiple level keys, specify a key as the primary key for the level. All dimension sources that have aggregate content at a specified level need to contain the column that's the primary key of that level. Each logical level should have one level key that's displayed when an Oracle BI Server user selects the object to drill down. You can use any level key to provide user access to the level.

  You must create an unique level key. Month isn't an unique level key. To create an unique level key with month include the year attribute as part of the key.

  If you don't ensure that your level key is unique by including higher-level attributes, then queries might return unexpected results. For example, when the Oracle BI Server needs to combine result sets from multiple physical queries, some expected rows might be dropped because they aren't considered unique according to the level key definition.

  Create meaningful level keys using common business keys such as *Month_name='2010 July'*, rather than generated surrogate keys such as *time_key='1023793'*. The generated surrogate keys are physical artifacts that only apply to a single instance of a source table. A business key can map to any physical instance for that logical column, for example, *month_name* might map to a detailed table, an aggregate table from an aggregate star, or a column in a federated spreadsheet. The Physical layer uses the surrogate keys in the joins. Using a business key doesn't impose a performance or flexibility penalty in the business model.

- Time dimensions and chronological keys

  You can identify a dimension as a time dimension. At least one level of a time dimension must have a chronological key. Use the following guidelines when setting up and using time dimensions:

  – At least one level of a time dimension must have a chronological key, see Select and Sort Chronological Keys in a Time Dimension.

  – All time series measures using the `AGO`, `TODATE`, and `PERIODROLLING` functions must be on time levels. `AGO`, `TODATE`, and `PERIODROLLING` aggregates are created as derived logical columns.

  – `AGO`, `TODATE`, and `PERIODROLLING` functionality isn't supported either on fragmented dimensional logical table sources, or on fact sources fragmented on the same time dimension. Fact sources may be fragmented on other dimensions.

    See About Time Series Functions.

- **Unbalanced or ragged hierarchy**

An unbalanced or ragged hierarchy is a hierarchy where the leaves (members with no children) might not have the same depth. For example, a site can choose to have data for the current month at the day level, previous months data at the month level, and the previous 5 years data at the quarter level.

User applications can use the ISLEAF function to determine whether to allow moving down from any particular member.

A missing member is implemented in the data source with a null value for the member value. All computations treat the null value as a unique child within its parent. Level-based measures and aggregate-by calculations group all missing nodes together.

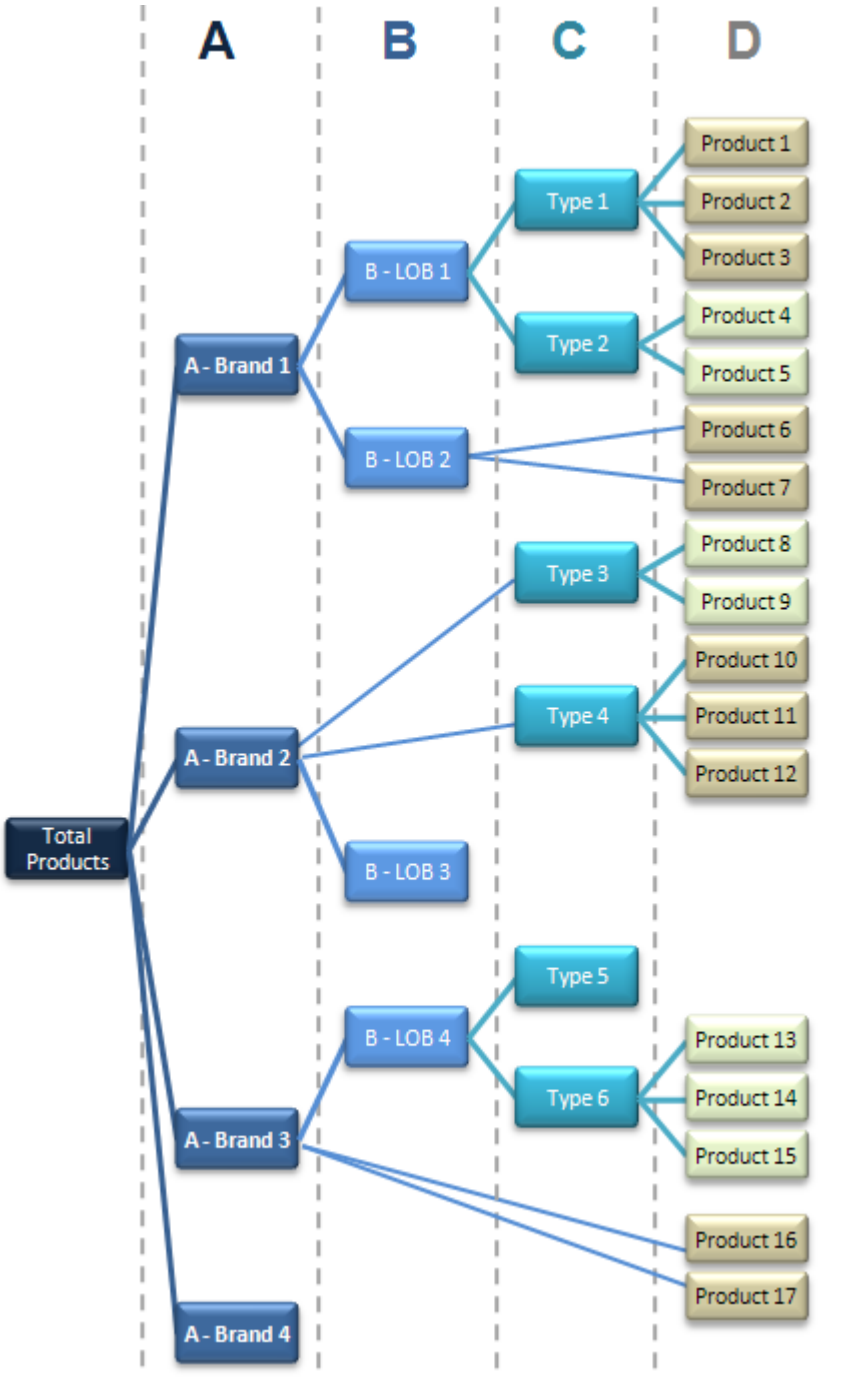
Unbalanced hierarchies aren't necessarily the same as parent-child hierarchies. Parent-child hierarchies are unbalanced by nature. Unbalanced level-based hierarchies are possible.

- **Skip-level hierarchy**
  A skip-level hierarchy is a hierarchy where there are members that don't have a value for a particular ancestor level. For example, in a Country-State-City-District hierarchy, the city *Washington, D.C.* doesn't belong to a State. In this case, you can drill down from the Country level (USA) to the City level (Washington, D.C.) and below.

  In a query, skipped levels aren't displayed, and don't affect computations. When sorted hierarchically, members appear under their nearest ancestors.

  A missing member at a particular level is implemented in the data source with a null value for the member value. All computations treat the null value as a unique child within its parent. Level-based measures and aggregate-by calculations group all skip-level nodes together.

The image shows a hierarchy with both unbalanced and skip-level characteristics. For example, A-Brand 4, B-LOB 3, and Type 5 are unbalanced branches, while skips are present between A-Brand 2 and Type 3, B-LOB 2 and Product 6, and others.



About Using Dimension Hierarchy Levels in Level-Based Hierarchies

Learn how to use dimension hierarchical levels.

Dimension hierarchical levels can be used to perform the following actions:

- Set up aggregate navigation.
- Configure level-based measure calculations, see Level-Based Measure Calculations.

- Determine what attributes appear when Oracle BI Presentation Services users drill down in their data requests.

Manually Create Dimensions, Levels, and Keys with Level-Based Hierarchies

Learn how to create and manage hierarchy level in level-based hierarchies.

Perform the tasks described in the following sections:

- Create Dimensions in Level-Based Hierarchies
- Create Logical Levels in a Dimension
- Associate a Logical Column and Its Table with a Dimension Level
- Identify the Primary Key for a Dimension Level
- Select and Sort Chronological Keys in a Time Dimension
- Add a Dimension Level to the Preferred Drill Path
- Add Sequence Numbers to a Time Dimension's Logical Level

Create Dimensions in Level-Based Hierarchies

After creating a dimension, each dimension can be associated with attributes (columns) from one or more logical dimension tables and level-based measures from logical fact tables.

After you associate logical columns with a dimension level, the tables in which these columns exist appear in the Tables tab of the Dimension dialog. See Work with Physical Hierarchy Objects.

It's a best practice to ensure that the physical hierarchy type set in the Physical layer matches the dimension properties you select in the Business Model and Mapping layer. In addition, you must ensure that the Ragged and Skipped Levels dimension properties are set correctly for queries to work.

1. In the Business Model and Mapping layer of the Administration Tool, right-click a business model, select **New Object**, select **Logical Dimension**, and then select **Dimension with Level-Based Hierarchy**.

   This option is only available when there is at least one dimension table that has no dimension associated with it.
2. In the Logical Dimension dialog, in the General tab, type a name for the dimension.

   The **Default root level** field is automatically populated after you associate logical columns with a dimension level.
3. If the dimension is a time dimension, select **Time**.
4. If the dimension is an unbalanced dimension, select **Ragged**.
5. If the dimension is a skip-level dimension, select **Skipped Levels**.
6. Optional: Type a description of the dimension.
7. Click **OK**.

Create Logical Levels in a Dimension

When creating logical levels in a dimension, you also create the hierarchy by identifying the type of level and defining child levels.

See Automatically Create Business Model Objects.

If you're defining the level as a `Grand Total level`, leave this field blank. The default value is 1.

The number doesn't have to be exact, but ratios of numbers from one logical level to another should be accurate. For relational sources, you can retrieve the row count for the level key and use that number as the number of elements. For multidimensional sources, you can use the number of members at that level.

The Oracle BI Server uses this number when selecting which aggregate source to use. For example, when aggregate navigation is used, multiple fact sources exist at different grains. The Oracle BI Server multiplies the number of elements at each level for each qualified source as a way to estimate the total number of rows for that source. Then, the Oracle BI Server compares the result for each source and selects the source with the lowest number of total elements to answer the query. The source with the lowest number of total elements is assumed to be the fastest.

1. In the Business Model and Mapping layer of the Administration Tool, right-click a dimension and select **New Object**, then select **Logical Level**.
2. In the Logical Level dialog, in the General tab, specify a name for the logical level.
3. For **Number of elements at this level**, specify the number of elements that exist at this logical level.
4. Choose one of the following options, if appropriate:
   - If the logical level is the Grand Total level, select **Grand total level**. There should be only one Grand Total level for a dimension.
   - If measure values at a particular level fully constitute aggregated measures at its parent level, select **Supports rollup to higher level of aggregation**
5. Click **Add** to define child logical levels.
6. In the Browse dialog, select the child logical levels and click **OK**.
7. In the Child Level pane, remove a previously defined child level, select the level in the Child Levels pane and click **Remove**.
8. Optional: Type a description of the logical level.
9. Click **OK**.

Associate a Logical Column and Its Table with a Dimension Level

After you create all logical levels within a dimension, drag and drop one or more columns from the dimension table to each logical level, except the Grand Total level.

The first time you drag a column to a dimension it associates the logical table to the dimension. The drag and drop action associates the logical column with that level of the dimension. To associate the logical level with that logical column, drag a column from one logical level to another.

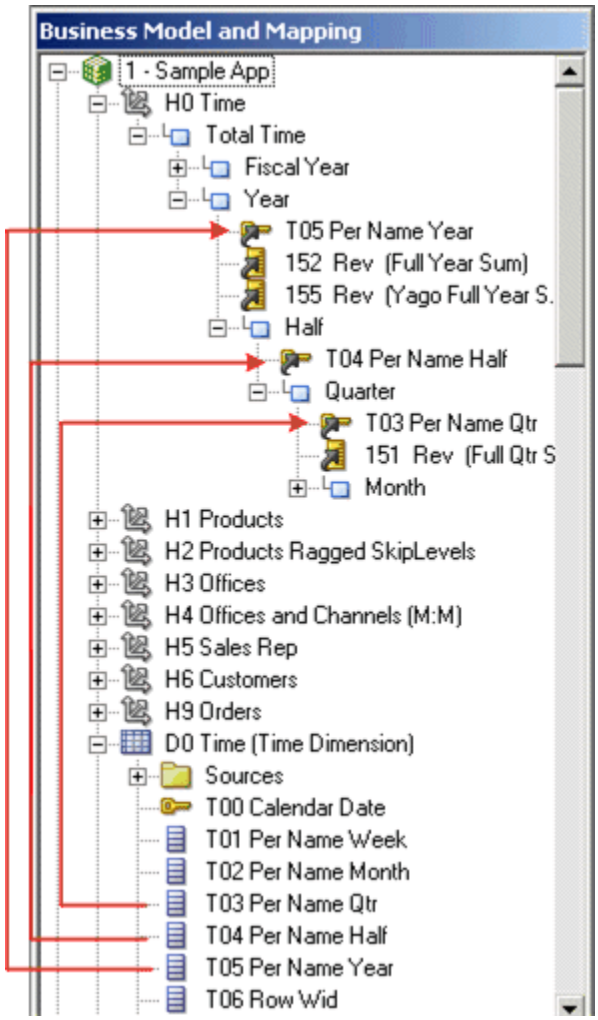
You must associate the logical column or columns that comprise the logical key of a dimension table with the lowest level of the dimension.

After you associate a logical column with a dimension level, the tables in which these columns exist appear in the Tables tab of the Dimensions dialog.

For examples, see:

- Level-Based Measure Calculations
- Grand Total Dimensional Hierarchy

For time dimensions, ensure that all time-related logical columns in the source table are defined in the time dimension. For example, if a time-related logical table contains the columns Month Name and Month Code, you must ensure that both columns are dragged to the appropriate level within the dimension. The image shows how to associate logical columns with a logical level.



1. In the Business Model and Mapping layer of the Administration Tool, double-click a dimension to verify tables that are associated with a dimension.
2. In the Dimensions dialog, click the **Tables** tab.

The Tables tab list contains tables that you associated with that dimension. If you created level-based measures, the list only includes one logical dimension table and one or more logical fact tables.

3. Click **OK** or **Cancel** to close the Dimensions dialog.

Level-Based Measure Calculations

A level-based measure is a column whose values are always calculated to a specific level of aggregation.

You can set up columns to measure CountryRevenue, RegionRevenue, and CityRevenue. For example, a company might want to measure its revenue based on the country, region, and city.

When a query containing a presentation hierarchy includes a level-based measure column, and the query grain is higher than the level of aggregation specific to the column, the query results return null. If the request only contains ordinary columns and no hierarchical columns, the level-based measure isn't replaced with null.

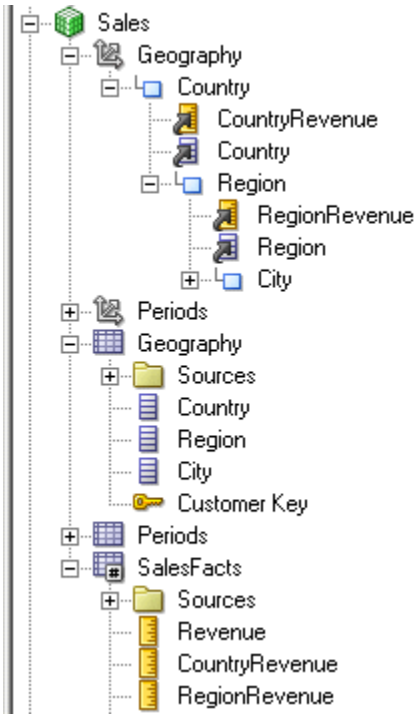
The AllProductRevenue measure is an example of a level-based measure at the Grand Total level. Level-based measures allow a single query to return data at multiple levels of aggregation. Level-based measures are also useful in creating share measures, calculated by taking some measure and dividing it by a level-based measure to calculate a percentage. For example, you can divide salesperson revenue by regional revenue to calculate the share of the regional revenue each salesperson generates.

For example, to set up these calculations, you need to build a dimensional hierarchy in your repository that contains the Grandtotal, Country, Region, and City levels. This hierarchy contains the metadata that defines a one-to-many relationship between Country and Region and a one-to-many relationship between Region and City. For each country, there are many regions, but each region is in only one country. Similarly, for each region, there are many cities, but each city is in only one region.

After building a dimensional hierarchy, you need to create three logical columns one each for CountryRevenue, RegionRevenue, and CityRevenue. The columns use the Revenue logical column as its source. The Revenue column has a default aggregation rule of `SUM` and has sources in the underlying databases.

Drag the CountryRevenue, RegionRevenue, and CityRevenue columns into the Country, Region, and City levels, respectively. Each query that requests one of these columns returns the revenue aggregated to its associated level.

The image shows the business model in the Business Model and Mapping layer for this example.



Grand Total Dimensional Hierarchy

Learn how to use a grand total dimensional hierarchy with revenue.

If you've a product dimensional hierarchy with TotalProducts (Grand Total level), Brands, and Products levels, and a Revenue column defined with a default aggregation rule of Sum, you can then create an AllProductRevenue logical column. The AllProductRevenue column uses Revenue as its source. Drag the AllProductRevenue column to the Grand Total level. Each query that includes the AllProductRevenue column returns the total revenue for all products. The value is returned regardless of any constraints on Brands or Products.

If you've constraints on columns in other tables, the grand total is limited to the scope of the query. For example, if the scope of the query asks for data from 1999 and 2000, the grand total product revenue is for all products sold in 1999 and 2000.

If you've three products, A, B, and C with total revenues of 100, 200, and 300 respectively, then the grand total product revenue is 600, the sum of each product's revenue. If you've set up a repository as described in this example, the following query produces the results listed:

```
SELECT product, productrevenue, allproductrevenue
FROM sales_subject_area
WHERE product IN 'A' or 'B'
```

The results are as follows:

```
PRODUCT;;PRODUCTREVENUE;;ALLPRODUCTREVENUE
A;;;;;;;;100;;;;;;;;;;;;;600
B;;;;;;;;200;;;;;;;;;;;;;600
```

The AllProductRevenue column always returns a value of 600, regardless of the products on which the query constrains.

Identify the Primary Key for a Dimension Level

Use the Keys tab in the Logical Level dialog to identify the primary key for a level.

1. In the Business Model and Mapping layer of the Administration Tool, expand a dimension and then expand the highest level (Grand Total level) of the dimension.
2. Double-click a logical level below the Grand Total level.
3. In the Logical Level dialog, click the Keys tab.
4. In the Keys tab, from the **Primary key** list, select a level key.

   If only one level key exists, it's the primary key by default.
5. To add a column to the list, perform the following steps:
   a. In the Logical Level dialog, click **New**.
   b. In the Logical Level Key dialog, type a name for the key.
   c. In the Logical Level Key dialog, select a column or click **Add**.
   d. If you click **Add**, in the Browse dialog, select the column, and then click **OK**.

      The column you selected appears in the **Columns** list of the Logical Level Key dialog and is automatically selected.

   You can't use a derived logical column that's the result of a `LOOKUP` function as part of a primary logical level key. This limitation exists because the `LOOKUP` operation is applied after aggregates are computed, but level key columns must be available before the aggregates are computed because they define the granularity at which the aggregates are calculated.

   You can use a derived logical column that's the result of a `LOOKUP` function as a secondary logical level key.
6. If the level is in a time dimension, you can select chronological keys and sort the keys by name.
7. (Optional) Type a description for the key and then click **OK**.
8. Repeat Step 2 through Step 7 to add primary keys to other logical levels.
9. In the Logical Level dialog, click **OK**.

Select and Sort Chronological Keys in a Time Dimension

At least one level of a time dimension must have a chronological key. Although you can select one or more chronological keys for any level and then sort keys in the level, only the first chronological key is used.

Pay attention when the column order in a chronological key has many columns. You set the column order using a SQL `ORDER BY` clause on the columns reflecting the real-world chronological order in the Chronological Key dialog of the Oracle BI Administration Tool. Since the range for quarters is 1 to 4 with 4 quarters in a year, using an `ORDER BY` clause with the Quarter before the Year (*Quarter*, *Year*) is incorrect. The incorrect order shows all first quarters across all years, before displaying any second quarters, and so on. To correct the results, use (*Year*, *Quarter*) in the `ORDER BY` clause.

To recognize a dimension as a time dimension, you must select **Time** on the General tab of the Dimension dialog.

1. In the Business Model and Mapping layer of the Administration Tool, expand a time dimension and then expand the highest level (Grand Total level) of the dimension.
2. Double-click a logical level below the Grand Total level.
3. In the Logical Level dialog, click the Keys tab.
4. To select a chronological key, in the Keys tab, select the **Chronological Key** option.
5. To sort chronological keys, in the Keys tab, select a chronological key and then click **Edit**.
6. In the Chronological Key dialog, select a chronological key column, click **Up** or **Down** to reorder the column, and then click **OK**.

Add a Dimension Level to the Preferred Drill Path

You can use the Preferred Drill Path tab to identify the drill path to use when Oracle BI Presentation Services users drill down in their data requests.

You should use this only to specify a drill path that's outside the normal drill path defined by the dimensional level hierarchy. It's most commonly used to drill from one dimension to another. You can delete a logical level from a drill path or reorder a logical level in the drill path.

1. To add a dimension level to the preferred drill path, click **Add** to open the Browse dialog, then select the logical levels to include in the drill path. You can select logical levels from the current dimension, or from other dimensions.
2. Click **OK** to return to the Level dialog.

Add Sequence Numbers to a Time Dimension's Logical Level

Adding absolute or relative sequence numbers to time dimensions optimizes time series functions and in some cases improves query time.

By default Oracle BI Server uses a complex RANK Physical SQL expression to generate sequence numbers for time dimensions. Adding an absolute or relative sequence number to the time dimension's logical level provides direct column references in the Time dimension table that contain the precomputed results of the rank expressions. This mapping, while optional, generates a simpler query that's easier for Oracle BI Server to run against the data source.

Sequence numbers are enumerations of time dimensional members at a certain level. Use an enumeration without gaps (dense). The enumeration must correspond to a real time order, for example, you can enumerate the months in a year from 1 to 12.

The sequence number type options are:

- **Absolute** - Choose this option to configure an absolute sequence number when the column enumerates the members of the time dimension without any reference, for example, calendar year.
- **Relative** - Chose this option to configure relative sequence numbers when you've a column that enumerates members of the time dimension relative to some parent level, for example, months in year from 1 to 12.

1. In the Business Model and Mapping layer of the Administration Tool, locate a time dimension and then double click a corresponding logical level.

2. In the Logical Level dialog, click the **Sequence Numbers** tab, specify the type of sequence numbers to add to the logical level.
3. Click **OK**.

Rules for Automatically Created Dimensions with Level-Based Hierarchies

The Create Dimensions option is only available if the selected logical table is a dimension table as defined by 1:N logical joins, and a dimension hasn't been associated with the table.

The following rules are applied:

- An automatically created dimension uses the same name as the logical table, adding Dim as a suffix. For example, if a table is named Periods, the dimension is named Periods Dim.
- A Grand Total level is automatically named *logical_table_name* Total. For example, the Grand Total level of the Periods Dim table is Periods Total.
- When there are multiple tables in a source, the join relationships between tables in the source determine the physical table containing the lowest-level attributes. The lowest level in the hierarchy is named *logical_table_name* Detail. For example, the lowest level of the periods table is Periods Detail.
- The logical key of the dimension table is mapped to the lowest level of the hierarchy and specified as the level key. This logical column should map to the key column of the lowest level table in the dimension source.
  - If there are two or more physical tables in a source, the columns that map to the keys of those tables become additional logical levels. These additional level names use the logical column names of these key columns.
  - The order of joins determines the hierarchical arrangement of the logical levels. The level keys of these new logical levels are set to the logical columns that map to the keys of the tables in the source.
- If there are multiple logical table sources, the tool uses attribute mappings and physical joins to determine the hierarchical order of the tables in the physical sources. For example, you might have three sources (A, B, C) each containing a single physical table and attribute mappings for 10, 15, and 3 attributes, respectively, not counting columns that are constructed from other logical columns. The following is a list of the results of creating a dimension for this table automatically:
  - The Administration Tool creates a dimension containing four logical levels, counting the Grand Total and detail levels.
  - The key of the table in source B, which has the greatest number of columns mapped and contains the column mapping for the logical table key, is the level key for the detail level.
  - The parent of the detail level is the logical level named for the logical column that maps to the key of the physical table in source A.
  - Any attributes that are mapped to both A and B should be associated with level A.
  - The parent of level A should be the logical level named for the logical column that maps to the key of the physical table in source C.
  - Any columns that are mapped to both A and C should be associated with level C.
- Table joins in a physical source might represent a pattern that results in a split hierarchy. For example, the Product table can join to the Flavor table and a Subtype table. This

would result in two parents of the product detail level, one flavor level and one subtype level.

- You can't create a dimension automatically in the following situations:
  - If a dimension with joins and levels has already been created, Create Dimension doesn't appear on the right-click menu.
  - If the table isn't yet joined to any other table, the option isn't available because it's considered a fact table.
- In a snowflake schema, if you use a table with only one source and create the dimension automatically, the child tables are automatically incorporated into a hierarchy. The child tables form intermediate levels between the Grand Total level and detail level. If more than one child table exists for a dimension table, the hierarchy is a split hierarchy.

Automatically Create Dimensions with Level-Based Hierarchies

You can set up a dimension automatically from a logical dimension table if a dimension for that table doesn't exist.

To create a dimension automatically, the Administration Tool examines the logical table sources and the column mappings in those sources and uses the joins between physical tables in the logical table sources to determine logical levels and level keys. As a best practice, create a dimension table after all the logical table sources have been defined for a dimension table.

1. In the Administration Tool, open a repository.
2. In the Business Model and Mapping layer, right-click a logical dimension table that isn't associated with any dimension.
3. From the right-click menu, select **Create Logical Dimension**, then select the **Dimension with Level-Based Hierarchy** or **Dimension with Parent-Child Hierarchy**.

   The new dimension is displayed in the Business Model and Mapping layer.

Populate Logical Level Counts Automatically

You can use Estimate Levels to automatically populate level counts for one or more dimension hierarchies.

Level counts are used by the query engine to determine the optimal query plan and to improve overall system performance.

You must open the repository in online mode and ensure that the business model is available for queries. In the Business Model and Mapping layer, you can select any of the following logical layer elements, and then run the Estimate Levels command:

- **Business model.** If you select the business model object, the Oracle BI Administration Tool attempts to check out all objects in the business model.
- **Dimension.** Run a consistency check on dimensions to ensure that the dimension is logically sound.
- **A combination of business models and dimensions.** You can select multiple dimensions and multiple business models individually.

When run, the Estimate Levels command also launches a consistency check on the level counts as described in the following list:

- Checks that a level key is valid. Columns in levels have referential integrity.
- Checks the parent-child relationship. If the parent level count is greater that the child level count, an error is returned.
- Generates a run report that lists all the counts that were estimated and any errors or consistency warnings.
- The queries and errors are logged in the `obis1_query.log` located in the *`DOMAIN_Home`*`/servers/obis1/logs`.

  Set the log level at 4 or higher to write this information to the log file. See Diagnose and Resolve Issues in *Administering Oracle Analytics Server*.

1. In the Administration Tool, open a repository in online mode.
2. Right-click one or more business models and dimension objects, and select **Estimate Levels**.
3. In the Check Out Objects dialog, click **Yes** to check out the objects that appear in the list.

   If you click **No**, the action terminates because you must check out items to run Estimate Levels.

   In the Administration Tool dialog, a list of the dimension level counts and any errors or warning messages appear.

When you check in the objects, you can check the global consistency of your repository.

Create and Manage Dimensions with Parent-Child Hierarchies

A parent-child hierarchy is a hierarchy of members that all have the same type.

This contrasts with level-based hierarchies, where members of the same type occur only at a single level of the hierarchy.

This section contains the following topics:

- About Parent-Child Hierarchies
- Create Dimensions with Parent-Child Hierarchies
- Define Parent-Child Relationship Tables
- Model Aggregates for Parent-Child Hierarchies
- Add the Parent-Child Relationship Table to the Model
- Maintain Parent-Child Hierarchies Based on Relational Tables

About Parent-Child Hierarchies

A common real-life parent-child hierarchy occurrence is an organizational reporting hierarchy chart.

In an organizational reporting hierarchy chart, the following can apply:

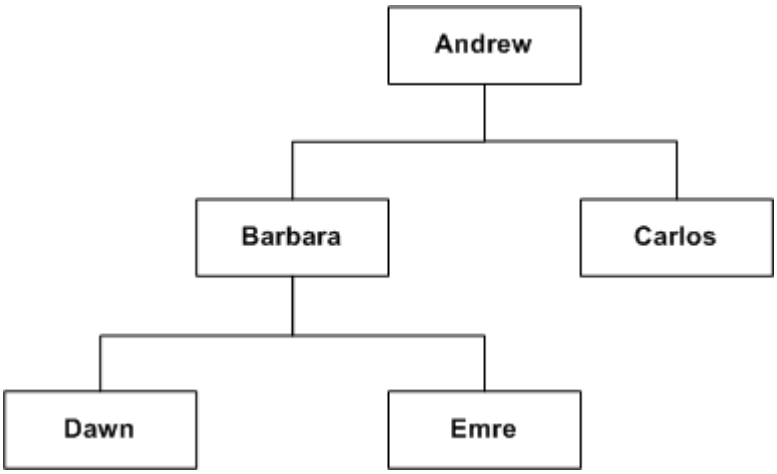
- Each individual in the organization is an employee.
- Each employee, apart from the top-level managers, reports to a single manager.

- The reporting hierarchy has many levels.

These conditions illustrate the basic features that define a parent-child hierarchy, namely:

- A parent-child hierarchy is based on a single logical table, for example, the *Employees* table
- Each row in the table contains two identifying keys, one to identify the member itself, the other to identify the parent of the member, for example, *Emp_ID* and *Mgr_ID*.

The image shows an example of a multi-level parent-child hierarchy.



The following table shows how this parent-child hierarchy could be represented by the rows and key values in an Employees table.

Emp_ID	Mgr_ID
Andrew	*null*
Barbara	Andrew
Carlos	Andrew
Dawn	Barbara
Emre	Barbara

You can expose logical dimensions with parent-child hierarchies to Answers users by creating presentation hierarchies that are based on particular logical dimensions. Creating hierarchies in the Presentation layer enables users to create hierarchy-based queries. See Work with Presentation Hierarchies and Levels.

This section contains the following topics:

- About Levels and Distances in Parent-Child Hierarchies
- About Parent-Child Relationship Tables

About Levels and Distances in Parent-Child Hierarchies

All the dimension members of a parent-child hierarchy occur in a single logical column.

In a parent-child hierarchy, the parent of a member is in another row in the same logical column, pointed to by the parent key. In a level-based hierarchy, the parent of a member is in a different logical column in the same row. Navigation in a parent-child hierarchy follows data values, while navigation in a level-based hierarchy follows the metadata structure.

In level-based hierarchies, each level is named, and occupies a position in the hierarchy that corresponds to a real-world attribute or category useful for analysis. In level-based hierarchies the number of levels is fixed at design time. There is no limit to the depth of a parent-child hierarchy, and the depth can change at run time due to new data.

Every parent-child hierarchy has two system-generated entities, Total and Detail, that are automatically defined when the logical dimension is created. The Detail entity contains all the hierarchy members. These two system-generated entities are different from the implicit, inter-member levels between ancestors and descendants in a parent-child hierarchy. The implicit levels are referred to as parent-child hierarchical levels.

Closely associated with levels is the concept of distance in parent-child hierarchies. The distance of one member from another is the number of parent-child hierarchical levels from the member to an ancestor or to a descendant. For example, the distance from a member to its parent is always 1. See About Parent-Child Hierarchies for an example.

About Parent-Child Relationship Tables

In relational tables, the relationships between different members in a parent-child hierarchy are implicitly defined by the identifier key values in the associated base table.

For each Oracle BI Server parent-child hierarchy defined on a relational table, you must also explicitly define the inter-member relationships in a separate parent-child relationship table.

The parent-child relationship table must include four columns, as follows:

- A column that identifies the member
- A column that identifies an ancestor of the member

  An ancestor is the parent of the member, or a higher-level ancestor.
- A distance column that specifies the number of parent-child hierarchical levels from the member to the ancestor
- A leaf column that indicates if the member is a leaf member (`1`=*Yes*, `0`=*No*)

The column names can be user defined. The data types of the columns must satisfy the following conditions:

- The member and ancestor identifier columns have the same data type as the associated columns in the logical table that contains the hierarchy members.
- The distance and leaf columns are `INTEGER` columns.

For the rows in a parent-child relationship table:

- Each member must have a row pointing at itself, with distance zero.
- Each member must have a row pointing at each of its ancestors. For a root member, this is a termination row with null for the parent and distance values.

The example shown in the table uses text strings for readability, but you normally use integer surrogate keys for `member_key` and `ancestor_key`, if they exist in the source dimension table.

The table shows an example of a parent-child relationship table with rows that represent the inter-member relationships for the employees. See the figure in About Parent-Child Hierarchies.

Member_Key	Ancestor_Key	Distance	Isleaf
Andrew	Andrew	0	0
Barbara	Barbara	0	0
Carlos	Carlos	0	0
Dawn	Dawn	0	0
Emre	Emre	0	0
Andrew	null	null	0
Barbara	Andrew	1	0
Carlos	Andrew	1	1
Dawn	Barbara	1	1
Dawn	Andrew	2	1
Emre	Barbara	1	1
Emre	Andrew	2	1

You generate scripts to create and populate the parent-child relationship table through a wizard that you can invoke when you define the parent-child hierarchy. Note the following about the create and load scripts:

- You run the create script only once, to create the parent-child relationship table in the data source.
- You must run the load script after each time the data changes in the dimension table. Because of this, you typically call the load script in your ETL processing. The load script reloads the entire parent-child relationship table; it isn't incremental.

If you don't choose to use the wizard, then you must create the parent-child relationship table manually and then import it into the Physical layer before associating it with the parent-child hierarchy. In this latter case, it's also your responsibility to populate the table with the data required to describe the inter-member relationships in the parent-child hierarchy.

Create Dimensions with Parent-Child Hierarchies

The key elements that you must define for a parent-child hierarchy are the identifier columns for the member and the parent of the member.

This basic principle applies to all parent-child hierarchies, regardless of the data source from which the hierarchy is derived.

Parent-child hierarchies that are based on relational tables must have an accompanying parent-child relationship table. See About Parent-Child Relationship Tables and Define Parent-Child Relationship Tables.

The Dimension with Parent-Child Hierarchy option is only available if there's at least one logical dimension table in the business model that doesn't have an associated dimension. The Browse window shows the logical dimension tables in the business model, each with their primary keys and the other columns in the table.

1. In the Business Model and Mapping layer of the Administration Tool, do one of the following:
   - Right-click a business model, select **New Object**, select **Logical Dimension**, and then select **Dimension with Parent-Child Hierarchy**.
   - Right-click a dimension table that isn't associated with any dimension and select **Create Logical Dimension**, then select **Dimension with Parent-Child Hierarchy**.
2. In the Logical Dimension dialog on the General tab, type a name for the dimension.
3. Click **Browse** located next to the **Member Key** field.
4. Select a **Member Key** for the parent-child hierarchy, and click **OK**.
5. Click **Browse** beside the **Parent Key** field.
6. Select a column for the Parent Key for the parent-child hierarchy and click **OK**.
7. If the logical table isn't from a relational table source, click **OK** to finish the process of creating the dimension.

If the logical table is from a relational table source, you must continue the dimension definition process by setting up the parent-child relationship table for the hierarchy.

Define Parent-Child Relationship Tables

Use these steps to define a parent-child relationship table for parent-child hierarchies based on relational tables.

When you create the parent-child relationship table, you must choose one of the following options:

- (Recommended method) Use a wizard that generates scripts to create and populate the parent-child relationship table.

  When you select **Create Parent-Child Relationship Table**, the Generate Parent-Child Relationship Table Wizard generates SQL scripts for creating and populating the parent-child relationship table. At the end of the wizard, the Oracle BI Server stores the scripts into directories chosen during the wizard session. The scripts, when run, make the parent-child relationship table available to the metadata repository.

  In the Generate Parent-Child Relationship Table wizard, you must provide a enter name for the DDL Script to Generate the Parent-Child Table, and select the location for storing the generate script. You must also supply a name for the parent-child relationship table and select the catalog or schema for the parent-child relationship table. You can preview the generated scripts.
- Select a previously-created parent-child relationship table.

The parent-child relationship table must have at least four columns that describe how the inter-member relationships are derived in the logical table selected for the hierarchy. See About Parent-Child Relationship Tables.

1. In the Logical Dimension dialog, click **Parent-Child Settings**.
2. Do one of the following to define the parent-child relationship table for the hierarchy:
   - (Recommended method) Click **Create Parent-Child Relationship Table** follow the wizard prompts.
   - Click **Select Parent-Child Relationship Table** to start the manual method of defining the parent-child relationship table for the parent-child hierarchy.

3. When using the manual method, select the physical table that acts as the parent-child relationship table for your parent-child hierarchy.

   The table must already exist in the Physical layer.

4. Map the four columns from the physical parent-child relationship table to the fields in the Parent-Child Table Column Details area, as follows:

   a. Select the Member Key column.

   b. Select the Parent Key column.

   c. Select the Relationship Distance column.

   d. Select the Leaf Node Identifier column.

5. Click **OK**, then click **OK** again to finish the manual process of defining the parent-child relationship table.

6. If you used the Generate Parent-Child Relationship Table Wizard to generate create and load scripts, run the scripts to create and load the parent-child relationship table in your data source.

Model Aggregates for Parent-Child Hierarchies

Fact tables in level-based hierarchies might only contain facts for a single level of the hierarchy.

Facts for higher-level dimension members can be calculated by aggregating the facts from the lower-level fact table or from a higher-level summary table.

In contrast, parent-child hierarchies require data modelers to make some additional decisions related to the following:

- How to store the base facts in the fact table
- How to aggregate the base facts to obtain the facts for higher-level members of the parent-child hierarchy

This section describes how to store and aggregate facts for parent-child hierarchies and contains the following topics:

- Store Facts for Parent-Child Hierarchies
- Aggregate Parent-Child Hierarchies

Store Facts for Parent-Child Hierarchies

There are two options for storing the base facts in the fact table for parent-child hierarchies.

You can use the following options:

- Store facts for only the leaf members of the parent-child hierarchy.
- Store facts for members at any level of the parent-child hierarchy, including non-leaf members.

The first option is more appropriate if the facts for the non-leaf members of the parent-child hierarchy can be derived entirely from the facts of the leaf members. For example, if you've a parent-child product hierarchy in which the actual product members appear only as leaf members of the hierarchy, then it makes sense for a revenue fact table to only record revenue facts for the leaf members of this product

hierarchy. The revenue figures for the non-leaf members of the product hierarchy such as the product categories can be derived entirely by aggregating the facts for the leaf product members at the bottom of the hierarchy.

The image shows example data for a situation where facts are stored only for leaf members in a parent-child hierarchy.

The following table shows example data for the dimension table PRODUCT_DIM:

MemberKey	Name	ParentKey
P1	Product1	C1
P2	Product2	C1
C1	Category1	C2
C2	Category2	C3
C3	Category3	-

The following table shows example data for the fact table REVENUE_FACTS:

ProductKey	YearKey	Revenue
P1	2011	100,000
P1	2012	105,000
P2	2011	75,000
P2	2012	80,000

The second option in which facts are stored for members at any level of the parent-child hierarchy is necessary when the facts for the non-leaf members aren't completely derived from facts of the leaf members. A good example is a sales person hierarchy in which a sales person might report to a manager who is also a sales person. Each individual sales person, including the manager, could have a different revenue figure stored in the fact table.

The table shows example data for this situation.

Facts Stored for Both Leaf and Non-Leaf Members

The following table shows example data for the dimension table SALES_REP_DIM:

MemberKey	Name	ParentKey
101	Phillip	201
102	Vivian	201
201	Jacob	301
202	Audrey	301
301	Ryan	-

The following table shows example data for the fact table REVENUE_FACTS:

SalesRepKey	YearKey	Revenue
101	2012	1,200,000
102	2012	1,100,000

SalesRepKey	YearKey	Revenue
201	2012	250,000
202	2012	1,400,000

Another case in which storing facts for both leaf and non-leaf members is appropriate is when the rules for aggregating the parent-child hierarchy are complex, or when aggregating the hierarchy at query time is expensive and would lead to unacceptably long query response times. In this case, the fact table would store preaggregated facts for the non-leaf members in addition to the facts stored for the leaf members.

Aggregate Parent-Child Hierarchies

As a data modeler, you must determine how to aggregate the stored facts to calculate the aggregated facts for higher level members of the parent-child hierarchy.

In addition to choosing the correct aggregation function for the measure, you must decide if you need to roll up the fact values recorded for lower-level members to calculate the values for higher-level members. In some cases, rolling up the facts of lower-level members of the parent-child hierarchy makes sense. In other cases such as with a pre-aggregated fact table or a measure that's intended to show each member's individual contribution, rolling up the facts from lower-level members of the parent-child hierarchy is incorrect.

Rolling up Facts from Lower-Level Members of a Parent-Child Hierarchy

If a fact table only stores facts for the leaf members of a parent-child hierarchy or if the fact table only records each member's individual contribution, then most likely the values stored in the fact table must be rolled up to obtain the correct aggregated value for higher-level members of the parent-child hierarchy. Rolling up the facts along a parent-child hierarchy is achieved by joining the fact table to the dimension table through the parent-child relationship table, see Add the Parent-Child Relationship Table to the Model.

For a fact table that stores facts only for the leaf members such as the product revenue fact table, this modeling technique calculates aggregate values that correctly summarize all the facts for the leaf-level members.

For a fact table that stores the individual contribution of both leaf members and non-leaf members, this technique computes a hierarchical aggregate that summarizes the individual contributions of the member and all its members.

Modeling Individual Contribution Measures

To report the individual contribution of each member, in addition, to reporting the summarized hierarchical aggregate that rolls up the individual contributions of multiple members, you must create two separate fact logical table sources. One fact logical table source maps the base fact table and the parent child relationship table. This is the logical table source for the hierarchical aggregate measure. The second fact logical table source maps only an alias of the fact table. This fact table alias should join directly with the dimension table rather that joining indirectly through the parent-child relationship table. This is the logical table source for the individual contribution measure.

Modeling Pre-aggregated Measures

Some fact tables contain pre-aggregated data that's populated for all members of the parent-child hierarchy. For example, the fact value for a root member might be populated with the aggregation of the data for all of its descendent members. It's important to ensure that queries don't aggregate the members from this dimension to avoid erroneous results.

To correctly model this type of parent-child hierarchy, you must create a parent-child relationship table to support hierarchical filter functions like `IsAncestor` and `IsDescendant`. You can join the parent-child dimension table directly with the fact table rather than joining through the parent-child relationship table to ensure that the pre-aggregated member value is returned, rather than rolling up all the descendants.

Don't modify the parent-child relationship table script to only include the *self* rows, because doing so would break the `IsAncestor` and `IsDescendant` functions.

To achieve the correct aggregation for dimensions of this type, you must determine what you want to see as a grand total when the parent-child hierarchy is aggregated. For example, assume that your hierarchy contains a single root member, and you want to display the pre-aggregated value for this root member. You must first create an additional fact logical table source mapped at the Total level of the parent-child hierarchy. Next, in the logical table source, create a `WHERE` clause filter that selects only the root member.

With this model in place, for queries that are at the Total level of the parent-child hierarchy, the Oracle BI Server selects the aggregate logical table source and applies the root member `WHERE` clause filter. For queries that are at the Detail level, the Oracle BI Server selects the detailed logical table source and returns the pre-aggregated member values. In either case, it doesn't matter how the aggregation rule is set, because there is a pre-aggregated source at each level.

Use this approach only if the queries are at the Total or Detail level of the parent-child dimension. For queries that group by some non-unique attribute of the parent-child dimension, the aggregation might not be correct. For example, if an Employee dimension has a Location attribute, and a query groups by *Employee.Location*, then double counting is likely because an employee often reports to other employees at the same location. Because of this, when fact tables contain pre-aggregated member values, you should avoid grouping by non-unique attributes of the parent-child dimension. If grouping by those attributes is unavoidable, then you should model them as separate dimensions.

Add the Parent-Child Relationship Table to the Model

For measures in fact tables that are aggregated by rolling up the facts from lower-level members, you must edit Physical layer joins to include the parent-child relationship table.

You need to add the parent-child relationship table to the appropriate logical table source.

For fact tables containing pre-aggregated data for a parent-child hierarchy or for individual contribution measures, you should join the parent-child dimension table directly with the fact table rather than joining through the parent-child relationship table.

Joining the parent-child dimension table directly with the fact table ensures that the pre-aggregated value or individual contribution value is returned, rather than rolling up all the descendants. When pre-aggregated data is populated for all members, don't add the parent-child relationship table to the logical table source to avoid over counting.

1. In the Administration Tool, right-click a physical table, select **Physical Diagram** , and then select **Selected Object(s) Only**.

2. Delete the direct joins from the dimension table to each of the fact tables.
3. Create a join from the parent-child relationship table to the dimension table using the ancestor key.
4. Create joins from the fact tables to the parent-child relationship table using the member key.
5. In the Business Model and Mapping layer, double-click the logical table source for the logical fact table that's used in your parent-child hierarchy.
6. In the **General** tab on the Logical Table Source dialog, click the **Add** button.
7. Click **Browse** to locate the parent-child relationship table in the Physical layer and click **Select**.
8. Click **OK** in the Logical Table Source dialog.

Maintain Parent-Child Hierarchies Based on Relational Tables

For parent-child hierarchies based on relational tables, you must ensure that the data in the parent-child relationship table accurately reflects the inter-member relationships in the dimension.

If you wrote scripts to create and populate the parent-child relationship table or used the Generate Parent-Child Relationship Table Wizard to create the scripts, you must run these scripts, adapting them to guarantee the integrity of the parent-child relationships in the hierarchy. You should add the Populate script to your extract-transform-load (ETL) process so that the script runs after the dimension table is updated.

Model Time Series Data

Time series functions provide the ability to compare business performance with previous time periods, allowing you to analyze data that spans multiple time periods.

For example, time series functions enable comparisons between current sales and sales a year ago, a month ago, and so on.

Because SQL doesn't provide a direct way to make time comparisons, you must model time series data in the Oracle BI repository. First, set up time dimensions based on the period table in your data warehouse. Then, you can define measures that take advantage of this time dimension to use the `AGO`, `TODATE`, and `PERIODROLLING` functions. At query time, the Oracle BI Server then generates highly optimized SQL that pushes the time offset processing down to the database whenever possible, resulting in the best performance and functionality.

This section contains the following topics:

- About Time Series Functions
- Create Logical Time Dimensions
- Create AGO, TODATE, and PERIODROLLING Measures

About Time Series Functions

Time series functions operate on time-oriented dimensions.

To use these functions on a particular dimension, you must designate the dimension as a Time dimension and set one or more keys at one or more levels as chronological keys. These keys identify the chronological order of the members within a dimension level.

Time series functions include `TODATE`, and `PERIODROLLING`. These functions let you use Expression Builder to call a logical function to perform time series calculations instead of aliasing physical tables and modeling logically. The time series functions calculate `AGO`, `TODATE`, and `PERIODROLLING` functions based on the calendar tables in your data warehouse, not on standard SQL date manipulation functions.

The image shows a sample report that includes several measures derived using time series functions.

	2008 Q1			2008 Q2			2008 Q3			2008 Q4		
	2008 / 01	2008 / 02	2008 / 03	2008 / 04	2008 / 05	2008 / 06	2008 / 07	2008 / 08	2008 / 09	2008 / 10	2008 / 11	2008 / 12
Dollars	100	200	300	101	202	303	110	220	330	444	555	666
Dollars Qago				100	200	300	101	202	303	110	220	330
Dollars QTD	100	300	600	101	303	606	110	330	660	444	999	1,665
Dollars 3-Period Rolling Sum	100	300	600	601	603	606	615	633	660	994	1,329	1,665
Dollars 3-Period Rolling Avg	33.3	100.0	200.0	200.3	201.0	202.0	205.0	211.0	220.0	331.3	443.0	555.0

Several different grains may be used in the time query, such as:

- Query grain

  The lowest time grain of the request.

- Time Series grain

  The time series grain indicates the aggregation or offset is requested, for the `AGO` and `TODATE` functions. In the report example shown in the image, the time series grain is Quarter. Time series queries are valid only if the time series grain is at the query grain or higher. The `PERIODROLLING` function doesn't have a time series grain; instead, you specify a start and end period in the function.

- Storage grain

  You can generate the report, shown in the example, from daily sales or monthly sales. The grain of the source is called the storage grain. A chronological key must be defined at this level for the query to work, but performance is generally much better if a chronological key is also defined at the query grain.

Queries against time series data must exactly match to access the query cache.

The following sections describe the time series conversion functions:

- About the AGO Function
- About the TODATE Function
- About the PERIODROLLING Function

About the AGO Function

The AGO function offsets the time dimension to display data from a past period.

This function is useful for comparisons such as *Dollars* compared to *Dollars a Quarter Ago*. The value of *Dollars Qago* for month 2008/08 equals the value of *Dollars* for month 2008/05.

The image shows example values for the Dollars and Dollars Qago measures.

	2008 Q1			2008 Q2			2008 Q3			2008 Q4		
	2008 / 01	2008 / 02	2008 / 03	2008 / 04	2008 / 05	2008 / 06	2008 / 07	2008 / 08	2008 / 09	2008 / 10	2008 / 11	2008 / 12
Dollars	100	200	300	101	202	303	110	220	330	444	555	666
Dollars Qago				100	200	300	101	202	303	110	220	330

In the example shown above, the Dollars Qago measure is derived from the Dollars measure.

In Expression Builder, the `AGO` function has the following template:

```
Ago(<<Measure>>, <<Level>>, <<Number of Periods>>)
```

`<<Measure>>` represents the logical measure column from which you want to derive. In this example, you would select the measure "Dollars" from your existing logical fact tables.

`<<Level>>` is the optional time series grain you want to use. In this example, you would select "Quarter" from your time dimension.

`<<Number of Periods>>` is the size of the offset, measured in the grain you provided in the `<<Level>>` argument. For example, if the `<<Level>>` is Quarter and the `<<Number of Periods>>` is 2, the function displays dollars from two quarters ago.

Using this function template, you can create an expression for a One Quarter Ago measure, as follows:

```
Ago("Sales"."Base Measures"."Dollars" , "Sales"."Time MonthDim"."Quarter" , 1)
```

The `<<Level>>` parameter is optional. If you don't want to specify a time series grain in the `AGO` function, the function uses the query grain as the time series grain.

For example, you could define Dollars_Ago as `Ago(Dollars, 1)`. Then, you could perform the following logical query:

```
SELECT Month, Dollars, Dollars_Ago
```

The result is the same as if you defined Dollars_Ago as `Ago(Dollars, Month, 1)`, or you could perform the following logical query:

```
SELECT Quarter, Dollars, Dollars_Ago
```

The result is the same as if you defined Dollars_Ago as `Ago(Dollars, Quarter, 1)`.

See *Logical SQL Reference Guide for Oracle Business Intelligence Enterprise Edition*.

About the TODATE Function

The TODATE function accumulates the measure from the beginning of the time series grain period to the current displayed query grain period.

For example, the image shows a report with the measure *Dollars QTD*, the Quarter To Date version of the *Dollars* measure.

	2008 Q1			2008 Q2			2008 Q3			2008 Q4		
	2008 / 01	2008 / 02	2008 / 03	2008 / 04	2008 / 05	2008 / 06	2008 / 07	2008 / 08	2008 / 09	2008 / 10	2008 / 11	2008 / 12
Dollars	100	200	300	101	202	303	110	220	330	444	555	666
Dollars QTD	100	300	600	101	303	606	110	330	660	444	999	1,665

In the example, Dollars QTD for Month 2008/05 is the sum of Dollars for 2008/04 and 2008/05. Dollars QTD is the sum of the values for all the query grain periods (month) for the current time series grain period (quarter). The accumulation starts over for the next quarter.

In the example, the Dollars QTD measure is derived from the Dollars measure.

In Expression Builder, the `TODATE` function uses the following format:

`ToDate(<<Measure>>, <<Level>>)`

<<*Measure*>> represents the logical measure column from which you want to derive. In this example, you select the measure *Dollars* from your existing logical fact tables.

<<*Level*>> is the time series grain you want to use. In this example, you select *Quarter* from your time dimension.

Using this function format, you can create the following expression for the measure:

`ToDate("Sales"."Base Measures"."Dollars" , "Sales"."Time MonthDim"."Quarter" )`

The query grain is specified in the query itself at run time. For example, this measure can display *Quarter To Date* at the *Day* grain, and accumulates up to the end of the Quarter.

See *Logical SQL Reference Guide for Oracle Business Intelligence Enterprise Edition*.

About the PERIODROLLING Function

The PERIODROLLING function lets you perform an aggregation across a specified set of query grain periods, rather than within a fixed time series grain.

The most common use is to create rolling averages such as a 13-week Rolling Average.

The `PERIODROLLING` function doesn't have a time series grain, the length of the rolling sequence is determined by the query grain. For example, the `Dollars 3-Period Rolling Average` calculates the mean of values from the last 3 months if the query grain is Month, but calculates the mean of the last 3 years if the query grain is Year.

The image shows a report with these two measures.

	2008 Q1			2008 Q2			2008 Q3			2008 Q4		
	2008 / 01	2008 / 02	2008 / 03	2008 / 04	2008 / 05	2008 / 06	2008 / 07	2008 / 08	2008 / 09	2008 / 10	2008 / 11	2008 / 12
Dollars	100	200	300	101	202	303	110	220	330	444	555	666
Dollars 3-Period Rolling Sum	100	300	600	601	603	606	615	633	660	994	1,329	1,665
Dollars 3-Period Rolling Avg	33.3	100.0	200.0	200.3	201.0	202.0	205.0	211.0	220.0	331.3	443.0	555.0

In the example above , the Dollars 3-Period Rolling Sum and Dollars 3-Period Rolling Avg measures are derived from the Dollars measure.

In Expression Builder, the `PERIODROLLING` function has the following format:

```
PeriodRolling(<<Measure>>, <<Starting Period Offset>>, <<Ending Period
Offset>>)
```

`<<Measure>>` represents the logical measure column from which you want to derive. To create the measure *Dollars 3-Period Rolling Sum*, you select the measure, *Dollars* from your existing logical fact tables.

`<<Starting Period Offset>>` and `<<Ending Period Offset>>` identify the first period and last period used in the rolling aggregation. The integer is the relative number of periods from the displayed period. In this example, the query grain is month, and the 3-month rolling sum starts 2 periods in the past and includes the current period, that is, for month 2008/07, the rolling sum includes 2008/05, 2008/06 and 2008/07. To create the measure, `Dollars 3-Period Rolling Sum`, the integers to indicate these offsets are -2 and 0.

Using this function format, you can create the following expression for the measure:

```
PeriodRolling("Sales"."Base Measures"."Dollars" , -2, 0)
```

The example also shows a 3-month rolling average. To compute this measure, you can divide the rolling sum that you previously created by 3 to get a 3-period rolling average. The assumption to divide the rolling sum by 3 results from the `<<Starting Period Offset>>` and `<<Ending Period Offset>>` fields for the rolling sum that are -2 and 0.

The expression for the 3-month rolling average is:

```
PeriodRolling("Sales"."Base Measures"."Dollars" , -2, 0) /3
```

Don't use the `AVG` function to create a rolling average. The `AVG` function computes the average of the database rows accessed at the storage grain. To perform the rolling average, you need an average where the denominator is the number of rolling periods at the query grain.

The `PERIODROLLING` function includes a fourth optional hierarchy argument that lets you specify the name of a hierarchy in a time dimension such as `yr, mon, day`, that you want to use to compute the time window. This option is useful when there are multiple hierarchies in a time dimension, or when you want to distinguish between multiple time dimensions. See *Logical SQL Reference Guide for Oracle Business Intelligence Enterprise Edition*.

Create Logical Time Dimensions

Creating time dimensions requires selecting a Time option and designating a chronological key for every level of every dimension hierarchy.

Use these guidelines when modeling time series data:

- Use a time series function when the data source contains history. A relational database that contains history might use a star or snowflake schema with an explicit time dimension table. A normalized, historical database might include a time hierarchy with levels in a schema similar to a snowflake. A simple date field isn't adequate for use with a time series function.
- Oracle Analytics Server requires the time dimension physical table or set of normalized tables that are separate from its related physical fact table.

  A common source schema pattern is a fully denormalized relational table or flat file that has time dimension columns are in the same table as facts and other

dimensions. This common source schema pattern can't qualify as a time dimension, because the time dimension table is combined with the fact table. Because you can't change the source model, you can create an Opaque View of the physical table containing the time columns to act as the distinct physical time dimension table. You must join the Opaque View time dimension to the physical table that contains the facts.

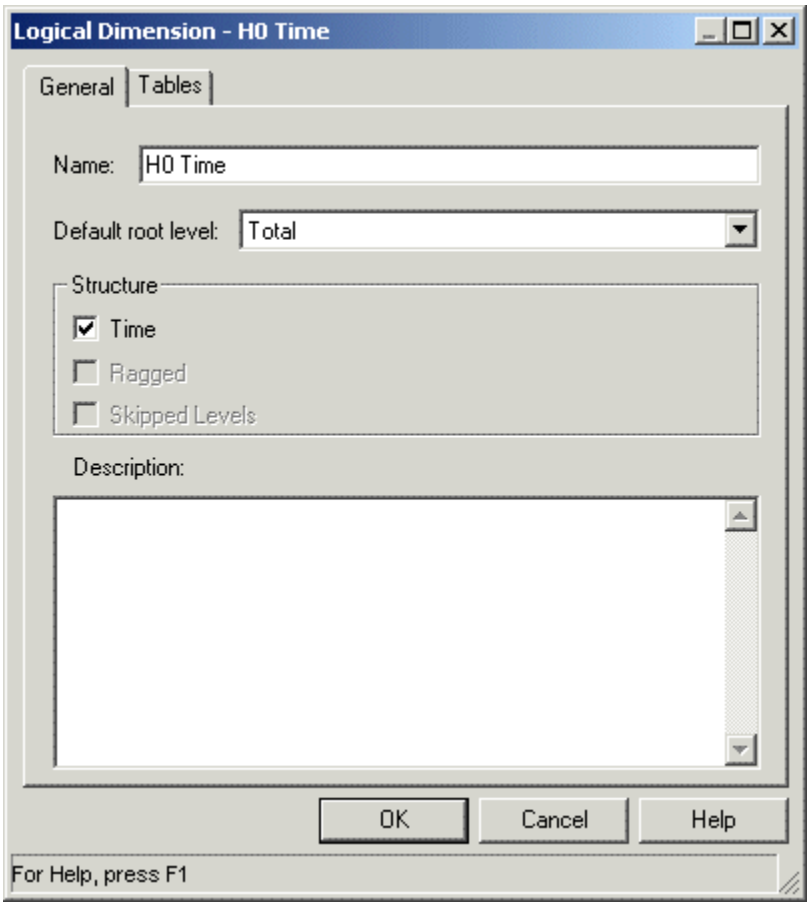
- In the Physical layer, the time dimension table or lowest-level table in the normalized/ snowflake must join directly to the fact table without an intervening tables. Create the join as a foreign key join.
- The tables in the physical model containing the time dimension can't join to other data sources, except at the most detailed level.
- A member value, a row in relational sources, must be physically present for every period at every hierarchy level. They must not contain rows that are skipped in the sequence. You don't need a fact data for every period. Only the dimension data must be complete.
- You must model each unit of distance between members such as *month*, *half*, or *year*, in a separate hierarchy level.

Select the Time Option in the Logical Dimension Dialog

Select the Time option in the General tab of the Logical Dimension dialog to enable time series functions on this dimension.

You can only use logical dimensions with the **Time** option selected as the time dimension for the time series functions `AGO`, `TODATE`, and `PERIODROLLING`.

The image shows the Time option in the Logical Dimension dialog.



Set Chronological Keys for Each Level

Designate a chronological key for every level of each dimension hierarchy.

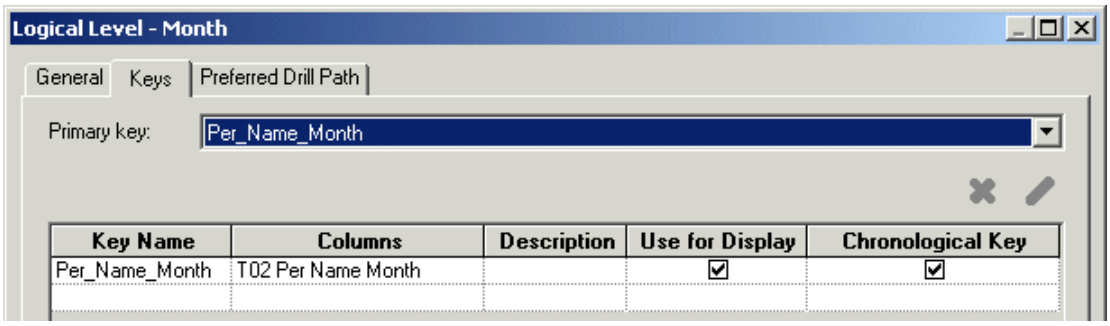
The chronological keys must be comparable with the standard SQL `ORDER BY` clause. The `ORDER BY` clause on the chronological key must reflect the real world chronological order of the time dimension members represented by the key. For example, if the time dimension members are: Jan-3-2013, Jan-4-2013, Jan-5-2013 then the following chronological keys can be assigned to them in the same order: 1, 5, 9. However, assigning chronological keys such as 2,1,3 would result in Jan-4-2013, Jan-3-2013, Jan-5-2013, which is an incorrect chronological order.

The Oracle BI Server uses the chronological key to create mathematically correct time series predictions, such as Jan + 2 months = Mar. You should set a chronological key for every level, except for the Grand Total level, so that you can perform time series operations on all levels with good performance. This enables you to use an `AGO`, `TODATE`, or `PERIODROLLING` function for any level of any time dimension hierarchy, such as fiscal month ago, calendar year ago, and day ago.

Theoretically, time series functions operate correctly if only the bottom level key in the Logical Dimension is chronological. In practice, however, this causes performance problems because it forces the physical query to use the lowest grain, causing joins of orders of magnitude more rows, for example, 365 times more rows for a "year ago" joining at the "day" grain.

As with any level key, be sure the key is unique at its level. For example, a column containing simple month names such as "January" isn't unique unless it's concatenated to a column containing year names.

The image shows how to designate a chronological key in the Logical Level dialog.



Create AGO, TODATE, and PERIODROLLING Measures

You can build time series measures by creating derived expressions from base measures.

To create a derived expression, you must create a new logical column and select **Derived from existing columns using an expression**, then open Expression Builder to build the appropriate time series function.

Follow these guidelines when modeling time series functions:

- You can't derive time series functions from measures that use the fragmentation form of federation. This rule prevents some complex boundary conditions and cross-source assumptions in the query generation and result merging, such as the need to join some time dimension rows from one source to some of the fact rows in a different source. To reduce maintenance and increase accuracy, it's best to create a single base measure, and then derive a family of time series measures from it. For example, start with a base measure, then define variations for month-ago, year-ago, and month-to-date.
- You must define the unit as a level of the time dimension, so that it can take advantage of the chronological keys built in the time dimension.

The following example shows how to build the `AGO` measure. See the *Logical SQL Reference Guide for Oracle Business Intelligence Enterprise Edition* for detailed syntax for the other time series functions, `TODATE` and `PERIODROLLING`.

11

Manage Logical Table Sources (Mappings)

This chapter explains how to work with logical table source objects and their mappings in the Business Model and Mapping layer of the Oracle BI repository.
This chapter contains the following topics:

- About Logical Table Sources
- Create Logical Table Sources
- Set Priority Group Numbers for Logical Table Sources
- Define Physical to Logical Table Source Mappings and Creating Calculated Items
- Define Content of Logical Table Sources
- About Working with Parent-Child Settings in the Logical Table Source
- Set Up Aggregate Navigation by Creating Sources for Aggregated Fact Data
- Set Up Fragmentation Content for Aggregate Navigation

About Logical Table Sources

Logical table sources define the mappings from a single logical table to one or more physical tables.

Use the physical to logical mapping to specify transformations that occur between the Physical layer and the Business Model and Mapping layer, and to enable aggregate navigation and fragmentation.

You can view logical table sources in the Business Model and Mapping layer.

Logical tables can have many physical table sources. A single logical column might map to many physical columns from multiple physical tables, including aggregate tables that map to the column such as if a query asks for the appropriate level of aggregation on that column.

This section contains the following topics:

- How Fact Logical Table Sources Are Selected to Answer a Query
- How Dimension Logical Table Sources Are Selected to Answer a Query
- Consistency Among Data in Multiple Sources

How Fact Logical Table Sources Are Selected to Answer a Query

The system uses criteria to select the fact logical table source to answer a query.

The following criteria is listed from the highest precedence to the lowest precedence:

- **Logical table source priority group**. A higher priority logical table source is used before a lower priority logical table source, even if the higher priority source is at a more detailed grain. A lower number indicates a higher priority. See Set Priority Group Numbers for Logical Table Sources.

- **The grain of the logical table source**. A higher-grain logical table source is used before a lower-grain logical table source, given that the priority group numbers are the same.
- **Number of elements at this level**. If the grains aren't comparable, the number specified for the **Number of elements at this level** field is considered.

  For example, assume you've the following two logical table sources with grains that aren't comparable: LTS1(year, city) and LTS2(month, state). If you've 10 years, 100 cities, 120 months, and 9 states, the worst case size of LTS1 is 10 x 100 = 1000, and the worst case size of LTS2 is 120 x 9 = 1080. In this scenario, LTS1 is selected because the source with the lowest estimated number of total elements is assumed to be the fastest.

  See Create Logical Levels in a Dimension.
- **First logical table source listed**. If all other criteria are equal, the first logical table source listed is selected, as shown in the Business Model and Mapping layer.

Every column in a query is sourced from a single logical table source based on these expected performance factors. Queries aren't load-balanced across multiple logical table sources.

How Dimension Logical Table Sources Are Selected to Answer a Query

After the appropriate fact logical table sources have been selected, the system selects the best dimensional logical table sources to answer the query.

Oracle BI Server uses the following criteria to select the dimension logical table source. The criteria are listed from the highest precedence to the lowest precedence:

- Logical table source priority group

  A higher priority dimension logical table source is used before a lower priority dimension logical table source. A lower number indicates higher priority.
- Lower join cost

  The dimension logical table source with the lowest join cost is selected before dimension logical tables sources with higher join costs, given that the priority group numbers are the same.
- Higher level

  If the priority group and join cost are the same, the higher level logical table source is chosen, because that logical table source could require joining fewer rows.

Change the Default Selection Criteria for Dimension Logical Table Sources

You can change the default logical table source selection criteria to favor dimension logical table sources that are at the same level as the fact logical table source before considering the higher level logical table source.

In Oracle BI Administration Tool, set the `DIMENSION_LTS_JOIN_RESTRICTIONS` session variable to `PREFER_SAME_LEVEL`.

If a suitable dimension logical table source at the same level as the fact logical table source doesn't exists, then Oracle BI Server selects the highest level dimension logical

table source that's joinable to the fact. These factors are only considered after priority group and join cost.

The `PREFER_SAME_LEVEL` value for the `DIMENSION_LTS_JOIN_RESTRICTIONS` session variable sets the following criteria for selecting the dimension logical table source to answer the query:

- Logical table source priority group
- Lower join cost
- Same level as the fact logical table source
- Higher level than other dimension logical table sources if no other logical table source is at the same level as the fact logical table source

When `DIMENSION_LTS_JOIN_RESTRICTIONS` is set to *NONE*, the default value, you can join fact logical table sources to a higher level dimension logical table source even if there is another joinable dimension logical table source at the same level as the fact.

Consistency Among Data in Multiple Sources

It's important to ensure that the data in your sources is consistent.

For example, your year-level logical table source and your month-level logical table source for your time dimension should cover the same time period.

Be aware that consistency issues with data in your sources might become apparent when you issue queries that override null suppression, in other words, when you create an analysis in Answers and select **Include Null Values**. For example, some aggregate tables might not include the dimension records that correspond to the null fact values such as a yearly sales aggregate table that doesn't include years with no sales. All years in the year dimension must exist for the null values to be included in the result.

Create Logical Table Sources

When you create logical tables and columns by dragging and dropping from the Physical layer, the logical table sources are generated automatically.

When you create logical tables and columns by dragging and dropping from the Physical layer, the logical table sources are generated automatically. If you create the logical tables manually, you need to also create the sources manually.

You also add new logical table sources when multiple physical tables can be the source of information. For example, many tables could hold information for revenue. You might have three different business units, each with its own order system, where you get revenue information. In another example, you might periodically summarize revenue from an orders system or a financial system and use this table for high-level reporting.

Use the **Allow Unmapped Tables** option for snowflake physical tables in an A > B > C configuration, where a logical table only maps to columns in A and C, but B needs to be included in the logical table source because it's in the join path between A and C.

1. In the Business Model and Mapping layer of the Administration Tool, right-click a logical table and select **New Object**, then select **Logical Table Source**.
2. In the Logical Table Source dialog, on the General tab, type a name for the logical table source.
3. Optional: Select **Disabled** to make this logical table source inactive.

4. Optional: Select **Allow Unmapped Tables** to enable this logical table source to have physical tables that aren't mapped to logical columns.
5. Click the **Add** button. In the Browse dialog, you can view joins and select tables for the logical table source. When there are two or more tables in a logical table source, all of the participating tables must have joins defined between them.
6. Optional: To view the joins, in the Browse dialog, select a table and click **View**. After reviewing the joins in the Physical Table dialog, click **Cancel**.
7. Optional: To add tables to the table source, select the desired tables in the **Name** list and click **Select**.
8. Click **OK**.

Set Priority Group Numbers for Logical Table Sources

You can set priority group numbers to determine which logical table source to use for queries for which there is more than one logical table source that can satisfy the requested set of columns.

For example, you might have user queries that are fulfilled by both a data warehouse and an OLTP source. Often, access to an operational system is expensive, while access to a data warehouse is cheap. In this situation, you can assign a higher priority to the data warehouse to ensure that all queries are fulfilled by the data warehouse if possible.

The priority group of a given logical table source doesn't always ensure that a particular query is fulfilled by that source. Priority group assignments are only one of many factors used by the Oracle BI Server to determine which logical table source to select for a given query. However, the logical table source priority is the most significant metric and is considered before any other cost metric.

To assign priority group numbers, rank your logical table sources in numeric order, with 0 being the highest-priority source. You can assign the same number to multiple sources. For example, you can have two logical table sources in priority group 0, two logical table sources in priority group 1, and so on. Often, only two priority groups are necessary (0 and 1).

Assigning priority groups is optional. All logical table sources are set to priority 0 by default. It's important that you don't use priority groups as a method fine tuning the choice of logical table sources used to answer queries. Oracle BI Server tries to automatically use the most optimal logical table sources, but only within the same priority group. When you set a different priority group to each logical table source, it might cause Oracle BI Server to use suboptimal logical table sources.

In some situations, you might want to allow users to reverse the normal logical table source priority ranking at query time. To accomplish this, you can use a combination of session variables and request variables with logical table source priority groups. This feature provides a way to dynamically select a source at run time, depending on user preference.

To enable this dynamic selection, you must first create the `REVERSIBLE_LTS_PRIORITY_SA_VEC` session variable in the repository. Create this variable as a string vector session variable that uses a row-wise session initialization block. `REVERSIBLE_LTS_PRIORITY_SA_VEC` should list the subject areas for which you want to allow users to reverse the logical table source priority ranking. You must define this variable to enable priority ranking reversal.

After you've defined the set of subject areas for which you want to allow priority ranking reversal, users can include the request variable `REVERSE_LTS_PRIORITY` with their queries to reverse the logical table source priority ranking. You can set this request variable to 1 to reverse the logical table source priority, or 0 to keep the normal logical table source priority.

As an alternative to using a request variable at query time, you can define a predetermined set of subject areas for which the logical table source priority is permanently reversed. To do this, create the session variable `REVERSED_LTS_PRIORITY_SA_VEC` in the repository. Create this variable as a string vector session variable that uses a row-wise session initialization block. `REVERSED_LTS_PRIORITY_SA_VEC` should list the subject areas for which you want the logical table source priority set to permanently reversed.

See Create Session Variables.

REVERSIBLE_LTS_PRIORITY_SA_VEC

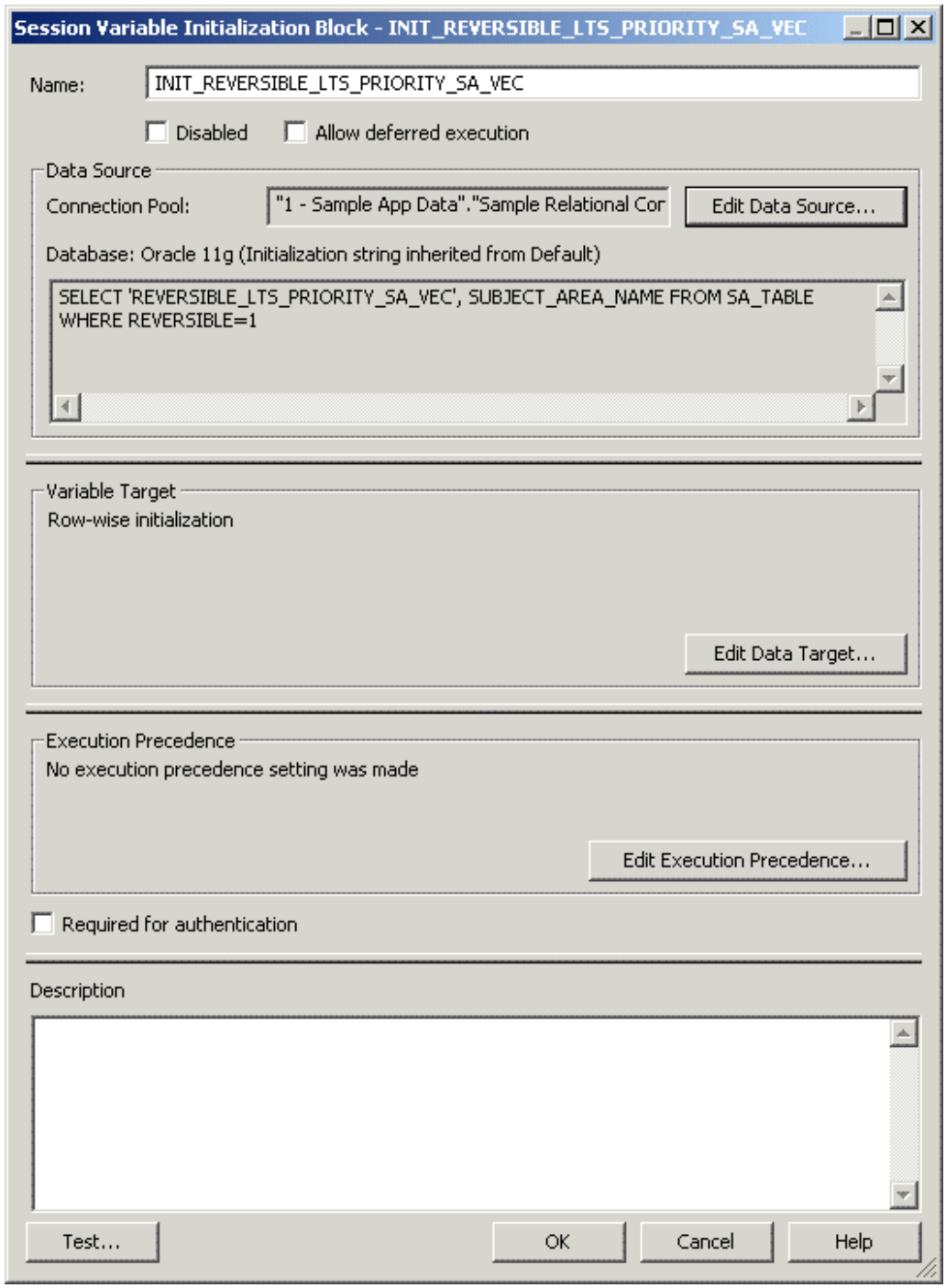
You could create a table called SA_TABLE that contains two columns: SUBJECT_AREA_NAME and REVERSIBLE. This table could contain rows mapping subject area names to their reversible values (1 or 0), as follows:

SUBJECT_AREA_NAME	REVERSIBLE
my_sa_1	1
my_sa_2	0

Then, you would create a string vector session variable called REVERSIBLE_LTS_PRIORITY_SA_VEC with a row-wise session initialization block. The initialization string for this initialization block is similar to the following:

```
SELECT 'REVERSIBLE_LTS_PRIORITY_SA_VEC', SUBJECT_AREA_NAME FROM SA_TABLE
WHERE REVERSIBLE=1
```

The image shows the Session Variable Initialization Block dialog for this example.


Session Variable Initialization Block - INIT_REVERSIBLE_LTS_PRIORITY_SA_VEC
Name:
INIT_REVERSIBLE_LTS_PRIORITY_SA_VEC
Disabled
Allow deferred execution
Data Source
Connection Pool:
"1 - Sample App Data"."Sample Relational Cor
Edit Data Source...
Database: Oracle 11g (Initialization string inherited from Default)
SELECT 'REVERSIBLE_LTS_PRIORITY_SA_VEC', SUBJECT_AREA_NAME FROM SA_TABLE
WHERE REVERSIBLE=1
Variable Target
Row-wise initialization
Edit Data Target...
Execution Precedence
No execution precedence setting was made
Edit Execution Precedence...
Required for authentication
Description
Test...
OK
Cancel
Help

Define Physical to Logical Table Source Mappings and Creating Calculated Items

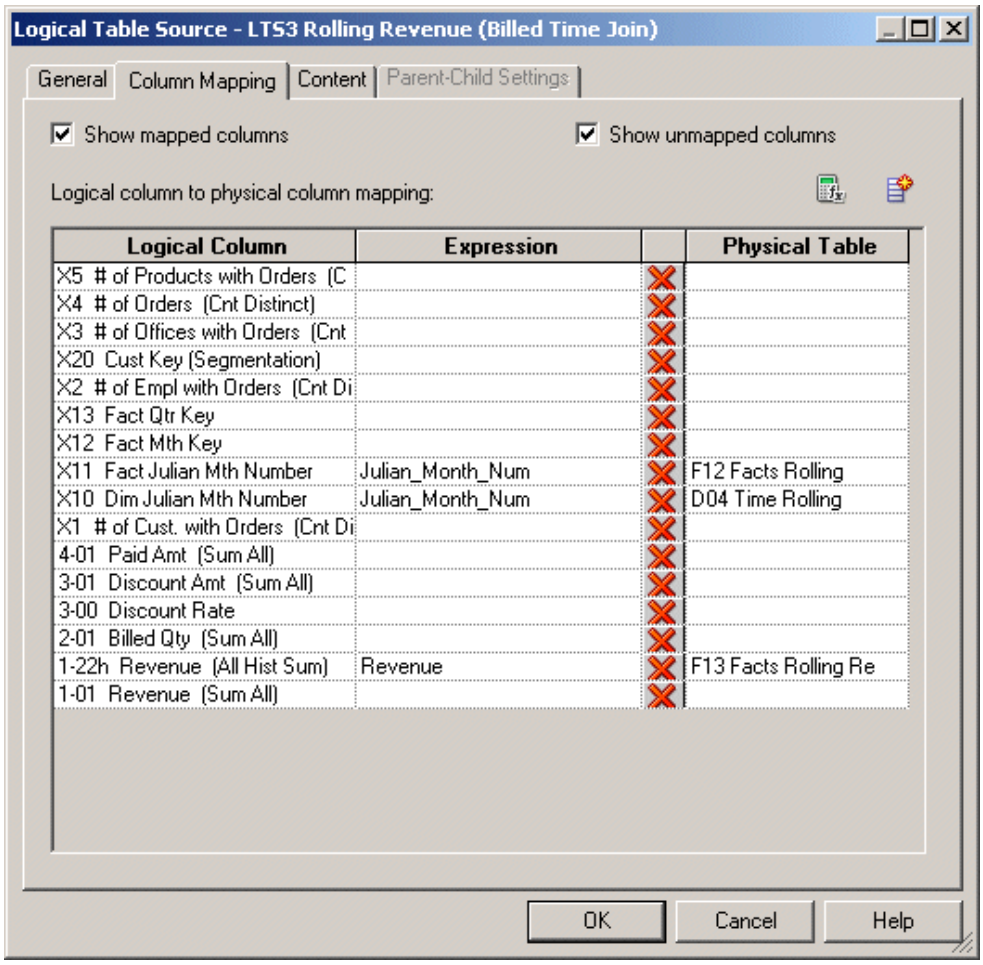
Use the Column Mapping tab of the Logical Table Source dialog to map logical columns to physical columns.

The physical to logical mapping can also be used to specify transformations that occur between the Physical layer and the Business Model and Mapping layer. The transformations can be simple, such as changing an integer data type to a character, or more complex, such as applying a formula to find a percentage of sales per unit of population. Applying these transformations is typically referred to as creating calculated items.

The data type of a logical column is determined by its logical table source mappings. For example, if a logical column has one physical source with a data type of `VARCHAR(50)` *not-nullable*, and another physical source with a `VARCHAR(20)` data type, *nullable*, then the data type of the logical column is `VARCHAR(50)` *nullable*. This final type is called a promoted type. Because of the rules governing logical table source mappings, you can't map physical sources with data types that are promotable such as an `INT` with a `VARCHAR`.

1. In the Business Model and Mapping layer of the Administration Tool, double-click a logical table source.
2. In the Logical Table Source dialog, click the Column Mapping tab.
3. In the Column Mapping tab, maximize or enlarge the dialog to show all the contents, as shown in the image.

   In the Column Mapping tab, in the **Logical column to physical column mapping** area, sort the rows by clicking a column heading.



4. In the **Physical Table** column, select the table that contains the column you want to map.

   When you select a cell in the Physical Table column, a list appears. It contains a list of tables currently included in this logical table source.

5. In the **Expression** column, select the physical column corresponding to each logical column.

   When you select a cell in the **Expression** column, a list appears. It contains a list of physical columns currently included in this logical table source.

6. To open Expression Builder, click the **Expression Builder** button.

   All columns used in creating physical expressions must be in tables included in the logical table source. You can't create expressions involving columns in tables outside the source.

   You can use Expression Builder to create calculated items, in which formulas are applied pre-aggregation. For example, to create the measure *tons sold* using the columns *units_sold* and *unit_weight*, you apply a pre-aggregation formula (`fact.units_sold*product.unit_weight`), and then apply the aggregation rule `SUM` in the measure object. Another example is using `CAST` to transform a column of type `TIMESTAMP` to type `DATE` for faster display in Answers and other clients, for example, `CAST("DB"."TABLE"."COL" AS DATE)`.

   You can also change data sources by creating expressions that perform transformations on physical data. For example, you can use the `CAST` function to

transform a column with a character data type to an integer data type, to match data coming from a second logical table source. Other examples include using `CONCATENATE` or math functions to make similar transformations on physical data.

See Answers for calculations that need to occur post-aggregation.

7. To remove a column mapping, click the **Delete** button.
8. After you map the appropriate columns, click **OK**.

Unmap a Logical Column from Its Source

You can edit the logical table sources from which the column derives its data, or unmap it from its sources.

In the Logical Column dialog, the Column Source tab contains information about the logical column.

1. In the Business Model and Mapping layer of the Administration Tool, double-click a logical column.
2. In the Logical Column dialog, click the Column Source tab.
3. In the **Logical Table Source** list, select a source and click **Unmap**.
4. Click **OK**.

Define Content of Logical Table Sources

To use a source correctly, the Oracle BI Server has to know what each source contains in terms of the business model.

Therefore, you need to define aggregation content for each logical table source of a fact table. The aggregation content rule defines at what level of granularity the data is stored in this fact table. For each dimension that relates to this fact logical table, define the level of granularity, making sure that every related dimension is defined. See Set Up Aggregate Navigation by Creating Sources for Aggregated Fact Data.

If a logical table is sourced from a set of fragments, it isn't required that every individual fragment maps the same set of columns. However, the server returns different answers depending on how columns are mapped.

- If all the fragments of a logical table map the same set of columns, then the set of fragmented sources is considered to be the whole universe of logical table sources for the logical table. This means that measure aggregations can be calculated based on the set of fragments.
- If the set of mapped columns differ across the fragments, then the Oracle BI Server assumes that it doesn't have the whole universe of fragments, and therefore it would be incorrect to calculate aggregate rollups, since some fragments are missing. In this case, the server returns NULL as measure aggregates.

> **Note:**
>
> Oracle highly recommends that all the fragments map to the same set of columns.

Use the Content tab of the Logical Table Source dialog to define any aggregate table content definitions, fragmented table definitions for the source, and `WHERE` clauses, if you want to limit the number of rows returned. See Set Up Fragmentation Content for Aggregate Navigation.

Verify Joins from Dimension Tables to Fact Tables

Joins tells the Oracle BI Server where to send queries for physical aggregate fact tables joined to and constrained by values in the physical aggregate dimension tables.

Oracle recommends that you use logical levels exclusively as the **Aggregation content, group-by** option. Don't mix aggregation by logical level and column in the same business model.

See About WHERE Clause Filters and Set Up Fragmentation Content for Aggregate Navigation.

You can type the formula directly into the **Fragmentation content** text area , or click Expression Builder. In the Expression Builder for Fragmentation Content, you can specify content in terms of existing logical columns. See Set Up Fragmentation Content for Aggregate Navigation.

Choose **This source should be combined with other sources at this level** if all fragments on this level are disjointed. Consider the following examples:

- Example 1 - Suppose you've two fragments, all sales including current year, and current year sales with the fragmentation predicate set to year = 2015. You shouldn't select the **This source should be combined with other sources at this level** option because the two fragments overlap. Oracle BI Server can use any single fragment based on query predicate or fragmentation predicate compatibility.
- Example 2 - Suppose you've two fragments, sales for year 2000 and before, according to the fragmentation predicate, and sales for year 2001 and after, according to the fragmentation predicate. You should select the **This source should be combined with other sources at this level** option because the fragments don't overlap. The Oracle BI Server creates a union of all the logical table sources on this level that can't be disqualified based on query predicate or fragmentation predicate compatibility.

See Logical Table Source Options Reference to learn which option to use in the Logical Table Source dialog.

1. In the Business Model and Mapping layer of the Administration Tool, expand a logical fact table, expand **Sources**, and double-click a logical fact source table.
2. In the Logical Table Source dialog, click the **Content** tab.
3. If a logical source is an aggregate table and you've defined logical dimensions, in the Logical Table Source dialog, select **Logical Level** from the **Aggregation content, group-by** list.
4. Specify a logical level for each dimension, unless you're specifying the Grand Total level. Dimensions with no level specified are interpreted as being at the most detailed level, in the **Logical Level** list, select the appropriate level for each logical dimension table to which the logical fact table is joined.

    - After specifying the appropriate logical level, skip to step 8.

5. Optional: (Not recommended) To specify aggregate content by column, from the **Aggregation content, group-by** list, select **Column**.
6. Click the **Table** column, and select each logical dimension table that defines the aggregation level of the source.
7. Click **Column**, and select the logical column for each dimension that defines how the aggregations were grouped.

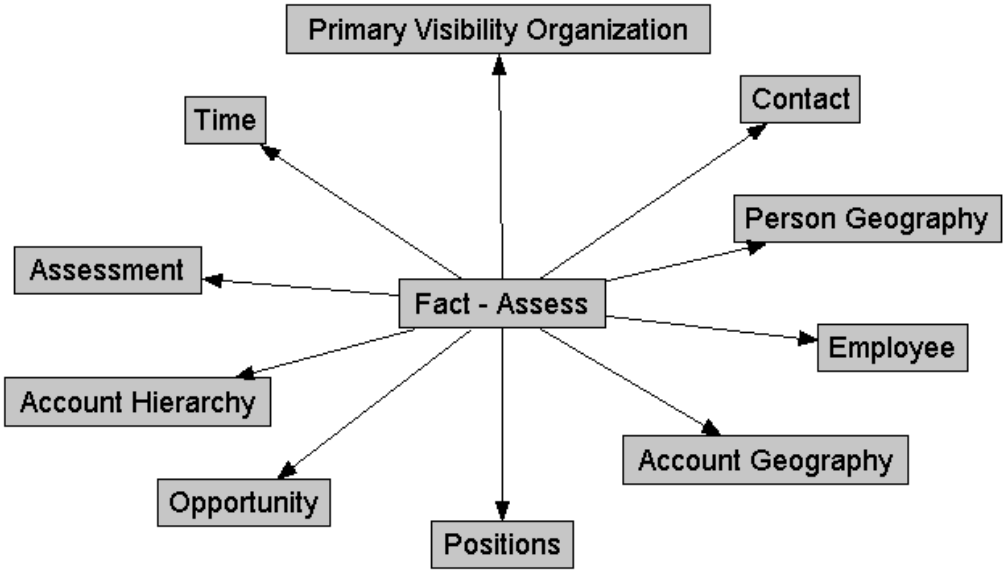
   When there are many logical columns to choose from, select the column that maps to the key of the source physical table. For example, if data has been aggregated to the Region logical level, pick the logical column that maps to the key of the Region table.
8. Optional: Use the **Fragmentation content** text are to describe the range of values included in the source when a source represents a portion of the data at a given level of aggregation to specify fragmented table definitions for the source.
9. Optional: Select **This source should be combined with other sources at this level** if all fragments on this level are disjointed.
10. Optional: To limit the number of rows the source uses in the resultant table, specify `WHERE` clause filters in **Use this "WHERE clause" filter to limit rows returned (exclude the "WHERE")**. You can enter the `WHERE` clause directly, or you can click the Expression Builder button to open the Expression Builder, create the `WHERE` clause, and click **OK**.
11. Optional: If the values for the source are unique, select the option **Select distinct values**.

Joins from Dimension Tables to Fact Tables

You must create joins between the aggregate fact tables and the aggregate dimension tables in the Physical layer.

You can verify joins by selecting a fact logical table and opening a Business Model Diagram (Selected Tables and Direct Joins). Only the dimension logical tables that are directly joined to this fact logical table appear in the diagram. It doesn't show dimension tables if the same physical table is used in logical fact and dimension sources.

The image shows a Fact - Assess logical fact table in a Business Model Diagram in the Selected Tables and Direct Joins view.



The table contains a list of the logical level for each dimension table that's directly joined the Fact - Assess fact table.

Dimension	Logical Level
Account Geography	Postal Code Detail
Person Geography	Postal Code Detail
Time	Day Detail
Account Organization	Account Detail
Opportunity	Opty Detail
Primary Visibility Organization	Detail
Employee	Detail
Assessment	Detail
Contact (W_PERSON_D)	Detail
FINS Time	Day
Positions	Details

Logical Table Source Options Reference

Learn how to use the options from the Logical Table Source dialog.

Options	Description
Aggregation content, group by	Specifies how the content is aggregated.

Options	Description
Copy	The **Copy** option is only available with fact tables. Copies aggregation content to the Windows clipboard. You can paste the `Dimension.Level` info into a text editor and use it for searching or for adding to documentation. **Copy** isn't available if the expression is empty.
Copy from	The **Copy from** option is available for fact tables and dimension tables. Copies aggregation content from another logical table source in the same business model. You need to specify the source from which to copy the aggregation content. Multiple business models appear but only the logical table sources from the current business model are selectable.
Get Levels	The **Get Levels** option is only available for fact tables. Changes aggregation content. If joins don't exist between fact table sources and dimension table sources, for example, if the same physical table is in both sources, the aggregation content determined by the Administration Tool doesn't include the aggregation content of this dimension.
Check Levels	The **Check Levels** option is only available for fact tables. Checks the aggregation content of logical fact table sources, not dimension table sources. The information returned depends on the existence of dimensions and hierarchies with logical levels and level keys, and physical joins between tables in dimension table sources and the tables in the fact table source. If the same tables exist in the fact and dimension sources and there are no physical joins between tables in the sources, Check Levels doesn't include the aggregation content of this dimension.
Fragmentation content	A description of the contents of a data source in business model terms. Data is fragmented when information at the same level of aggregation is split into multiple tables depending on the values of the data. A common situation would be to have data fragmented by time period.
This source should be combined with other sources at this level	Select this option when data sources at the same level of aggregation don't contain overlapping information. In this situation, all sources must be combined to get a complete picture of information at this level of aggregation.
Select distinct values	Used if the values for the source are unique.

About WHERE Clause Filters

The WHERE clause filter is used to constrain the physical tables referenced in the logical table source.

If there are no constraints on the aggregate source, leave the `WHERE` clause filter blank.

Each logical table source should contain data at a single intersection of aggregation levels. You wouldn't want to create a source, for example, that had sales data at both the Brand and Manufacturer levels. If the physical tables include data at multiple levels, add an appropriate `WHERE` clause constraint to filter values to a single level.

Any constraints in the `WHERE` clause filter are made on the physical tables in the source.

About Working with Parent-Child Settings in the Logical Table Source

Learn about when a logical table is part of a dimension with a parent-child hierarchy that's based on relational tables.

When this is the case, the logical table includes both a physical source and a source for the parent-child relationship table required for the parent-child hierarchy. Parent-child relationship tables explicitly define the inter-member relationships for parent-child hierarchies.

Typically, logical table sources for parent-child relationship tables are created automatically when you run the scripts created by the Generate Parent-Child Table Wizard. You access this wizard from the Parent-Child Table Settings dialog, available in the dimension object.

The Generate Parent-Child Table Wizard feature isn't available from the Logical Table Source dialog. You must go to the dimension object to create scripts to generate the parent-child relationship table.

You can view details for the parent-child relationship table source in the Parent-Child Settings tab of the Logical Table Source dialog. The following information appears in the tab:

- **Parent-Child Table:** Shows the name of the parent-child relationship table on which this source is based.
- **Member Key:** The name of the column in the parent-child relationship table that identifies the member.
- **Parent Key:** The name of the column in the parent-child relationship table that identifies an ancestor of the member.
- **Relationship Distance:** The name of the column in the parent-child relationship table that specifies the number of parent-child hierarchical levels from the member to the ancestor.
- **Leaf Node Identifier:** The name of the column in the parent-child relationship table that indicates if the member is a leaf member (1=Yes, 0=No).

See Create Dimensions with Parent-Child Hierarchies.

Set Up Aggregate Navigation by Creating Sources for Aggregated Fact Data

Aggregate tables store precomputed results from measures that have been aggregated over a set of dimensional attributes.

Each aggregate table column contains data at a given set of levels. For example, a monthly sales table might contain a precomputed sum of the revenue for each product in each store during each month. You configure this metadata in the Content tab of the Logical Table Source dialog.

When you create a logical table source for an aggregate fact table, you should create corresponding logical dimension table sources at the same levels of aggregation.

You need to have at least one logical dimension table source for each level of aggregation. If the sources at each level already exist, you don't need to create new ones.

For example, you might have a monthly sales fact table containing a precomputed sum of the revenue for each product in each store during each month. You need to have the following three other dimension sources, one for each of the logical dimension tables referenced in the example:

- A source for the Product logical table with one of the following content specifications:
  - By logical level: ProductDimension.ProductLevel
  - By column: Product.Product_Name
- A source for the Store logical table with one of the following content specifications:
  - By logical level: StoreDimension.StoreLevel
  - By column: Store.Store_Name
- A source for the Time logical table with one of the following content specifications:
  - By logical level: TimeDimension.MonthLevel
  - By column: Time.Month

At query time, the Oracle BI Server first determines which sources have enough detail to answer the query. Out of these sources, the Oracle BI Server chooses the most aggregated source to answer the query, because it's assumed to be the fastest. The most aggregated source is the one with the lowest multiplied number of elements. See Create Logical Levels in a Dimension to learn how to specify the number of elements at each level.

Set Up Fragmentation Content for Aggregate Navigation

When a logical table source doesn't contain the entire set of data at a given level, you need to specify the portion, or fragment, of the set that it does contain.

You describe the content in terms of logical columns in the **Fragmentation content** box in the Content tab of the Logical Table Source dialog.

The examples in this section illustrate techniques and rules for specifying the fragmentation content of sources.

This section contains the following topics:

- Specify Fragmentation Content for Single Column, Value-Based Predicates
- Specify Fragmentation Content for Single Column, Range-Based Predicates
- Specify Fragmentation Content for Aggregate Table Fragments

Specify Fragmentation Content for Single Column, Value-Based Predicates

You can replace the `IN` predicates with either an equality predicate or multiple equality predicates separated by the `OR` connective.

Fragment 1:

```
logicalColumn IN <valueList1>
```

Fragment n:

```
logicalColumn IN <valueList_N>
```

Specify Fragmentation Content for Single Column, Range-Based Predicates

When a logical table source doesn't contain the entire set of data at a given level, you need to specify the fragment of the set that it does contain.

Use >= and < predicates to ensure that the fragment content descriptions don't overlap. For each fragment, you must express the upper value as <. An error occurs if you use <=. You can't use the `BETWEEN` predicate to describe fragment range content.

Fragment 1:

```
logicalColumn >= valueof(START_VALUE) AND logicalColumn < valueof(MID_VALUE_1)
```

Fragment 2:

```
logicalColumn >= valueof(MID_VALUE_1) AND logicalColumn < valueof(MID_VALUE_2)
```

Fragment n:

```
logicalColumn >= valueof(MID_VALUE_N-1) AND logicalColumn < valueof(END_VALUE)
```

Pick your start point, midpoints, and endpoint carefully.

The `valueof` referenced here is the value of a repository variable. If you use repository values in your expression, the following construct doesn't work for Fragment 2:

```
logicalColumn >= valueof(MID_VALUE_1)+1 AND logicalColumn < valueof(MID_VALUE_2)
```

Use another repository variable instead of `valueof(MID_VALUE1)+1`.

The same variables, for example, `valueof(MID_VALUE1)`, aren't required to appear in the content of both fragments. You could set another variable, and create statements of the following form:

Fragment 1:

```
logicalColumn >= valueof(START_VALUE) AND logicalColumn < valueof(MID_VALUE_1)
```

Fragment 2:

```
logicalColumn >= valueof(MID_VALUE_2) AND logicalColumn < valueof(MID_VALUE_3)
```

See Use Variables in the Oracle BI Repository.

Specify Multicolumn Content Descriptions

An arbitrary number of predicates on different columns can be included in each content filter. Each column predicate can be value-based or range-based.

Fragment 1:

```
<logicalColumn1 predicate> AND <logicalColumn2 predicate > ... AND
<logicalColumnM predicate>
```

Fragment n:

```
<logicalColumn1 predicate> AND <logicalColumn2 predicate > ... AND <logicalColumnM
predicate>
```

Ideally, all fragments have predicates on the same M columns. If there is no predicate constraint on a logical column, the Oracle BI Server assumes that the fragment contains data for all values in that logical column. See Specify Parallel Content Descriptions for exceptions using the `OR` predicate.

Specify Parallel Content Descriptions

Use the parallel OR to handle dates that cross logical columns such as across years, or across months in a date range.

Use the `parallel OR technique` to handle the multiple hierarchical relationships across logical columns such as from year to year month to date, and from month to year month to date. For example, consider fragments delineated by different points in time such as year and month. Constraining sufficiently far back in a year is enough to drive the selection of just the historical fragment. The `parallel OR technique` supports this, as shown in the next example. This example assumes that the snapshot month was April 1, 12:00 a.m. in the year 1999.

Fragment 1 (Historical):

```
EnterpriseModel.Period."Day" < VALUEOF("Snapshot Date") OR
EnterpriseModel.Period.MonthCode < VALUEOF("Snapshot Year Month") OR
EnterpriseModel.Period."Year" < VALUEOF("Snapshot Year") OR
EnterpriseModel.Period."Year" = VALUEOF("Snapshot Year") AND
  EnterpriseModel.Period."Month in Year" < VALUEOF("Snapshot Month") OR
EnterpriseModel.Period."Year" = VALUEOF("Snapshot Year") AND
  EnterpriseModel.Period."Monthname" IN ('Mar', 'Feb', 'Jan')
```

Fragment 2 (Current):

```
EnterpriseModel.Period."Day" >= VALUEOF("Snapshot Date") OR
EnterpriseModel.Period.MonthCode >= VALUEOF("Snapshot Year Month") OR
EnterpriseModel.Period."Year" > VALUEOF("Snapshot Year") OR
EnterpriseModel.Period."Year" = VALUEOF("Snapshot Year") AND
  EnterpriseModel.Period."Month in Year" >= VALUEOF("Snapshot Month") OR
EnterpriseModel.Period."Year" = VALUEOF("Snapshot Year") AND
  EnterpriseModel.Period."Monthname" IN ('Dec', 'Nov', 'Oct', 'Sep', 'Aug', 'Jul',
  'Jun', '', 'Apr')
```

If the logical model doesn't go down to the date level of detail, then omit the predicate on `EnterpriseModel.Period."Day"` in the preceding example.

Notice the use of the `OR` connective to support parallel content description tracks.

Examples of Parallel Content Descriptions

These examples explain how to use labels with fragment content statements.

The Track number labels in the examples are shown to help relate the examples to the discussion that follows. You wouldn't include these labels in the actual fragmentation content statement.

Fragment 1 (Historical)

```
Track 1  EnterpriseModel.Period."Day" < VALUEOF("Snapshot Date") OR
Track 2  EnterpriseModel.Period.MonthCode < VALUEOF("Snapshot Year Month") OR
```

```
Track 3  EnterpriseModel.Period."Year" < VALUEOF("Snapshot Year") OR
Track 4  EnterpriseModel.Period."Year" = VALUEOF("Snapshot Year") AND
  EnterpriseModel.Period."Month in Year" < VALUEOF("Snapshot Month") OR
Track 5  EnterpriseModel.Period."Year" = VALUEOF("Snapshot Year") AND
  EnterpriseModel.Period."Monthname" IN ('Mar', 'Feb', 'Jan')
```

For example, consider the first track on `EnterpriseModel.Period."Day."` In the historical fragment, the < predicate tells the Oracle BI Server that any queries that constrain on Day before the Snapshot Date fall within the historical fragment. Conversely, the >= predicate in the current fragment on Day indicates that the current fragment doesn't contain data before the Snapshot Date.

The second track on `MonthCode`, for example, 199912, is similar to Day. It uses the < and >= predicates, as there is a non-overlapping delineation on month because the snapshot date is April 1. The key rule to remember is that each additional parallel track must reference a different column set. You can use common columns, but the overall column set must be unique. The Oracle BI Server uses the column set to select the most appropriate track.

The third track on `Year`, < in the historical fragment and > in the current fragment, tells the Oracle BI Server that optimal (single) fragment selections can be made on queries that just constrain on year. For example, a logical query on Year IN (1997, 1998) should only hit the historical fragment. Likewise, a query on Year = 2000 should only hit the current fragment. However, a query that hits the year 1999 can't be answered by the content described in this track, and therefore hits both fragments, unless additional information can be found in subsequent tracks.

The fourth track describes the fragment set for Year and Month in Year (month integer). Notice the use of the multi-column content description technique, described previously. Notice the use of < and >= predicates, as there is no ambiguity or overlap for these two columns.

The fifth track describes fragment content in terms of Year and Month name. It uses the value-based `IN` predicate technique.

As an embellishment, suppose the snapshot date fell on a specific day within a month: therefore, multi-column content descriptions on just year and month would overlap on the specific snapshot month. To specify this ambiguity, <= and >= predicates are used.

Fragment 1 (Historical):

```
EnterpriseModel.Period."Day" < VALUEOF("Snapshot Date") OR
EnterpriseModel.Period.MonthCode <= VALUEOF("Snapshot Year Month") OR
EnterpriseModel.Period."Year" < VALUEOF("Snapshot Year") OR
EnterpriseModel.Period."Year" = VALUEOF("Snapshot Year") AND
  EnterpriseModel.Period."Month in Year" <= VALUEOF("Snapshot Month") OR
EnterpriseModel.Period."Year" = VALUEOF("Snapshot Year") AND
  EnterpriseModel.Period."Monthname" IN ('Apr', 'Mar', 'Feb', 'Jan')
```

Fragment 2 (Current):

```
EnterpriseModel.Period."Day" >= VALUEOF("Snapshot Date") OR
EnterpriseModel.Period.MonthCode >= VALUEOF("Snapshot Year Month") OR
EnterpriseModel.Period."Year" > VALUEOF("Snapshot Year") OR
EnterpriseModel.Period."Year" = VALUEOF("Snapshot Year") AND
  EnterpriseModel.Period."Month in Year" >= VALUEOF("Snapshot Month") OR
EnterpriseModel.Period."Year" = VALUEOF("Snapshot Year") AND
  EnterpriseModel.Period."Monthname" IN ('Dec', 'Nov', 'Oct', 'Sep', 'Aug',
'Jul',
  'Jun', '', 'Apr')
```

Specify Unbalanced Parallel Content Descriptions

In an order entry application, time-based fragmentation between historical and current fragments is insufficient.

For example, records might still be volatile, even though they're historical records entered into the database before the snapshot date.

Assume, in the following example, that open orders can be directly updated by the application until the order is shipped or canceled. After the order has shipped, however, the only change that can be made to the order is to type a separate compensating return order transaction.

There are two parallel tracks in the following content descriptions. The first track uses the multicolumn, parallel track techniques described in the preceding section. Notice the parentheses nesting the parallel calendar descriptions within the Shipped-or-Canceled order status multicolumn content description.

The second parallel track is present only in the Current fragment and specifies that all Open records are in the Current fragment only.

Fragment 1 (Historical):

```
Marketing."Order Status"."Order Status" IN ('Shipped', 'Canceled') AND
  Marketing.Calendar."Calendar Date" <= VALUEOF("Snapshot Date") OR
Marketing.Calendar."Year" <= VALUEOF("Snapshot Year") OR
Marketing.Calendar."Year Month" <= VALUEOF("Snapshot Year Month")
```

Fragment 2 (Current):

```
Marketing."Order Status"."Order Status" IN ('Shipped', 'Canceled') AND
  Marketing.Calendar."Calendar Date" > VALUEOF("Snapshot Date") OR
Marketing.Calendar."Year" >= VALUEOF("Snapshot Year") OR
Marketing.Calendar."Year Month" >= VALUEOF("Snapshot Year Month") OR
Marketing."Order Status"."Order Status" = 'Open'
```

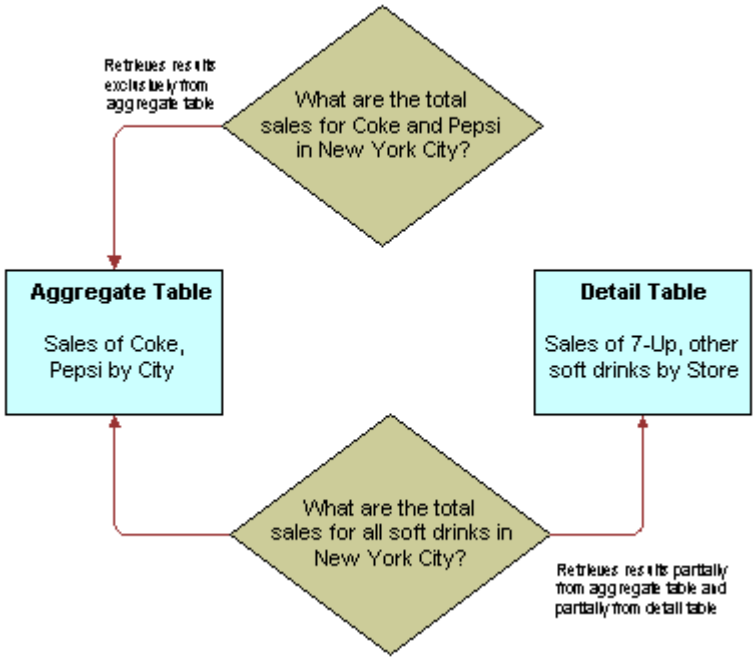
The overlapping Year and Month descriptions in the two fragments don' cause a problem, as overlap is permissible when there are parallel tracks. The rule is that at least one of the tracks has to be non-overlapping. The other tracks can have overlap.

Specify Fragmentation Content for Aggregate Table Fragments

Information at a given level of aggregation is sometimes stored in multiple physical tables.

Information at a given level of aggregation is sometimes stored in multiple physical tables. When individual sources at a given level contain information for a portion or fragment of the domain, the Oracle BI Server needs to know the content of the sources in order to pick the appropriate source for the query.

For example, suppose you've a database that tracks the sales of soft drinks in all stores. The detail level of data is at the store level. Aggregate information, as described in the image, is stored at the city level for the sales of Coke and Pepsi, but there is no aggregate information for the sales of 7-Up or any other of the sodas.



The goal of this type of configuration is to maximize the use of the aggregate table. If a query asks for sales figures for Coke and Pepsi, the data should be returned from the aggregate table. If a query asks for sales figures for all soft drinks, the aggregate table should be used for Coke and Pepsi and the detail data for the other brands.

The Oracle BI Server handles this type of partial aggregate navigation. To configure a repository to use aggregate fragments for queries whose domain spans multiple fragments, you need to define the entire domain for each level of aggregate data, even if you must configure an aggregate fragment as being based on a less summarized physical source.

This section contains the following topics:

- Specify the Aggregate Table Content
- Define a Physical Layer Table with a Select Statement to Complete the Domain
- Specify the SQL Virtual Table Content
- Creating Physical Joins for the Virtual Table

Specify the Aggregate Table Content

You configure the aggregate table navigation in the logical table source mappings.

In the soft drink example, the aggregate table contains data for Coke and Pepsi sales at the city level. Its Aggregate content specification, in the Content tab of the Logical Table Source window, is similar to the following:

Group by logical level:

```
GeographyDim. CityLevel, ProductDim.ProductLevel
```

Its Fragmentation content specification also in the Content tab of the Logical Table Source dialog is similar to the following:

```
SoftDrinks.Products.Product IN ('Coke', 'Pepsi')
```

This content specification tells the Oracle BI Server that the source table has data at the city and product level for two of the products. Additionally, because this source is a fragment of the data at this level, you must select **This source should be combined with other sources at this level**, in the Content tab of the Logical Table Source dialog, to indicate that the source combines with other sources at the same level.

Define a Physical Layer Table with a Select Statement to Complete the Domain

The data for the rest of the domain (the other types of sodas) is all stored at the store level.

To define the entire domain at the aggregate level, for example, city and product, you need to have a source that contains the rest of the domain at this level. Because the data at the store level is at a lower, more detailed level than at the city level, it's possible to calculate the city and product level detail from the store and product detail by adding up the product sales data of all of the stores in a city. You can use a query involving the store and product level table.

One way to do this is to define a table in the Physical layer with a Select statement that returns the store level calculations. To define the table, create a table in the Physical layer on the physical schema object that the `SELECT` statement uses for the query and selecting **New Physical Table**. Choose **Select** from the **Table Type** list, and type the SQL statement in the **Default Initialization String** box.

The SQL statement must define a virtual table that completes the domain at the level of the other aggregate tables. In this case, there is one existing aggregate table, and it contains data for Coke and Pepsi by city. Therefore, the SQL statement has to return all of the data at the city level, except for the Coke and Pepsi data.

Specify the SQL Virtual Table Content

Create a new logical table source for the Sales column that covers the remainder of the domain at the city and product level.

This source contains the virtual table created in the previous section. Map the Dollars logical column to the US Dollars physical column in this virtual table.

The Aggregate content specification, in the Content tab of the Logical Table Source dialog, for this source is:

Group by logical level:

```
GeographyDim.CityLevel, ProductDim.ProductLevel
```

This tells the Oracle BI Server this source has data at the city and product level.

The Fragmentation content specification might be:

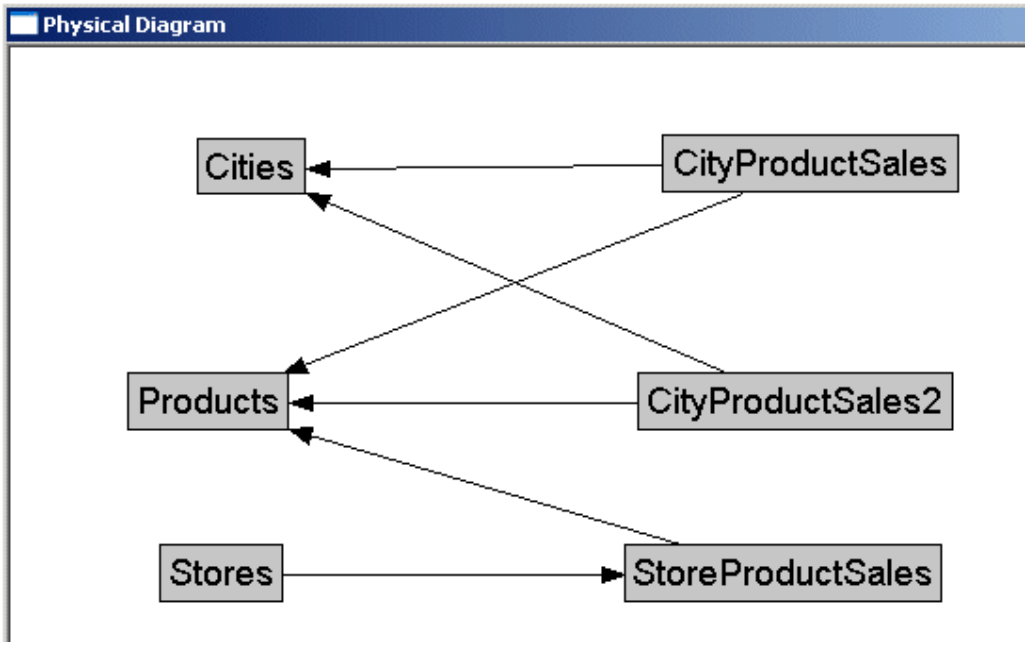
```
SoftDrinks.Products.Product = '7-Up'
```

Additionally, because it combines with the aggregate table containing the Coke and Pepsi data at the city and product level to complete the domain, you need to select the option in the Content tab of the Logical Table Source dialog indicating that the source is combined with other sources at the same level.

Create Physical Joins for the Virtual Table

Provides an image that shows how to construct physical joins.

Construct the correct physical joins for the virtual table. Notice that CityProductSales2 joins to the Cities and Products tables in the image below.



In this example, the two sources comprise the whole domain for soda sales. A domain can have many sources. The sources have to all follow the rule that each level must contain sources that, when combined, comprise the whole domain of values at that level. Setting up the entire domain for each level helps ensure that queries asking for Coke, Pepsi, and 7-Up don't leave out 7-Up. It also helps ensure that queries requesting information that has been precomputed and stored in aggregate tables can retrieve that information from the aggregate tables, even if the query requests other information that isn't stored in the aggregate tables.

12

Create and Maintain the Presentation Layer

You can learn how to use the Oracle BI Administration Tool to create, edit, and maintain objects in the Presentation layer of the Oracle BI repository.
This chapter contains the following topics:

- About the Presentation Layer
- Create and Customize the Presentation Layer
- Work with Subject Areas
- Work with Presentation Tables and Columns
- Work with Presentation Hierarchies and Levels
- Set Permissions for Presentation Layer Objects
- Create Aliases (Synonyms) for Presentation Layer Objects
- Control Presentation Object Visibility

About the Presentation Layer

You can provide customized, secure, role-based views of a business model to users in the Presentation layer.

Role-based views provide object security and also provide a way to hide some of the complexity of the business model.

In the Presentation layer, you can set an implicit fact column. The primary function of the Presentation layer is to provide custom names, dictionary entries, organization, and security for different groups of users.

Presentation layer views are called subject areas. In previous versions, subject areas were called presentation catalogs. You can create a subject area that's identical to your business model, or you can create role-based subject areas that show a single subject or that supports a specific business role. Subject areas aren't abstract views. You should create subject areas that organize your content in a way that benefits your users.

Subject areas in the Presentation layer appear as catalogs to client tools that use the Oracle BI Server as an ODBC data source. Subject areas contain presentation tables, columns, hierarchies, and levels.

Even though the Logical SQL requests from Answers and other clients query the presentation tables and columns, the logic for entities, relationships, and joins is in the Business Model and Mapping layer.

Create and Customize the Presentation Layer

After you've created the Business Model and Mapping layer, you can drag and drop entire business models to the Presentation layer in the Administration Tool to create subject areas.

You can create subject areas and other Presentation layer objects manually.

This section contains the following topics:

- About Creating Subject Areas
- About Removing Unneeded or Unwanted Columns
- Rename Presentation Columns to User-Friendly Names
- Export Logical Keys in the Subject Area
- Set an Implicit Fact Column in the Subject Area
- Maintain the Presentation Layer

About Creating Subject Areas

There are several ways to create subject areas in the Presentation layer.

The recommended method is to drag and drop a business model from the Business Model and Mapping layer to the Presentation layer, and then modify the Presentation layer based on what you want users to see. You can move columns between presentation tables, remove columns that don't need to be seen by the users, or even present all of the data in a single presentation table. You can create presentation tables to organize and categorize measures in a way that makes sense to your users.

You can also duplicate an existing subject area and its corresponding business model. See Duplicate a Business Model and Subject Area.

Although each subject area must be populated with contents from a single business model, you can create multiple subject areas for one business model. For very large business models, you may want to do this to help users work with the content. Users in Oracle BI Server can create queries that span multiple subject areas, as long as the subject areas correspond to the same business model.

There are many ways to create multiple subject areas from a single business model. One method is to drag a particular business model to the Presentation layer multiple times, then edit the properties or objects of the resulting subject areas as needed.

For example, if you've a business model called `ABC` that contains the `Geography` and `Products` dimensions, you can drag it to the Presentation layer twice. Two subject areas are created, with the default names `ABC` and `ABC#1`. You can then edit the subject areas as follows:

- Rename the `ABC` subject area to `DEF`, then delete the `Geography` presentation hierarchy
- Rename the `ABC#1` subject area to `XYZ`, then delete the `Products` presentation hierarchy

Users in Oracle BI Server can run queries that span the `DEF` subject area containing the `Products` hierarchy, and the `XYZ` subject area containing the `Geography` hierarchy.

When you query a single subject area, all the columns exposed in that subject area are compatible with all the dimensions exposed in the same subject area. However, when you combine columns and dimensions from multiple subject areas, you must ensure that you don't include combinations of columns and dimensions that are incompatible with one another.

For example, a column in one subject area might not be dimensioned by Project. If columns from the Project dimension from another subject area are added to the request along with columns that aren't dimensioned by Project, then the query might

fail to return results, or cause the Oracle BI Server error, "`No fact table exists at the requested level of detail: XXXX.`"

Automatically Create Subject Areas Based on Logical Stars and Snowflakes

You can automatically create one subject area for each logical star or logical snowflake in your business model.

Logical stars and logical snowflakes are both composed of a centralized fact table connected to multiple dimension tables. This feature provides another way to create multiple subject areas from a single business model.

To create a subject area for each fact table that's part of a logical star or snowflake, right-click the business model and select **Create Subject Areas for Logical Stars and Snowflakes**. The new subject areas are automatically created, each containing a fact table and only the dimension tables with which it's associated. This option is available for any business model that contains logical stars or logical snowflakes.

About Removing Columns

One important reason to use a custom Presentation layer is to make the schema as easy to use and understand as possible.

Customize the presentation layer prevent users from seeing columns that provide no meaningful content.

The following columns are examples of columns that you might want to remove from the Presentation layer:

- Key columns that have no business meaning.
- Columns that users don't need to view, for example, codes, when text descriptions exist.
- Columns that users aren't authorized to read.

Rename Presentation Columns to User-Friendly Names

You should try to keep presentation column names and their source logical column names synchronized to reduce maintenance.

By default, presentation columns have the same name as the corresponding logical column in the Business Model and Mapping layer.

To synchronize presentation column names and their source logical column names, select **Use Logical Column Name** in the Presentation Column dialog.

In some cases, however, you may want a different presentation column name to be shown to users. To do this, change the name of the presentation column in the Presentation Column dialog.

When you change the name of a presentation column, an alias is automatically created for the old name, so compatibility to the old name remains. See Create Aliases (Synonyms) for Presentation Layer Objects.

You can't rename a Presentation layer object to a name that's already in use as an alias for an object of the same type.

Export Logical Keys in the Subject Area

For each subject area in the Presentation layer, you can decide whether to export any logical keys as key columns to tools that access it.

Exporting logical keys is irrelevant to users of Oracle BI Presentation Services, but it may be advantageous for some query and reporting tools.

If you decide to export logical keys, make sure that the logical key columns exist in the table folders. In this situation, your business model should use logical key/foreign key joins.

When you select the option **Export logical keys** in the Subject Area dialog, any columns in the Presentation layer that are key columns in the Business Model and Mapping layer are listed as key columns to any ODBC client. This is the default selection. In most situations, this option should be selected.

If you're using a tool that issues parameterized SQL queries, such as Microsoft Access, don't select the option **Export logical keys**. This stops the tool from issuing parameterized queries.

Set an Implicit Fact Column in the Subject Area

For each subject area in the Presentation layer, you can set an implicit fact column.

The implicit fact column is added to a query when it contains columns from two or more dimension tables and no measures.

The column isn't visible in the results. It's used to specify a default join path between dimension tables when there are several possible alternatives or contexts.

If an implicit fact isn't configured, Oracle BI Server uses any fact Logical Table Source (LTS) to answer dimension-only subrequest that contains multiple dimensional references but no fact reference.

The Oracle BI Server can also use any fact LTS, if the configured implicit fact column isn't relevant to the dimensions that are joined. This could happen, for example, when implicit fact column is a level based measure at a level higher than the dimensional only subrequest.

Maintain the Presentation Layer

There is no automatic way to synchronize all changes between the Business Model and Mapping layer and the Presentation layer.

For example, if you add logical columns to an existing logical table, or edit existing columns, you must manually update the corresponding Presentation layer objects.

However, the Administration Tool can automatically synchronize the name of presentation columns with their corresponding logical column names. To take advantage of this feature, ensure that **Use Logical Column Name** is selected in the Presentation Column dialog.

In some cases, if there are many changes to a logical table or even to an entire business model, it's easiest to delete the corresponding presentation table or subject area, and then and drag and drop the updated logical objects to the Presentation layer.

For this reason, it's best to wait until the Business Model and Mapping layer is relatively stable before adding customizations in the Presentation layer.

Work with Subject Areas

In the Presentation layer, subject areas enable you to show different views of a business model to different sets of users.

Populate subject areas using the contents from a single business model. Subject areas can't span business models.

Subject areas are created automatically by dragging and dropping business models from the logical layer.

Aliases are created automatically whenever presentation objects are renamed, so that any queries using the original name don't break.

- See Set Permissions for Presentation Layer Objects.
- See Localize Oracle Analytics Server in *Administering Oracle Analytics Server*.
- See Control Presentation Object Visibility and Create Aliases (Synonyms) for Presentation Layer Objects.

1. In the Presentation layer, double-click a subject area.
2. In the General tab, you can change the name for the subject area.
3. To set permissions for this subject area, click **Permissions**.
4. Select **Custom display name** to dynamically display a name based on a session variable and to edit the **Translation Key** field. Select **Custom description** to dynamically display a custom description based on a session variable.

   These options are used typically for localization purposes. When you externalize strings in the Presentation layer and run the Externalize Strings utility, the results contain the session variable information and the translation key.
5. The **Business model** list displays the business model for this subject area.
6. Select the option **Export logical keys** to expose the logical keys to other applications.

   In most situations, this option should be selected. Many client tools differentiate between key and non-key columns, and the option **Export logical keys** provides client tools access to the key column metadata. Any join conditions the client tool adds to the query, however, are ignored, because the Oracle BI Server uses the joins defined in the repository.

   If you're using a tool that issues parameterized SQL queries, such as Microsoft Access, don't select the **Export logical keys** option. Not exporting logical keys stops the tool from issuing parameterized queries.
7. Optional: Set an **Implicit Fact Column**.

   This column is added to a query when it contains columns from two or more dimension tables and no measures. The column isn't visible in the results. It's used to specify a default join path between dimension tables when there are several possible alternatives or contexts.
8. Optional: Specify an expression in the **Hide object if** field that controls whether this subject area is visible in the Subject Area Tree in Answers and BI Composer. Leave this field blank (the default) to show the object.

9. Optional: Type a description. This description appears in a mouse-over tool tip for the subject area in Answers .
10. In the Presentation Tables tab, you can add, remove, edit, or reorder the presentation tables for this subject area.
11. Use the Aliases tab to specify or delete aliases for this subject area.
12. Click **OK**.

Work with Presentation Tables and Columns

Learn how to customize presentation tables and columns in these topics.

Presentation tables and presentation columns appear as folders and columns in Answers . You can customize presentation tables and presentation columns to help users craft queries based on their business needs.

This section contains the following topics:

- Create and Manage Presentation Tables
- Reorder Presentation Layer Tables
- About Presentation Columns
- Nest Folders in Answers and BI Composer

Create and Manage Presentation Tables

You can use presentation tables to organize columns into categories that make sense to the user community.

A presentation table can contain columns from one or more logical tables. The names and object properties of the presentation tables are independent of the logical table properties. Presentation tables are created automatically by dragging and dropping logical tables from the logical layer. A presentation table can't have the same name as its parent subject area. For example, you can't have a subject area called Customer that has a Customer table within it.

Aliases are created automatically whenever presentation objects are renamed, so that any queries using the original name don't break. Use the Aliases tab to specify or delete aliases for this presentation table. See Create Aliases (Synonyms) for Presentation Layer Objects.

For localization when creating presentation tables:

- Use the **Custom display name** to dynamically display a name based on a session variable.
- Use **Custom description** to dynamically display a custom description based on a session variable.
- Use the **Translation Key** along with the custom display name to localize the user interface.

When you externalize strings in the Presentation layer and run the Externalize Strings utility, the results contain the session variable information and the translation key. See Localize Oracle Analytics Server in *Administering Oracle Analytics Server*.

Use the Child Presentation Tables tab to specify presentation tables that you want to show as nested folders in Answers and BI Composer. See Nest Folders in Answers and BI Composer and Control Presentation Object Visibility.

See Set Permissions for Presentation Layer Objects.

1. In the Presentation layer, from a subject area, double-click a presentation table to open the Properties dialog.
2. Optional: In the Properties dialog for the presentation table, in the General tab, change the name for the presentation table.
3. Leave the **Hide object if** field blank.
4. Optional: In the Columns tab, add, remove, edit, or reorder the presentation columns for the selected presentation table.
5. In the Hierarchies tab, add, remove, edit, or reorder the presentation hierarchies for the selected presentation table.
6. When you've completed your changes, click **OK**.

Reorder Presentation Layer Tables

Use this task to reorder a table or sort all tables in a subject area.

1. In the Presentation layer, double-click a subject area.
2. In the Subject Area dialog, click the Presentation Tables tab.
3. In the **Name** list, select the table and use drag-and-drop to reposition the table, or click the **Up** and **Down** buttons to move a table.
4. Click the **Name** column heading to sort all tables in alphanumeric order.

   The sort operation changes the order between ascending (A to Z) and descending (Z to A) alphanumeric order.

About Presentation Columns

Presentation columns provide analytics data for display in web clients.

You can create presentation columns by dragging and dropping logical columns from the Oracle BI Administration Tool Business Model and Mapping (logical) layer to the Presentation layer. New columns added to presentation tables can't use the same name or alias same name of an existing column.

You can drag and drop a column from a single logical table in the Business Model and Mapping layer onto multiple presentation tables. For example, you can create several presentation tables that contain different classes of measures such as one containing volume measures, one containing share measures, and one containing measures from a year ago.

You must enable the **Edit presentation names** in the Administration Tool option before you can edit the presentation column's name. See Set Administration Tool Options and Set Permissions for Presentation Layer Objects.

You can use the **Custom display name**, **Custom description**, and **Translation Key** fields for localization purposes. When you externalize strings in the Presentation layer and run the Externalize Strings utility, the results contain the session variable information and the translation key. See Localize Oracle Analytics Server in *Administering Oracle Analytics Server*.

You can use the **Custom display name** and **Custom description** fields to propagate UI hints (labels and tooltips) from an ADF data source to display in Answers . See Propagate Labels and Tooltips from ADF Data Sources.

Aliases are created automatically whenever presentation objects are renamed, so that any queries using the original name don't break.

Change the Presentation Column Name

The presentation column names, by default, are identical to the logical column names in the Business Model and Mapping layer. You can change the presentation column name.

1. Open the Oracle BI Administration Tool.
2. In the Presentation layer, double-click a presentation column to change.
3. In the Presentation Column dialog, remove the check from the **Use Logical Column Name** field.
4. Click **Edit** next to the **Logical Column Name** field.
5. In the Logical Column dialog, in the **Name** field, type a new name to use for the Presentation Column.
6. In the Logical Column dialog, implement other changes, and click **OK**.
7. In the Presentation Column dialog, click **Custom display name**, and then click **OK**.

Reorder Presentation Columns

You can change the order of the columns in your presentation.

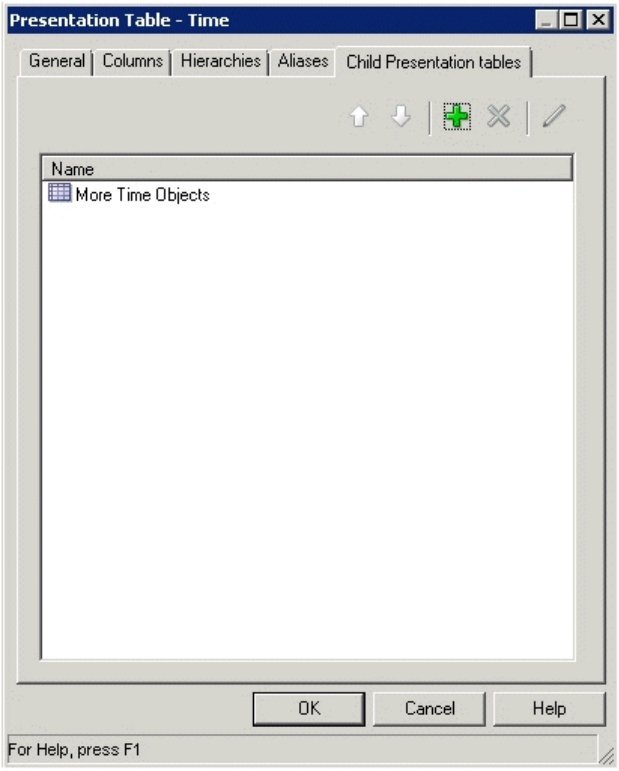
1. Open the Oracle BI Administration Tool
2. In the Presentation layer, right-click a presentation table and select **Properties**.
3. Click the **Columns** tab.
4. Select the column you want to reorder.
5. Use drag-and-drop to reposition the column, or click the **Up** and **Down** buttons.
6. Click **OK**.

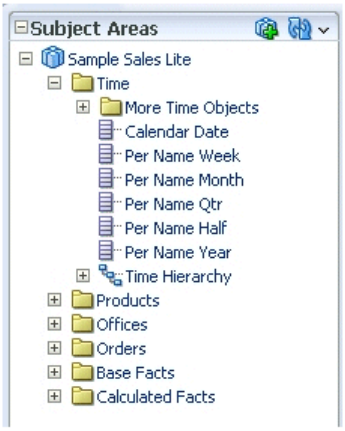
Nest Folders in Answers and BI Composer

You can designate child presentation tables using the Child Presentation Tables tab in the Presentation Table dialog.

Designate child presentation tables to give the appearance of nested folders in Answers and BI Composer. You can add multiple layers of nesting using this method.

The image shows how a designated child presentation table appears nested in Answers.





The folders only appear nested, but they aren't actually nested in drill-down, and the qualified names of the objects remain the same. The Presentation layer in the Administration Tool doesn't display the nesting; the nesting only appears in Answers and BI Composer. This feature only works for presentation tables, and not for other Presentation layer objects.

When you run a consistency check, the Consistency Check Manager detects any circularity in parent-child presentation table assignment. It also detects and reports project definitions that include child presentation tables without parent presentation tables.

If you previously used hyphens at the beginning of presentation table names or arrows at the beginning of presentation table descriptions to achieve nesting, you should run the Convert Presentation Folders utility to convert your metadata to the new structure. See Use the Convert Presentation Folders Utility.

Work with Presentation Hierarchies and Levels

Presentation hierarchies and presentation levels provide an explicit way to expose the multidimensional model in Answers .

When presentation hierarchies and levels are defined in the Presentation layer, roll-up information is displayed in the Answers navigation pane, providing users with important contextual information.

Members in a presentation hierarchy aren't visible in the Presentation layer. You can see hierarchy members in Answers .

Users can create hierarchy-based queries using objects in presentation hierarchies and levels. Presentation hierarchies expose analytic functionality such as member selection, custom member groups, and asymmetric queries.

You can also provide localization information and apply fine-grained access control to presentation hierarchies and levels.

If you've a repository from a previous release, the presentation hierarchies don't appear in the Presentation layer automatically as part of the Oracle BI repository upgrade process. You must manually create the presentation hierarchies and levels objects by dragging logical dimensions from the Business Model and Mapping layer to the appropriate presentation tables.

This section contains the following topics:

- Create and Manage Presentation Hierarchies
- Create and Manage Presentation Levels

Create and Manage Presentation Hierarchies

To create a presentation hierarchy, you can drag a logical dimension hierarchy from the Business Model and Mapping layer to a table in the Presentation layer.

The presentation hierarchy object must be located within a presentation table, unlike in the Business Model and Mapping layer, where logical dimensions are peer objects of tables. Presentation hierarchies are also displayed within their associated tables in Answers , providing a conceptually simpler model.

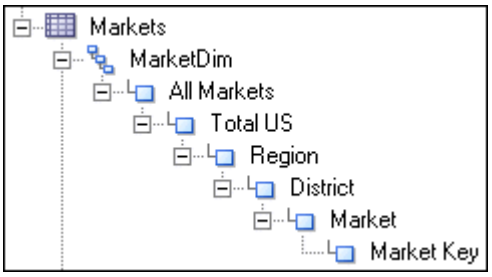
If a logical dimension spans multiple logical tables in the Business Model and Mapping layer, it's a best practice to model the separate logical tables as a single presentation table in the Presentation layer.

There are several ways to create presentation hierarchies:

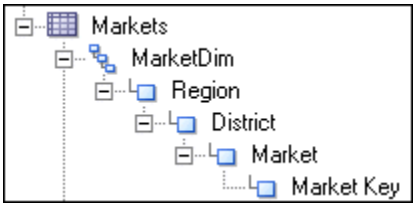
- When you drag an entire business model to the Presentation layer, the presentation hierarchies and constituent levels appear automatically, along with other presentation objects.
- When you drag a logical dimension table to the Presentation layer, presentation hierarchies and levels based on those dimensions are created automatically.
- You can also drag individual logical dimensions to the appropriate presentation tables to create corresponding presentation hierarchies within those tables.
- As with most other objects in the repository, you can right-click a presentation table, select **New Object**, and then select **Presentation Hierarchy** to manually define the object.

You can also drag an individual logical level from the Business Model and Mapping layer to a presentation table to create a presentation hierarchy that's a subset of the logical dimension hierarchy.

For example, suppose a logical dimension has the levels All Markets, Total US, Region, District, Market, and Market Key. Dragging and dropping the entire logical dimension to the corresponding presentation table appears as follows:



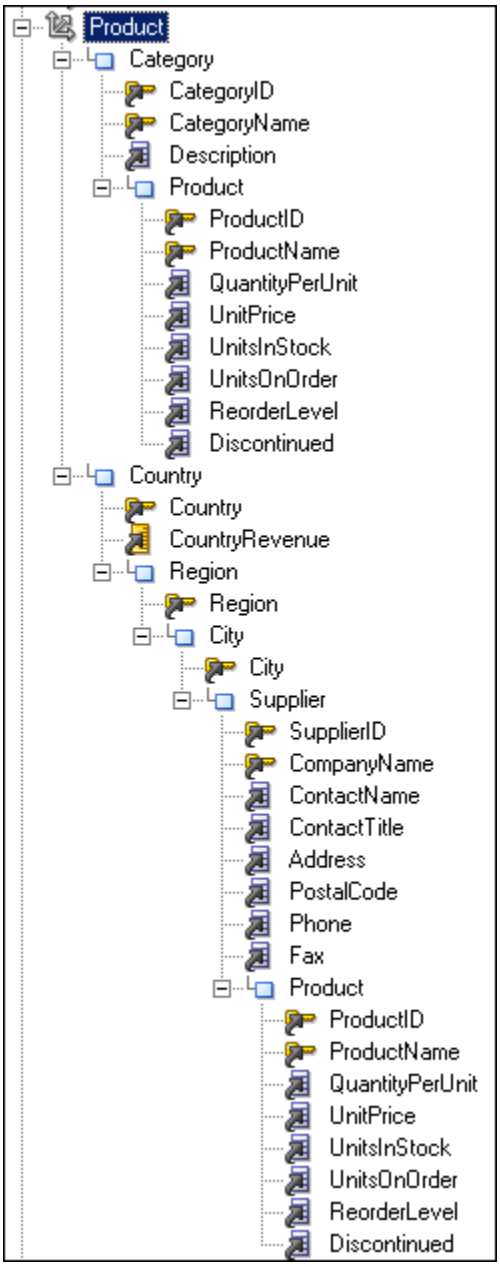
However, dragging and dropping the Region level to the same presentation table appears as follows:



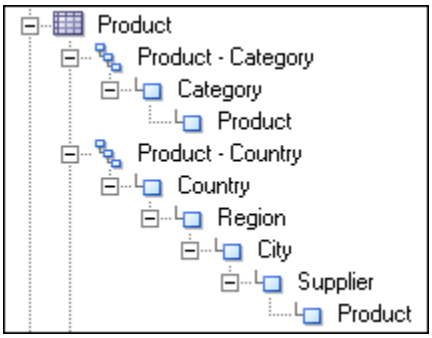
Model Dimensions with Multiple Hierarchies in the Presentation Layer

For logical dimensions that contain multiple logical hierarchies, multiple separate presentation hierarchies are created.

For example, the following logical dimension called Product contains the two hierarchies Category and Country:



In the Business Model and Mapping layer, this logical dimension is modeled as a single dimension object that contains multiple hierarchies. In contrast, the Presentation layer models this dimension as two separate objects: one that displays the drill path through the Category level, and another that shows the drill path through the Country level, as follows:



Edit Presentation Hierarchy Objects

Learn how to edit presentation hierarchy properties.

You can edit presentation hierarchy properties, including setting permissions to apply role-based access control, setting a custom display name for localization purposes, and changing the levels in a hierarchy.

The Display Columns tab is only available for parent-child hierarchies. Because parent-child hierarchies don't contain levels, display columns are defined for the presentation hierarchy object as a whole. Use the Display Columns tab to define which columns should be used for display for this parent-child hierarchy.

You can add, delete, or reorder display columns. You can also click the **Edit** button to edit properties for a particular column.

The Levels tab lists the levels within the hierarchy and their order. This tab isn't available for parent-child hierarchies. You can add, delete, or reorder levels. You can also click the **Edit** button to edit properties for a particular level. See Create and Manage Presentation Levels.

See Set Permissions for Presentation Layer Objects.

Use the Aliases tab to specify or delete aliases for this presentation hierarchy. See Create Aliases (Synonyms) for Presentation Layer Objects.

See Control Presentation Object Visibility and Localize Oracle Analytics Server in *Administering Oracle Analytics Server*.

1. In the Presentation layer, double-click a presentation hierarchy to display the Presentation Hierarchy dialog.
2. In the General tab, you can change the following:
   - **Name.** Aliases are created automatically whenever presentation objects are renamed, so that any queries using the original name don't break.

     You must enable the **Edit presentation names** Administration Tool option before you can edit the presentation hierarchy's name.
   - **Permissions.**
   - **Custom display name** and **Custom description.** Select **Custom display name** to dynamically display a name based on a session variable and to edit the **Translation Key** field. Select **Custom description** to dynamically display a custom description based on a session variable.

These options are used typically for localization purposes. When you externalize strings in the Presentation layer and run the Externalize Strings utility, the results contain the session variable information and the translation key.

- **Logical Dimension.** This field displays the name of the logical dimension for this presentation hierarchy. Click **Browse** to select a different logical dimension.
- **Hide object if.** This field lets you specify an expression that controls whether this presentation hierarchy is visible in the Subject Area Tree in Answers and BI Composer. Leave this field blank (the default) to show the object.

Create and Manage Presentation Levels

Presentation levels are typically created automatically when presentation hierarchies are created.

Presentation levels are displayed within hierarchical columns in Answers .

See the following:

- Set Permissions for Presentation Layer Objects
- Localize Oracle Analytics Server in *Administering Oracle Analytics Server*.
- Create Aliases (Synonyms) for Presentation Layer Objects

You must enable the **Edit presentation names** Administration Tool option before you can edit a presentation level's name.

Aliases are created automatically whenever presentation objects are renamed, so that any queries using the original name don't break.

The Logical Level field displays the name of the logical level for this presentation level.

The **Custom display name** and the **Custom description** options are used typically for localization purposes. When you externalize strings in the Presentation layer and run the Externalize Strings utility, the results contain the session variable information and translation key.

Specifying an expression in the **Hide object if** field has no effect on the visibility of presentation levels in the Subject Area Tree in Answers and BI Composer.

The Drill To Levels and Drill From Levels tabs, and the **Generate Drill Graph** aren't currently used.

Use the Display Columns tab to define columns used for display for that level on drill-down. For example, if two columns called *Name* and *ID* at the same level, you can choose to display *Name* because it's the more user-friendly option. The display columns that appear by default when a presentation level is created are based on which key columns for the corresponding logical level have the **Use for display** option selected.

You can add, delete, or reorder display columns. You can also click the **Edit** button to edit properties for a particular column.

1. In the Presentation layer, double-click a presentation level to display the Presentation level dialog.
2. In the General tab, type a **Name** for the presentation level.

3. Click **Permissions** to specify access to the presentation level by application roles and users.
4. Optional: Select **Custom display name** to dynamically display a name based on a session variable and to edit the **Translation Key** field.
5. Optional: Select **Custom description** to dynamically display a custom description based on a session variable.
6. Optional: Click **Browse** to select a different **Logical Level**.
7. Drag a presentation column to the presentation level in the Presentation layer to automatically add the column as a display column for the presentation level.
8. In the Aliases tab to specify or delete aliases for a presentation level.

Set Permissions for Presentation Layer Objects

You can apply access control to restrict which individual users or application roles (groups) can access particular presentation layer objects.

For example, you can provide read-only access to a set of presentation tables for a particular application role, read-write access for a second application role, and no access for a third application role.

You can also use the Identity Manager to set up privileges and permissions. The Identity Manager is useful for setting permissions for individual application roles to many objects at once, unlike permissions in the Presentation layer, which you can only set for one object at a time. See Set Up Object Permissions and Apply Data Access Security to Repository Objects.

You can control what level of privilege is granted by default to the `AuthenticatedUser` application role, which is the default application role associated with new repository objects. To do this, set the `DEFAULT_PRIVILEGES` parameter in the `NQSConfig.INI` file.

To set permissions for presentation layer objects:

1. In the Presentation layer, double-click a presentation object, such as a subject area, table, column, or hierarchy.
2. In the General tab, click **Permissions**.
3. In the Permissions dialog, any users or application roles with the **Default** permission don't appear in the User/Application Roles list. Select **Show all users/application roles** to see users and application roles with the Default permission.

   In online mode only, by default, no users are retrieved, even when **Show all users/application** roles is selected. Click **Set online user filter** to specify the set of users you want to retrieve.

   The filter is empty by default, which means that no users are retrieved. Enter * to retrieve all users, or enter a combination of characters for a specific set of users, such as A* to retrieve all users whose names begin with the letter A. The filter isn't case-sensitive.
4. For each user and application role, you can allow or disallow access privileges for this presentation object by selecting one of the following options:
   - **Read.** Only allows read access to this object.
   - **Read/Write.** Provides both read and write access to this object.
   - **No Access.** Explicitly denies all access to this object.

- **Default.** The permission is inherited from the parent object. For subject areas, because they're a top-level object, Default is equivalent to the permission granted to the AuthenticatedUser application role.

5. Click **OK**.
6. Click **OK** in the Properties dialog for this presentation object.

Generate a Permission Report for Presentation Layer Objects

You can generate a permission report for individual presentation layer objects to see a summary of how permissions have been applied for that object.

The Permission Report displays the name and a description of the presentation object, along with a list of users/application roles and their permissions.

1. In the Oracle BI Administration Tool, open a repository in online or offline mode.
2. In the Presentation layer, right-click an object and select **Permission Report**.

Sort Columns in the Permissions Dialog

There are six ways that you can sort the types and User/Application Role names in the Permissions dialog.

To change the sort, click the heading of the first or second column. The first column has no heading and contains an icon that represents the type of user or application role. The second column contains the name of the User/Application Role object.

You can't sort on the columns for individual object permissions such as Read, and Read/Write.

There are three ways to sort by type, and two ways to sort the list of user and application role names. This results in a total of six possible sort results (3 x 2 = 6). The following list shows the sort results available by clicking the type column:

- AuthenticatedUser, Application Roles, Users, ascending by name of type
- Users, Application Roles, AuthenticatedUser, descending by name of type
- Type column is in no particular order. The Type value is ignored, as all names in User/Application Role column are sorted in ascending order by value in User/Application Role column.

The following list shows the sort results available by clicking the User/Application Role column:

- Ascending within the type
- Descending within the type

Create Aliases (Synonyms) for Presentation Layer Objects

Each presentation object can have a list of aliases (synonyms) for its name that you can use in Logical SQL queries.

Use the Alias tab in the Properties dialog for the appropriate presentation object such as subject area, presentation table, presentation hierarchy, presentation level, or presentation column to create the list of aliases.

Because Presentation layer objects are often deleted and then re-created during the repository development process, it's best to wait until your logical business model is relatively stable before creating aliases for presentation objects.

You can use this feature to rename presentation objects without breaking references that any existing requests have to the old names, including requests from Answers, Publisher, or other Logical SQL clients. If you're still developing a new repository, you might want to wait until the repository is stable before renaming objects.

For example, consider a subject area called *Sample Sales Reduced* that contains a presentation table called *Facts Other*. If you rename the presentation column called *# of Customers* to *Number of Customers*, any requests that use *# of Customers* fail. However, if you add *# of Customers* to the list of synonyms in the Alias tab for the *Number of Customers* column, then queries containing both *# of Customers* and *Number of Customers* succeed and return the same results.

- Aliases for presentation objects don' appear in Answers or other query clients when creating new queries. Only the primary names of subject areas, hierarchies, levels, tables, and columns appear.
- This feature works in a different way from SQL aliases or the alias feature in the Physical layer. It simply provides synonyms for object names, much like synonyms in SQL.
- Aliases are created automatically when you rename presentation objects. For example, if you change Catalog to Catalog1, the original name Catalog is added to the Aliases list.
- You can't rename a Presentation layer object to a name that's already in use as an alias for an object of the same type.

1. In the Oracle BI Administration Tool, in the Presentation layer, double-click a presentation object such as a subject area, table, column, or hierarchy.
2. In the Properties dialog for the presentation object, click the Aliases tab.
3. Click the **New** button to create an alias, and then type the text string to use for the alias.
4. Click **OK**.

Control Presentation Object Visibility

You can use the Hide object if field to hide selected Presentation layer objects in the Subject Area Tree in Answers and BI Composer. You can hide subject areas, tables, columns, and hierarchies.

Although the **Hide object if** field is shown for presentation levels, it's a placeholder for a future release and currently has no effect on presentation level objects.

The **Hide object if** field only controls object visibility and doesn't affect object access. For example, you can query objects that are hidden using tools like `nqcmd`.
There are three different types of expressions that you can use in the **Hide object if** field to determine Presentation layer object visibility:

- **Constant**. Use any non-zero constant in the field to hide the object. Use zero (0) or leave the field blank to display the object.
- **Session variable**. You can use a session variable in the expression to control whether the object is hidden. If the expression evaluates to a non-zero value, the object is hidden. If the expression evaluates to zero, is empty, or has no value definition, the object is displayed. The session variable must be populated using a session initialization block or

a row-wise initialization block. You must properly populate the session variable to control the visibility.

The SQL for the `init` block can use `CASE` statements to control whether to return zero or a non-zero number in the session variable, for example:

```
VALUEOF(NQ_SESSION."VISIBLE")
```

Session variable names that include periods must be enclosed in double quotes.

- **Session variable comparison**. You can use an equality or inequality comparison to control whether the object is hidden, using the following form:

  ```
  'session_variable_expression' '=|<>' 'constant'
  ```

  If the expression evaluates to zero, null, or empty, the object is displayed. If the expression evaluates to a non-zero value, the object is hidden, for example:

  ```
  NQ_SESSION."VISIBLE" = 'ABC'
  NQ_SESSION."VISIBLE" <> 'ABC'
  ```

  You must enclose session variable names that include periods in double quotes.

  You can use any scalar function supported by Oracle Analytics Server in the `Hide object if` expression. Scalar functions include any function that accepts a simple value for each of its arguments and returns a single value. You can use the functions listed, except for functions that return non-deterministic results like `RAND`, `NOW`, `CURRENT_DATE`, `CURRENT_TIMESTAMP`, and `CURRENT_TIME`.

  - All String Functions.
  - Math Functions, except RAND.
  - Calendar Date/Time Functions, except NOW, and CURRENT_DATE.
  - Conversion Functions such as CAST, IFNULL, TO_DATETIME, and VALUEOF.

  For example, the following expression checks to see if the *NQ_SESSION.VISIBLE* session variable begins with the letters *ABC*:

  ```
  LEFT(VALUEOF(NQ_SESSION."VISIBLE"), 3) = 'ABC'
  ```

  The following expression checks to see if the given variable begins with *ExtnAttribute*:

  ```
  Left(VALUEOF(NQ_SESSION."ADF_LABEL_ORACLE.APPS.CRM.MODEL.ANALYTICS.
  APPLICATIONMODULE.CRMANALYTICSAM_CRMANALYTICSAMLOCAL_CRMANALYTICSAM.
  OPPORTUNITYAM.OPPORTUNITY_EXTNATTRIBUTECHAR001"), 13) = 'ExtnAttribute'
  ```

Run the Consistency Check Manager to detect any inconsistencies in the visibility filter expression.

13

Create and Persist Aggregates for Oracle BI Server Queries

Learn how to set up and use aggregate persistence in Oracle Analytics Server.
Most data warehouse practitioners create aggregated data tables to improve the performance of highly summarized queries. The aggregate tables store precomputed results that are combined measures, usually summed, over a set of dimensional attributes. Use aggregate tables to improve query response times in decision support systems.

If you write SQL queries or use a tool that only understands what physical tables exist and not their meaning, then using aggregate tables becomes more complex as the number of aggregate tables increases. The Oracle BI Server's aggregate navigation capability enabled queries to use the information stored in aggregate tables automatically. The Oracle BI Server lets you concentrate on asking the right business question, and then the server decides which tables provide the fastest answers.

Oracle Analytics Server takes advantage of the aggregates in source databases. See Manage Logical Table Sources (Mappings). The aggregate persistence automates the creation and loading of the aggregate tables and their corresponding metadata mappings to minimize the time required to create and maintain the data aggregation, as well as load database scripts and the corresponding metadata mappings.

This chapter contains the following topics:

- About Aggregate Persistence
- Aggregate Persistence Improvements
- About Aggregate Persistence Errors
- Identify Query Candidates for Aggregation
- Use Oracle BI Summary Advisor to Identify Query Candidates for Aggregation
- Use the Aggregate Persistence Wizard to Generate the Aggregate Specification
- Use Model Check Manager to Check for Modeling Problems
- Write the Create Aggregates Specification
- Run the Aggregate Specification Script
- Life Cycle Use Cases for Aggregate Persistence
- Use Double Buffering to Refresh Highly Available Aggregates
- Create Aggregates on TimesTen Sources

About Aggregate Persistence

Use the Aggregate Persistence feature to create aggregates for Oracle BI Server queries.

The Aggregate Persistence Wizard lets you automate the creation of the aggregate specification script. When you run this script against a live Oracle BI Server, aggregate tables are created by the aggregate persistence engine and are mapped into the metadata for

navigation. When aggregates are persisted, indexes and statistics are created on relational tables for greater performance.

The Aggregate Persistence Wizard creates a SQL script that you can run on a scheduled basis against the Oracle BI Server. In the Aggregate Persistence Wizard, you specify the measures, dimensionality, and other parameters of each star or cube based on your performance design. The script should run after each load of the base-level tables, so that the aggregates are always synchronized with the detail-level data when the load window completes and users begin to run queries.

Aggregate creation runs against the primary server in a cluster. It takes some time for the metadata changes to propagate to the secondary servers. The cluster refresh time is a user controlled option and you could get incorrect results if a query hits a dependent child server before it's refreshed. It's the administrator's responsibility to set an appropriate cluster refresh interval.

Aggregate persistence requires a dedicated connection pool to create tables or cubes in the target database that holds the aggregates. Because the Oracle BI Repository enables federation, the aggregated target can use the same database as the detailed source, or in a completely different database. You must create the dedicated connection pool before you run the Aggregate Persistence Wizard, so the correct connection pool is selected during the appropriate step of the wizard.

The default prefix `SA_` is automatically added to dimension (level) aggregates. You can change this default prefix by updating the `AGGREGATE_PREFIX` parameter in the `AGGREGATE_PERSISTENCE` section of the NQSConfig.INI file:

```
AGGREGATE_PREFIX = "prefix_name" ;
```

You must appropriately secured and restrict access to the target schema used to store aggregates. The schema should have privileges to connect, create, and drop tables and indexes. By default, only users who have administrator privileges can manage aggregates.

Don't use aggregate persistence against tables with active Virtual Private Database (VPD) security filters. There's a possibility that the aggregate information could persist without the VPD filter, posing a security risk.

Aggregate Persistence Improvements

Oracle Analytics Server automatically creates more usable aggregates and creates aggregates without the need to fix data set errors or modeling problems.

Surrogate Keys

Aggregate persistence can create surrogate keys for joining dimensions to fact aggregate tables.

In most cases the source and the target databases aren't the same instance.

The Oracle BI Server uses the hash join method to improve surrogate key creation. Where possible, a new request variable is automatically added to the fact aggregate population query and when this request variable is set, the query engine builds hash joins for the dimension tables in parallel before joining to the fact table.

The Oracle BI Summary wizard displays the **Use surrogate keys** option to suggests when you should use surrogate keys. When this option is selected, the `using_surrogate_key` clause is added to all levels in the aggregate specification.

Auto Correction (Hardening) of Level Keys

Aggregate persistence auto-corrects or hardens level keys that aren't unique.

The Oracle BI Summary Advisor recommends aggregates with level keys that are unique as defined, or with level keys that are auto-corrected (hardened) to make unique keys. Modifications to underlying data might impact such aggregates.

To improve performance, Oracle suggests creating aggregates using surrogate key rather than natural keys. Auto-correction, or hardening, isn't as effective when natural keys are used, especially in the prepare-create mode of operation.

Unbalanced (Ragged) and Skip-Level Hierarchies

Aggregate persistence creates aggregates for logical dimensions with unbalanced or skip-level hierarchies. You can create aggregates with or without using surrogate keys. The Oracle BI Summary Advisor recommends aggregates that contain logical dimensions with unbalanced and skip-level hierarchies.

Chronological Keys

The Oracle BI Server requires chronological keys to support time series functions such as `AGO`, `TODATE`, and `PERIODROLLING`.

Time series functions operate correctly when only the lowest key in the logical dimension is chronological.

Aggregate persistence generates chronological keys with the `CK_` prefix for time levels without chronological keys. A new column is added to the physical dimension aggregate table to store the chronological key value, and a new logical column is added to the logical table of the time dimension. The column is mapped to the new column added to the physical dimension aggregate table.

The `delete aggregates` statement automatically removes all metadata created to support generated chronological keys.

Count Distinct Measures

The Oracle BI Server uses aggregates with count distinct measures to serve queries for these measures at higher grains.

The Aggregate Persistence wizard includes the **Persist Count Distinct Measures as raw values** option, when selected appends `as_raw_values` to all the valid count distinct measures specified. When the **Persist Count Distinct Measures as raw values** option is selected, aggregate persistence sets an aggregation expression override on the corresponding logical column for the system-generated aggregate logical table source. The Oracle BI Summary Advisor recommends both methods of persistence for count distinct measures.

About Aggregate Persistence Errors

Occurrences such as a network failure, no disk space on the database, or a bad aggregate request result in aggregate persistence errors.

When a series of aggregates are being created, and the creation of one aggregate fails, the aggregate persistence engine skips creation of the failed aggregate and its dependencies and proceeds to the next aggregate in the list. Check the log files to identify failed aggregates.

If there are errors, you must remove the failed aggregates in one of the following ways:

- Manually remove the aggregates from the metadata and the database. To identify the aggregate metadata, you can query the repository using the `Is System Generated` filter for physical tables and logical table sources. See Query the Repository.
- Automatically remove the failed aggregates using the Delete Aggregates specification. In particular, use this technique to remove any orphan aggregate dimensions, those not joined to any other fact table.

Run the Model Check Manager to ensure that your repository doesn't contain modeling problems that can affect Oracle BI Summary Advisor and aggregate persistence performance and results. See Use Model Check Manager to Check for Modeling Problems.

Identify Query Candidates for Aggregation

When creating aggregates, you must identify which queries would benefit substantially from aggregated data.

You can achieve the best results by aggregating to the highest level possible.

To identify slow-running queries, perform the following tasks:

- **Enable usage tracking in the Oracle BI Server**. Usage tracking statistics can be used in a variety of ways, such as database optimization, aggregation strategies, and billing users or departments based on the resources they consume. The Oracle BI Server tracks usage at the detailed query level. When you enable usage tracking, statistics for every query are written to a usage tracking log file or inserted into a database table.

  It's strongly recommended that you use the direct insertion into a database method for usage tracking. See *Administering Oracle Analytics Server*.
- **Analyze the query run times and identify the slowest running queries as candidates for aggregation**. The run time for creating aggregates is dependent on the type of aggregates selected by the user. Creating aggregates from large fact tables is slower than from smaller tables. You should carefully select the aggregates to create.

Use Oracle BI Summary Advisor to Identify Query Candidates for Aggregation

If you're running Oracle Analytics Server on the Oracle Exalytics Machine, you can use the Oracle BI Summary Advisor feature to identify which aggregates increase query performance and to generate a script for creating the recommended aggregates. If you aren't running Oracle Analytics Server on the Oracle Exalytics Machine, the Oracle BI Summary Advisor feature isn't available.

This section contains the following topics:

- About Oracle BI Summary Advisor
- Set Up the Statistics Database
- Turn On Usage Tracking

- Turn On Summary Advisor Logging
- Generate an Aggregate Specification Script
- Summary Advisor Stop Criteria Run Constraints
- Use the nqaggradvisor Utility to Run the Oracle BI Summary Advisor

About Oracle BI Summary Advisor

To reduce query time, you can create aggregate tables that store precomputed results for queries that include rolled-up data.

Before creating aggregates, you need to analyze usage tracking statistics to identify which aggregates could increase query performance. You can use the Summary Advisor to get an optimal list of aggregate tables based on query patterns that might achieve maximum query performance gain while meeting specific resource constraints. The Summary Advisor generates an aggregate creation script that you can run to create the recommended aggregate tables.

This section contains the following topics:

- Gather Summary Advisor Statistics
- Generate and Use Summary Advisor Recommendations
- About Measure Subset Recommendations

Gather Summary Advisor Statistics

Before Summary Advisor can generate recommendations, you must obtain a representative sample of usage statistics for Summary Advisor to use.

Enabling Usage Tracking and Summary Advisor Logging has a minor system performance impact on production systems.

Use one of the following approaches to gather Summary Advisory statistics:

- Enable Usage Tracking and Summary Advisor Logging on a production system, and let users run queries against Oracle BI Server for several days. The Summary Advisor Statistics Table is populated with usage statistics. See Turn On Usage Tracking and Turn On Summary Advisor Logging.
- In a test environment, run a representative workload against the Oracle BI Server to gather Summary Advisor statistics. A representative workload is a list of commonly requested Logical SQL statements. You typically obtain a representative workload from your production environment.

  After you've the representative workload, enable Usage Tracking and Summary Advisor Logging on the Oracle BI Server in your test environment, and use the `nqcmd` utility to run the workload against the Oracle BI Server. See Use nqcmd to Test and Refine the Repository. The Summary Advisor Statistics Table is populated with usage statistics.

Generate and Use Summary Advisor Recommendations

After the Summary Advisor Statistics table is populated with representative data, the Summary Advisor can analyze the data and generate aggregate recommendations to speed up queries.

Run the Oracle BI Summary Advisor Wizard in the Administration Tool to generate an aggregate specification, and then use the aggregate specification to create aggregates using `nqcmd`. See Generate an Aggregate Specification Script and Run the Aggregate Specification Script.

Oracle BI Summary Advisor supports aggregate creation on Oracle TimesTen In-Memory Database, Oracle Analytics Server or when using Oracle Database In-Memory on Oracle Exalytics.

You can also save your Summary Advisor options to a file, and re-run the Oracle BI Summary Advisor Wizard later without re-entering the same options.

About Measure Subset Recommendations

Learn about using the Summary Advisor with aggregates and specific measures.

When the **Only include measures used in queries** option is set in the Summary Advisor wizard, the Summary Advisor only recommends aggregates that contain specific measures that are both present in the analyzed query workload, and that can optimize the query workload if aggregates are created.

- The Summary Advisor doesn't include measures that aren't used in the query workload in its recommended aggregates.
- The size estimation of aggregate fact tables is based on the recommended measure subset, instead of using all the measures of a logical fact table during estimation.
- The Summary Advisor doesn't include measures that are invalid for aggregate persistence in its recommended aggregates.

Set Up the Statistics Database

Before you can use the Oracle BI Summary Advisor feature, you must set up a database to store the collected statistics.

You must run the Repository Creation Utility (RCU) on the target database to create the required statistics schema.

See *Installing and Configuring Oracle Analytics Server*.

- You use the database you installed for use with Oracle Analytics Server as the statistics database because this database already has the RCU-created schemas. The RCU-created table name for Summary Advisor is `S_NQ_SUMMARY_ADVISOR`.
- You also need to import the database into the Physical layer of the Oracle BI repository.
- You must use the same database for Summary Advisor that you use for usage tracking. If you already have a database and schema set up for usage tracking, you can skip the steps in this section.

See Import Metadata from Relational Data Sources.

1. Run the Repository Creation Utility on an external database of your choice.

   You can skip this step if you choose to use the database you installed for use with Oracle Analytics Server for Summary Advisor statistics, because this database has the RCU-created tables already.
2. Open the Administration Tool and import the database into the Physical layer.
3. Save and close the repository.

Use the Upload Repository Command to upload the repository and make it available for queries. See Make the Repository Available for Queries.

Columns in the S_NQ_SUMMARY_ADVISOR Table

Review the columns in the S_NQ_SUMMARY_ADVISOR table.

Column	Description
GROUPBYCOLUMNIDVECTOR	Upgrade IDs for logical column objects that represent group-by columns in a processing path. A *processing path* is an internal Oracle BI Server term. It represents a subquery that involves a single fact logical table source.
LOGICALFACTTABLEID	Upgrade ID of the logical fact table.
LOGICALTABLESOURCEIDVECTOR	Upgrade IDs of the logical table sources.
LOGICAL_QUERY_ID	Foreign key that references the ID column in `S_NQ_ACCT`. This column helps identify the Logical SQL that generated this processing path.
MEASURECOLUMNIDVECTOR	Upgrade IDs for logical column objects that represent measures in a processing path.
PROCESSINGTIMEINMILLISEC	Time spent on this processing path, in milliseconds.
QUERYLEVELIDVECTOR	Upgrade IDs of the logical levels in a processing path.
QUERYSTATUS	For internal use only.
ROW_COUNT	The number of rows retrieved in a processing path. Data in this column is reserved for use by Oracle BI Summary Advisor.
SOURCECELLLEVELIDVECTOR	Upgrade IDs of the logical levels in the logical table source.
VERSION	Version number of the Oracle BI Server.

Turn On Usage Tracking

You must enable usage tracking before collecting Summary Advisor statistics.

See *Administering Oracle Analytics Server*.

Turn On Summary Advisor Logging

When you're ready to collect statistics, you can enable Summary Advisor logging. For new (non-upgraded) installations, the Summary Advisor parameters are centrally managed. For upgrading customers, the Summary Advisor parameters aren't centrally managed by default.

Enabling Summary Advisor logging

You can manage the Summary Advisor parameters using `NQSConfig.INI`.

To enable Summary Advisor logging in `NQSConfig.INI` when central management is disabled for these parameters, follow these steps:

1. On the Oracle BI Server computer, open the `NQSConfig.INI` file in a text editor. You can find this file at:

   ORACLE_INSTANCE`/config/OracleBIServerComponent/coreapplication_obis`*n*

   Make a backup copy of the file before editing.
2. In the `[USAGE_TRACKING]` section, update the following parameters:
   - Set `SUMMARY_STATISTICS_LOGGING` to one of the following options:
     - `YES`: Enables Summary Advisor logging.
     - `LOG_OUTER_JOINT_QUERIES_ONLY`: Enables Summary Advisor logging only for logical queries that contain outer joins. Consider using this option when the minor performance impact of enabling full Summary Advisor logging is a concern.
   - Set `SUMMARY_ADVISOR_TABLE_NAME` to the name of the fully-qualified database table for collecting statistics, as it appears in the Physical layer of the Oracle BI repository. For example:

     ```
     SUMMARY_ADVISOR_TABLE_NAME =
     "My_DB"."DEV_BIPLATFORM"."S_NQ_SUMMARY_ADVISOR";
     ```

     The table name you specify must belong to the same database object and connection pool that you're using for usage tracking.
3. Save and close the file.
4. Restart the Oracle BI Server.
5. If you've multiple Oracle BI Server instances, then repeat these steps in each `NQSConfig.INI` file for all Oracle BI Server instances.

Adding Summary Advisor to the Administration Tool Menu

If you open the repository file in online mode on an Exalytics Server, and the **Oracle BI Summary Advisor** menu option isn't listed under **Tools/Utilities**, you may need to enable it manually.

1. Open `bi-config.xml`
   - Linux: `$`*DOMAIN_HOME*`/config/fmwconfig/biconfig/core/bi-config.xml`

- Windows: `%DOMAIN_HOME%\config\fmwconfig\biconfig\core\bi-config.xml`

2. Search for the following attribute: `<bi:hw-acceleration>`.
3. Set this attribute to `true`.
4. Restart the Business Intelligence services. The menu option should now be visible.

Generate an Aggregate Specification Script

After generating Summary Advisor statistics, you can run the Oracle BI Summary Advisor Wizard to generate an aggregate specification script that you can later run to create the aggregates.

You can only run the Summary Advisor Wizard in online mode. You can also run the Oracle BI Summary Wizard from the command line. See Use the nqaggradvisor Utility to Run the Oracle BI Summary Advisor.

Before you run the Summary Advisor Wizard, you must map the target database, used for creating the aggregates, into the Physical layer. You must manually create the necessary database, connection pool, and physical schema objects.

If you've used the Oracle BI Summary Advisor Wizard previously and saved your filter criteria, targets, and other options as an XML file, you can click **Load Parameters from File** to load the previously saved options into your current wizard session.

The Oracle BI Summary Advisor Wizard is available if you're running Oracle Analytics Server or when using Oracle Database In-Memory on Oracle Exalytics. You can run the aggregate script, recommended by Summary Advisor or manually defined aggregates using Oracle Database In-Memory on Oracle Exalytics as the target.

If your Summary Advisor table, specified in the `SummaryAdvisorTableName` in the **System MBean Browser**, or the *SUMMARY_ADVISOR_TABLE_NAME* parameter in `NQSConfig.INI` is empty, Summary Advisor can't proceed.

Summary Advisor Setting Recommendations

In the Summary Advisor's Miscellaneous page, Oracle recommends:

- Selecting **Use surrogate keys** to improve the performance of queries using the aggregates.
- Selecting **Prefer optimizer estimates** to improve performance during the Summary Advisor process.

  The **Prefer optimizer estimates** option enables using cardinality estimates that originate out of the database query optimizer whenever possible, rather than issuing actual count queries. You can use the **Prefer optimizer estimates** option with Oracle Database.

  For Summary Advisor to use database query optimizer estimates, obtain up-to-date statistics on the concerned database objects. See the Oracle Database documentation for more information.

  If you don't select the **Prefer optimizer estimates** option, Summary Advisor issues count queries to the back-end data sources to obtain row counts (cardinality) for certain queries on the data sources that can sometimes take a long time to run. Refer the appropriate database documentation for guidelines on how to obtain the best estimates. For example, when using Oracle Database, you might want to use the column group feature to improve cardinality estimates for multiple column queries.

A query that attempts to sample a particular grain isn't issued by Summary Advisor if an entry for that particular grain already exists in the Summary Advisor cache files, regardless of whether it's an actual count query or a cardinality estimate query.

You should remove the Summary Advisor cache files when selecting or deselecting the **Prefer optimizer estimates** option. To do this, delete `NQAggregate.Stats.Cache.txt` and `NQAggregate.LTS.Stats.Cache.txt` in the following directory on the Oracle BI Administration Tool computer:

ORACLE_INSTANCE`\bifoundation\OracleBIServerComponent\coreapplication_obis`*n*`\aggr`

- Select **Only include measures used in queries** to include measures used in queries. See About Measure Subset Recommendations.

  If you don't select this option, all measures in a logical fact table are included in the recommendation, including measures that weren't used in the workload analyzed by Summary Advisor.

See Summary Advisor Stop Criteria Run Constraints.

Oracle recommends running the Model Check Manager to ensure that your repository doesn't contain modeling problems that could affect Oracle BI Summary Advisor performance and results. See Use Model Check Manager to Check for Modeling Problems.

Oracle recommends running the Model Check during off-peak periods. The Model Check Manager runs queries against back-end data sources for some checks. Running the Model Check Manager for large repositories can take a long time. Use **Filtered by Statistics**, or run it only for selected objects, to improve performance.

1. Open your repository in the Administration Tool in online mode.
2. Select **Tools**, and then select **Utilities**.
3. Select **Oracle BI Summary Advisor**, and then click **Execute**.
4. Optional: In Filter Logs - Logical Fact Tables, generate Summary Advisor recommendations for all logical fact tables, or select specific logical fact tables, and click **Next**.
5. Optional: In Filter Logs - Time Window, enter a **Start Date** and **End Date** to filter the Summary Advisor logging statistics based on time period, and click **Update** to refresh the view after entering a time period.
6. Optional: In Filter Logs - Execution Time Threshold, specify the number of seconds for **Minimum Cumulative Time** to filter by a minimum query time threshold for each logical table source.
7. In Targets, select the target container and associated connection pool for the location of aggregate tables.

   You can specify more than one target container.
8. Specify the **Database Schema**, **Connection Pool**, and **Capacity** for the target in megabytes, then click **Add Target** to add it to the list.
9. In Select File Location, click **Browse** to select the location for storing the aggregate specification, a SQL script, and click **Next**.
10. Optional: In Stopping Criteria, specify run constraints for the set of recommendations.

11. Optional: In Miscellaneous, specify the maximum size of any single aggregate, in megabytes.

    You can also specify the location of an XML output file that stores the criteria and options from this session to re-use in a future Summary Advisor session.

12. In Run, click **Run** to generate recommendations using the Summary Advisor process.

    (Optional) You can click **Stop** at any point to stop the process. When Summary Advisor stops or runs to completion, the aggregate recommendations are displayed.

13. When the process completes, click **Next**.

14. In Filter Aggregates, review the current set of aggregate recommendations.

    You can exclude certain aggregates from the creation process by deselecting the Include option for that row.

15. On the Finish Script screen, review the script, and then click **Finish** to save the script.

See Run the Aggregate Specification Script.

Summary Advisor Stop Criteria Run Constraints

In the Summary Advisor wizard Stopping Criteria page, you can specify run constraints for the set of recommendations.

Consider the following:

- You can specify the maximum time that Summary Advisor runs before returning results.
- You can specify a minimum percentage improvement to performance gain of all affected queries in the workload when adding a new aggregate.

  Summary Advisor uses an iterative optimization algorithm. For each round of the iteration, Summary Advisor Summary Advisor evaluates a different set of aggregates. When you specify a minimum percentage improvement on this screen, Summary Advisor compares the estimated gain between two consecutive rounds, and stops when the incremental improvement is less than the specified minimum percentage.

  The following formula describes the estimated gain between rounds:

  ```
  Estimated Gain = [(total query time at the beginning of the round) -
  (total query time at the end of the round)] / (Initial total query time
  prior to the first round)
  ```

  For example:

  – Initial total query time = 1000s
  – End of Round 1:

    Total query time = 500s

    Gain = (1000 - 500)/1000 = 50%
  – End of Round 2:

    Total query time = 250s

    Gain = (500 - 250)/1000 = 25%

Use the nqaggradvisor Utility to Run the Oracle BI Summary Advisor

You can use the Oracle BI Server utility `nqaggradvisor` to run the Summary Advisor from the command line instead of using the Oracle BI Administration Tool.

After Summary Advisor statistics have been generated, use `nqaggradvisor` to generate an aggregate specification script that you can then run to create the aggregates. The `nqaggradvisor` utility is only available if you're running Oracle Analytics Server on the Oracle Exalytics Machine.

The location of the `nqaggradvisor` utility is:

BI_DOMAIN`/bi/bitools/bin`

Syntax

The `nqaggradvisor` utility takes the following parameters.

```
nQAggrAdvisor -d dataSource | -u userName | -o outputFile |
-c tupleInQuotes [-p password] [-F factFilter] [-z maxSizeAggr]
[-g gainThreshold] [-l minQueryTime] [-t timeoutMinutes]
[-s startDate] [-e endDate] [-C on/off] [-M on/off] [-K on/off]
```

Where:

dataSource is the ODBC data source name for the Oracle BI Server to which you want to connect and run Summary Advisor.

userName is the user name with which to log into the data source. The specified user must have the privilege required to open the Administration Tool in online mode and use the Oracle BI Summary Advisor Wizard.

outputFile is the fully qualified path and file name of the output aggregate specification script.

tupleInQuotes is the aggregate persistence target. You must specify the fully qualified connection pool, fully qualified schema name, and capacity in megabytes.

password is the password corresponding to the *userName*. If not specified, the user is prompted for a password when running `nQAggrAdvisor`.

factFilter is the fact filter file name. The fact filter file contains the fully qualified names of logical fact tables for which to generate Summary Advisor recommendations. Add each logical fact table's fully qualified name on a separate line. If a fact filter file isn't specified, then all logical fact tables in the repository are included in the analysis.

maxSizeAggr is the maximum size of an aggregate in megabytes.

gainThreshold is the minimum percentage improvements to performance gain of all affected queries in the workload required by Summary Advisor when adding a new aggregate in its iterative optimization algorithm. Summary Advisor stops when this value isn't satisfied. The default value is *1*.

minQueryTime is the minimum query time threshold in seconds for each logical table source before it's included in the Summary Advisor process. The default value is *0*.

`timeoutMinutes` is the maximum time in minutes that Summary Advisor runs before returning results. Specify *0* for unlimited. The default value is *0*.

startDate is the start date for statistics to include in the Summary Advisor process.

endDate is the end date for statistics to include in the Summary Advisor process.

`-C` specifies whether to use optimizer estimates. Specify *on* or *off*. The default is `off`.

`-M` specifies which measures to include in the recommendation. Specify *on* to include measures used in the workload. Specify *off* to include all measures in a logical fact table including those measures that weren't used in the workload analyzed by Summary Advisor. The default is *off*.

`-K` specifies whether to use surrogate keys. Specify *on* or *off*. The default is *on*.

Examples

The following example shows how to correctly specify the `tupleInQuotes` parameter:

```
nQAggrAdvisor -d "AnalyticsWeb" -u "Administrator" -p "ADMIN" -o
"C:\temp\aggr_advisor.out.txt" -c "DW_Aggr"."Connection Pool","DW_Aggr".."AGGR",1000
```

The following example shows how to correctly specify the `gainThreshold` , `startDate` , and `endDate` parameters.

```
nQAggrAdvisor -d "AnalyticsWeb" -u "Administrator" -p "ADMIN" -o
"C:\temp\aggr_advisor.out.txt" -F "C:\temp\fact_filter.txt" -g 10 -c
"TimesTen_instance1"."Connection Pool","dbo",2000 -s "2011-05-02 08:00:00" -e
"2011-05-07 18:30:00" -C on -M on -K off
```

Use the Aggregate Persistence Wizard to Generate the Aggregate Specification

You can use the Aggregate Persistence Wizard to create the SQL file used to create and load aggregate tables and map them into the metadata.

Run the resulting SQL file against a running Oracle BI Server.

Oracle recommends that you use the Aggregate Persistence Wizard because it automatically enforces many of the constraints necessary when generating the aggregate specification. However, you can manually write the aggregate Logical SQL as an alternative to using the wizard.

Before you run the Aggregate Persistence Wizard, you must map the target database where you plan to create the aggregates into the Physical layer. To do this, manually create the necessary database, connection pool, and physical schema objects. If you're running Oracle Analytics Server on Oracle Exalytics machine, you can use the Summary Advisor feature instead of the Aggregate Persistence Wizard to identify which aggregates increase query performance and to generate a script for creating the recommended aggregates.

Because Model Check Manager runs queries against back-end data sources for some checks, it's recommended to run it during off-peak periods. In addition, it can take a long time to run Model Check Manager for large repositories. Use **Filtered by Statistics** (where available), or run it only for selected objects, to improve performance.

See the following:

- Use Model Check Manager to Check for Modeling Problems

- Run the Aggregate Specification Script
- Add Surrogate Keys to Dimension Aggregate Tables
- Use Oracle BI Summary Advisor to Identify Query Candidates for Aggregation
- Write the Create Aggregates Specification Manually

1. Run Model Check Manager to ensure that your repository doesn't contain modeling problems that can affect aggregate creation and performance.
2. Open your repository in the Administration Tool, if it's not open already.

   You must run Model Check Manager in online mode. However, you can run the Aggregate Persistence Wizard in either online or offline mode.
3. Select **Tools** , select **Utilities** , select **Aggregate Persistence**, and then click **Execute**.
4. In **Select File Location**, specify the complete path and file name of the aggregate creation script.

   You can specify a new or an existing file name.

   Typically, when you run the SQL script against the Oracle BI Server, it creates DDL and runs it against the target database schema to create the aggregate tables, then loads them from the source, and finally creates the Oracle BI Server metadata so the aggregate navigation feature can use the new tables.

   You can select **Generate target DDL in a separate file** if you want the DDL stored in a separate file from the Oracle BI Server SQL script. Selecting this option gives you the flexibility to alter the auto-generated DDL and run it independently of the Oracle BI Server. For example, you may want to alter the storage parameter or index settings.

   When you select **Generate target DDL in a separate file**, two SQL scripts are generated in the directory you specify in the **Location** field:

   - The create aggregates script (*script_name*)
   - The prepare aggregates script (*script_name*_DDL)

   After selecting **Generate target DDL in a separate file** and completing the wizard steps, you typically do the following:

   a. Run the prepare aggregates script against the server. This action creates a DDL file at the following location:

      ```
      ORACLE_INSTANCE\bifoundation\OracleBIServerComponent\coreapplication_obisn\
      aggr
      ```

   b. Run the generated DDL file against the target database to create the table.

   c. Run the create aggregates script to populate the table.

   Click **Next** after you've finished specifying options on the Select File Location screen.
5. In the Select Business Measures screen, select the measures on which you want to aggregate. To do this, select a business model in the upper pane, then select a single fact table or a set of measures in the lower pane. You can't select measures that span multiple fact tables. Use Ctrl-click to select multiple measures, or use Shift-click to select a range of consecutive measures.

Select **Persist 'Count Distinct' measures as raw values** to add the `as_raw_values` clause to all valid count distinct measures and to set an aggregation expression override on the corresponding logical column for each system-generated aggregate logical table source. Setting this option enables aggregate persistence to store actual values that are distinct-counted. If you don't select this option, then aggregate persistence stores pre-computed counts for the specified level combinations.

The **View Script** button isn't available during the creation of the first aggregate table block.

Click **Next** after you've selected the appropriate measures.

6. In the Select Levels screen, specify the level of aggregation by selecting a logical level for one or more dimensions. You can specify a surrogate key to use for the fact-dimension join.

   The default join option between the aggregated fact and dimension tables is the primary key defined in the logical level you selected. If the primary key of the level is large and complex, the join to the fact table is expensive, so using a surrogate key is recommended in this case. A surrogate key is an artificially generated key, usually a number. For example, a surrogate key in the level aggregate table would simplify this join, removing unnecessary (level primary key) columns from the fact table and resulting in a leaner fact table.

   Using a surrogate key only changes the query response time, not the logical results of the queries. However, generating the surrogate keys can have the side effect of increasing the aggregate table load time. Therefore, the recommended setting is as follows:

   - If the primary key for the logical level you've selected is already a single, numeric column, you typically shouldn't select the **Use Surrogate Key** option since it may add to load processing time without producing a performance benefit.
   - If the primary key for the logical level you've selected is a text string, or consists of multiple logical columns, you typically should use a surrogate key to improve the performance of the queries that join to that aggregate dimension. However, keep in mind that generating the surrogate key can increase the load time for that aggregate dimension table.

   Click **Next** after you've selected the appropriate level of aggregation.

7. In the Select Connection Pool screen, select the appropriate items to specify a location for the aggregate table.

   A default aggregate table name is provided, and a prefix is added to the table name. The default prefix for the generated fact table is `ag`. For tables created for dimension (level) aggregates, the default prefix is `SA_` . You can changed the prefix by updating the `AGGREGATE_PREFIX` property in `NQSConfig.INI`.

   Click **Next** after you've provided connection pool information.

8. In the Finish screen, the **View Script** button becomes available for use, and the Logical SQL script appears for your review. Choose whether to define another aggregate (default) or end the wizard, and then click **Next**.

9. In the Finish Script screen, the complete path and file name appears. Click **Finish**.

Use Model Check Manager to Check for Modeling Problems

Learn how to use Model Check Manager to check for modeling problems that might affect Oracle BI Summary Advisor and the aggregate persistence engine.

This section contains the following topics:

- About Model Check Manager
- Run Model Check Manager
- Resolve Model Errors
- Check Models Using the validaterpd Utility

About Model Check Manager

You can use the Model Check Manager to check your repository metadata for issues that might affect the success of the Oracle BI Summary Advisor or the aggregate persistence engine.

- The Model Check Manager requires access to the summary statistics table, when using **Filtered by Statistics**, and back-end data sources for some checks. Some of the back-end queries can impact performance, you should run the Model Check Manager during off-peak periods.
- You can only run the Model Check Manager in online mode.
- The Model Check Manager doesn't make any changes to repository metadata. The Model Check Manager only flags possible problems.

The Model Check Manager returns both error and warning messages. You must fix errors identified by Model Check Manager. If you don't fix the errors, the Oracle BI Summary Advisor could provide incorrect recommendations, and the aggregate persistence engine could fail to create aggregates. You should fix warnings. Issues identified by warnings result in suboptimal recommendations from Oracle BI Summary Advisor, or suboptimal performance from the aggregate persistence engine.

Model Check Manager runs parallel queries against the database for better performance. By default, 24 threads are enabled. To change the default number of threads for model check manager, create and set an operating system environment variable called `MODEL_CHECKER_MAX_THREADS`. The maximum number of threads you can specify is 100.

Run Model Check Manager

For Oracle BI Summary Advisor, run Model Check Manager after you've gathered Summary Advisor statistics, but before you run the Oracle BI Summary Advisor Wizard.

To run Model Check Manager globally using the Administration Tool, select the File menu, then select Check Models. You can use the following options:

- **Complete:** Checks all objects in the Business Model and Mapping layer of the Oracle BI repository.
- **Filtered by Statistics:** Checks only fact table objects and associated dimensions in the Business Model and Mapping layer that have been actively queried according to the statistics table. Select this option to speed up the process for large repositories.

  This option is only available on the Oracle Exalytics Machine. If you attempt to filter by statistics on a non-Exalytics system, or if you attempt to filter when the

statistics table isn't available, a warning appears explaining that Model Check Manager can't filter by statistics.

See the following sections for information about setting up the Summary Advisor statistics table:

- Set Up the Statistics Database
- Turn On Usage Tracking
- Turn On Summary Advisor Logging

To run Model Check Manager for selected objects using the Administration Tool, right-click one or more business models, dimension objects, or logical fact tables and select **Check Model**. Then, choose **Complete** or **Filtered by Statistics** from the submenu, as described in the preceding list. The **Filtered by Statistics** menu option is only available for fact table objects and business model objects.

When using Model Check Manager with large repositories, it's recommended that you use **Filtered by Statistics**, or run it only for selected objects, to improve performance.

- In the Oracle BI Administration Tool, from the **File** menu, select **Check Models**.

Resolve Model Errors

After running the Model Check Manager for one or more objects, the Model Check Manager opens so that you can correct errors in the repository.

Run the Oracle BI Administration Tool in online mode.

1. In the Model Check Manager results, double-click a row to open the **Properties** dialog, or select a row and click **Go To**.
2. Correct the problems using the information in the **Error Description**.
3. Rerun the Model Check to verify that all of the issues are resolved.

Check Models Using the validaterpd Utility

You can check models from the command line using the Oracle BI Server validaterpd utility with the -L option.

Running this utility with `-L` performs the same model checks as Model Check Manager in the Administration Tool. The `validaterpd` utility is available on both Windows and Linux systems.

To run `validaterpd` in Model Check mode, you must specify the DSN of a running Administration Tool.

The location of the `validaterpd` utility is:

BI_DOMAIN`/bi/bitools/bin`

See Use the validaterpd Utility to Check Repository Consistency.

Syntax

The `validaterpd` utility takes the following parameters in Model Check mode:

```
validaterpd -L -D DSN_name -U DSN_user_name [-P DSN_password]
{-O output_txt_file_name |-C output_csv_file_name | -X output_xml_file_name} [-W]
[-S] [-8]
```

Where:

`-L`: Specifies Model Check mode.

`-D`: The DSN of a running Oracle BI Server.

`-U`: The user name for the Oracle BI Server DSN.

`-P`: The password for the Oracle BI Server DSN.

The password argument is optional. If you don't provide the password argument, you're prompted to enter the password when you run the command. To minimize the risk of security breaches, Oracle recommends that you don't provide password arguments either on the command line or in scripts.

The password argument is supported for backward compatibility only. For scripting purposes, you can pass the password through standard input.

`-O` Use this option to output the results in a text file.

`-C` Use this option to output the results in a CSV file.

`-X` Use this option to output the results in an XML file.

`-8` Use this option to specify UTF-8 output (optional).

`-W` You can include an allowed list objects file. This text file specifies a limited number of logical objects that you want to check. Enter the fully-qualified name of each logical object on a single line. If `-W` isn't specified, all logical objects are checked.

`-S` Use this option to check only objects that have been actively queried according to the statistics table. If `-S` isn't specified, all objects are checked. If `-W` is also specified, the allowed list file can only contain business models and logical fact tables, other objects aren't checked. This option is only available on the Oracle Exalytics machine.

Examples

```
validaterpd -L -D DSNName -U Username -O results.txt
Give password: my_dsn_password
```

The preceding example connects to an Oracle BI repository using the *DSNName* connection, checks all models in the Oracle BI repository, and writes output to *results.txt*.

```
validaterpd -L -D DSNName -U Username -O results.txt -W whitelist.txt -S
Give password: my_dsn_password
```

The preceding example connects to an Oracle BI repository using the *DSNName* connection, performs a model check, and writes output to *results.txt*. Only objects listed in *whitelist.txt* are checked. Furthermore, because -S is specified, only objects that have been actively queried according to the statistics table are checked.

When `-W` and `-S` are both specified, the allowed list can only contain business models and logical fact tables. Other objects aren't checked.

Write the Create Aggregates Specification Manually

You can write the script file manually, instead of using the Aggregate Persistence Wizard to create the script file. Oracle recommends that you use the Aggregate Persistence Wizard.

If you don't want the Oracle BI Server to modify your databases during aggregate creation, then you can specify this in the Aggregate Persistence Wizard by selecting the option **Generate target DDL in a separate file**. The Aggregate Persistence Wizard creates a DDL file, the *prepare aggregates* script, that you can use to create the empty aggregate tables. After this, you need to run the *create aggregates* script to populate the aggregate tables. This option provides some flexibility in case the database access to create tables is restricted. You must run the prepare aggregates script before you run the create aggregates script.

This section contains the following topics:

- What Constraints Are Imposed During the Create Process?
- Write the Create Aggregates Specification
- Add Surrogate Keys to Dimension Aggregate Tables

What Constraints Are Imposed During the Create Process?

You can learn about constraints are imposed during the create process.

The following constraints are imposed during the create process:

- Valid measures

  A valid measure must have a valid aggregation rule. The following constraints apply to level-based measures:

  – If the level is grand total alias, then that dimension mustn't be present in the list of levels for that aggregate specification.

  – Any other level defined for this measure must be present in the list of levels for that aggregate specification.

  If the above constraints aren't met, then the entire aggregate specification is discarded. In addition, a measure is ignored by the create process if any of the following conditions are true:

  – Measure is mapped to a session or repository variable.

  – Measure is a derived measure.

  – Measure has a default aggregation rule of *None*.

  Measures that are ignored don't necessarily affect the aggregate specification. The remaining measures are used to create the aggregate.

- Valid levels

  A valid level must have a valid primary key. If a level is invalid, the aggregate specification is discarded. Attributes of a level or its primary key are ignored if any of the following conditions are true:

  – Attribute is mapped to session or repository variables.

  – Attributes aren't from the same logical table.

- Valid aggregate specification

A valid aggregate specification has the following properties:

- Name length is between 1 and 18 characters (inclusive).
- Specify at least one valid level.
- Specify at least one valid measure.
- Must have a valid connection pool.
- Must have a valid output container (database/catalog/schema).
- Connection pool and container must belong to the same database.
- Only one level per dimension can be specified.
- Measures can only be from the same fact table.
- All logical components of the specification must be from the same subject area.

An aggregate specification is ignored if the name already exists in the output container, because level aggregates are reviewed by the entire database. However, if different catalogs or schemas are specified for the same fact aggregate name, it's allowed to have multiple facts with the same name but different scope in the same database.

The aggregate specification is discarded if any dimension isn't joined to a fact.

Write the Create Aggregates Specification

All metadata names, except logical fact columns, are fully qualified.

There are two modes of operation: Create and Delete. It's strongly recommended that you place all aggregate specifications under a single Create Aggregates statement.

See Add Surrogate Keys to Dimension Aggregate Tables.

Delete Statement for Aggregate Specification

Begin the script file with a Delete statement. It's essential to delete system-generated aggregates before creating new ones.

This ensures that data is consistent and removes invalid or incomplete aggregates before you run the Create operation. The following statement is the syntax for deleting aggregates:

```
Delete aggregates [list of fully qualified physical table names];
```

For example:

```
Delete aggregates "src".."INCR"."fact_1", "src".."INCR"."fact_2";
```

You can include a comma-separated list of physical tables to delete. You must include system-generated tables from a previous run of the aggregate creation script. Any dimension tables joined to listed fact tables are also deleted.

If a dimension table is joined to more than one fact table, you can't delete the table unless the other joined table is also deleted.

In addition to fact tables, you can also use the Delete statement to delete orphan dimension tables, these are dimension tables that aren't joined to any other fact table. Orphan dimension tables sometimes occur when aggregate creation fails.

The Delete statement also removes the logical key and logical column that were added to the time dimension's logical table when chronological keys were added for aggregate persistence.

Create Statement for Aggregate Specification

The Create statement should follow the Delete statement.

The following is the syntax for creating aggregates:

```
Create|Prepare aggregates
aggr_name_1
for  logical_fact_table_1 [(logical_fact_column_1, logical_fact_column_2,
count_distinct_logical_fact_column_1 as_raw_values, ...)]
at levels (level_1, level_2, …)
using connection pool connection_pool_name_1
in schema_name_1
[ ,aggr_name_2
for logical_fact_table_3 [(logical_fact_column_5, logical_fact_column_2,…)]
at levels (level_3, level_2, …)
using connection pool connection_pool_name_2
in schema_name_2] ;
```

The `as_raw_values` must accompany a count-distinct measure with a simple aggregate rule. A simple aggregate rule has only one rule, which isn't dimension-based.

Multiple Aggregates in Aggregate Specification

Use these guideline to specify multiple aggregates in a single Create Aggregates statement.

- Ensure that each of the multiple aggregate specifications are separated by a comma, and the entire aggregate creation script is terminated with a semicolon.
- In this file, only one Delete Aggregates statement should be specified at the beginning. Make sure that only one delete is issued per ETL run, unless a reset is needed.

  Any aggregate scripts that are run after the first one shouldn't have a Delete Aggregates statement, or all previously created aggregates are removed.

Where Clause for Aggregate Specification

You can add an optional Where clause to the Create statement.

The Where clause filters the data that you want to aggregate and creates fragmented aggregates, or aggregates for only a fragment of data in the base fact table. The Where clause also sets the Fragmentation content field located on the Logical Table Source dialog. In most cases, the creation of fragmented aggregates maximizes query acceleration while minimizing the cost of creating and maintaining the aggregate.

The following examples show when you would use fragmented aggregates:

- If you're working with the time dimension and want your aggregates to include data only from the last three years.
- If your company reports primarily on revenue in the United States and wants the aggregates to include only United States data.

The following is an example of a valid Create statement with the Where clause:

```
Create Aggregates
Revenue_By_Year
for  "sales"."sales" at levels("sales".timedim.year)
where("sales"."time"."calendar year"=2007)
using connection pool aggrtarget.cp1
in aggrtarget..schema1
```

Logical Column Requirements

The logical column that you specify in the Where clause must meet the following requirements:

- The logical column must belong to a dimension.
- If the logical column belongs to a dimension included in the aggregate specification, then it must be at or above the level of the aggregate.
- If you use `operational_oper`, then the logical column's data type must match the constant's data type in `inlist`.

  If you use `inclusion_oper`, then the logical column's data type must match all the constants' data type in `inlist`.

Where Clause Grammar

The grammar for the `Where` clause in the create aggregate specification is a subset of the `Where` grammar for a Logical SQL filter. The grammar the Oracle BI Server supports for the create aggregate specification differs slightly from the Logical SQL filter.

Review the following Create statement and its Where clause:

```
create aggregate aggr1 for fact1 at levels(11,12...) where (filter_list)
using ....
```

The following are the acceptable grammar rules:

```
filter_list ::= filter logical_oper filter_list

    | filter

    | '(' filter_list ')'

filter ::= logical_column relational_oper constant

    | logical_column inclusion_oper '(' inlist ')'

relational_oper ::= '=' | '!=' | '<' | '>' | '<=' | '>=' | 'like'

inlist ::= constant ',' inlist

    | constant

logical_oper ::= 'and' | 'or'

inclusion_oper ::= 'in' | 'not in'
```

Add Surrogate Keys to Dimension Aggregate Tables

The join option default between fact and level aggregate tables uses primary keys from the level aggregate.

If the primary key of the level is large and complex, that is composed of many columns, then the join to the fact table is expensive. A surrogate key is an artificially generated key, usually a number. A surrogate key in the level aggregate table simplifies the join and removes unnecessary columns (level primary key) from the fact table, resulting in a smaller-sized fact table. Adding surrogate keys to the dimension (level) aggregate tables can simplify joins to the fact tables and might improve query performance. A surrogate key ensures that each aggregate table has a unique identifier.

It's possible for sharing a level among multiple fact tables. One fact might use surrogate keys, and another might use primary keys from the dimension aggregate. The following are some options for resolving this issue:

- Set a metadata property for levels that indicates whether to use surrogate keys or primary keys.
- Always create a surrogate key for a level aggregate. You can decide later after observing performance if you should create a fact aggregate using a surrogate or primary key.

You could specify using surrogate keys for the entire star which results in simpler syntax, restricts the available user options, and slows the aggregate creation process.

About the Create/Prepare Aggregates Syntax

The syntax for create/prepare aggregates contains the change for Using_Surrogate_Key.

You can specify a surrogate key option for each level. If unspecified, the fact and dimension tables are joined using the primary key from the level aggregate.

```
Create|Prepare aggregates
aggr_name_1
[file output_file_name]
for  logical_fact_table_1 [(logical_fact_column_1, logical_fact_column_2,…)]
at levels (level_1 [Using_Surrogate_Key], level_2, …)
using connection pool connection_pool_name_1
in schema_name_1
[ ,aggr_name_2
for logical_fact_table_3 [(logical_fact_column_5, logical_fact_column_2,…)]
at levels (level_3, level_2, …)
using connection pool connection_pool_name_2
in schema_name_2] ;
```

About Surrogate Key Output from Create/Prepare Aggregates

The changes to the current process are restricted to the physical metadata layer in the repository and the database.

- For a level aggregate in the physical metadata, the `Using_Surrogate_Key` join option does the following:

  The level aggregate table has a new column called *`levelName_upgradeIDSK`*, check for collisions. This is the surrogate key column for the dimension aggregate. The *levelName* is truncated if the total number of characters exceeds 18.

- For a level aggregate in the database, the `Using_Surrogate_Key` join option does the following:

  The level aggregate table also has a corresponding column called *`levelName_upgradeIDSK`*. You can populate the table using `RCOUNT()`.

- For a fact aggregate in the physical metadata, the `Using_Surrogate_Key` join option does the following:
  - The fact aggregate table no longer contains columns from the level's primary keys.
  - Instead, a new column that corresponds to the level aggregate's surrogate key is added to the table.
  - The type of this column is identical to the level's surrogate key.
  - The column has the same name as that in the level aggregate, and checks for collisions.
  - The fact table and the level table are joined using this surrogate key only.
- For a fact aggregate in the database, the `Using_Surrogate_Key` join option does the following:

  The fact aggregate table has the corresponding surrogate key. The table is populated using new capabilities available through `Populate`.

Run the Aggregate Specification Script

Learn how to run the aggregate specification script against the Oracle BI Server.

Before you run the script, you must create an ODBC DSN (data source name) for the Oracle BI Server and ensure that the correct log level is set. You must manually create a DSN for the Oracle BI Server to run the aggregate specification against for a single-node deployment. When the deployment is a multi-node cluster, you must run the aggregate specification directly against the source Oracle BI Server. Create a non-clustered DSN for the source Oracle BI Server to run the aggregate specification against. Use the Cluster Manager in the Administration Tool in online mode to determine which Oracle BI Server is the source.

In a clustered environment, the aggregate specification script performs a rolling restart of the destination Oracle BI Servers in the background. As a best practice, you should avoid making other configuration changes in Fusion Middleware Control or the configuration files while running the aggregate persistence script. Only the destination servers are restarted in the rolling restart. Changing the configuration, might send the Oracle BI Server a different set of configuration settings than the destination Oracle BI Servers. If the configuration changed, restart the source Oracle BI Server.

After creating a DSN, you can run the script using `nqcmd` as a user who is a member of the BI Administrators group. See Use nqcmd to Test and Refine the Repository.

The queries and errors are logged in the `obis1_query.log` located in the `DOMAIN_Home/servers/obis1/logs`.

See Integrating Other Clients with Oracle Business Intelligence in *Integrator's Guide for Oracle Business Intelligence Enterprise Edition* for information about how to create an ODBC DSN for the Oracle BI Server.

Trace logs are recorded if the logging level is at least 2. The logging events include the aggregate execution plan and the order in which the aggregates are created and deleted. Higher logging levels provide more details about the query and execution plans - for example, specify logging level 4 or higher to see the queries being run. Error logs are recorded if the logging level is at least 1, and to `nqserver.log` regardless of the logging level.

Use one of the following methods to set the logging level:

- Set the logging level in the repository user object for the user who plan to run the script. See Manage the Query Log in *Administering Oracle Analytics Server*.
- Create and set the LOGLEVEL system session variable. LOGLEVEL overrides logging levels set on individual users. See Create Session Variables.

1. Connect directly to a DSN for a running Oracle BI Server and not to a clustered DSN.
2. Set an appropriate logging level, least 2, before running the script.
3. edit the aggregate creation script directly to set the logging level as a request variable in each delete aggregates or create aggregates statement, for example:

   ```
   set variable LOGLEVEL=7 : delete aggregates;
   set variable LOGLEVEL=7 : create aggregates... ;
   ```

   Use a colon as a delimiter when setting request variables using `nqcmd`.
4. As a member of the BI Administrators group, use `nqcmd` to connect to the non-clustered DSN for the Oracle BI Server that you created in step 1.
5. Run the aggregate specification script.

After running the SQL script, aggregates are created and persisted in the Oracle BI Server metadata, as well as, in the back-end databases.

When a series of aggregates are being created, and the creation of one aggregate fails, the aggregate persistence engine skips creation of the failed aggregate and its dependencies and proceeds to the next aggregate in the list.

Check the log files to identify failed aggregates. If orphan aggregate dimensions, those not joined to any other fact table, were created, use the `Delete Aggregates` command to remove them.

Lifecycle Use Cases for Aggregate Persistence

The table summarizes the user tasks to persist aggregates for different lifecycle use cases.

Lifecycle use cases focus on operations against single or multiple aggregate persistence targets, and don't describe operations for single or multiple-node deployments. User tasks are the same for both single-node deployments and multiple node deployments with the only difference is related to a clustered deployment. In a clustered deployment, you must connect to the controller Oracle BI Server. A rolling restart of the subordinate servers is performed in the background. See Run the Aggregate Specification Script.

Number	Use Case	Description
1	Creating aggregates for a single aggregate persistence target	To only create aggregates, modify the aggregate creation script to remove the delete aggregates statement at the beginning. Then, use `nqcmd` to run the script.

Number	Use Case	Description

2	Deleting aggregates for a single aggregate persistence target	To delete aggregates, use `nqcmd` to run the delete aggregates statement directly, as follows:
		`Delete aggregates [list of fully qualified physical fact table names];`
		For example:
		`Delete aggregates;`
		or
		`Delete aggregates "src".."INCR"."fact_1", "src".."INCR"."fact_2";`

3	Refreshing aggregates for a single aggregate persistence target	Use `nqcmd` to run the aggregate creation script, which contains statements to first delete, then create the aggregates. Alternatively, you can manually delete the aggregates as described in use case 2, then create aggregates as shown in use case 1. This manual method is useful for situations where you want to delete all aggregates, but the aggregate creation script only specifies certain aggregates to be deleted.

4	Creating aggregates for multiple redundant aggregate persistence targets	To create aggregate clones on multiple targets, modify the aggregate creation script to copy the create aggregates statements as many times as you've targets.
		For example, say you've a script containing the following create aggregates statement:
		`set variable LOGLEVEL=7 : create aggregates "myfactaggr" for "FACT_1"("MEASURE_1") at levels ("INCR"."DIM1_LEVEL1Dim"."DIM1_LEVEL1 Detail") using connection pool "tgt1"."cp" in "tgt1".."double1";`
		You would then copy the block, paste it below the first block, and modify the connection pool and schema information for your second target. For example:
		`set variable LOGLEVEL=7 : create aggregates "myfactaggr" for "FACT_1"("MEASURE_1") at levels ("INCR"."DIM1_LEVEL1Dim"."DIM1_LEVEL1 Detail") using connection pool "tgt2"."cp" in "tgt2".."double2";`
		After you've copied and modified the block for all your targets, save the script. Then, use `nqcmd` to run the aggregate creation script.

5	Deleting aggregates for multiple aggregate persistence targets	To delete aggregates on multiple targets, use `nqcmd` to run the delete aggregates statement directly for the affected fact tables. For example:
		`set variable LOGLEVEL=7 : delete aggregates "tgt1".."double1"."myfactaggr";`
		`set variable LOGLEVEL=7 : delete aggregates "tgt2".."double2"."myfactaggr";`

6	Refreshing aggregates for multiple redundant aggregate persistence targets	See Use Double Buffering to Refresh Highly Available Aggregates.

Number	Use Case	Description
7	Refreshing aggregates for multiple partitioned aggregate persistence targets	In some cases, you might have different aggregates partitioned across multiple targets. This approach maximizes memory use, but doesn't provide highly available aggregates. To refresh partitioned aggregates, use one of the following methods as appropriate for your deployment: • Run the Aggregate Persistence Wizard multiple times against the different targets to generate a set of aggregate creation scripts, then run the scripts. • If you're running Oracle Analytics Server on the Oracle Exalytics Machine, run Oracle BI Summary Advisor and specify multiple targets in the Targets screen. Then, run the aggregate creation script.

Use Double Buffering to Refresh Highly Available Aggregates

When you've aggregate clones across multiple aggregate persistence targets, you can use double buffering to avoid downtime when refreshing the aggregates.

You manually call the aggregate create and delete SQL statements in a way that controls the refresh to set up double buffering.

You start by deleting aggregates on the first target. Next, you create the aggregates on the first target, specifying the targets where aggregates haven't yet been deleted as inactive schemas, so that the old data isn't used in the refresh. Then, you repeat this process for each target. You don't need to specify inactive schemas when refreshing the last target because by that point, the data in the other schemas has already been refreshed.

When specifying inactive schemas, set the request variable INACTIVE_SCHEMAS before the create aggregates statement, for example:

```
set variable INACTIVE_SCHEMAS='"tgt2".."double2"' :
```

Only specify schemas that haven't yet been refreshed as inactive schemas. Don't specify a schema that has already been refreshed or that you've just deleted.

When specifying multiple inactive schemas, use a comma-separated list. Make sure there are no spaces in the list.

The Refreshing Aggregate Clones on Two Targets example show to use double buffering to refresh aggregates on two targets. When you've aggregate clones across multiple aggregate persistence targets, the additional instances are hot-spares that take over the query load while the initial instance is being refreshed. The aggregate clones aren't used for load balancing the incoming queries.

Refreshing Aggregate Clones on Two Targets

Assume that you've the following aggregate clones on targets tgt1 and tgt2:

```
"myfactaggr"
for "FACT_1"("MEASURE_1")
at levels ("INCR"."DIM1_LEVEL1Dim"."DIM1_LEVEL1 Detail")
using connection pool "tgt1"."cp"
in "tgt1".."double1",

"myfactaggr"
for "FACT_1"("MEASURE_1")
at levels ("INCR"."DIM1_LEVEL1Dim"."DIM1_LEVEL1 Detail")
```

```
using connection pool "tgt2"."cp"
in "tgt2".."double2";
```

1. Delete the aggregate clone for the first target:

   ```
   set variable LOGLEVEL=7 : delete aggregates "tgt1".."double1"."myfactaggr";
   ```

2. Create the aggregate for the first target, making sure to specify the second target as an inactive schema so that the data isn't used in the refresh:

   ```
   set variable LOGLEVEL=7, INACTIVE_SCHEMAS='"tgt2".."double2"' : create
   aggregates
   "myfactaggr"
   for "FACT_1"("MEASURE_1")
   at levels ("INCR"."DIM1_LEVEL1Dim"."DIM1_LEVEL1 Detail")
   using connection pool "tgt1"."cp"
   in "tgt1".."double1";
   ```

3. Delete the aggregate clone for the second target:

   ```
   set variable LOGLEVEL=7 : delete aggregates "tgt2".."double2"."myfactaggr";
   ```

4. Create the aggregate for the second target. Because the first target has already been refreshed, don't specify any inactive schemas:

   ```
   set variable LOGLEVEL=7 : create aggregates
   "myfactaggr"
   for "FACT_1"("MEASURE_1")
   at levels ("INCR"."DIM1_LEVEL1Dim"."DIM1_LEVEL1 Detail")
   using connection pool "tgt2"."cp"
   in "tgt2".."double2";
   ```

Create Aggregates on TimesTen Sources

These topics describe configuration steps and features related to aggregate creation on TimesTen sources.

To create aggregates with compressed tables in TimesTen, you must enable COMPRESSED_COLUMNS in the Features tab of the Database dialog in the Oracle BI Administration Tool. See SQL Features Supported by a Data Source.

This section contains the following topics:

- Enable PL/SQL for TimesTen
- Enable Performance Enhancement Features for TimesTen

See Oracle Exalytics for specific instructions on setting up TimesTen sources.

Enable PL/SQL for TimesTen

To create aggregates on TimesTen sources, you must ensure that PL/SQL is enabled for the instance, and that the PL/SQL first connection attribute PLSQL is set to 1.

You can enable PL/SQL at install time, or run the `ttmodinstall` utility to enable it post-install, see *Oracle TimesTen In-Memory Database Installation Guide*.

Enable Performance Enhancement Features for TimesTen

You can disable redo logging and run database checkpoints in the background to enhance performance.

1. Edit the `obis.properties` file located at *`BI_DOMAIN`*`\config\fmwconfig\bienv\obis`.
2. Add the following TimesTen variables:
   - `ORACLE_BI_TT_DISABLE_REDO_LOGGING=1`

     The `ORACLE_BI_TT_DISABLE_REDO_LOGGING=1` element disables redo-logging, enabling faster creation of aggregates.

     Set the `ORACLE_BI_TT_DISABLE_REDO_LOGGING=1` element to *0* (zero), to disable the feature.
   - `ORACLE_BI_TT_BACKGROUND_CHECKPOINT_INTERVAL=`*`10`*

     The `ORACLE_BI_TT_BACKGROUND_CHECKPOINT_INTERVAL` element changes how often TimesTen flushes its data to disk. If the element is missing, the default is every 10 seconds. If you explicitly set `ORACLE_BI_TT_BACKGROUND_CHECKPOINT_INTERVAL=N`, for example, to 10, the flush to disk occurs every *N* seconds, 10 as in this example. If you set `ORACLE_BI_TT_BACKGROUND_CHECKPOINT_INTERVAL` to *0* (zero) background flushing is disabled. Enabling background flushing speeds up creation of aggregates, by avoiding a large blocking flush at the end of the aggregate creation process.
3. Save and close the `obis.properties` file.
4. Restart the Oracle BI Server.

14

Apply Data Access Security to Repository Objects

Learn about data access security available for Oracle BI repository objects.

Data access security controls the right to view and modify data. You can use the following data access security methods:

- Row-level security.
- Object permissions.
- Query limits.

Tasks required to implement data access security include managing users, groups, and application roles, setting up custom LDAP servers, and managing custom authenticators, are covered in *Managing Security for Oracle Analytics Server*.

- Setting up SSL connections.
- Defining application roles and functional groups.

  You must create users and application roles before you can implement data access security.
- Assigning users to application roles and functional groups.
- Setting up LDAP servers.
- Defining and managing custom authenticators.

In the Oracle BI Administration Tool use online mode to implement data access security. If you're using offline mode, see About Applying Data Access Security in Offline Mode. Oracle Analytics Server usage tracking performs data access security auditing. See *Administering Oracle Analytics Server*.

This chapter contains the following topics:

- About Data Access Security
- Row-Level Security
- Set Up Row-Level Security
- Object Permissions
- Set Up Object Permissions
- Overview of User and Application Role Commands
- Rename Application Role Command
- Delete Application Role Command
- Rename Users Command
- Delete Users Command
- Set Query Limits

- About Applying Data Access Security in Offline Mode

About Data Access Security

After developing your metadata repository, you need to set up your data access security architecture.

Data access security accomplishes the following goals:

- Protects business data from unauthorized access.
- Protects your repository metadata such as measure definitions.
- Prevents individual users from damaging overall system performance.

You can implement and enforce row-level data security in both the repository and in the database. Object permissions and query limits are set up in the repository and are enforced only by the Oracle BI Server.

If you choose to implement row-level security in the database, you should also implement object permissions and query limits in the repository. Database-level object restrictions on individual tables or columns, and other objects don't prevent users without access from seeing these repository objects. However, queries against those tables, columns, and other objects fail. You should set up object permissions in the repository to hide these objects from all clients.

Because a variety of clients can connect to the Oracle BI Server, you can't implement or enforce data security in Oracle BI Presentation Services. You can use the Oracle BI Presentation Services set of security controls that enable setting up privileges to access functionality in the Oracle Analytics Server user interface, as well as dashboards and analyses objects. If you only implement security controls in Oracle BI Server, the repository and database are exposed to SQL injection hacker attacks and other security vulnerabilities. You must provide object-level security in the repository to create rules that apply to all incoming clients.

Where to Find Information About Security Tasks

The table lists the location of security task information for Oracle Analytics Server.

Task	Location
Setting up user authentication with the default authentication provider or an alternative authentication provider	Manage Security Using the Default Security Configuration in *Security Guide for Oracle Business Intelligence Enterprise Edition*
Creating and managing users and groups in the default authentication provider	Manage Users and Groups in the Embedded WebLogic LDAP Server in *Managing Security for Oracle Analytics Server*
Creating application roles and managing policies in the default policy store	Managing the Policy Store in *Securing Applications with Oracle Platform Security Services*
Viewing and understanding the default permissions used with application roles in the policy store	Default Permissions in *Security Guide for Oracle Business Intelligence Enterprise Edition*

Task	Location
Applying data access security in offline mode and setting up placeholder application roles	About Applying Data Access Security in Offline Mode
Setting up row-level data security	Set Up Row-Level Security
Setting repository object permissions	Set Up Object Permissions
Setting query limits (governors)	Set Query Limits
Setting up single sign-on (SSO)	Enable SSO Authentication in *Managing Security for Oracle Analytics ServerSecurity Guide for Oracle Business Intelligence Enterprise Edition*
Enabling SSL communication	SSO Configuration in Oracle Business Intelligence in *Managing Security for Oracle Analytics Server*
Managing custom authenticators	Authenticate by Using a Custom Authenticator Plug-In in *Security Guide for Oracle Business Intelligence Enterprise Edition*

Row-Level Security

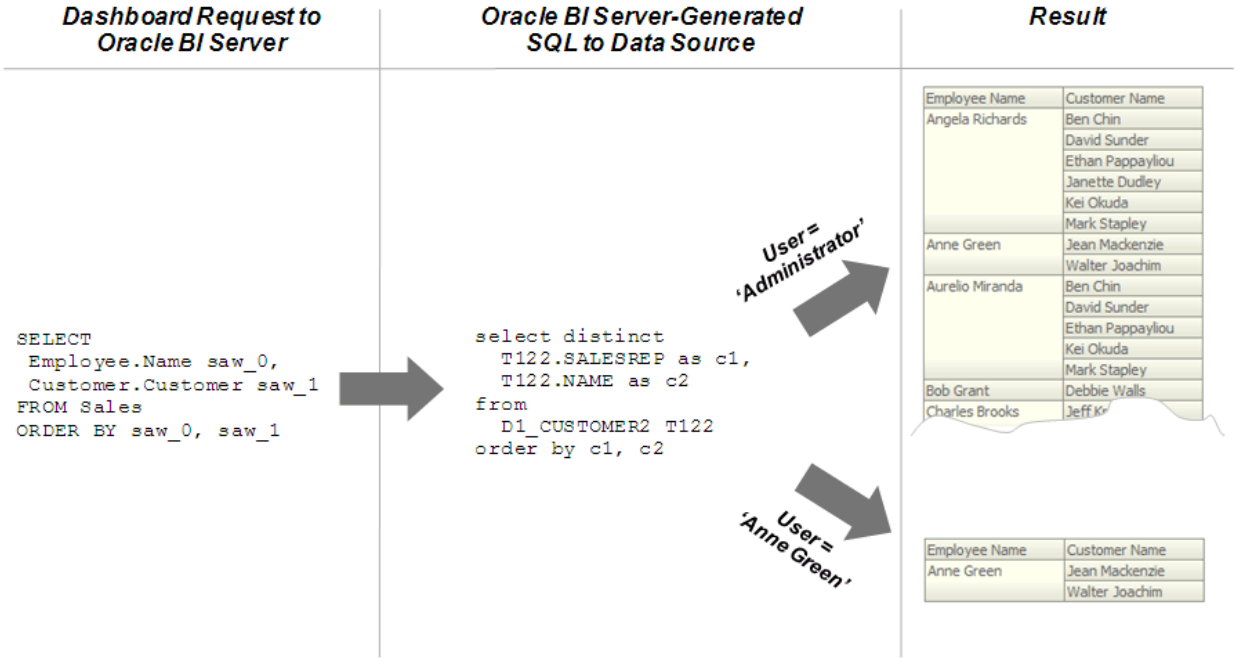
Oracle Analytics Server enables you to configure data security.

Some data sources apply data security policies to determine what data can be queried by an individual user. Data security is described using various terms such as row-level security, data-level security, or Virtual Private Database (VPD) policies. This document uses the term, row-level security.

Some data sources support connections using a privileged user that can impersonate the end user, who is running a query. Oracle Analytics Server connection pools allow parameterization of connection string information, and on-connection and on-query scripts that run prior to data queries. When Oracle Analytics Server connects to a data source by using a privileged user that can impersonate the actual end user, the data source's data security policies apply to the end user queries.

In addition to the connection string and query script configuration, the Oracle Business Intelligence connection pools include a Virtual Private Database (VPD) option. If you use Virtual Private Database (VPD), you can prevent sharing of Oracle Analytics Server query cache between users, because each user needs to retrieve only the data they are permitted to query.

The image shows how row-level security is enforced in the database for queries. The security rules are applied to all incoming clients and can't be breached, even when the Logical SQL query is modified. In this example, the results returned are different depending on the user that generated the query, even though the SQL query generated by the Oracle BI Server is the same. The returned results are based on rules created and enforced in the database.



You must define the users, permissions, and security policies in the database. Refer to your database documentation for more information.

When setting up row-level security consider the following configuration information:

- Row-level security doesn't work when SSO is used, or for any cases that involve impersonation such as Delivers, because the password for the end user isn't available to the Oracle BI Server.
- A connection script can be used to achieve the same functionality for Oracle Database data sources.
- For Essbase or Hyperion Financial Management data sources, the connection pool displays an additional option to implement SSO.

Set Up Row-Level Security

You can choose to set up row-level security in the repository, or in the database.

Implementing row-level security in the repository provides many benefits, including the following:

- All users share the same database connection pool for better performance
- All users share cache for better performance
- You can define and maintain security rules that apply across many federated data sources

Implementing row-level security in the database, in contrast, is good for situations where multiple applications share the same database. When you design and implement row-level security in the database, you should also define and apply object permissions in the repository.

Although it's possible to set up row-level security in both the repository and in the database, you typically don't enforce row-level security in both places unless you've a particular need to do so.

This section contains the following topics:

- Set Up Data Filters in the Repository
- Set Up Row-Level Security in the Database

Data Filters

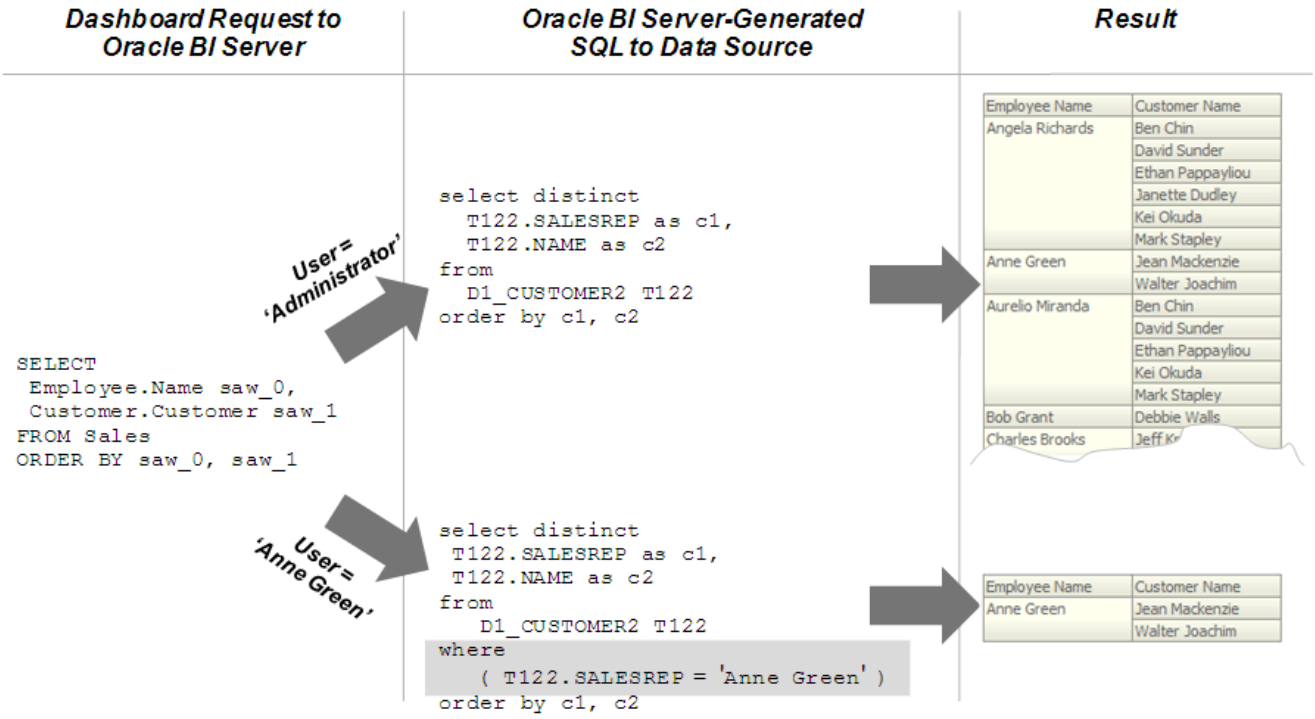
Use the Oracle BI Administration Tool to define data filters on repository objects for specific application roles.

You typically don't set up data filters if you've implemented row-level security in the database, because in this case, your row-level security policies are being enforced by the database rather than the Oracle BI Server.

You can set data filters for objects in the Business Model and Mapping layer and the Presentation layer. Applying a filter on a logical object impacts all Presentation layer objects that use the object. If you set a filter on a Presentation layer object, it's applied to the object along with any other filters that are set on the underlying logical objects.

The image shows how data filter rules are enforced in the Oracle BI Server. The security rules are applied to all incoming clients and can't be breached, even when the Logical SQL query is modified.

In this example, a filter has been applied to an application role. When Anne Green, who is a member of that role, sends a request, the return results are limited based on the filter. Because no filters have been applied to the application roles for the Administrator user, all results are returned. The Oracle BI Server-generated SQL takes into account any data filters that have been defined.



Set Up Data Filters in the Repository

Use these steps to assign data filters to enforce row-level security rules in the repository.

You should always set up data filters for a specific application roles rather than for individual users.

To create filters, you first select objects from subject areas on which you want to apply the filters. Then, you provide the filter expression information for the individual objects. For example, you might want to define a filter like `"Sample Sales"."D2 Market"."M00 Mkt Key" > 5` to restrict results based on a range of values for another column in the table.

If you're in offline mode, and application roles don't appear in the Identity Manager, see About Applying Data Access Security in Offline Mode.

You can also use repository and session variables in filter definitions. Use Expression Builder to include these variables to ensure the correct syntax.

When a repository object such as a logical fact table is accessed by multiple application roles with different levels of access, create functional groups to prevent application roles from viewing data restricted from view by that specific application role. For example, you want your regional sales associates to see the revenue for a quarter in their assigned region, but you don’t want your regional sales associate to see to total segment sales for all of the regions, to avoid exposing sensitive information, you create functional groups with different levels of access as appropriate for the specific application role to the filter. See Specify a Functional Group for an Application Role.

1. In the Oracle BI Administration Tool, open your repository.
2. Select **Manage**, then select **Identity**.
3. In the Identity Manager dialog, double-click an application role.
4. In the Application Role dialog, click **Permissions**.
5. In the Application Role Permissions dialog, click the Data Filters tab.
6. From the **Subject Area** list, select a repository object to use in the filter.
7. Do one of the following:
   - Click **Add** button to browse to locate the object to use, and then click **Select**.
   - Double-click the **Name** field in an empty row, then browse to locate the object, and double-click to select the object.
8. Select the data filter to define, click the **Expression Builder** icon.
9. In the Expression Builder, define the condition using the repository objects and operators.
10. Optional: From the Status list.
11. Click **OK**, then click **OK** again to return to the Identity Manager.

Specify a Functional Group for an Application Role

Use these steps to specify a functional group for application roles with different data access filters on the same repository object, usually a logical fact table.

When there are no functional groups defined, all the security filters applied to a given table, regardless of the associated role, and are combined using the `OR` operator. Using the `OR` operator works in most cases because a user can view a union of all the rows selected by the security filters. For example, consider the following filters:

Role A is assigned the filter, `Product = 'Coke'`

Role B is assigned the filter, `Product = 'Pepsi'`

If a user is given Role A and Role B, then the user can view data for both the Coke and Pepsi products.

When the two security filters from the same table are combined in the query, the filter conditions are combined using the `OR` operator, this is appropriate for most security filters defined on dimension tables, for example:

```
Product = 'Coke' OR Product = 'Pepsi'
```

Using functional groups are necessary is when securing a single fact table, using data filters from different dimensions.

In this example, a fact table is secured using the following filters:

Role A is assigned the filter, `Product = 'Coke'`

Role B is assigned the filter, `Product = 'Pepsi'`

Role C is assigned the filter, `Region = 'Southwest'`

If you don't use functional groups, a user with roles A, B, and C would have all three filter conditions combined in the query using the `OR` operator, for example:

```
(Product = 'Coke' OR Product = 'Pepsi' OR Region = 'Southwest')
```

Combining the results of Role A, B, and C doesn't make sense because Product and Region are independent dimensions. Combining data filters from different dimensions using `OR` operator provides the user access to more data values than the user should view.

In this example, the user can see data for all products within the Southwest region as well as data for all regions within the Pepsi and Coke products.

To get the expected behavior, that is allowing the user to see data only for the Pepsi and Coke products within the Southwest region, you need to change the filter to combine the product filters with the region filter using the `AND` operator, for example:

```
(Product = 'Coke' OR Product = 'Pepsi') AND (Region = 'Southwest')
```

To achieve this using functional groups, assign the security filters to functional groups as follows:

Role A is assigned the filter, `Product = 'Coke' with functional group "Product"`

Role B is assigned the filter, `Product = 'Pepsi' with functional group "Product"`

Role C is assigned the filter, `Region = 'Southwest' with functional group "Region"`

All the filters in the same functional group are combined using the `OR` operator and all sets of filters in different functional groups are combined using the `AND` operator. By choosing the functional groups associated with each security filter, you can control how the filters are combined using the `OR` and `AND` operators.

To create a data filter, see Set Up Data Filters in the Repository.

1. In the Oracle BI Administration Tool, from **Manage**, select **Identity**.
2. In the Identity Manager, double-click an application role.
3. In Application Role, click **Permission**s.
4. In Application Role Permissions, click the Data Filters tab.
5. In the Data Filter tab, select the filter to assign to a functional group.
6. In the **Functional Group** column, select an existing group, or typing the name of a new group to use.
7. Click **OK**.

Set Up Row-Level Security in the Database

If multiple applications share the same database, it's good to implement row-level security in the database.

This procedure is applicable for making database queries through RPD queries (Common Enterprise Information Model), if you have configured the database to use the Virtual Private Database (VPD) feature.

1. Open your repository in the Administration Tool.
2. Double-click the connection pool associated with the database for which you want to set up database-level security.
3. In the General tab of the Connection Pool dialog, select **Shared logon**, and then enter the user credentials.

   The user details provided in the shared logon will be used to connect to the database.

   You can use the database session context to pass end user identity to the database. Use a connection pool script to set up session context.
4. Click **OK** in the Connection Pool dialog.
5. Double-click the database object for which you want to set up database-level security.
6. In the Database dialog, select **Virtual Private Database**. Selecting this option ensures that the Oracle BI Server protects cache entries for each user. Oracle BI Server matches a list of security-sensitive variables to each prospective cache hit. Cache hits would only occur on cache entries that included and matched all security-sensitive variables.
7. Click **OK** in the Database dialog.

After you've set up row-level security for the data the database, you can set up object permissions in the repository for Presentation layer or other objects. You can also set query limits (governors). See Set Up Object Permissions and Set Query Limits.

Object Permissions

You can set up object permissions in your repository to control access to Presentation layer and Business Model and Mapping layer objects

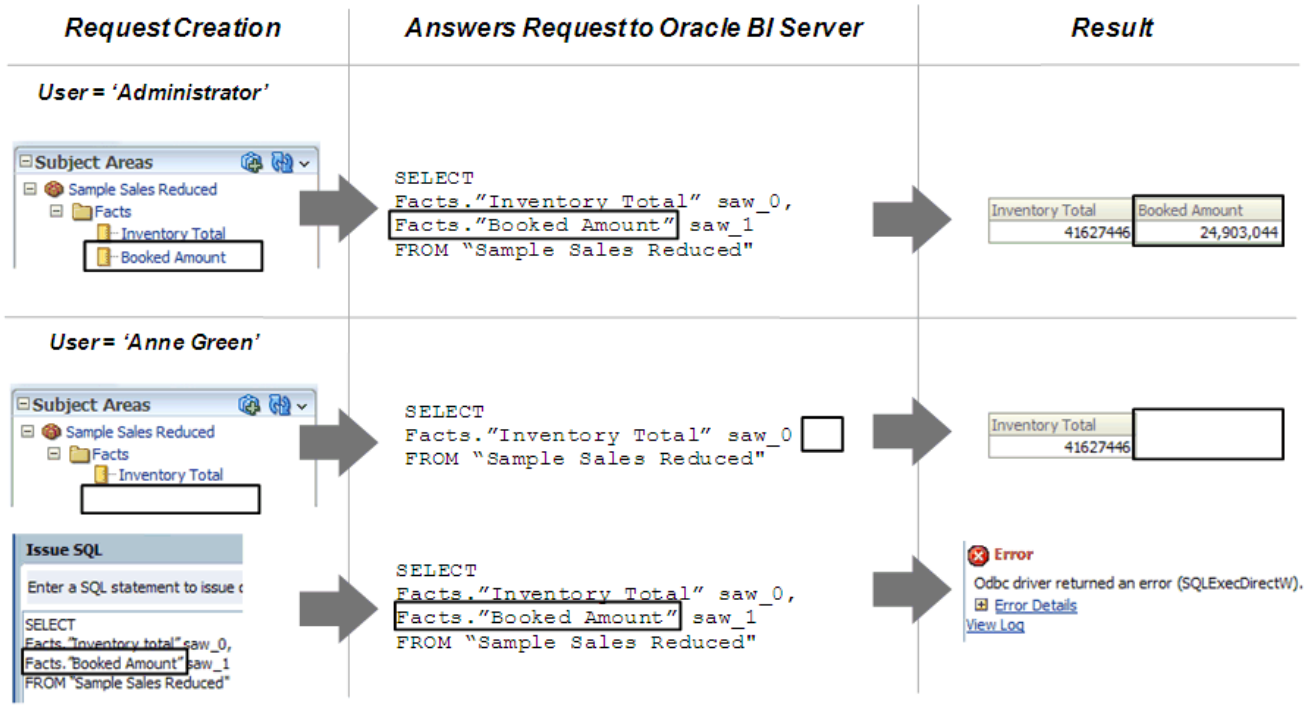
You set object permissions using the Oracle BI Administration Tool.

To set up object permissions:

- Set the data access for specific application roles.
- Specify functional groups when multiple application roles have different levels of access to the same object.
- Select individual objects in the Presentation layer.

Set up object permissions for application roles when you want to define data access permissions for a set of objects that are common to users assigned the specific application role. You should set up object permissions for specific application roles rather than for individual users to simplify data access management.

The following image shows how object permissions can restrict users from viewing specific repository object. Security rules are applied to all incoming client queries, and can't be breached, even when the Logical SQL query is modified. In this example, the Administrator application role has been granted access to the Booked Amount column allowing the Administrator to view the returned results. The user, Anne Green, who isn't a member of an application role with access to the Booked Amount column, can't see the column in the Subject Area pane of Answers . Even if the query is modified, results aren't returned for the Booked Amount column because of the application role-based object permissions have been set.



- If an application role has permissions on an object from multiple sources, for example, explicitly and through one or more additional application roles, the permissions are applied based on the order of precedence.
- If you explicitly deny access to an object that has child objects, users who are members of the individual application role are denied access to the child objects. For example, if you explicitly deny access to a particular logical table, you're implicitly denying access to all of the logical columns associated with that table.
- Object permissions don't apply to repository and session variables, so values in these variables aren't secure. Anyone who knows or can guess the name of the variable can use it in an expression in Answers or in a Logical SQL query. Don't put sensitive data like passwords in session or repository variables.
- You can control the level of privilege is granted by default to the AuthenticatedUser application role. The AuthenticatedUser is the default application role associated with new repository objects.

  The AuthenticatedUser application role means any authenticated user. The AuthenticatedUser application role is internal to the Oracle BI Repository. The AuthenticatedUser application role appears in the Permissions dialog for connection pools and Presentation layer objects. The AuthenticatedUser doesn't appear in the list of application roles in the Identity Manager.

  Update the `DEFAULT_PRIVILEGES` parameter in the `NQSConfig.INI` file. See Security Section Parameters in *Administering Oracle Analytics Server*.

Set Up Object Permissions

Use these steps to set up object permissions for individual application roles in your repository to control access to Presentation layer and Business Model and Mapping layer objects.

Application roles aren't displayed if you're using offline mode unless you've first modified them in online mode. See About Applying Data Access Security in Offline Mode.

1. Open your repository in the Administration Tool.
2. Select **Manage**, then select **Identity**.
3. In the Identity Manager dialog, in the tree pane, select **BI Repository**.
4. In the right pane, select the Application Roles tab, then double-click the application role for which you want to set object permissions.
5. In the Application Role dialog, click **Permissions**.
6. In the User/Application Role Permissions dialog, in the Object Permissions tab, do one of the following to select an object:
   - Click the **Add** button, locate the object, and then click **Select**.
   - Click the **Name** field in an empty row, locate the object, and then click **Select**
7. Assign the appropriate permission for each object. You can choose one of the following options:
   - **Read:** Only allows read access to this object.
   - **Read/Write:** Provides both read and write access to this object.
   - **No Access:** Explicitly denies all access to this object.

8. Click **OK**, then click **OK** again to return to the Identity Manager.

About Permission Inheritance for Users and Application Roles

Users can have explicitly granted permissions. They can also have permissions granted through membership in application roles, that in turn can have permissions granted through membership in other application roles.

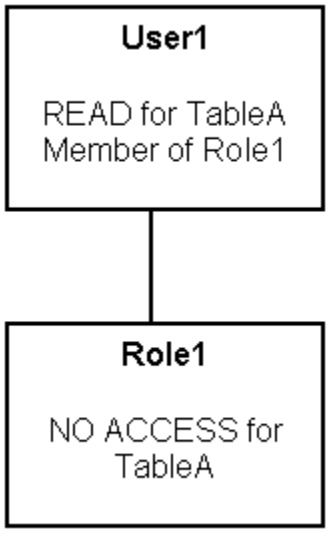
Permissions granted explicitly to a user have precedence over permissions granted through application roles, and permissions granted explicitly to the application role take precedence over any permissions granted through other application roles.

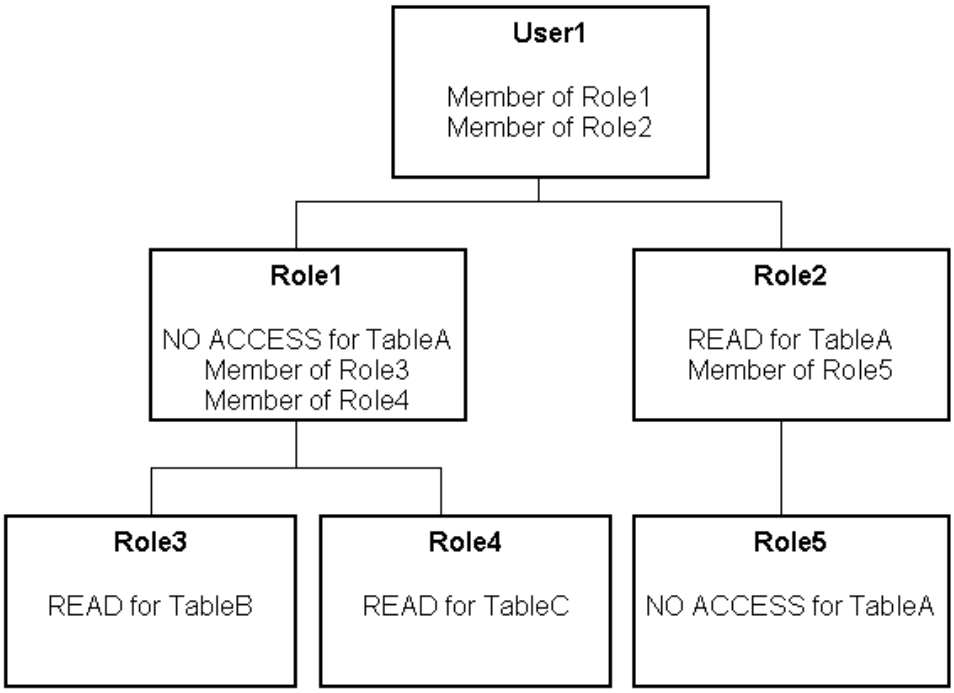
If there are multiple application roles acting on a user or application role at the same level with conflicting security attributes, then the user or application role is granted the least restrictive security attribute. Oracle currently requires that the application role with access to an object also have access to the object's container. For example, if ApplicationRole 1 has permission to access Column A, which is part of Table B, then ApplicationRole1 must also have permission to access Table B. Any explicit permissions acting on a user take precedence over any permissions on the same objects granted to that user through application roles.

In previous releases, the application role didn't require access to an object's container, as described above. To revert to this behavior, go to the Oracle BI Server machine and create environment variable `OBIS_SECURITY_10g_COMPATIBLE` and set it to 1.

Filter definitions, however, are always inherited. For example, if User1 is a member of Role1 and Role2, and Role1 includes a filter definition but Role2 doesn't, the user inherits the filter definition defined in Role1.

You should always define object permissions for application roles rather than for individual users.





These are the resulting permissions:

- User1 is a direct member of Role1 and Role2, and is an indirect member of Role3, Role4, and Role5.
- Because Role5 is at a lower level of precedence than Role2, its denial of access to TableA is overridden by the `READ` permission granted through Role2. The result is that Role2 provides `READ` permission on TableA.
- The resultant permissions from Role1 are `NO ACCESS` for TableA, `READ` for TableB, and `READ` for TableC.
- Because Role1 and Role2 have the same level of precedence and because the permissions in each cancel the other out (Role1 denies access to TableA, Role2 allows access to TableA), the less restrictive level is inherited by User1. In other words, User1 has `READ` access to TableA.
- The total permissions granted to User1 are `READ` access for TableA, TableB, and TableC.

Permission Inheritance 1

You might have a user (User1) who is explicitly granted permission to read a given table (TableA). Suppose also that User1 is a member of Role1, and Role1 explicitly denies access to TableA. The resultant permission for User1 is to read TableA.

Because permissions granted directly to the user take precedence over those granted through application roles, User1 has the permission to read TableA.

Overview of User and Application Role Commands

Learn about commands that you can use with the repository and the Oracle BI Presentation Catalog.

You can use the following commands to update application roles and users stored in the repository and the Oracle BI Presentation Catalog:

- Rename Application Role Command
- Delete Application Role Command
- Rename Users Command
- Delete Users Command

The remaining upload and update commands, for example, Upload Repository Command and Update Repository Variables Command, only update the repository.

The application roles and users update commands use two plugins, the Oracle BI repository plugin, which updates the application roles and users in the repository, and the WEBCAT plugin, which updates application roles and the users in the Oracle BI Presentation Catalog.

These plugins function separately, and therefore the failure of one doesn't impact the other. In the event of a partial failure, or one of the two plugins failing, Oracle recommends that you address the root cause of the failure and then re-run the command as you initially ran it. Reapplying the successful plugin has no impact on the results, but re-running the command reruns the failed plugin.

By default, the application roles and users update commands run the two plugins, and the order in which they're run is Oracle BI repository and then WEBCAT. However, the commands include the `-L` option which allows you to specify an individual plugin or to reverse the default order in which the plugins are run. Run the commands in the default order. You might need to run only one plugin or reverse the order of the plugins.

Rename Application Role Command

Use the `renameapproles` command to upload a JSON file containing information about the application roles that you want to rename for a specific server instance.

You run the `renameapproles` command through a launcher script, `datamodel.sh` on Linux and `datamodel.cmd` on Windows.

If the domain is installed in default folder then the location of the launcher script looks like the following:

Oracle_Home`/user_projects/domains/`*Domain_Name*`/bitools/bin/datamodel.sh` or `datamodel.cmd` on Windows.

If the client install doesn't have domain names, the launcher script location is as follows:

```
Oracle_Home\bi\bitools\bin\datamodel.cmd
```

See What You Need to Know Before Using the Command and Overview of User and Application Role Commands.
Note that if you rename an application role in Oracle BI repository (RPD) and Oracle BI Presentation Catalog (WEBCAT), make sure that both the old role and the new role are present in Enterprise Manager.

Syntax

The `renameapproles` command takes the following parameters:

```
renameapproles -T inputfile.json[-L plugin list] -SI service instance -U cred
username[-P cred password] [-S hostname] [-N <port number] [-SSL] [-H]
```

Where

`T` specifies the name of the JSON input file containing the application role name changes for the server instance.

`SI` specifies the name of the service instance.

`L` specifies a single plugin to run or to reverse the default plugin processing order. The plugins determine where the system applies the updates: to the repository, the Oracle BI Presentation Catalog, or both. See Overview of User and Application Role Commands.

The following options are for `L`:

- `RPD`: Specify this option to rename application roles in the repository, only.
- `WEBCAT`: Specify this option to rename application roles in the Oracle BI Presentation Catalog. For example, you must use the `-L` WEBCAT option when renaming application roles in the Oracle BI Presentation Catalog.
- `WEBCAT,RPD`: Specify this option to reverse the default plugin run order.

  The default plugin run order is Oracle BI repository (RPD) and then Oracle BI Presentation Catalog (WEBCAT).
- Omit this option to run the plugins in their default order, which is Oracle BI repository (RPD) and then Oracle BI Presentation Catalog.

`U` specifies a valid user's name to be used for authentication.

`P` specifies the password corresponding to the user's name that you specified for `U`. If you don't supply the password, you're prompted for the password when the command is run. Oracle recommends that you include a password in the command only if you're using automated scripting to run the command.

`S` specifies the host name. Only include this option when you're running the command from a client installation.

`N` specifies the port number. Only include this option when you're running the command from a client installation.

`SSL` specifies to use SSL to connect to the Oracle WebLogic Server to run the command. Only include this option when you're running the command from a client installation.

`H` displays the usage information and exits the command. Use `-H` or run `.sh` without any parameters to display the help content.

Example

```
datamodel.sh renameapproles -T approlenames.json -SI bi -U weblogic -P
password -S server1.example.com -N 7777 -SSL
```

Creating a JSON Rename Application Role Input File

Use the following syntax to create the JSON rename application role input file.

```
{
    "Title":"Target Application Roles",
    "App-Roles":[
          { "oldname":"<current_approle1>", "newname":"<new_approle1>" },
          { "oldname":"<current_approle2>", "newname":"<new_approle2>" },
          { "oldname":"<current_approle3>", "newname":"<new_approle3>" }
   ]
}
```

Delete Application Role Command

Use the `deleteapproles` command to upload a JSON file containing a list of application roles that you want to delete from a specific server instance.

You run the utility through a launcher script, `datamodel.sh` on Linux and `datamodel.cmd` on Windows.

If the domain is installed in default folder then the location of the launcher script looks like the following:

Oracle_Home`/user_projects/domains/`*Domain_Name*`/bitools/bin/datamodel.sh` or `datamodel.cmd` on Windows.

If the client install doesn't have domain names, the launcher script location is as follows:

Oracle_Home`\bi\bitools\bin\datamodel.cmd`

See What You Need to Know Before Using the Command and Overview of User and Application Role Commands.

Syntax

The `deleteapproles` command takes the following parameters:

`deleteapproles -T` *inputfile.json*`[-L` *plugin list*`] -SI` *service_instance* `-U` *cred_username*`[-P` *cred_password*`] [-S` *hostname*`] [-N` *port_number*`] [-SSL} [-H]`

Where

`T` specifies the name of the JSON input file containing the application roles to be deleted from the server instance.

`L` specifies a single plugin to run or to reverse the default plugin processing order. The plugins determine where the system applies the updates: to the repository, the Oracle BI Presentation Catalog, or both. For Oracle BI repository file (RPD) and WEBCAT plugin usage information, see Overview of User and Application Role Commands.

The following options are available for `L`:

- `RPD`: Specify this option to delete application roles in the repository, only.
- `WEBCAT`: Specify this option to delete application roles in the Oracle BI Presentation Catalog, only. You must use the `-L` WEBCAT option when deleting application roles.

- `WEBCAT,RPD`: Specify this option to reverse the default plugin run order. The default plugin run order is Oracle BI repository (RPD) and then Oracle BI Presentation Catalog (WEBCAT).
- Omit this option to run the plugins in their default order.

`SI` specifies the name of the service instance.

`U` specifies a valid user's name to be used for authentication.

`P` specifies the password corresponding to the user's name that you specified for `U`. If you don't supply the password, then you're prompted for the password when the command is run. Oracle recommends that you include a password in the command only if you're using automated scripting to run the command.

`S` specifies the host name. Only include this option when you're running the command from a client installation.

`N` specifies the port number. Only include this option when you're running the command from a client installation.

`SSL` specifies to use SSL to connect to the Oracle WebLogic Server to run the command. Only include this option when you're running the command from a client installation.

`H` displays the usage information and exits the command. Use `-H` or run `.sh` without any parameters to display the help content.

Example

```
datamodel.sh deleteapproles -T approlenames.json -SI bi -U weblogic -P
password -S server1.example.com -N 7777 -SSL
```

Creating a JSON Delete Application Role Input File

Use the following syntax to create the JSON delete application role input file.

```
{
    "Title":"Target Application Roles",
    "App-Roles":[
         { "name":"<approle1>" },
         { "name":"<approle2>" },
         { "name":"<approle3>" }
   ]
}
```

Rename Users Command

Use the `renameusers` command to upload a JSON file containing a list of information about the users that you want to rename for a specific server instance.

You run the utility through a launcher script, `datamodel.sh` on Linux and `datamodel.cmd` on Windows.

If the domain is installed in default folder then the location of the launcher script looks like the following:

Oracle_Home`/user_projects/domains/`*Domain_Name*`/bitools/bin/datamodel.sh` or `datamodel.cmd` on Windows

If the client install doesn't have domain names, the launcher script location is as follows:

Oracle_Home\bi\bitools\bin\datamodel.cmd

See What You Need to Know Before Using the Command.

See Overview of User and Application Role Commands.

Syntax

The `renameusers` command takes the following parameters:

```
renameusers -T usernames.json[-L plugin list] -SI service instance -U cred
username[-P cred password] [-S hostname] [-N port number] [-SSL] [-H]
```

Where

`T` specifies the name of the JSON input file containing the user name changes for the server instance.

`L` specifies a single plugin to run or to reverse the default plugin processing order. The plugins determine where the system applies the updates: to the repository, the Oracle BI Presentation Catalog, or both.

The following options are for `L`:

- `RPD`: Specify this option to rename users in the Oracle BI repository, only.
- `WEBCAT`: Specify this option to rename users in the Oracle BI Presentation Catalog. For example, you must use the `-L WEBCAT` option when renaming users in the Oracle BI Presentation Catalog.
- `WEBCAT,RPD`: Specify this option to reverse the default plugin run order. The default plugin run order is Oracle BI repository (RPD) and then Oracle BI Presentation Catalog (WEBCAT).
- Omit this option to run the plugins in their default order.

`SI` specifies the name of the service instance.

`U` specifies a valid user's name to be used for authentication.

`P` specifies the password corresponding to the user's name that you specified for `U`. If you don't supply the password, then you're prompted for the password when the command is run. For security purposes, Oracle recommends that you include a password in the command only if you're using automated scripting to run the command.

`S` specifies the host name. Only include this option when you're running the command from a client installation.

`N` specifies the port number. Only include this option when you're running the command from a client installation.

`SSL` specifies to use SSL to connect to the Oracle WebLogic Server to run the command. Only include this option when you're running the command from a client installation.

`H` displays the usage information and exits the command. Use `-H` or run `.sh` without any parameters to display the help content.

Example

```
datamodel.sh renameusers -T usernames.json -SI bi -U weblogic -P password
-S server1.example.com -N 7777 -SSL
```

Creating a JSON Rename Users Input File

Use the following syntax to create the JSON rename users input file.

```
{
    "Title":"Target Users",
    "Users":[
          { "oldname":"<current_user1>", "newname":"<new_user1>" },
          { "oldname":"<current_user2>", "newname":"<new_user2>" },
          { "oldname":"<current_user3>", "newname":"<new_user3>" }
    ]
}
```

Delete Users Command

Use the `deleteusers` command to upload a JSON file containing a list of users that you want to delete from a specific server instance.

You run the utility through a launcher script, `datamodel.sh` on Linux and `datamodel.cmd` on Windows.

If the domain is installed in default folder then the location of the launcher script looks like the following:

Oracle_Home`/user_projects/domains/`*Domain_Name*`/bitools/bin/datamodel.sh` or `datamodel.cmd` on Windows

If the client install doesn't have domain names, the launcher script location is as follows:

Oracle_Home`\bi\bitools\bin\datamodel.cmd`

See What You Need to Know Before Using the Command and Overview of User and Application Role Commands.

Syntax

The `deleteusers` command takes the following parameters:

```
deleteusers -T usernames.json [-L plugin list] -SI service_instance-U
cred_username[-P cred_password] [-S hostname] [-N port_number] [-SSL]
[-H]
```

Where

`T` specifies the name of the JSON input file containing the users to be deleted from the server instance. See the syntax in *Create a JSON Delete Users Input File* for information about the correct syntax for the application role input file.

`L` specifies a single plugin to run or to reverse the default plugin processing order. The plugins determine where the system applies the updates: to the repository, the Oracle BI Presentation Catalog, or both.

The following options are for `L`:

- `RPD`: Specify this option to delete users in the repository, only.
- `WEBCAT`: Specify this option to delete users in the Oracle BI Presentation Catalog. For example, you must use the`-L WEBCAT` option when deleting users from the Oracle BI Presentation Catalog.
- `WEBCAT,RPD`: Specify this option to reverse the default plugin run order. The default plugin run order is repository (RPD) and then Oracle BI Presentation Catalog (WEBCAT).
- Omit this option to run the plugins in their default order, which is repository (RPD) then Oracle BI Presentation Catalog (WEBCAT).

`SI` specifies the name of the service instance.

`U` specifies a valid user's name to be used for authentication.

`P` specifies the password corresponding to the user's name that you specified for `U`. If you don't supply the password, then you're prompted for the password when the command is run. Oracle recommends that you include a password in the command only if you're using automated scripting to run the command.

`S` specifies the host name. Only include this option when you're running the command from a client installation.

`N` specifies the port number. Only include this option when you're running the command from a client installation.

`SSL` specifies to use SSL to connect to the Oracle WebLogic Server to run the command. Only include this option when you're running the command from a client installation.

`H` displays the usage information and exits the command. Use `-H` or run `.sh` without any parameters to display the help content.

Example

```
data-model-cmd.sh deleteusers -T usernames.json -SI bi -U weblogic -P password -
S server1.us.example.com -N 777 -SSL
```

Create a JSON Delete Users Input File

Use the following syntax to create the JSON delete users input file.

```
{
    "Title":"Target Users",
    "Users":[
         { "name":"<user1>" },
         { "name":"<user2>" },
         { "name":"<user3>" }
    ]
}
```

Set Query Limits

You can manage the query environment by setting query limits (governors) in the repository for particular application roles.

You can limit queries by the number of rows received, by maximum run time, and by restricting to particular time periods. You can also allow or disallow direct database requests or the Populate privilege.

You should always set query limits for particular application roles rather than for individual users.

This section contains the following topics:

- Access the Query Limits Functionality in the Administration Tool
- Limit Queries By the Number of Rows Received
- Limit Queries By Maximum Run Time and Restricting to Particular Time Periods
- Allow or Disallow Direct Database Requests
- Allow or Disallow the Populate Privilege

Access the Query Limits Functionality in the Administration Tool

Learn how to access the Query Limits tab of the User/Application Role Permissions dialog.

If you're in offline mode, no application roles appear in the list unless you've first modified them in online mode, see About Applying Data Access Security in Offline Mode.

1. Open your repository in the Administration Tool to access the query limits functionality for an application role.
2. Select **Manage**, then select **Identity**.
3. In the Identity Manager dialog, in the tree pane, select **BI Repository**.
4. In the right pane, select the Application Roles tab, then double-click the application role for which you want to set query limits.
5. In the Application Role dialog, click **Permissions**.
6. In the User/Application Role Permissions dialog, click the Query Limits tab.

Limit Queries By the Number of Rows Received

You can control runaway queries by limiting queries to a specific number of rows.

Any query limits you set should exceed the Presentation Server settings for Maximum Number of Rows Processed when Rendering a Table View and Maximum Number of Rows to Download by at least 500 to avoid error messages. If you choose to impose data source rows limits on certain users or Application Roles using the repository Max Rows query limits setting, then those users may receive Max Row Limit Exceeded messages.

See Use Fusion Middleware Control to Set Configuration Options for Data in Tables and Pivot Tables and Use Fusion Middleware Control to Set the Maximum Number of Rows Processed to Render a Table in *Administering Oracle Analytics Server*.

The options for Status Max Rows are:

- **Enable:** This limits the number of rows to the value specified. If the number of rows exceeds the **Max Rows** value, the query is terminated.

- **Disable:** Disables any limits set in the **Max Rows** field.
- **Warn:** Doesn't enforce limits, but logs queries that exceed the set limit in the Query log.
- **Ignore:** Limits are inherited from the parent application role. If there is no row limit to inherit, no limit is enforced.

Follow the steps in Accessing the Query Limits Functionality in the Administration Tool to access the Query Limits tab.

1. In the **Max Rows** column, type the maximum number of rows for users to retrieve from each source database object.
2. In the **Status Max Rows** field, select an option for each database.
3. Click **OK**, then click **OK** again to return to the Identity Manager.

Limit Queries By Maximum Run Time and Restricting to Particular Time Periods

You can forbid queries during certain time periods, or you can specify the maximum time a query can run on a database.

If you don't select a particular time period, access rights remain unchanged. If you allow or disallow access explicitly in one or more application roles, users are granted the least restrictive access for the defined time periods. For example, if a user is a member of an application role that's explicitly allowed access all day on Mondays, but that user also belongs to another application role that's disallowed access during all hours of every day, then the user has access on Mondays only.

1. Follow the steps in Access the Query Limits Functionality in the Administration Tool to access the Query Limits tab.
2. To specify the maximum time a query can run on a database, in the **Max Time (Minutes)** column, enter the maximum number of minutes you want queries to run on each database object. Then, in the **Status Max Time** field, select one of the following options for each database:
   - **Enable:** This limits the time to the value specified.
   - **Disable:** Disables any limits set in the **Max Time** field.
   - **Warn:** Doesn't enforce limits, but logs queries that exceed the set time limit in the Query log.
   - **Ignore:** Limits are inherited from the parent application role. If there is no time limit to inherit, no limit is enforced.
3. To restrict access to a database during particular time periods, in the **Restrict** column, click the **Ellipsis** button. Then, in the Restrictions dialog, perform the following steps:
   a. To select a time period, click the start time and drag to the end time.
   b. To explicitly grant access, click **Allow**.
   c. To explicitly deny access, click **Disallow**.
   d. Click **OK**.
4. Click **OK**, then click **OK** again to return to the Identity Manager.

Allow or Disallow Direct Database Requests

Use this task to allow or disallow the ability to run direct database requests for a particular application role.

For the selected role, this privilege overrides the `Allow direct database requests by default` property for the database object in the Physical layer.

The options for the **Execute Direct Database Requests** field are:

- *Allow*
  Explicitly grants the ability to run direct database requests for this database.
- *Disallow*
  Explicitly denies the ability to run direct database requests for this database.
- *Ignore*
  Limits are inherited from the parent application role. If there is no limit to inherit, then direct database requests are allowed or disallowed based on the `Allow direct database requests by default` property for the database object.

Follow the steps in Access the Query Limits Functionality in the Administration Tool to access the Query Limits tab.

1. In the Query Limits tab for each database object, in the **Execute Direct Database Requests** field, select an option.
2. Click **OK**, then click **OK** again to return to the Identity Manager.

Allow or Disallow the Populate Privilege

When a criteria block is cached, the Populate stored procedure writes the Cache/ Saved Result Set value to the database.

You can grant or deny this Populate privilege to particular application roles.

For the selected application role, this privilege overrides the `Allow populate queries by default` property for the database object in the Physical layer.

The options for the **Populate Privilege** field are:

- *Allow*
  Explicitly grants the Populate privilege for this database. For all Marketing data warehouses, select Allow.
- *Disallow*
  Explicitly denies the Populate privilege for this database.
- *Ignore*
  Limits are inherited from the parent application role. If there is no limit to inherit, then the Populate privilege is allowed or disallowed based on the `Allow populate queries by default` property for the database object.

Follow the steps in Access the Query Limits Functionality in the Administration Tool to access the Query Limits tab.

1. For each database object, in the **Populate Privilege** field, select an option.
2. Click **OK**, and then click **OK** again to return to the Identity Manager.

About Applying Data Access Security in Offline Mode

You should perform data access security tasks in the Oracle BI Administration Tool in online mode.

The Administration Tool doesn't store users in the repository, and you can't create a query that returns repository users.

When you open the repository In online mode, you can retrieve the latest list of application roles from the policy store by selecting **Action**, then selecting **Synchronize Application Roles** in the Identity Manager.

Set Up Placeholder Application Roles for Offline Repository Development

Application roles are created and managed in the policy store using the Oracle WebLogic Administration Console and Fusion Middleware Control.

These application roles are displayed in the Administration Tool in online mode so that you can use them to set data filters, object permissions, and query limits for particular roles. The application roles in the policy store are retrieved by the Oracle BI Server when it starts.

In some cases, you may want to proceed with setting up data access security in your repository for application roles that haven't yet been defined in the policy store. You can do this by creating placeholder application roles in the Administration Tool, then proceeding with setting up data access security in the repository.

If you create placeholder application roles in the Administration Tool, you must eventually add them to the policy store. Run a consistency check in online mode to identify application roles that have been defined in the Administration Tool, but that haven't yet been added to the policy store. Be sure to use the same name in the policy store that you used for the placeholder role in the Administration Tool.

Use caution when defining and using placeholder roles. If you make changes to a role in offline mode that also exists in the policy store, the changes are overwritten the next time you connect to the Oracle BI Server.

1. Open your repository in the Administration Tool.
2. Select **Manage**, then select **Identity**.
3. In the Identity Manager dialog, select **Action**, select **New**, and then select **Application Role**.
4. In the Application Role dialog, provide the following information:
   - **Name:** Provide a name for the role.
   - **Display Name:** Enter the display name for the role.
   - **Description:** (Optional) provide a description of this application role.
   - **Members:** Use the **Add** and **Remove** buttons to add or remove users and other application roles as appropriate.
   - **Permissions:** Set object permissions, data filters, and query limits for this application role as appropriate. Refer to the other sections in this chapter for detailed information.
5. Click **OK** to return to the Identity Manager.

Perform the steps in Run the Consistency Check Manager. Record any entries related to application roles, then add the appropriate roles to the policy store as appropriate. See Use Tools to Configure Security in Oracle Business Intelligence in the *Managing Security for Oracle Analytics Server* for information about adding application roles to the policy store. You can check an individual application role by right-clicking the application role in the Identity Manager dialog and then selecting **Check Consistency**.

15

Complete Oracle BI Repository Setup

This chapter explains how to perform final Oracle BI repository setup tasks like configuring for Oracle Scorecard and Strategy management, saving the repository and checking consistency, testing the repository, and uploading the repository using the upload repository command.
This chapter contains the following topics:

- Save the Repository and Check Consistency
- Use nqcmd to Test and Refine the Repository
- Upload Repository Command
- Make the Repository Available for Queries
- Create Data Source Connections to the Oracle BI Server for Client Applications
- Publish to the User Community

Save the Repository and Check Consistency

In offline editing, remember to save your repository from time to time.

You can save a repository in offline mode even though the business models may be inconsistent.

To determine if business models are consistent, use the Check Consistency command to check for compilation errors. You can check for errors in the whole repository or in a particular logical business model by selecting a business model and then selecting **Check Consistency** from the right-click menu.

The consistency check analyzes the repository for certain kinds of errors and inconsistencies. For example, the consistency check finds any logical tables that don't have logical sources configured or any logical columns that aren't mapped to physical sources, checks for undefined logical join conditions, determines whether any physical tables referenced in a business model aren't joined to the other tables referenced in the business model, and checks for existence of a subject area for each business model.

Passing a consistency check doesn't guarantee that a business model is constructed correctly, but it does rule out many common problems.

When you check for consistency, any errors or warnings that occur are displayed in a dialog. Correct any errors and check for consistency again, repeating this process until there are no more errors. An error message indicates a problem that must be corrected. A warning message identifies a possible problem. See Check the Consistency of a Repository or a Business Model.

After upgrading from a previous software version and checking the consistency of your repository, you might observe messages that you hadn't received in previous consistency checks. This typically indicates inconsistencies that had been undetected before the upgrade, not new errors.

- In the Oracle BI Administration Tool with a repository open, from the File menu, select **Check Global Consistency**.

Use nqcmd to Test and Refine the Repository

When your repository is complete, you can run sample queries against it to test that it's created properly.

Correct any problems you find and test again, repeating this process until you're satisfied with the results.

You can use the Oracle BI Server utility `nqcmd` to run test queries against the repository. The utility connects using an Oracle BI Server ODBC DSN. The Oracle BI Server must be running to use `nqcmd`.

The `nqcmd` utility is available on Windows and Linux systems.

This utility is intended for sanity testing. For heavier load testing, use Answers or another client. Queries with many thousands of rows don't work with `nqcmd`.

Although you can use `nqcmd` to run queries against other ODBC data sources, this section only describes how to use this utility to query the Oracle BI Server.

> **Tip:**
>
> On Windows, you can see the available local ODBC data source names in **Data Sources (ODBC)** available in **Administrative Tools**. The System DSN tab displays a list of the available DSNs, for example, *AnalyticsWeb_coreapplications*.

You can pass a text file with SQL statements to the utility (script mode), or you can enter SQL at the command line (interactive mode). Queries are run against the default subject area, unless the object names used in the query are fully qualified.

See nqcmd Command Line Arguments.

1. On Windows launch `nqcmd` from the following location:

   BI_DOMAIN`/bitools/bin`

2. At `nqcmd`, type the argument options to use, for example:

```
nqcmd -dmy_dsn -umy_username [-pmy_password] -ssql_input_file -
omy_result_file
```

nqcmd Command Line Arguments

Review the table to learn about the valid the command-line arguments for `nqcmd`.

If you run `nqcmd` in interactive mode rather than script mode, that is, if you don't pass a SQL input file, `nqcmd` shows a menu of options after you provide the data source name and user credentials. Although many options are shown, you only use Q, T, and C against the Oracle BI Server.

Use `Q` to type a query at the command line. You must enter the query on a single line, and you can't use a semicolon as a delimiter. Pressing **Enter** sends the SQL to the Oracle BI Server.

Use `T` to browse presentation tables, or `C` to browse presentation columns. The utility prompts you for catalog pattern, user pattern, table pattern, and table type pattern before returning results.

For catalog pattern, enter the subject area that contains the tables you want to see. For table pattern, enter the specific table. You can enter percent (%) to see all subject areas or all tables, use `%` with other characters to replace a set of characters, or use underscore (_) with other characters to replace a single character.

User pattern and table type pattern aren't used in queries against the Oracle BI Server, use `%` for these options.

You can use `D` to view a static list of data types supported by the Oracle BI Server.

The arguments, `-C`, `-R`, `-f`, `-H`, `-q`, and `-NoFetch` are listed by the utility as available arguments, these options aren't used.

Argument	Description
-?	Lists the available command-line arguments.
`-d`*`data_source_name`*	Specifies the ODBC data source name for the targetOracle BI Server. If you omit this parameter, you're prompted at the command line to enter the data source name (DSN).
`-u`*`user_name`*	Specifies a valid Oracle Analytics Server user name.
`-p`*`password`*	Specifies the corresponding Oracle Analytics Server user password. The password argument is optional. If you don't provide a password argument, you're prompted to enter a password when you run the command. To minimize the risk of security breaches, Oracle recommends that you don't provide a password argument either on the command line or in scripts. The password argument is supported for backward compatibility only. For scripting purposes, you can pass the password through standard input.
`-s`*`sql_input_file_name`*	The name and path of a text file that includes your test SQL queries.
`-o`*`output_result_file_name`*	The name and path of a file where the utility writes the query results. This option is only used with `-s`.
`-D`*`delimiter`*	The delimiter used in the SQL input file, for example, semicolon (;) or colon (:). This option is only used with `-s`.
`-a`	Enables asynchronous processing. This option is typically used with `-s`, when you're passing a SQL input file with multiple SQL statements.

Argument	Description
`-z`	Enables UTF8 output instead of ANSI Code Page (ACP) in the output result file. You might need to include this option to display international characters in query results.
`-utf16`	Enables UTF16 instead of ACP for communication between `nqcmd` and the Oracle BI ODBC driver. You might need to include this option to display international characters in query results.
`-NotForwardCursor`	Disables the ODBC forward only cursor. Including this argument overrides the setting specified in the ODBC DSN.
`-v`	Displays the version of the `nqcmd` utility.
`-SessionVar` *`session_variable_name=session_variable_value`*	Includes the specified session variable and sets it to the specified value.

Upload Repository Command

Use the `uploadrpd` command to upload the repository to Oracle BI Server.

Uploading the repository to Oracle BI Server allows BI Server to load the repository into memory on startup and makes the repository available for queries.

You can only upload the repository to a specific service instance.

Oracle provides the `downloadrpd` and `uploadrpd` commands for offline repository diagnostic and development purposes such as testing, only. In all other repository development and maintenance situations, you should use BAR to utilize BAR's repository upgrade and patching capabilities and benefits.

You can use this command to upload the Oracle BI repository in RPD format. You can't use this command to upload a repository composed of MDS XML documents.

You run the utility through a launcher script, `datamodel.sh` on Linux and `datamodel.cmd` on Windows. If the domain is installed in default folder then the location of the launcher script looks like the following:

`Oracle_Home``/user_projects/domains/`*`Domain_Name`*`/bitools/bin/datamodel.sh or datamodel.cmd`

on Windows.
If the client install doesn't have domain names, the launcher script location is as follows:

`Oracle_Home``\bi\bitools\bin\datamodel.sh` or `datamodel.cmd` on Windows.

See What You Need to Know Before Using the Command.

Syntax

The `uploadrpd` command takes the following parameters:

```
uploadrpd -I <RPD filename> [-W <RPD password>] [-D] [KG <groups>] [-RG <groups>] -U
<cred_username> [-P <cred_password>][SI <service_instance>] [-S <host>] [-N <port>] [-
SSL] [-H]
```

Where

`I` specifies the name of the repository that you want to upload.

`W` is the repository's password. If you don't supply the password, then you're prompted for the password when the command is run. Oracle recommends that you include a password in the command only if you're using automated scripting to run the command.

`SI` specifies the name of the service instance.

`U` specifies a valid user's name to be used for authentication.

`P` specifies the password corresponding to the user's name that you specified for `U`. If you don't supply the password, then you're prompted for the password when the command is run. Oracle recommends that you include a password in the command only if you're using automated scripting to run the command.

`S` specifies the host name. Only include this option when you're running the command from a client installation.

`N` specifies the port number. Only include this option when you're running the command from a client installation.

`SSL` specifies to use SSL to connect to the Oracle WebLogic Server to run the command. Only include this option when you're running the command from a client installation.

`H` displays the usage information and exits for the command. Use `-H` or run `.sh` without any parameters to display the help content.

`-D` removes all the existing customization groups on the server, for example:

```
datamodel.sh uploadrpd -I orders.rpd -SI ssi -U weblogic -D
```

You can select to keep some of the existing groups and other all of the other groups, using `-D -KG`, for example:

```
datamodel.sh uploadrpd -I orders.rpd -SI ssi -U weblogic -D -KG "group1, group2"
```

The upload repository command keeps group1 and group2, and deletes any other existing groups.

You can delete some of the existing groups, but keep all of the other groups, using `-D -RG`, for example:

```
datamodel.sh uploadrpd -I orders.rpd -SI ssi -U weblogic -D -RG "group1, group2"
```

The upload repository command deletes group1 and group2, and keeps any other existing groups.

Example

```
datamodel.sh uploadrpd -I repository.rpd -SI bi -U weblogic -S
server1.example.com -N 7777 -SSL
```

Make the Repository Available for Queries

Use the upload repository command to make the repository available for queries.

After you build a repository and it's consistent, you need to upload the repository using the Upload Repository Command so that all Oracle BI Server instances can access it. Uploading the repository allows the Oracle BI Server to load the repository into memory upon startup and makes the repository available for queries.

You must upload an Oracle BI repository in RPD format. You can't upload a repository composed of MDS XML documents.

When the repository is uploaded and you can connect to it, run sample queries against it to test that it's created properly. Correct any problems you find and test again, repeating this process until you're satisfied with the results.

Create Data Source Connections to the Oracle BI Server for Client Applications

If you want to enable end user client applications to connect to the new repository, you must define an ODBC data source connection to the Oracle BI Server for each application.

The Oracle BI Presentation Services has the same relationship to the Oracle BI Server as any other client application.

See Integrating Other Clients with Oracle Business Intelligence in *Integrator's Guide for Oracle Business Intelligence Enterprise Edition* to learn about creating ODBC data source connections for the Oracle BI Server.

Publish to the User Community

After testing is complete, notify the user community that the data sources are available for querying.

Presentation Services users only need to know the URL to type in their browser. Client/server users, for example, users accessing the Oracle BI Server with a query tool or report writer client application, need to know the subject area names, the computer on which the server is running, and their user IDs and passwords. They also need to have the ODBC DSN for the Oracle BI Server installed on their computers, and they may need to know the logical names of repositories if multiple repositories are used and the data source name (DSN) being created doesn't point to the default repository.

16

Set Up Data Sources on Linux

You can learn how to set up data sources for use with Oracle Analytics Server when the Oracle BI Server is running on Linux.
Most repository development is performed on Windows, because the Administration Tool runs only on Windows. When you move to a production system, however, you can choose to run the Oracle BI Server on a Linux platform.

This chapter contains the following topics:

- About Setting Up Data Sources on Linux
- Configure Data Source Connections Using Native Gateways
- Use DataDirect Connect ODBC Drivers on Linux
- Configure Database Connections Using Native ODBC Drivers
- Set Up Oracle TimesTen In-Memory Database on Linux
- Configure Essbase Data Sources on Linux

About Setting Up Data Sources on Linux

When the Oracle BI Server is running on Linux, most data source connections are for query-only access.

The Administration Tool is used for importing objects and is a Windows-only tool. You must set up data source connections for import on Windows.

When the Oracle BI Server is running on Linux and you need to update database object settings such as the database type or connection pool settings, you can copy the repository file to a Windows computer, make the changes using the Administration Tool on Windows, and then copy the repository file back to the Linux computer.

There are three types of data source connections on Linux platforms:

- Native data source gateway connections, such as OCI for Oracle Database
- ODBC connections using the DataDirect Connect ODBC drivers that are bundled with Oracle Analytics Server
- Native ODBC connections using external drivers, such as for Teradata data sources

You can have a single repository that contains both DataDirect Connect ODBC connections and native ODBC connections. If you're using the native ODBC drivers and DataDirect ODBC drivers, you must manage the drivers with the same DataDirect ODBC driver manager. For example, the Teradata ODBC drivers include their required ODBC driver managers. When the Teradata ODBC driver is used with Oracle Analytics Server, you must manage the driver with the DataDirect ODBC driver manager that's bundled with Oracle Analytics Server.

Settings for Data Source Connections Using Native Gateways

Learn about the environment variable settings that you must configure for Oracle Database using native gateways.

For Oracle Database:

- The Oracle BI Server uses the Oracle Call Interface (OCI) to connect to the database. OCI is installed by default with Oracle Analytics Server. You must use the bundled version to connect.
- In the `tnsnames.ora` file, the Oracle Database alias, the defined entry name, must match the Data Source Name used in the repository connection pools of all physical Oracle databases.

  When connecting to an Oracle Database data source, you can include the entire connect string, or you can use the net service name defined in the `tnsnames.ora` file. If you choose to enter only the net service name, you must set up a `tnsnames.ora` file in the following location within the environment, so that the Oracle Analytics Server can locate the entry:

  BI_DOMAIN`/bidata/components/core/serviceinstances/ssi/oracledb`
- Edit the `obis.properties` file to set environment variables for the database client.

For an example, see Sample obis.properties Entries for Oracle Database.

Sample obis.properties Entries for Oracle Database

This example shows sample entries in obis.properties for Oracle Database on various platforms.

The shell script excerpts shown are examples only and aren't recommendations for particular software platforms. See Configure Data Source Connections Using Native Gateways.

```
################################################################
# Linux: Oracle BI 32 bit mode
################################################################
#set +u

# Oracle Parameters
#--------------------------
# Make sure that Oracle DB 32 bit Client is installed
#ORACLE_HOME=/export/home/oracle/12c
#export ORACLE_HOME
#TNS_ADMIN=$ORACLE_HOME/network/admin
#export TNS_ADMIN
#PATH=$ORACLE_HOME/bin:/opt/bin:$PATH
#export PATH
#LD_LIBRARY_PATH=$ORACLE_HOME/lib:$LD_LIBRARY_PATH
#export LD_LIBRARY_PATH

# If you have Linux 64 bit Platform, and would like to run Oracle BI 32 bit
# then you must install Oracle DB 64 bit client, and this client comes with
# 32 bit libraries under $ORACLE_HOME/lib32. The LD_LIBRARY_PATH in this case
```

```
# shall be like this:
#LD_LIBRARY_PATH=$ORACLE_HOME/lib32:$LD_LIBRARY_PATH
#export LD_LIBRARY_PATH

# Oracle Parameters
#----------------------------
# Make sure to install Oracle DB 64 bit Client
#ORACLE_HOME=/export/home/oracle/12c
#export ORACLE_HOME
#TNS_ADMIN=$ORACLE_HOME/network/admin
#export TNS_ADMIN
#PATH=$ORACLE_HOME/bin:/opt/bin:$PATH
#export PATH
#LD_LIBRARY_PATH_64=$ORACLE_HOME/lib:$LD_LIBRARY_PATH_64:/opt/j2se/jre/lib/sparc
#export LD_LIBRARY_PATH_64
#----------------------------

# Oracle Parameters
#----------------------------
#ORACLE_HOME=/export/home/oracle12c
#export ORACLE_HOME
#TNS_ADMIN=$ORACLE_HOME/network/admin
#export TNS_ADMIN
#PATH=$ORACLE_HOME/bin:/opt/bin:$PATH
#export PATH
#SHLIB_PATH=$ORACLE_HOME/lib:$SHLIB_PATH:/opt/j2se/jre/lib/hp700
#export SHLIB_PATH
#----------------------------

################################################################
# AIX: Oracle BI 64 bit mode
################################################################
#set +u

# Oracle Parameters
#----------------------------
#ORACLE_HOME=/export/home/oracle/12c
#export ORACLE_HOME
#TNS_ADMIN=$ORACLE_HOME/network/admin
#export TNS_ADMIN
#PATH=$ORACLE_HOME/bin:/opt/bin:$PATH
#export PATH
#LIBPATH=$ORACLE_HOME/lib:$LIBPATH:/opt/j2se/jre/lib/sparc
#export LIBPATH
#----------------------------
```

Configure Data Source Connections Using Native Gateways

You can connect to Oracle Database using native gateways.

1. Open the `obis.properties` file located at:

   BI_DOMAIN`/config/fmwconfig/bienv/obis`

2. Include the appropriate environment variable settings for the database client of your choice. Ensure that you point to the appropriate libraries, depending on whether you're using a 32-bit or 64-bit database.
3. Save and close the file.
4. Restart OBIS1.

About Updating Row Counts in Native Databases

Learn when you can use the Update Rowcount function.

If the following are true:

- You're using the **Update Rowcount** in the Administration Tool in offline mode.
- You're running a heterogeneous environment such as the Oracle BI Server on Linux, while remote administrators run the Administration Tool on Windows computers.

When using the **Update Rowcount** functionality in offline mode, the Administration Tool uses local data source connection definitions on the client computer, not the server data sources. Configure Oracle Database client on the Windows computer running the Administration Tool so that the following conditions are true:

- Data sources point to the same database identified in the Oracle BI `obis.properties` file on the Linux server.
- The name of the local data source matches the name of the data source defined in the Connection Pool object in the physical layer of the Oracle BI repository (.rpd) file.

If the above conditions aren't true, and if the server and client data sources are pointing at different databases, then erroneous updated row counts or incorrect results appear.

Troubleshoot OCI Connections

There are several reasons why you might have trouble connecting to an Oracle Database using OCI.

Check to ensure that the following conditions are true:

- The computer running the Oracle BI Server must use Oracle Call Interface (OCI) to connect to the database.
- If you choose not to use the entire connect string in the repository connection pool, you must ensure that a valid `tnsnames.ora` file is set up in the following location within the environment, so that the Oracle BI Server can locate the entry:

  BI_DOMAIN`/bidata/components/core/serviceinstances/ssi/oracledb`
- If you choose not to use the entire connect string in the repository connection pool, ensure that the net service name in the `tnsnames.ora` file matches the Data Source Name used in the connection pool.

  For example, in the following example of a `tnsnames.ora` entry, the corresponding repository connection pool Data Source Name is ITQA2.

```
ITQA2 =
   (DESCRIPTION =
      (ADDRESS_LIST =
         (ADDRESS = (PROTOCOL = TCP)(HOST = ITQALAB2)(PORT = 1521))
   (CONNECT_DATA =
      (SERVICE_NAME = ITQALAB2.corp)
      )
   )
```

The following procedure shows how to check repository database and connection pool settings against the Oracle tnsnames.ora settings.

1. Open the repository in the Administration Tool.
2. In the Physical layer, double-click the database you want to check to display the Database dialog.
3. On the General tab, in the **Data source definition: Database** field, ensure that the appropriate Oracle Database version is selected. Then, click **OK**.
4. Open the Connection Pool dialog for this data source. You might need to expand the database object in the Physical layer to see the connection pool object.
5. In the Connection Pool dialog, check that the following is true:
   - The **Call interface** field displays the appropriate value for the release of the Oracle Database you're using.
   - The **Data source name** field displays the Oracle Database net service name that you defined in the tnsnames.ora entry.
   - The **User name** and **password** fields contain the correct values.

   Change the values if necessary, then click **OK**.
6. In the environment, open the `tnsnames.ora` file located in the following directory:

   BI_DOMAIN`/bidata/components/core/serviceinstances/ssi/oracledb`
7. Check that a valid net service name exists with the following characteristics:
   - Matches the connection pool settings for the Data Source Name
   - Specifies the targeted Oracle physical database

Use DataDirect Connect ODBC Drivers on Linux

Oracle Analytics Server provides DataDirect Connect ODBC drivers and driver managers for Linux operating systems for connectivity to MySQL, Sybase ASE, Informix, Hive, and Impala databases.

Amazon Redshift data sources are also supported. You need to use the Amazon Redshift ODBC driver available from Amazon Web Services. Configure the Amazon Redshift data source using the steps documented for other data sources.

The DataDirect drivers are installed in the Oracle Analytics Server installation process. You can find the DataDirect Connect ODBC drivers in the *MW_HOME*`/bi/modules/oracle.bi.datadirect.odbc/7.1.6/lib` directory.

You don't need to set the ODBCINI environment variable to set up the DataDirect Connect ODBC drivers. This variable is set automatically during installation.

Configure Oracle Analytics Server to Use DataDirect

When you install Oracle Analytics Server, the required DataDirect 7.1.6 drivers are installed and automatically configured.

You can define the default settings in the `obis.properties` and `odbc.ini` files.

You need to modify your existing database configurations to use the DataDirect drivers. For information about modifying your existing database configuration, see the following procedures:

- Configure the DataDirect Connect ODBC Driver for DB2 Database
- Configure the DataDirect Connect ODBC Driver for MySQL Database
- Configure the DataDirect Connect ODBC Driver for Sybase ASE Database
- Configure the DataDirect Connect ODBC Driver for Informix Database
- Configure the DataDirect Connect ODBC Driver for Cloudera Impala Database
- Configure the DataDirect Connect ODBC Driver for Apache Hive Database

Additional DataDirect Configuration for Oracle Essbase

Modify the DataDirect configuration to connect to Essbase data sources.

The name of the DataDirect 7.1.6 driver file to use with Essbase is `essbase.cfg`.

1. Open `essbase.cfg` for editing from the following location:

   BI_DOMAIN`/config/fmwconfig/biconfig/essbase`
2. In the configuration file, locate the `BPM_ORACLE_DriverDescriptor` element and change the value to `"DataDirect 7.1.6 Oracle Wire Protocol"`.
3. Use Fusion Middleware Control to restart Essbase.

Configure the DataDirect Connect ODBC Driver for DB2 Database

Use these steps to connect to a DB2 database.

The name of the DataDirect ODBC driver file to connect to a MySQL database is `ARdb227.so`.

1. Open the `obis.properties` file located in:

   BI_DOMAIN`/config/fmwconfig/bienv/OBIS`
2. Locate the `LD_LIBRARY_PATH` variable. Use the following:
   - For Linux, the library path variable is `LD_LIBRARY_PATH`.
   - For AIX, the library path variable is `LIBPATH`.
3. If necessary, update the `LD_LIBRARY_PATH` variable to include the DataDirect driver path.
4. In `obis.properties`, locate the `PATH` variable and if necessary, include the DataDirect driver path.
5. Save and close the file.
6. Open the `odbc.ini` file. You can find this file at:

   BI_DOMAIN`/config/fmwconfig/bienv/core`
7. Create an entry for the database:

   Use the same ODBC connection name to the data source name specified in the connection pool defined in the repository.

Set the `Driver` parameter to the file name and location of the DataDirect Connect driver for DB2 Database. For the `NetworkAddress` use the IP address or fully qualified host name and the port number.

In the following example, the Driver parameter is set to the DataDirect Connect driver, and the data source name is ABCDEF.

```
[ABCDEF]
Driver=/u01/app/product/12.2.1.4.0/obi_1/bi/modules/oracle.bi.datadirect.odbc/
7.1.6/lib/ARdb227.so
Description=Oracle 7.1 DB2 Wire Protocol
AccountingInfo=
AddStringToCreateTable=
AlternateID=
AlternateServers=
ApplicationName=
ApplicationUsingThreads=1
AuthenticationMethod=0
BulkBinaryThreshold=32
BulkCharacterThreshold=-1
BulkLoadBatchSize=1024
BulkLoadFieldDelimiter=
BulkLoadRecordDelimiter=
CatalogSchema=
CharsetFor65535=0
ClientHostName=
ClientUser=
#Collection applies to z/OS and iSeries only
Collection=
ConcurrentAccessResolution=0
ConnectionReset=0
ConnectionRetryCount=0
ConnectionRetryDelay=3
CurrentFuncPath=
#Database applies to DB2 UDB only
Database=ABCDEF
DefaultIsolationLevel=1
DynamicSections=1000
EnableBulkLoad=0
EncryptionMethod=0
FailoverGranularity=0
FailoverMode=0
FailoverPreconnect=0
GrantAuthid=PUBLIC
GrantExecute=1
GSSClient=native
HostNameInCertificate=
IpAddress=10.116.26.27
KeyPassword=
KeyStore=
KeyStorePassword=
LoadBalanceTimeout=0
LoadBalancing=0
#Location applies to z/OS and iSeries only
Location=
LogonID=
MaxPoolSize=100
MinPoolSize=0
Password=
PackageCollection=NULLID
PackageNamePrefix=DD
```

```
PackageOwner=
Pooling=0
ProgramID=
QueryTimeout=0
ReportCodePageConversionErrors=0
TcpPort=50002
TrustStore=
TrustStorePassword=
UseCurrentSchema=0
ValidateServerCertificate=1
WithHold=1
XMLDescribeType=-10
```

8. Save and close the `odbc.ini` file.

Configure the DataDirect Connect ODBC Driver for MySQL Database

Use these steps to connect to a MySQL database.

The name of the DataDirect ODBC driver file to connect to a MySQL database is `ARmysql27.so`.

1. Open the `obis.properties` file located in:

   BI_DOMAIN`/config/fmwconfig/bienv/OBIS`

2. Locate the `LD_LIBRARY_PATH` variable. Use the following:

   - For Linux, the library path variable is `LD_LIBRARY_PATH`.
   - For AIX, the library path variable is `LIBPATH`.

   For example, to set the library path variable for the driver on Linux:

   ```
   LD_LIBRARY_PATH=$MW_HOME/bi/bifoundation/server/bin,
   $MW_HOME/bi/bifoundation/web/bin,
   $MW_HOME/clients/epm/Essbase/EssbaseRTC/bin,
   $MW_HOME/bi/bifoundation/odbc/lib,
   $ORACLE_INSTANCE,
   $MW_HOME/lib
   ```

3. If necessary, update the `LD_LIBRARY_PATH` variable to include the DataDirect driver path. For example, to update the variable for the driver on Linux:

   ```
   LD_LIBRARY_PATH=$MW_HOME/bi/modules/oracle.bi.datadirect.odbc/7.1.6/lib,
   $MW_HOME/bi/bifoundation/server/bin,
   $MW_HOME/bi/bifoundation/web/bin,
   $MW_HOME/clients/epm/Essbase/EssbaseRTC/bin,
   $MW_HOME/bi/bifoundation/odbc/lib,
   $ORACLE_INSTANCE
   $ORACLE_HOME/lib:$MW_HOME/lib
   ```

4. In `obis.properties`, locate the `PATH` variable and if necessary, include the DataDirect driver path.
5. Save and close the file.
6. Open the `odbc.ini` file. You can find this file at:

   BI_DOMAIN`/config/fmwconfig/bienv/core`

7. Create an entry for the database:

Use the same ODBC connection name to the data source name specified in the connection pool defined in the repository.

Set the `Driver` parameter to the file name and location of the DataDirect Connect driver for MySQL Database. For the `NetworkAddress` use the IP address or fully qualified host name and the port number.

In the following example, the Driver parameter is set to the DataDirect Connect driver, and the data source name is MySQL_DB.

```
[MySQL Wire Protocol]
Driver=/refresh/home/oracle/middleware/oracle_home/bi/modules/
oracle.bi.datadirect.odbc/7.1.6/lib/ARmysql27.so
Description=Oracle 7.1 MySQL Wire Protocol
AlternateServers=
ApplicationUsingThreads=1
ConnectionReset=0
ConnectionRetryCount=0
ConnectionRetryDelay=3
Database=<database_name>
DefaultLongDataBuffLen=1024
EnableDescribeParam=0
EncryptionMethod=0
FailoverGranularity=0
FailoverMode=0
FailoverPreconnect=0
HostName=<MySQL_host>
HostNameInCertificate=
InteractiveClient=0
LicenseNotice=You must purchase commercially licensed MySQL database software or a
MySQL Enterprise subscription in order to use the DataDirect Connect for ODBC for
MySQL Enterprise driver with MySQL software.
KeyStore=
KeyStorePassword=
LoadBalanceTimeout=0
LoadBalancing=0
LogonID=
LoginTimeout=15
MaxPoolSize=100
MinPoolSize=0
Password=
Pooling=0
PortNumber=<MySQL_server_port>
QueryTimeout=0
ReportCodepageConversionErrors=0
TreatBinaryAsChar=0
TrustStore=
TrustStorePassword=
ValidateServerCertificate=1
```

8. Save and close the `odbc.ini` file.

Configure the DataDirect Connect ODBC Driver for Sybase ASE Database

The name of the DataDirect ODBC driver file to connect to a Sybase ASE database is ARase27.so.

1. Open the `obis.properties` file located in:

   BI_DOMAIN`/config/fmwconfig/bienv/OBIS`

2. Locate the `LD_LIBRARY_PATH` variable use the following information:
   - For Linux, the library path variable is `LD_LIBRARY_PATH`.
   - For AIX, the library path variable is `LIBPATH`.

   For example, to set the library path variable for the driver on Linux:

   ```
   LD_LIBRARY_PATH=$MW_HOME/bi/bifoundation/server/bin,
   $MW_HOME/bi/bifoundation/web/bin,
   $MW_HOME/clients/epm/Essbase/EssbaseRTC/bin,
   $MW_HOME/bi/bifoundation/odbc/lib,
   $ORACLE_INSTANCE$:$MW_HOME/lib
   ```

3. If necessary, update the `LD_LIBRARY_PATH` variable to include the DataDirect driver path.

   To update the variable for the driver on Linux, review the following example:

   ```
   LD_LIBRARY_PATH=$MW_HOME/bi/modules/oracle.bi.datadirect.odbc/7.1.6/lib,
   $MW_HOME/bi/bifoundation/server/bin,
   $MW_HOME/bi/bifoundation/web/bin,
   $MW_HOME/clients/epm/Essbase/EssbaseRTC/bin,
   $MW_HOME/bi/bifoundation/odbc/lib,$ORACLE_INSTANCE$:$MW/lib
   ```

4. Locate the `PATH` variable and if necessary, include the DataDirect driver path.
5. Save and close the file.
6. Open the `odbc.ini` file located in:

   `BI_DOMAIN``/config/fmwconfig/bienv/core`

7. Create an entry for the database.

   Use the same ODBC connection name to the data source name specified in the connection pool defined in the repository.

   Set the `Driver` parameter to the file name and location of the DataDirect Connect driver for Sybase ASE Database. For the `NetworkAddress` provide the IP address or fully qualified host name and the port number.

   The following example shows the Driver parameter set to the DataDirect connect driver, and *SybaseASE_DB* as the data source name.

   ```
   [SybaseASE_DB]
   Driver=/scratch/aime1/work/mw3108/bi/modules/oracle.bi.datadirect.odbc/7.1.6
   Description=DataDirect 7.1 Sybase Wire Protocol
   AlternateServers=
   ApplicationName=
   ApplicationUsingThreads=1
   ArraySize=50
   AuthenticationMethod=0
   Charset=
   ConnectionRetryCount=0
   ConnectionRetryDelay=3
   CursorCacheSize=1
   Database=Paint
   DefaultLongDataBuffLen=1024
   EnableDescribeParam=0
   EnableQuotedIdentifiers=0
   EncryptionMethod=0
   GSSClient=native
   HostNameInCertificate=
   InitializationString=
   ```

```
Language=
LoadBalancing=0
LogonID=my_id
NetworkAddress=111.111.111.111,5005
OptimizePrepare=1
PacketSize=0
Password=
RaiseErrorPositionBehavior=0
ReportCodePageConversionErrors=0
SelectMethod=0
ServicePrincipalName=
TruncateTimeTypeFractions=0
TrustStore=
TrustStorePassword=
ValidateServerCertificate=1
WorkStationID=
```

8. Save and close the `odbc.ini` file.

Configure the DataDirect Connect ODBC Driver for Informix Database

Use these steps to configure the DataDirect ODBC driver file to connect to an Informix database. The file to use is ARifcl27.so.

1. Open the `obis.properties` file located in:

   BI_DOMAIN`/config/fmwconfig/bienv/OBIS`

2. Locate the `LD_LIBRARY_PATH` variable, using the following information:

   - For Linux, the library path variable is `LD_LIBRARY_PATH`.
   - For AIX, the library path variable is `LIBPATH`.

   For example, to set the library path variable for the driver on Linux:

   ```
   LD_LIBRARY_PATH=$MW_HOME/bi/bifoundation/server/bin,
   $MW_HOME/bi/bifoundation/web/bin,
   $MW_HOME/clients/epm/Essbase/EssbaseRTC/bin,
   $MW_HOME/bi/bifoundation/odbc/lib,
   $ORACLE_INSTANCE$:$MW_HOME/lib
   ```

3. If necessary, update the `LD_LIBRARY_PATH` variable to include the DataDirect driver path.

   To update the variable for the driver on Linux, review the following example:

   ```
   LD_LIBRARY_PATH=$MW_HOME/bi/modules/oracle.bi.datadirect.odbc/7.1.6/lib
   $MW_HOME/bi/bifoundation/server/bin,
   $MW_HOME/bi/bifoundation/web/bin,
   $MW_HOME/clients/epm/Essbase/EssbaseRTC/bin,
   $MW_HOME/bi/bifoundation/odbc/lib,
   $MW_INSTANCE$:$MW_HOME/lib
   ```

4. In obis.properties, locate the `PATH` variable and if necessary, include the DataDirect driver path.
5. Save and close the file.
6. Open the `odbc.ini` file located in:

   BI_DOMAIN`/config/fmwconfig/bienv/core`

7. Create an entry for the database.

You must use the identical ODBC connection name to the data source name specified in the connection pool defined in the repository.

Set the `Driver` parameter to the file name and location of the DataDirect Connect driver for Informix. You must specify the `HostName` parameter, you can use the fully qualified host name or the IP address, and the `PortNumber` parameter.

In the following example, the `Driver` parameter is set to the DataDirect Connect driver, and the data source name is *Informix_DB*.

```
[Informix_DB]
Driver=/scratch/aime1/work/mw3108/bi/modules/oracle.bi.datadirect.odbc/7.1.6
Description=DataDirect Informix Wire Protocol
AlternateServers=
ApplicationUsingThreads=1
CancelDetectInterval=0
ConnectionRetryCount=0
ConnectionRetryDelay=3
Database=
HostName=111.111.111.111
LoadBalancing=0
LogonID=informix
Password=mypassword
PortNumber=1526
ReportCodePageConversionErrors=0
ServerName=
TrimBlankFromIndexName=1
```

8. Save and close the `odbc.ini` file.

Configure the DataDirect Connect ODBC Driver for Cloudera Impala Database

The DataDirect ODBC driver file name, to connect to a Cloudera Impala database is `ARimpala27.so`.

See Configure Impala 1.3.x to Include a LIMIT Clause.

1. Open the `obis.properties` file located at:

   `BI_DOMAIN``/config/fmwconfig/bienv/OBIS`

2. Locate the `LD_LIBRARY_PATH` variable.

   The library path variable is:

   - For Linux : `LD_LIBRARY_PATH`.
   - For AIX: `LIBPATH`

   For example, to set the library path variable for the driver on Linux, use:

   ```
   LD_LIBRARY_PATH=$MW_HOME/bi/bifoundation/server/bin,
   $MW_HOME/bi/bifoundation/web/bin,
   $MW_HOME/clients/epm/Essbase/EssbaseRTC/bin,
   $MW_HOME/bi/bifoundation/odbc/lib,
   $ORACLE_INSTANCE,
   $MW_HOME/lib
   ```

3. If necessary, update the `LD_LIBRARY_PATH` variable to include the DataDirect driver path.

For example, to update the variable for the driver on Linux:

```
LD_LIBRARY_PATH=$MW_HOME/bi/modules/oracle.bi.datadirect.odbc/7.1.6/lib,
$MW_HOME/bi/bifoundation/server/bin,
$MW_HOME/bi/bifoundation/web/bin,
$MW_HOME/clients/epm/Essbase/EssbaseRTC/bin,
$MW_HOME/bi/bifoundation/odbc/lib,
$ORACLE_INSTANCE,
$MW_HOME/lib
```

4. In `obis.properties`, locate the `PATH` variable and, if necessary, include the DataDirect driver path.
5. Save and close the file.
6. Open the `odbc.ini` file located at:

   `BI_DOMAIN``/config/fmwconfig/bienv/core`
7. Create an entry for the database and specify the `HostName` parameter.

   You can use the fully qualified host name or the IP address as the `HostName` parameter, and the `PortNumber` parameter.

   - The ODBC connection name is identical to the data source name specified in the connection pool defined in the repository.
   - The `Driver` parameter is set to the file path of the DataDirect Connect driver for Cloudera Impala.
   - The *HostName* parameter uses the fully qualified host name, or the IP address as the *HostName* parameter and the *PortNumber* parameter.

   The following example shows the Driver parameter set to the DataDirect Connect driver, and the Impala_DB data source name.

   ```
   [Impala_DB]
   Driver=/scratch/aime1/work/mw3108/bi/modules/oracle.bi.datadirect.odbc/7.1.6/
   ARimpala27.so
   Description=Oracle 7.1 Cloudera Impala Wire Protocol
   ArraySize=16384
   Database=default
   DefaultLongDataBuffLen=1024
   DefaultOrderByLimit=-1
   EnableDescribeParam=0
   HostName=localhost
   LoginTimeout=30
   MaxVarcharSize=2000
   PortNumber=21050
   RemoveColumnQualifiers=0
   StringDescribeType=12
   TransactionMode=0
   UseCurrentSchema=0
   WireProtocolVersion=2
   ```
8. Save and close the `odbc.ini` file.

   If you're using Impala 1.3.*x*, you must configure include a `LIMIT` clause section. If you're using Impala 1.4 (CDH 5.1) or later, then you can skip the additional steps.

Configure Impala 1.3.x to Include a LIMIT Clause

Impala 1.3.*x* requires that queries with an ORDER BY clause contain a LIMIT clause.

There are three methods to specify this clause in the configuration. Oracle recommends using the Modify the Impala daemon's default query options method. For the second and third methods, see Modify the Impala DefaultOrderByLimit Alternate Methods.

Specifying a default order by limit using any of the following methods returns a maximum of 2,000,000 rows for queries with an ORDER by clause.

If you specify the LIMIT clause using the Modify the Impala daemon's default query options method, and your queries include an ORDER BY clause, then Impala returns a maximum of 2,000,000 rows. If this limit is exceeded, then Impala throws an exception.

For queries over 2,000,000 rows, specify a higher `default_order_by_limit` value.

You can also specify the `Default Order By Limit` by using the client instead of the Impala server.

Use this method if you don't have rights to modify the Impala daemon using the previous methods. If you use this method, then Impala silently truncates your value to 2,000,000 rows.

1. Go to the Cloudera Manager's home page and click the **Impala** service.
2. In the Impala service page, click **Configuration**, and the select **View and Edit**.
3. In the Configuration page, select **Impala Daemon Default Group**.
4. Locate Impala Daemon Query Options Advanced, also known as the **default_query_options**, and add the following entries:

   ```
   default_order_by_limit=2000000
   abort_on_default_limit_exceeded=true
   ```

5. Click **Save Changes**.
6. In the Cloudera Manager's home page, restart the Impala service.

Modify the Impala DefaultOrderByLimit Alternate Methods

Use the first DefaultOrderByLimit option if your Impala environment isn't managed by Cloudera Manager. Use the second DefaultOrderByLimit option if you don't have rights to modify the Impala daemon.

Modify the Impala Daemon's Default Query Options Without Cloudera Manager

If your environment is managed by Cloudera Manager and you've the required permissions, use `DefaultOrderByLimit`, the recommended method for updating the Impala daemon. See Configure Impala 1.3.x to Include a LIMIT Clause.

If your Impala environment isn't managed by Cloudera Manager, use the Impala product documentation to help you modify the `LIMIT` clause. See "Configuring Impala Startup Options Through the Command Line" in the *CDH 5 Installation Guide*.

If you specify the LIMIT clause using this method and your queries include an ORDER BY clause, then Impala returns a maximum of 2,000,000 rows. If this limit is exceeded, then Impala throws an exception.

- After completing the steps in the Configuring Impala Startup Options Through the Command Line task, add the following entry in `IMPALA_SERVER_ARGS`:

```
-default_query_options
'default_order_by_limit=2000000;abort_on_default_limit_exceeded=true'
```

Modify the DefaultOrderByLimit Parameter in the odbc.ini Impala DSN Entry

Use this method if you don't have rights to modify the Impala daemon using the previous methods. If you use this method, then Impala silently truncates your value to 2,000,000 rows.

If you need your query to return more than 2,000,000 rows, then specify a higher `DefaultOrderByLimit` parameter value.

You can specify the Default Order By Limit using the client instead of the Impala server.

1. Open the `odbc.ini` file from the *`BI_DOMAIN`*`/config/fmwconfig/bienv/core` directory.
2. Locate the Impala_DB database entry, and then locate the `DefaultOrderByLimit` parameter.
3. Update the value to `2000000`, for example, `DefaultOrderByLimit=2000000`.
4. Save and close the `odbc.ini` file.

Configure the DataDirect Connect ODBC Driver for Apache Hive Database

The name of the DataDirect ODBC driver file to connect to a Apache Hive is libARhive28.so.

See Limitations on the Use of Apache Hive.

See *Quick Start: Progress DataDirect for ODBC for Apache Hive Wire Protocol Driver for Linux* located in the `mwhome\bi\common\ODBC\Merant\8.0.2\help` directory.

1. Open the `obis.properties` file located in:

   `BI_DOMAIN``/config/fmwconfig/bienv/obis`
2. Locate the `LD_LIBRARY_PATH` variable, use the following information:
   - For Linux, the library path variable is `LD_LIBRARY_PATH`.
   - For AIX, the library path variable is `LIBPATH`.

   For example, to set the library path variable for the driver on Linux:

   ```
   LD_LIBRARY_PATH=$MW_HOME/bi/bifoundation/server/bin,
   $ORACLE_HOME/bi/bifoundation/web/bin,
   $ORACLE_HOME/clients/epm/Essbase/EssbaseRTC/bin,
   $ORACLE_HOME/bi/bifoundation/odbc/lib,
   $ORACLE_INSTANCE,
   $ORACLE_HOME/lib
   ```
3. If necessary, update the `LD_LIBRARY_PATH` variable to include the DataDirect driver path.

   To update the variable for the driver on Linux, review the following example:

```
LD_LIBRARY_PATH=$bi/modules/oracle.bi.datadirect.odbc/8.0.2/lib,
$ORACLE_HOME/bi/bifoundation/server/bin,
$ORACLE_HOME/bi/bifoundation/web/bin,
$ORACLE_HOME/clients/epm/Essbase/EssbaseRTC/bin,
$ORACLE_HOME/bi/bifoundation/odbc/lib,
$ORACLE_INSTANCE,
$ORACLE_HOME/lib
```

4. In `obis.properties`, locate the `PATH` variable and if necessary, include the DataDirect driver path.
5. To point to the DataDirect driver, create the `HADOOP_DLL` variable either above or below the `LD_LIBRARY_PATH` variable.

   For example:

   `HADOOP_DLL=`*`MW_HOME`*`/bi/modules/oracle.bi.datadirect.odbc/8.0.2/lib/ARhive28.so`
6. Save and close the file.
7. Open the `odbc.ini` file located in:

   `BI_DOMAIN``/config/fmwconfig/bienv/core`
8. Create an entry for the database, ensuring that the ODBC connection name is identical to the data source name specified in the connection pool defined in the repository.

   Ensure that you set the Driver parameter to the file name and location of the DataDirect Connect driver for Hive. You must specify the `HostName` parameter. You can use the fully qualified host name, or the IP address, and the `PortNumber` parameter.

   In the following example, the Driver parameter is set to the DataDirect Connect driver, and the data source name is *Hive*.

   ```
   [Hive]
   Driver=MW_HOME/bi/modules/oracle.bi.datadirect.odbc/8.0.2/lib
   Description=Oracle 8.0 Apache Hive Wire Protocol
   ArraySize=16384
   Database=default
   DefaultLongDataBuffLen=1024
   EnableDescribeParam=0
   HostName=localhost
   LoginTimeout=30
   MaxVarcharSize=2000
   PortNumber=10000
   RemoveColumnQualifiers=0
   StringDescribeType=12
   TransactionMode=0
   UseCurrentSchema=0
   ```
9. Save and close the `odbc.ini` file.
10. Restart OBIS1.

Configure Database Connections Using Native ODBC Drivers

Oracle Analytics Server bundles Linux ODBC drivers for some data sources, but not all.

For these data sources, including Teradata and Oracle TimesTen In-Memory Database, you must install your own ODBC driver, then update the `obis.properties` and `odbc.ini` files to configure the data source.

If you're using Teradata, see Avoid Spool Space Errors for Queries Against Teradata Data Sources.

See Create or Change Connection Pools.

1. Open the `obis.properties` file, located in:

   BI_DOMAIN`/config/fmwconfig/bienv/obis`

2. Locate the `LD_LIBRARY_PATH` variable, use the following information:

   - For Linux, the library path variable is `LD_LIBRARY_PATH`.
   - For AIX, the library path variable is `LIBPATH`.

   For example, to set the library path variable for the driver on Linux:

   ```
   LD_LIBRARY_PATH=$ORACLE_HOME/opt/teradata/client/15.10/odbc_64/lib,
   $ORACLE_HOME/bi/bifoundation/web/bin,
   $ORACLE_HOME/clients/epm/Essbase/EssbaseRTC/bin,
   $ORACLE_HOME/bi/bifoundation/odbc/lib,
   $ORACLE_INSTANCE,
   $ORACLE_HOME/lib
   ```

3. If necessary, update the `LD_LIBRARY_PATH` variable to include the driver path.

   To update the variable for the driver on Linux, review the following example:

   ```
   LD_LIBRARY_PATH=$ORACLE_HOME/opt/teradata/client/15.10/odbc_64/lib,
   $ORACLE_HOME/bi/bifoundation/server/bin,
   $ORACLE_HOME/bi/bifoundation/web/bin,
   $ORACLE_HOME/clients/epm/Essbase/EssbaseRTC/bin,
   $ORACLE_HOME/bi/bifoundation/odbc/lib,
   $ORACLE_INSTANCE,
   $ORACLE_HOME/lib
   ```

4. In `obis.properties`, locate the `PATH` variable and if necessary, include the DataDirect driver path.
5. Save and close the file.
6. Open the `odbc.ini` file. You can find this file at:

   BI_DOMAIN`/config/fmwconfig/bienv/core`

7. Create an entry for the database, ensuring that the ODBC connection name is identical to the data source name specified in the connection pool defined in the repository.

   Ensure that you set the Driver parameter to the file name and location of the native ODBC driver for the database, with the library suffix that's appropriate for the operating system, for example, `.so` for AIX.

   The following example provides details for a Teradata data source on Linux, with a data source name of Tera_Northwind.

```
[Tera_Northwind]
Driver=/opt/teradata/client/15.10/odbc_64/lib/tdata.so
Description=NCR 3600 running Teradata V2R6.2
DBCName=10.345.67.899
astUser=
Username=northwind
Password=northwind
Database=northwind
DefaultDatabase=northwind
NoScan=no
```

If you've selected the option **Require fully qualified table names** in the General tab of the Connection Pool dialog for this data source in the Administration Tool, the `DefaultDatabase` parameter doesn't require a value.

8. In the `odbc.ini` file, add an entry to the section `ODBC Data Sources` with the details appropriate for the data source.

   The following example provides details for a Teradata data source with a data source name of Tera_Northwind.

   ```
   Tera_Northwind=tdata.so
   ```

9. Restart OBIS1.
10. Using the Administration Tool, open the repository and add the new DSN you created as the Connection Pool **Data source name** for the appropriate physical databases.

Set Up Oracle TimesTen In-Memory Database on Linux

You must perform some prerequisite tasks before setting up Oracle TimesTen In-Memory Database data sources.

To set up Oracle TimesTen In-Memory Database data sources, first follow the instructions in Configure TimesTen Data Sources to set up the TimesTen data source. Ensure that you go to the section Configure Database Connections Using Native ODBC Drivers to obtain the correct steps for Linux systems.

Next, review the best practices described in Improve Use of System Memory Resources with TimesTen Data Sources and implement them as needed.

Finally, if the user that starts OBIS1 doesn't have the path to the TimesTen DLL ($TIMESTEN_HOME/lib) in their operating system `LD_LIBRARY_PATH` variable, or `LIBPATH` on AIX, you must add the TimesTen DLL path as a variable in the `obis.properties` file.

1. Open `obis.properties` for editing. You can find `obis.properties` at:

   BI_DOMAIN`/config/fmwconfig/bienv/obis`

2. Add the required TimesTen variable `TIMESTEN_DLL`, and also update the `LD_LIBRARY_PATH` variable (or equivalent), as shown below:

   ```
   TIMESTEN_DLL=$TIMESTEN_HOME/lib/libttclient.so
   LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$TIMESTEN_HOME/lib
   ```

3. Save and close the file.
4. Repeat these steps on each computer that runs the Oracle BI Server process. If you're running multiple Oracle BI Server instances on the same computer, then

ensure that you update the ias-component tag appropriately for each instance in `obis.properties`, for example, `ias-component id="coreapplication_obis1"`, and `ias-component id="coreapplication_obis2"`.

5. Restart OBIS1.

Configure Essbase Data Sources on Linux

The Oracle BI Server uses the Essbase client libraries to connect to Essbase data sources.

The Essbase client libraries are installed by default with Oracle Analytics Server. No additional configuration is required to enable Essbase data source access for full installations.

17

Manage Oracle BI Repository Files

This chapter describes tasks related to managing your Oracle BI repository files, including comparing and merging repositories, equalizing objects, querying and managing metadata, and changing the repository password.
This chapter contains the following topics:

- Compare Repositories
- Equalize Objects
- Merge Repositories
- Query and Manage Repository Metadata
- Change the Oracle BI Repository Password

Compare Repositories

Learn how to compare all repository objects in two different repositories.

If you're using an Oracle BI Applications repository and have customized its content, you can use this feature to compare your customized repository to a new version of the repository received with Oracle BI Applications.

See Merge Repositories.

This section contains the following topics:

- Compare Repositories Using the Compare Dialog
- Compare Repositories Using comparerpd
- Turn Off Compare Mode

Compare Repositories Using the Compare Dialog

Use this task to compare repositories in the Oracle BI Administration Tool.

The repository that you open is referred to as the current repository. See Use Online and Offline Repository Modes for instructions on opening a repository.

1. In the Administration Tool, open a repository in offline mode.
2. From the **File** menu, select **Compare**.
3. In the Select Original Repository dialog, select the repository you want to compare to the open repository. Select **Repository** from the submenu to select a binary repository file in Oracle BI repository file format, or select **XML** to select a set of MDS XML documents.
4. In the Open Offline dialog, enter the repository password and click **OK**.
5. Use the Compare repositories dialog to review the differences between repositories.

Compare Repositories Using comparerpd

You can compare repositories and create patch files using the comparerpd utility.

Use `comparerpd` on Linux systems when the Administration Tool isn't available.

When you run the `comparerpd` utility, the system equalizes the repository objects. The equalizing process outputs the *my_current_rpd*`_equalized.rpd` file, an informational file containing the output of the equalization done between the compared repositories. Equalizing ensures that objects with the same qualified names have the same unique identifiers (UIDs). Objects with the same name but different UIDs are automatically updated to use the same UID or equalized. Including when equalization isn't needed, the *my_current_rpd*`_equalized.rpd` file is generated. The *my_current_rpd*`_equalized.rpd` file is saved to the original repository's location.

The location of the `comparerpd` utility is:

BI_DOMAIN`/bitools/bin`

Syntax

The `comparerpd` utility takes the following parameters:

```
comparerpd [-P modified_rpd_password] -C modified_rpd_pathname
[-W original_rpd_password] -G original_rpd_pathname {-O output_csv_file_name |
-D output_patch_file_name | -M output_mds_xml_directory_name} -A -E -8
```

Where:

`-P` *modified_rpd_password* is the repository password for the modified repository, also called the customer or customized repository.

`-C` *modified_rpd_pathname* is the name and location of the modified repository.

`-W` *original_rpd_password* is the repository password for the original repository.

`-G` *original_rpd_pathname* is the name and location of the original repository.

`-O` *output_csv_file_name* is the name and location of a csv file where you want to store the repository object comparison output.

`-D` *output_patch_file_name* is the name and location of an XML patch file where you want to store the differences between the two repositories.

`-M` *output_mds_xml_directory_name* is the top-level directory where you want to store diff information in MDS XML format. A list of removed XML files is stored in the directory tree under the top-level directory at:

```
oracle\bi\server\base\DeletedFiles.txt
```

You can specify an output CSV file using `-O`, an XML patch file using `-D`, or an MDS XML directory tree using `-M`. You can't specify more than one output type at the same time.

If the patch contains passwords, such as connection pool passwords, the patch file is encrypted using the repository password from the current repository. The current repository password effectively becomes the patch file password. You might need to

supply this patch file password when applying the patch, if it's different from the repository password for the original repository.

`-A` is an optional argument that ensures the XML patch file doesn't contain encrypted passwords for the connection pools. This parameter is used in conjunction with the `-D` parameter.

`-E` is an optional argument that causes UIDs to be used to compare expression text. Using this parameter ensures objects that have the same name but different UIDs are given the same UID. This action makes sure that objects are compared as modified instead of being reported as created and deleted. If `-E` isn't specified, strings are used.

`-8` specifies UTF-8 encoding.

The arguments for the *modified_rpd_password* and *original_rpd_password* are optional. If you don't provide password arguments, you're prompted to enter any required passwords when you run the command. To minimize the risk of security breaches, Oracle recommends that you don't provide password arguments either on the command line or in scripts. The password arguments are supported for backward compatibility only. For scripting purposes, you can pass the password through standard input.

For example:

```
comparerpd -C customer.rpd -G original.rpd -O diff.csv
Give password for customer repository: my_cust_password
Give password for original repository: my_orig_password
```

This example generates a comparison diff file in CSV format called *diff.csv* from the *customer.rpd* and *original.rpd* repositories.

```
comparerpd -C customer.rpd -G original.rpd -D my_patch.xml
Give password for customer repository: my_cust_password
Give password for original repository: my_orig_password
```

This example generates an XML patch file called *my_patch.xml* from the *customer.rpd* and *original.rpd* repositories.

Turn Off Compare Mode

You can remove marks applied to objects while using the Compare Repositories and Merge Repositories options.

The **Turn off Compare Mode** option is only available after you've clicked **Mark** during the Compare action. If no repository object is marked, this option isn' available.

- In the Administration Tool, select **File**, then select **Turn off Compare Mode**.

Equalize Objects

If you've objects in two repositories that have the same name but different Upgrade IDs, you may want to treat them as the same object.

Use the `equalizerpds` utility to give the objects in the repositories the same Upgrade ID. You can equalize objects as part of the merge process.

You can also use the Equalize Objects dialog, available from the Compare repositories dialog, to preview the repository after you run the `equalizerpds` utility.

This section contains the following topics:

- About Equalizing Objects
- Use the Equalize Objects Dialog
- Use the equalizerpds Utility

About Equalizing Objects

You might need to equalize objects because the Oracle BI Administration Tool tracks the history of each repository object using the Upgrade ID of the object.

The Upgrade ID is a unique identifier for each object.

Sometimes, the Upgrade ID can change because of user actions or during merge. When this occurs, and a subsequent comparison is done, the Oracle BI Administration Tool treats the new Upgrade ID as a new object, and the missing original Upgrade ID as a deleted object.

For example, assume you've two identical repositories. In one repository, delete a presentation column, then re-create it again. When you compare the two repositories using the Compare repositories dialog, there are two entries for the presentation column: one that shows the old object as deleted, and one that shows the new object as created. Without using the Compare repositories dialog, it's hard to tell that this action occurred, because the Administration Tool typically shows only the object name and properties, not the underlying Upgrade ID.

The Upgrade IDs aren't unique, in rare cases the repositories that you want to merge might contain the same Upgrade ID. Running the `equalizerpds` utility on the repositories corrects the duplicate Upgrade ID issue and prevents an error while performing the merge.

Run the `equalizerpds` utility on your repositories before merging them to equalize your changes. Equalizing any opposing changes such as a column that's been duplicated and then renamed, cleans up the underlying Upgrade IDs, and prevents unintended renaming during the merge.

When you equalize objects, you can lose track of object renames because legitimate object renames become different objects. Intentional renames you did in the repository might change to different Upgrade IDs, so subsequent merges erroneously treat the renamed object as a new object. To avoid the renaming and new object situation, enter the before and after names of intentionally renamed objects in a rename map file that you then pass to the utility. The `equalizerpds` utility uses the information in the file to ensure that the original IDs are used in the renamed current objects.

You can view the Upgrade ID for repository objects using the Query Repository dialog. See View the Upgrade ID for Repository Objects.

View the Upgrade ID for Repository Objects

You can view the Upgrade ID for repository objects using the Query Repository dialog. When you use this task, a new column showing the Upgrade IDs displays in the Results list.

The **Upgrade ID** isn' available as a column option unless you've selected **Show Upgrade ID in Query Repository** in the General tab of the Options dialog. See Set Administration Tool Options.

1. In the Oracle BI Administration Tool, open a repository in *offline mode*.
2. Select **Tools**, then select **Query Repository**.
3. In Query Repository, click **Columns**.
4. In Select columns, select **Upgrade ID** from the list or use **Find** to help locate the *Upgrade ID*.
5. Click **Query**.

Use the Equalize Objects Dialog

The Equalize Objects dialog provides a preview of what your repository looks like if you run the `equalizerpds` utility on it.

The Equalize Objects dialog provides a convenient way to compare changes related to objects that have the same name, but it doesn't persist any of the changes.

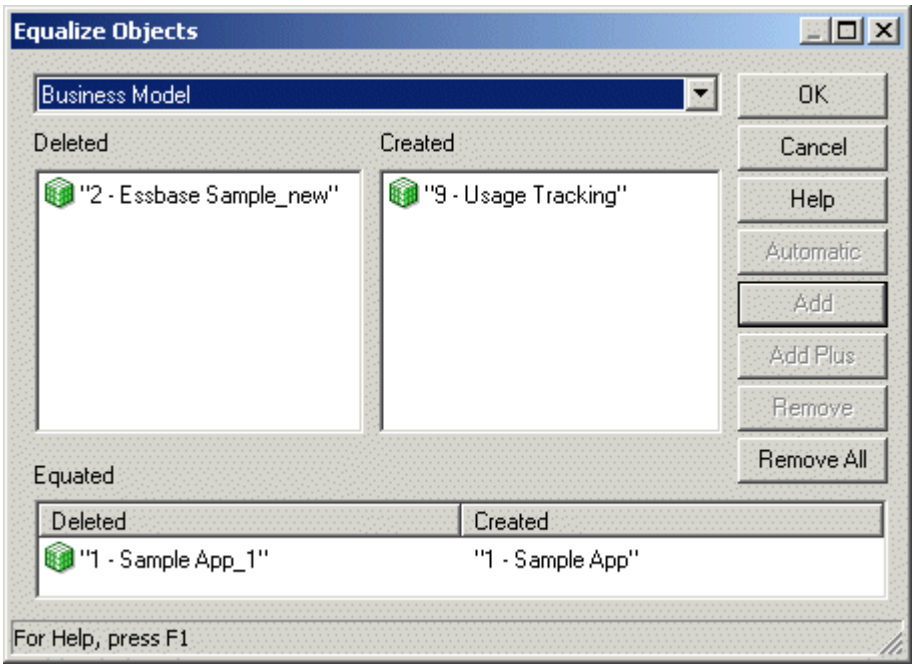
Using the Equalize Objects dialog can be a very slow process for larger repositories.

1. In the Administration Tool, open your repository in offline mode.
2. From the **File** menu, select **Compare**.
3. In the Select Original Repository dialog, click **Repository** from the submenu to select a binary repository file in Oracle BI repository file format, or select **XML** to select a set of MDS XML documents.
4. In the Open Offline dialog, enter the repository password and click **OK**.
5. In the Compare repositories dialog, click **Equalize**.
6. The Equalize Objects dialog shows a list of changes to consider objects with different Upgrade IDs to the same object. You can use the following options to model how the changes might get equalized:
   - Click **Automatic** to automatically equalize changes related to objects that have the same name. The changes appear in the Equated table.

     If no changes can be automatically equalized, nothing appears in the table, and the **OK** button remains disabled.
   - Select an object in the Deleted list, then select the equivalent object in the Created list and click **Add** or **Add Plus** to equate the objects. **Add Plus** adds the object along with its child objects to the Equated table, while **Add** simply adds the selected object. For example, if you select a Subject Area and click **Add Plus**, the underlying Presentation Tables and Presentation Columns are added as well.

     After you make a manual selection, the **Automatic** button is disabled.
   - Select a row in the Equated table and select **Remove** or **Remove All** to remove objects from the Equated table. **Remove All** removes the object along with its child objects, while **Remove** simply removes the selected object

     The **Automatic** button is enabled after all manual selections are removed.
7. When you're finished modeling the changes, click **OK**. The changes appear in the Compare Repositories dialog, but the changes don't persist after you close the dialog.



Use the equalizerpds Utility

You can use the `equalizerpds` utility to equalize the Upgrade ID of objects in two separate repositories. If objects have the same Upgrade ID, they're considered to be the same object. The utility compares Upgrade IDs from the first repository (the original repository) with Upgrade IDs from the second repository (the modified repository). Then, the utility equalizes the Upgrade IDs of objects with the same name, using the Upgrade ID from the original repository.

The `equalizerpds` utility is available on both Windows and Linux systems. You can only use `equalizerpds` with binary repositories in Oracle BI repository file format.

Provide the full path names to your repository files, both the input files and the output file, if they're located in a different directory.

The location of the `equalizerpds` utility is:

BI_DOMAIN`/bitools/bin`

Syntax

The `equalizerpds` utility takes the following parameters:

```
equalizerpds [-B original_repository_password] -C
original_repository_name
{-E modified_repository_password]|-G} -F modified_repository_name [-I
input_UDML_script_name][-J rename_map_file]
[-O output_repository_name] [-R output_apply_result_file] [-U
output_id_map_file][-Y equalStringSet]
```

Where:

`G` specifies to use the original repository password for the modified repository password.

rename_map_file is a text file containing a list of objects that were renamed and that you want to equalize. The format is a tab-separated file with the following columns:

```
TypeName      Name1      Name2
```

For example, to include a logical column in the map file that was renamed from Name1 to Name2, provide the following:

ATTRIBUTE *"BusinessModel"."Table"."Name1" "BusinessModel"."Table"."Name2"*

Don't cut and paste this example as the foundation for your own file, because the tab separators might not get copied properly. Create a new file with proper tabs.

See About Values for TypeName.

`equalStringSet` is a set of characters that you want to treat as equal.

The *original_repository_password* and *modified_repository_password* arguments are optional. If you don't provide these password arguments, you're prompted to enter the passwords when you run the command (*password1* and *password2*). To minimize the risk of security breaches, Oracle recommends that you don't provide password arguments on the command line or in scripts. The password arguments are supported for backward compatibility only. For scripting purposes, you can pass the password through standard input.

For example:

```
equalizedrpds -C original.rpd -F modified.rpd -O modified_equalized.rpd
password1: my_original_rpd_password
password2: my_modified_rpd_password
```

In this example, *original.rpd* is compared with *modified.rpd*, the Upgrade IDs are equalized using the Upgrade IDs from *original.rpd*, and the final result is written to modified_equalized.rpd.

About Values for TypeName

Learn about the available object types and their corresponding values for TypeName.

The table shows the available object types and their corresponding values for `TypeName`.

Object Type	Value for TypeName
Database	DATABASE
Connection Pool	CONNECTION POOL
Physical Catalog	CATALOG
Physical Schema	SCHEMA
Physical Display Folder	PHYSICAL DISPLAY FOLDER
Physical Table	TABLE
Physical Key	TABLE KEY
Physical Foreign Key	FOREIGN KEY
Physical Column	COLUMN
Physical Complex Join	JOIN

Object Type	Value for TypeName
Physical Hierarchy	HIERARCHY
Physical Level	PHYSICAL LEVEL
Cube Column	COLUMN
Cube Table	CUBE TABLE
LDAP Server	LDAP SERVER
Custom Authenticator	CUSTOM AUTHENTICATOR
Variable	VARIABLE
Application Role	SECURITY ROLE
User	USER
User Database Signon	USER DATABASE SIGNON
Project	PROJECT
Business Model	SUBJECT AREA
Logical Dimension	DIMENSION
Logical Level	LEVEL
Logical Display Folder	LOGICAL DISPLAY FOLDER
Logical Table	LOGICAL TABLE
Logical Source Folder	LOGICAL SOURCE FOLDER
Logical Table Source	LOGICAL TABLE SOURCE
Logical Column	ATTRIBUTE
Logical Join	ROLE RELATIONSHIP
Logical Key	LOGICAL KEY
Logical Foreign Key	LOGICAL FOREIGN KEY
Presentation Catalog	CATALOG FOLDER
Presentation Table	ENTITY FOLDER
Presentation Column	FOLDER ATTRIBUTE
Presentation Hierarchy	PRESENTATION HIERARCHY
Presentation Level	PRESENTATION LEVEL
Catalog Link	CATALOG LINK
Target Level	CUSTOMER TYPE
List Catalog	LIST CATALOG
Qualified Item	QUALIFIED ITEM
Qualifying Key	QUALIFYING KEY
Sampling Table	SAMPLING TABLE
Segmentation Catalog	SEGMENTATION CATALOG

Merge Repositories

You can use the Merge Repository Wizard in the Administration Tool to merge Oracle BI repositories.

You can merge repositories in binary Oracle BI repository (RPD) format, or MDS XML format. There are three types of merges:

- Full merges are typically used during development processes, when there are two different repositories that need to be merged. The Administration Tool provides a three-way merge feature that lets you merge two repositories that have both been derived from a third, original repository. Full merges can also be used to import objects from one repository into another.
- Patch merges are used when you're applying the differential between two versions of the same repository. For example, you might want to use a patch merge to apply changes from the development version of a repository to your production repository, or to upgrade your Oracle BI Applications repository.

  By default, the `patchrpd` utility's merge functionality uses patch mode. If you want to set up the Merge Repository Wizard to match the `patchrpd` utility's default merge functionality, then you must set up the Merge Repository Wizard to run in *patch* mode.
- Multiuser development merges are used when you're publishing changes to projects using a multiuser development environment, see About the Multiuser Development Merge Process.

See Merge Rules to learn how repository objects are merged.

This section contains the following topics:

- Perform Full Repository Merges
- Perform Patch Merges

Perform Full Repository Merges

You can use the Oracle BI Administration Tool to merge different repositories.

This section describes how to use the full (standard) repository merge feature in the Administration Tool.

This section contains the following topics:

- About Full Repository Merges
- Perform Full Repository Merges With a Common Parent
- Perform Full Repository Merges Without a Common Parent

About Full Repository Merges

The merge process typically involves three versions of an Oracle BI Repository: the original repository, modified repository, and current repository.

The original repository is the original repository (the parent repository), while the modified and current repository are the two repositories to merge. The current repository is the one open in the Administration Tool.

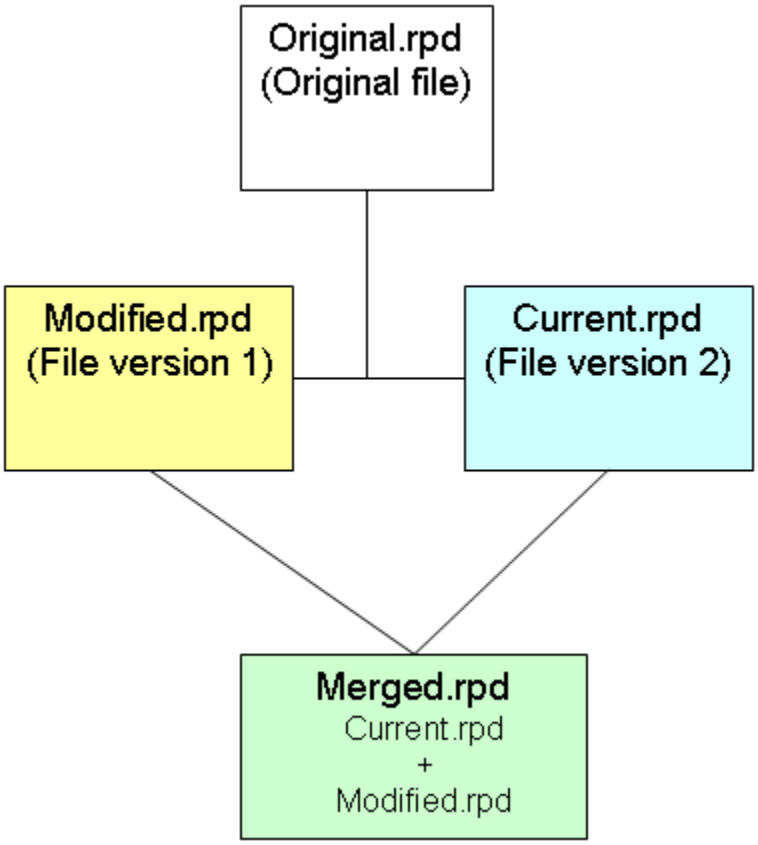
During the merge process, the Administration Tool compares the original repository with the modified repository and the original repository with the current repository. Conflicts occur when there are unresolved changes resulting from the two comparisons. For example, a conflict occurs if you rename object A to B in the modified repository, but you rename object A to C in the current repository.

The Merge Repository feature lets you decide on an object-by-object basis which changes you want to keep in the final merged repository. If there are no conflicts, merging is automatic.

There are two types of full merge:

- Common Parent. This merge, also called a three-way merge, is useful when you've a common parent repository and two derived repositories. There is a parent (original) repository, and two derived repositories. After the merge, a fourth merged repository is created.

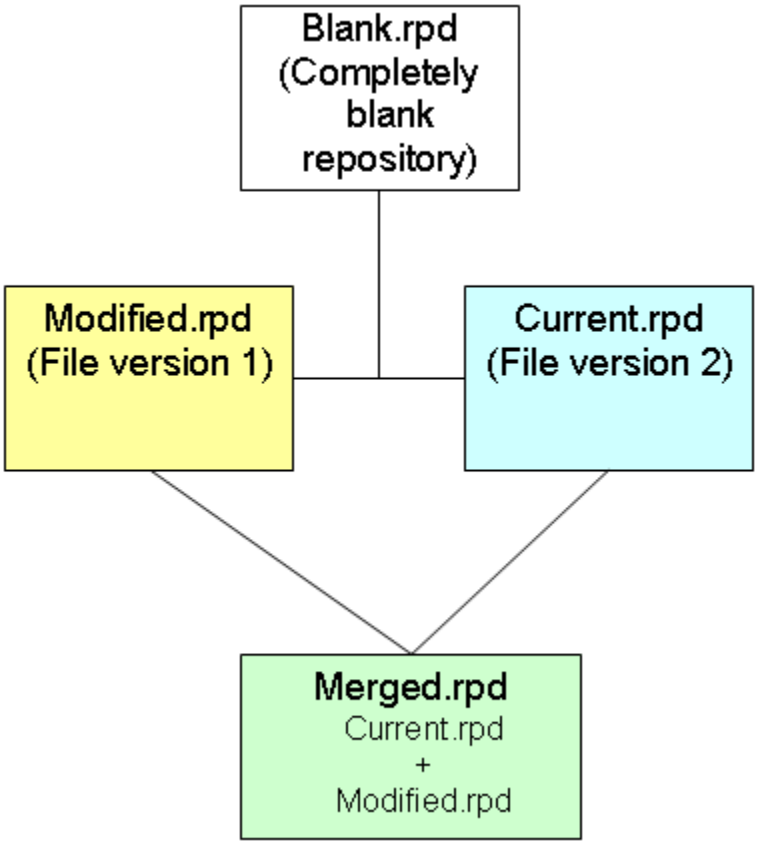
  The example shown in the figure below assumes you're merging binary Oracle BI (RPD) repositories, but you can also perform this type of merge with MDS XML repositories.



- No Common Parent

  A merge when the objects don't have a common parent is also called a two-way merge. Use a two-way merge when to import objects from one repository to another repository. To perform a merge of objects with no common parent use a three-way merge in the Administration Tool with a blank repository as the original repository.

  The example shown in the image below, you're merging binary Oracle BI (RPD) repositories. You can also perform this type of merge with MDS XML repositories.



Perform Full Repository Merges With a Common Parent

Learn how to use the Oracle BI Administration Tool to perform a full repository merge with a common parent.

Use this approach when you've an **original** parent repository and would like to merge the changes made to objects in two modified repository versions, **current** and **modified**. Objects that don't exist in the current repository are created as new objects.

See Equalize Objects and Merge Strategies Reference.

1. In the Administration Tool, open the **current** repository in offline mode.
2. From the Administration Tool menu, select **File**, then select **Merge**.
3. In the Select Input Files screen, for **Merge Type**, select **Full Repository Merge**.
4. Select the original parent repository by clicking **Select** next to **Original Master Repository**. Select **Repository** from the submenu to select a binary repository file in Oracle BI repository file format, or select **XML** to select a set of MDS XML documents.

   For binary repositories, browse to select the original repository, and then click **Open**. For MDS XML format repositories, use the Browse For Folder dialog to select the root folder location of the MDS XML documents, and then click **OK**.
5. Provide the password for the original repository in the appropriate **Repository Password** field.

6. Select the modified repository by clicking **Select** next to the **Modified Repository** field. Select **Repository** from the submenu to select a binary Oracle BI repository file in RPD format, or select **XML** to select a set of MDS XML documents.

   For binary repositories, browse to select the modified repository, and then click **Open**. For MDS XML format repositories, use the Browse For Folder dialog to select the root folder location of the MDS XML documents, and then click **OK**.

7. Provide the password for the modified repository in the appropriate **Repository Password** field.

8. Optional: You can change the default name and location of the saved (merged) repository by clicking **Select** next to the **Save Merged Repository as** field. Select **Repository** from the submenu to save the merged repository as a binary repository file in Oracle BI repository file format, or select *XML* to save the merged repository as a set of MDS XML documents.

   For binary repositories, provide a new name and location, and then click **Save**. For MDS XML format repositories, use the Browse For Folder dialog to select the root folder location of the MDS XML documents, and then click **OK**.

9. It's a good practice to equalize your changes to clean up underlying object IDs before merging. If you haven't yet equalized your changes, select **Equalize during merge** to equalize objects as part of the merge process. Selecting this option may affect merge performance.

10. Click **Next**. If there are any conflicts, the Define Merge Strategy screen of the Merge Repository Wizard appears. If there are no conflicts, the Merge Repository Wizard closes.

11. The Define Merge Strategy screen displays a decision table that shows conflicts for this merge.

    To make decisions about whether to include or exclude objects from the merged repository, choose **Current** or **Modified** from the **Decision** list. Choose **Current** to keep the change for the selected object in the current repository, or choose **Modified** to keep the change for the selected object in the modified repository.

    When you select an object in the decision table, the read-only text box below the decision table describes what changes were made to that object in the current repository. In addition, the tree panels at the bottom of the dialog show the affected objects for the selected row. Alternatively, you can select an object in one of the tree views to automatically highlight the corresponding row in the decision table.

    The **Modified** option in the **Decision** list displays a suffix that indicates whether the object is added to or deleted from the merged repository. **Modified (A)** indicates that the object is added, and **Modified (D)** indicates that the object is deleted.

    The type of conflict is displayed in the Description column of the Conflicts table. The decision choices you can make depend on the type of conflict shown in this column. The following list shows example results for different types of conflicts:

    - **Added to Current:** Choosing **Current** keeps the new object in the merged repository. Choosing **Modified (D)** deletes the new object from the merged repository.
    - **Deleted from Current:** Choosing **Current** keeps the repository as it's without adding the object to the merged repository. Choosing **Modified (A)** adds the object back into the merged repository.

- **Changed in both (different):** The object wasn't added or deleted, but at least one of its properties was modified. Click the plus sign (+) to the left of the row to view the property that was changed, as well as its value in the original, current, and modified versions of the repository. Property values are only shown for small-length strings. Longer-length strings like descriptions, features, and init strings aren't shown.

  Click the option for the value you want to retain in the merged version of the repository. For some properties, such as aliases, you can choose the **Merge Choices** option to merge the properties rather than choose one over the other. This option is only available if the properties can be merged.

  You typically don't need to make merge decisions regarding objects that have been added to or deleted from the Modified repository. However, you can view change statistics for this merge to see a summary of changes, including objects that have been added to or deleted from Modified.

After you make a merge decision, the row for that decision in the table changes from red to black. When all rows have a value in the Decision field, the **Finish** button is enabled.

12. In addition to making merge decisions, you can perform other operations in the Define Merge Strategies screen.
13. Click **Finish**.

Merge Strategies Reference

Use the table to help with merge decisions when merging repositories with a common parent.

The table lists and describes the elements in the Define Merge Strategies screen.

Element	Description
Conflicts table	The Conflicts table includes the following columns: • **Type:** The type of object for which there is a conflict, for example, Presentation Column. • **Name:** The name of the object for which there is a conflict. • **Description:** The reason for the conflict, such as Added to Current. See the previous step for a description of different conflict types. • **Decision:** Select the decision according to what change you want to keep in the merged repository, such as Current, Modified (A), Modified (D), or By Property. See the previous step for a description of the results of different decisions. For objects with properties that are modified in each repository, a sub-table (grid) is displayed with details of the changed properties. The grid includes the following columns: • **Property:** The name of the property that has been modified in each repository. • **Original:** The value of the property in the original repository. • **Modified:** The value of the property in the modified repository. Select this option to keep this value. • **Current:** The value of the property in the current repository. Select this option to keep this value. • **Merge Choices:** For some properties, like aliases, you can choose this option to merge the property values rather than choose one or the other.

Element	Description
Show qualified names	When selected, shows fully qualified names for objects in the decision table, for example, *"Paint"..."Month Year Ago fact"*. When the **Show qualified names** option is selected, some of the object names can be too long to fit into the cells of the decision table. Use the mouse to hover over the truncated name to see the full name of the object or property. Alternatively, you can manually resize columns, or double click the column separator to expand the column to the width of the object name.
Set Default Decisions	When clicked, allows you to choose how the Administration Tool resolves conflicts. Use this option when you don't want to set each conflict's decision manually. Select **All** if you want the Administration Tool to resolve all unresolved and resolved conflicts, that is conflicts for which you've already specified decisions. If you choose this option, then the Administration Tool checks the conflict decisions that you've already specified and changes them if necessary. Select **Only undecided** if you want the Administration Tool to resolve only the unresolved conflicts. That is, to assign decisions to all conflicts for which you haven't specified decisions. When you select this option, the Administration Tool preserves the conflict decisions that you've already specified.
Check consistency of the merged Oracle BI repository file	When selected, runs a consistency check before saving the merged file.
Hide object views	Select this option to hide the tree panels in the dialog. The tree panels show the affected objects for the row selected in the conflicts decision table. Hiding the tree panels can improve the performance of the Define Merge Strategies screen.
Save Decisions to File	Saves a file containing interim changes in the Repository subdirectory so that you can stop work on the merge and continue it later. After saving the changes (decisions), close the Merge repositories dialog by clicking **Cancel**. If there are a large number of decisions, you can save time by saving the merge decisions to a CSV file, opening the file in Excel or a text editor, and then modifying the merge decisions manually. Then, you can load the updated merge decisions file in the Define Merge Strategies screen, or include the decision file as an argument in `patchrpd`.
Load Decision File	Loads the saved decisions file from the Repository subdirectory so that you can continue processing a repository merge. This option is especially useful for resolving conflicts after running an automated patch merge using `patchrpd`. See Use patchrpd to Apply a Patch.
Find by Name or Type	Searches by an object Name and Type such as Initialization Block.
Find Again	Searches again for the most recent Find value.
View Change Statistics	Shows statistics for this merge, such as the number of objects deleted from the Modified repository, the number of objects that were changed in both repositories, and so on.
Details	Shows the text in the read-only text box below the decision table in a separate window.

Element	Description
View Original/Modified/ Current repository	Shows properties for the affected object in the selected repository.

Perform Full Repository Merges Without a Common Parent

Learn how to use the Oracle BI Administration Tool to perform a full repository merge without a common parent.

Use this method when you want to import objects from one repository, the **modified** repository, into another, the **current** repository.

In the repository you choose to define as **current**, make sure that the Presentation layer references any Physical layer or Business Model and Mapping layer objects that you want to keep. Objects like business models, databases, and connection pools in the current repository that aren't referenced by any Presentation layer objects are discarded during the merge. If necessary, you might want to add a placeholder subject area that references the objects before you perform the merge to ensure the desired objects are kept. See Merge Rules to learn which objects are retained or discarded during the merge process.

1. In the Administration Tool, select **File**, then select **New Repository** to serve as the original repository in the merge.
2. In the Create New Repository Wizard, provide a name for the repository, for example, `blank.rpd`.
3. In **Import Metadata**, choose **No**.
4. Enter and confirm the repository password you want to use for this repository.
5. Click **Finish**.
6. Close the blank repository.
7. Open the **current** repository containing the objects you want to import, in offline mode.
8. From the Administration Tool menu, choose **File**, then select **Merge**.
9. In Select Input Files, for **Merge Type**, select **Full Repository Merge**.
10. Click **Select** next to **Original Master Repository**, then click **Repository**.
11. Navigate to your blank repository file and click **Open** and leave the password field blank.
12. Next to the **Modified Repository** field, click **Select** to choose the destination repository.
    a. Select **Repository** from the submenu to select a binary Oracle BI repository file in RPD format, or select **XML** to select a set of MDS XML documents.
    b. For binary repositories, browse to select the modified repository, and then click **Open**.
    c. For MDS XML format repositories, use the Browse For Folder dialog to select the root folder location of the MDS XML documents, and then click **OK**.
13. Provide a password for the modified repository in the appropriate **Password** field.
14. Click **Next**.

    If there are any conflicts, the Merge Repository Wizard's Define Merge Strategy displays a message.
15. In Define Merge Strategy, choose **Current** or **Modified** from the **Decision** list.

When you select an object in the decision table, the read-only text box below the decision table describes what changes were made to that object in the current repository. Refer to the table for information about additional options in Define Merge Strategy such as saving merge decisions to a comma-separated values (.csv) file.

After you make a merge decision, the row for that decision in the table changes from red to black.

16. Click **Finish**.

Perform Patch Merges

You can use the patch merge to generate an XML patch file that contains only the changes made to a repository.

This patch can be then applied to the old (original) version of the repository to create the new version.

This merge method is very useful for development-to-production scenarios, and can also be used for Oracle BI Applications customers to upgrade their repository.

This section explains how to generate a patch that contains the differences between two repositories, and then apply the patch to a repository file.

This section contains the following topics:

- About Patch Merges
- Generate a Repository Patch
- Apply a Repository Patch

About Patch Merges

In a patch merge, you create a patch that contains the differences between the current repository and the original repository.

You apply the patch file to the modified repository. Differences between the current and original repository must exist in matching existing projects in both repositories.

In a development-to-production scenario, you've an original parent repository, a current repository that contains the latest development changes, and a modified repository that's the deployed copy of the original repository.

To generate a patch, you open the current repository and select the original repository, then create the patch.

To apply the patch, you open the modified repository and select the original repository, then apply the patch.

In an Oracle BI Applications repository upgrade scenario, the current repository is the latest version of the repository shipped by Oracle, and the original repository is the original repository shipped by Oracle. The modified repository is the one that contains the changes you made to the original repository.

Generate a Repository Patch

Use the Oracle BI Administration Tool to generate a patch that contains the differences between two repositories.

If changes are made using projects, you must add metadata changes to an existing project. Changes made to a new project are lost during the patch merge process.

See Equalize Objects.

1. In the Administration Tool, open the **current** Oracle Analytics Server repository in offline mode. In other words, open the updated repository that contains the changes you want to put in the patch.
2. Select **File**, then select **Compare**.
3. Select the **original** Oracle Analytics Server repository. Select **Repository** from the submenu to select a binary Oracle BI repository file in RPD file format, or select **XML** to select a set of MDS XML documents.
4. In the Open Offline dialog, enter the repository password and click **OK**.
5. It's a good practice to equalize your changes to clean up underlying object IDs before generating a patch.
6. In the Compare repositories dialog, review the changes between the repositories. Then, click **Create Patch**.
7. In the Create Patch dialog, enter a name for the patch file, for example, `my_patch.xml`, and click **Save**.

If the patch contains passwords, such as connection pool passwords, the patch file is encrypted using the repository password from the current repository. The current repository password effectively becomes the patch file password. You might need to supply this patch file password when applying the patch, if it's different from the repository password for the original repository.

You can also generate a patch at the command line using the `comparerpd` utility. See Comparing Repositories Using comparerpd.

Apply a Repository Patch

Use the Oracle BI Administration Tool to apply a patch that contains the differences between two repositories.

You can apply patches from a larger multiuser repository to a smaller subset extract repository. In this case, only the changes in the subset are applied from the patch.

1. In the Administration Tool, open the Oracle BI repository in offline mode to apply the patch.
2. Select **File**, then select **Merge**. The Merge Repository Wizard appears.
3. For **Merge Type**, select **Patch Repository Merge**. When you select this option, the **Partial Subset Merge** field displays. By default, this field is selected and the patch is applied as a partial subset merge. Clear this option if you want the patch applied to the whole repository.

4. Select the original parent repository by clicking **Select** next to **Original Master Repository**. Select **Repository** from the submenu to select a binary Oracle BI repository file in RPD format, or select **XML** to select a set of MDS XML documents.

   For binary repositories, browse to select the original repository, and then click **Open**. For MDS XML format repositories, use the Browse For Folder dialog to select the root folder location of the MDS XML documents, and then click **OK**.

   The original repository can't be the same as the modified (currently open) repository.

5. Enter the repository password for the original repository.
6. Click **Select** next to **Patch File**. Browse to select the patch file you want to apply, then click **Open**. The patch file must be in XML format.
7. In most cases, you can leave the **Patch Password** field blank. You only need to supply the patch file password when the patch file contains passwords for objects, such as connection pool passwords, and when the patch file password is different from the original repository password. The patch file password is the same as the password for the current repository.
8. Optional: You can change the default name and location of the saved (patched) repository by clicking **Select** next to the **Save Merged Repository as** field. Select **Repository** from the submenu to save the merged repository as a binary Oracle BI repository file in RPD format, or select **XML** to save the merged repository as a set of MDS XML documents.

   For binary repositories, provide a new name and location, and then click **Save**. For MDS XML format repositories, use the Browse For Folder dialog to select the root folder location of the MDS XML documents, and then click **OK**.

9. Click **Finish**.

Use patchrpd to Apply a Patch

You can apply a patch using the patchrpd utility.

Use `patchrpd` when you want to patch repositories on Linux systems where the Administration Tool isn't available. The `patchrpd` utility is available on both Windows and Linux systems. You can only run `patchrpd` with binary Oracle BI repositories in RPD format.

Unlike the Administration Tool patch feature, `patchrpd` provides an option to apply default decisions for conflicts automatically. `patchrpd` also provides the capability to save all conflicts to a decision file so that you can examine the results and determine whether additional manual changes are needed. See Using Patchrpd to Automate the Patch Process.

`Patchrpd` also provides the ability to select the set of merge rules to apply. By default, the merge rules for patches are used, but you can optionally select to use the full (upgrade) merge rules or the multiuser development merge rules.

The arguments for all passwords, including the *modified_rpd_password*, *original_rpd_password*, and *patch_file_password*, are optional. If you don't provide password arguments, you're prompted to enter any required passwords when you run the command. To minimize the risk of security breaches, Oracle recommends that you don't provide password arguments either on the command line or in scripts. The

password arguments are supported for backward compatibility only. For scripting purposes, you can send the password through standard input.

Provide the full path names to all files, including the repository files and the XML patch file, if they're located in a different directory.

The location of the `patchrpd` utility is:

BI_DOMAIN`/bitools/bin`

Syntax

The `patchrpd` utility takes the following parameters:

```
patchrpd [-P modified_rpd_password] -C modified_rpd_pathname
[-Q original_rpd_password] -G original_rpd_pathname [-S patch_file_password]
-I patch_file_pathname [-S patch_file_2_password -I patch_file_2_pathname ...]
[-D input_decision_file_pathname] -O output_rpd_pathname
[-V output_decision_file_pathname] [-E] [-M mode] [-R] [-A] [-U] [-N] [-8]
```

Where:

`-P` *modified_rpd_password* is the repository password for the modified repository, also called the customer or customized repository.

`-C` *modified_rpd_pathname* is the name and location of the modified repository.

`-Q` *original_rpd_password* is the repository password for the original repository.

`-G` *original_rpd_pathname* is the name and location of the original repository.

`-S` *patch_file_password* is the password for the patch file. The patch file password is the same as the password for the current repository. You only need to supply the patch file password when the patch file contains passwords for objects, such as connection pool passwords, and when the patch file password is different from the original repository password.

`-I` *patch_file_pathname* is the name and location of the XML patch file you want to apply. You can apply multiple patches by including multiple `-I` arguments with paired `-S` arguments, as needed.

`-D` *input_decision_file_pathname* is the name and location of a decision file in CSV format that you want to use to affect the behavior of this patch merge. This argument is optional.

`-O` *output_rpd_pathname* is the name and location of the Oracle BI repository output file you want to generate.

`-V` *output_decision_file_pathname* is an optional argument that lets you generate a CSV format decision file. The decision file shows all the conflicts from the merge. In other words, the decision file lists the decisions that would have been displayed in the Define Merge Strategy screen of the Merge Wizard in the Administration Tool. The decision file provides a record of all items that could be influenced by user input. This argument must be used in conjunction with the `-U` argument.

`-E` is an optional argument that enables you to skip the equalize step.

`-M mode` is an optional argument that enables you to change the mode of the merge. By default, `patchrpd` runs in patch mode, in which as many changes as possible are applied silently. To change this default, then for mode specify *mud* to use merge rules for multiuser development merges, or *upgrade* to use merge rules for full merges. See Merge Rules.

By default, the Administration Tool's merge functionality uses full merge. If you want to run the `patchrpd` utility to match the Administration Tool's default merge functionality, then you must specify "upgrade" in the `-M mode` argument.

`-R` is an optional argument that skips consistency checking for logical columns. This option can speed up the patch process when the patch file doesn't contain logical to physical column mappings.

`-A` is an optional argument that can be used in multiuser development environments to skip subset patching and instead apply the patch using input repository files.

`-U` is an optional argument that applies default decisions for conflicts automatically so that `patchrpd` can finish successfully. If you don't include this parameter, `patchrpd` displays a warning and exits if a conflict is detected.

`-N` is an optional argument that's used to ignore all non-fatal errors. Examples of non-fatal errors are unresolved objects, duplicated objects, and broken or incorrect expressions.

`-8` specifies UTF-8 encoding.

For example:

```
patchrpd -C customer.rpd -G original.rpd -I patch.xml -O patched.rpd
-V decision.csv -U
Give password for customer repository: my_modified_rpd_password
Give password for original repository: my_original_rpd_password
```

This example applies a patch called `patch.xml` to the *customer.rpd* repository, and then generates an output repository called *patched.rpd* and an output decision file called *decision.csv*.

Query and Manage Repository Metadata

You can use repository queries to help manage repository metadata and configure the repository to handle large complex queries.

This section contains the following topics:

- Query the Repository
- Query Related Objects
- Configure the Repository for Large Complex Queries
- Query Data Models Remotely

Query Related Objects

Query Related Objects enables querying objects related to one or more objects that you select from the Physical, Business Model and Mapping, or Presentation layer.

You can only use this feature with objects selected from the same layer. You can't, for example, query objects related to both a Physical layer object and a Business Model and Mapping layer object. See Repository Query Options.

1. In the Administration Tool, open your repository.

2. Select one or more objects of the same type from a single layer, for example, a set of logical columns from the Business Model and Mapping layer.
3. Right-click the objects and select **Query Related Objects**.
4. Select an object type to narrow your search to a particular type of object, or select **All Types** to query all objects related to your source objects.

After you select an object type, the Query Related Objects dialog is displayed, showing the objects related to your source objects in the **Name** list.

Repository Query Options

Review this topic to see options available in the Query Repository dialog.

Option	Description
Mark	Select one or more objects in the Name list and click **Mark** to mark the selected objects. To unmark the objects, select them and click **Mark** again. Marking objects makes them easier to visually identify as you develop metadata.
Set Icon	Select one or more objects in the Name list and click **Set Icon** to select a different icon for the objects. You can set special icons for objects to help visually identify them as having common characteristics. For example, you might want to pick a special icon to identify columns used only by a specific user group. To change the icons back to the original icons, select the objects and click **Set Icon** again. Then, select **Remove associated icon** and click **OK**.
Show Qualified Name	Use this option to display the fully qualified name of the objects found by the query. For example, if you query for logical columns, the default value in the **Name** list is the column name. However, if you select **Show Qualified Names**, the value in the **Name** list changes to `businessmodel.logicaltable.column`.
Show Parent	Select an object in the Name list and click **Show Parent** to view the parent hierarchy of an object. If the object doesn't have a parent, a message appears. You can't use Show Parent with users or application roles. In the Parent Hierarchy dialog, you can edit or delete objects. If you delete an object from this dialog, any child objects of the selected object are also deleted.
GoTo	Select one or more objects in the Name list and click **GoTo** to go to the objects in the Administration Tool view of the repository. The selected objects appear highlighted in the Physical, Business Model and Mapping, or Presentation layer. The Query Related Objects dialog closes when you choose this option.

Query the Repository

You can query for objects in the repository to examine and update the internal structure of the repository.

Query a repository and view reports that show such items as all tables mapped to a logical source, all references to a particular physical column, content filters for logical sources,

initialization blocks, and security and user permissions. Run a report before making any physical changes in a database that might affect the repository. You can save the report to a file in comma-separated value (CSV) or tab-delimited format.

You can construct a filter to restrict the results to display specific values, save a query, run a previously saved query, or create new repository objects. See Construct a Filter for Query Results.

1. In the Administration Tool, open your repository.
2. Select **Tools**, then select **Query Repository**.
3. In the Query Repository dialog, complete the query information using the table as a guide.
4. Click **Query**.

Construct a Filter for Query Results

Use the Query Repository Filter dialog to create criteria the select the data that you want returned in the results.

If you're constructing a complex filter, you might want to click **OK** after adding each constraint to verify that the filter construction is valid for each constraint.

You can construct multiple filters. When you do, the **Operator** field becomes active. When the **Operator** field is active, you can set `AND` and `OR` conditions.

See Query Filter Examples.

In the Options dialog on the General tab, select **Show Upgrade ID in Query Repository** to enable filtering by Upgrade ID.

1. In the Administration Tool, select **Tools**, then select **Query Repository**.
2. In the Query Repository dialog, select an item in the **Results** list or select an item from the **Type** list, and then click **Filter**.
3. In the Query Repository Filter dialog, click the **Item** field. The **Item** list contains the items by which you can filter.
4. In the **Item** list, select the filter that you want to apply to the Results or Type object you selected in the previous step.
5. Type information in the **Value** column, as appropriate.
6. Click **OK** to return to the Query Repository dialog.

Query Filter Examples

Review the examples to learn how to use filters in your queries.

Viewing All Databases Referenced In a Business Model

The following example shows how to create a filter that lets you view all databases referenced in a particular business model.

1. In the Query Repository dialog, select **Database** from the **Type** list, and then click **Filter**.
2. In the Query Repository Filter dialog, click the **Item** field, and then select **Related to**.

3. Click the ellipsis button to the right of the **Value** field, and in the list, choose **Select object**.
4. In the Select dialog, select the business model by which you want to filter, and then click **Select**. Your selection appears in the **Value** field.
5. Click **OK** to return to the Query Repository dialog. The filter appears in the box to the left of the **Filter** button.
6. Click **Query**. The Results list shows the databases referenced by the business model you selected.

Viewing All Presentation Columns Mapped to a Logical Column

The following example shows how to create a filter that lets you view all presentation columns mapped to a particular logical column.

1. In the Query Repository dialog, select **Presentation Column** from the **Type** list, and then click **Filter**.
2. In the Query Repository Filter dialog, click the **Item** field, and then select **Column**.
3. Click the ellipsis button to the right of the **Value** field, and in the list, choose **Select object**.
4. In the Select dialog, select the column by which you want to filter, and then click **Select**. Your selection appears in the **Value** field.
5. Click **OK** to return to the Query Repository dialog. The filter appears in the box to the left of the **Filter** button.
6. Click **Query**. The Results list shows the presentation columns mapped to the logical column you selected.

Nested Queries

The following example shows nested queries, where the filter itself is another query.

1. In the Query Repository dialog, select **Logical Column** from the **Type** list, and then click **Filter**.
2. In the Query Repository Filter dialog, click the **Item** field, and then select **Related to**.
3. Click the ellipsis button to the right of the **Value** field, and in the list, choose **Set Condition for Physical Column**.
4. In the new Query Repository Filter dialog, click the **Item** field, and then select **Source column**.
5. Click the ellipsis button to the right of the **Value** field, and in the list, choose **Select Object**.
6. In the Browse dialog, select a source physical column (for example, Column A) and click **Select**.
7. Click **OK** in the Query Repository Filter dialog for the subquery condition. This subquery queries all aliases for the source column you selected.
8. In the Query Repository Filter dialog for the main query, click the **Item** field in the next row and then select **Related to**.
9. Click the ellipsis button to the right of the **Value** field, and in the list, choose **Select Object**.

10. In the Browse dialog, select the same source physical column (for example, Column A) and click **Select**.
11. Select OR from the **Operator** list.
12. Click **OK** to return to the Query Repository dialog. The filter appears in the box to the left of the **Filter** button.
13. Click **Query**. The Results list shows a list of logical columns related to either Column A, or aliases of Column A.

Configure the Repository for Large Complex Queries

Change the `OBIS_DISABLE_QUERY_GOVERN_MEMORY` parameter value in the `obis.properties` file to support long and large complex queries that exceed the memory limits imposed by the Oracle BI Server.

The value of `OBIS_DISABLE_QUERY_GOVERN_MEMORY` itself doesn't prevent recursive queries from disrupting the availability of the Oracle BI Server. The memory limits are imposed for all queries by default to prevent service disruption. However, the limits might also prevent large, complex queries from running if the query would exceed the built-in limits. Disabling the memory limits of the query governor by setting the `OBIS_DISABLE_QUERY_GOVERN_MEMORY` value to 1 might allow large, complex queries to run. However, the Oracle BI Server is no longer protected from recursive queries that might disrupt its availability.

Consider changing the `OBIS_DISABLE_QUERY_GOVERN_MEMORY` value to bypass the memory limits if you see the following error message:

```
State: HY000. Code: 43119. [nQSError: 43119] Query Failed: (HY000)
State: HY000. Code: 59151. [nQSError: 59151] The user query with request
id:request_ID and logical hash:hash_number exceeded the maximum query governing
memory limit. (HY000)
```

1. Open the `obis.properties` file for editing from the following location:

   obiee_home`/user_projects/domains/bi/config/fmwconfig/bienv/OBIS/obis.properties`

2. Append the following to the file:

   ```
   OBIS_DISABLE_QUERY_GOVERN_MEMORY=1
   ```

3. Restart the Oracle BI Server, and retry the query.

Query Data Models Remotely

You can query data models remotely.

For information, see "About Integrating with the Oracle BI Server as a Data Source" in *Integrator's Guide for Oracle Business Intelligence Enterprise Edition*.

Change the Oracle BI Repository Password

Each Oracle BI repository has a password that's used to encrypt its contents.

You create the repository password when you create a new Oracle BI repository file.

You can change the repository password using the Administration Tool in offline mode, or using the `obieerpdpwdchg` utility. You can't change the repository password when the repository is open in the Administration Tool in online mode.

The `obieerpdpwdchg` utility is especially useful when you want to change the repository password on Linux systems where the Administration Tool isn't available.

This section contains the following topics:

- Change the Oracle BI Repository Password Using the Administration Tool
- Change the Oracle BI Repository Password Using the obieerpdpwdchg Utility

Change the Oracle BI Repository Password Using the Administration Tool

Learn how to change your password in the Administration Tool and publish a modified repository.

Use the Upload Repository Command to change the repository password and to publish the modified repository.

1. Open the repository in the Administration Tool in offline mode.
2. Select **File**, then select **Change Password**.
3. Enter the current (old) password.
4. Enter the new password and confirm it. The repository password must be longer than five characters and can't be empty.
5. Click **OK**.
6. Save and close the repository.

Change the Oracle BI Repository Password Using the obieerpdpwdchg Utility

Learn how to change the repository password.

Use these steps to change the repository password using the `obieerpdpwdchg` utility, and then publish the modified repository using the Upload Repository Command.

You must define the repository password with more than five characters. Passwords are masked on the command line unless you include the `-C` option in the command to disable masking.

1. Navigate to the `obieerpdpwdchg` utility, which is located here:

   BI_DOMAIN`/bitools/bin`

2. Type the following arguments for `obieerpdpwdchg`:

   - `-I` *name_and_path_of_existing_repository*
   - `-O` *name_and_path_of_new_repository*

   Then, enter the current (old) password and the new password when prompted, for example:

```
obieerpdpwdchg  -I my_repos.rpd -O my_changed_repos.rpd
Please enter the repository password:
```

```
Please enter a new repository password:
```

18

Use Expression Builder and Other Utilities

This chapter describes Expression Builder and provides instructions for creating constraints, aggregations, and other definitions within the Oracle BI repository. It also describes the various utilities and wizards contained in the Oracle BI Administration Tool.
This chapter contains the following topics:

- Use Expression Builder
- Use Administration Tool Utilities
- Use the Calculation Wizard

Use Expression Builder

You can use the Expression Builder dialogs in the Oracle BI Administration Tool to create constraints, aggregations, and other definitions within a repository.

Expression Builder provides automatic color highlighting and other formatting enhancements to make expressions easier to build and to read.

The expressions you create with Expression Builder are similar to expressions created with SQL. Except where noted, you can use all expressions constructed with Expression Builder in SQL queries against the Oracle BI Server.

This section contains the following topics:

- About the Expression Builder Dialogs
- About the Expression Builder Toolbar
- About the Categories in the Category Pane
- Set Up an Expression

About the Expression Builder Dialogs

When creating expressions in Expression Builder, you select a category from the Category pane and values are displayed in the lower panes depending on the value selected in the Category pane.

When you type a value into a Find field, it filters out the non-matching strings and displays matching strings only. You can only enter text in the Find field that matches the text of one of the available strings. For example, if the available string options begin with A11, A12, and A13, the text you enter in the Find field must begin with A. After typing search criteria in a Find field, you can move up and down the list using the scroll bar, and use the tab key to move between the Find fields. To return to the full list of results, delete the string from the Find field.

When you first open Expression Builder, the items aren't sorted. When selected, the **Sort Panes** option sorts all items in the panes. As soon as you select this option, the panes are automatically redrawn without changing the contents of the panes or your filtering criteria.

About the Expression Builder Toolbar

The Expression Builder toolbar is located at the bottom of Expression Builder.

The table describes each button and its function in an expression.

Operator	Description
+	Plus sign for addition.
-	Minus sign for subtraction.
*	Multiply sign for multiplication.
/	Divide by sign for division.
||	Character string concatenation.
(	Open parenthesis.
)	Close parenthesis.
>	Greater than sign, indicating values higher than the comparison.
<	Less than sign, indicating values lower than the comparison.
=	Equal sign, indicating the same value.
<=	Less than or equal to sign, indicating values the same or lower than the comparison.
>=	Greater than or equal to sign, indicating values the same or higher than the comparison.
<>	Not equal to, indicating values higher or lower, but different.
AND	`AND` connective, indicating intersection with one or more conditions to form a compound condition.
OR	`OR` connective, indicating the union with one or more conditions to form a compound condition.
NOT	`NOT` connective, indicating a condition isn't met.
,	Comma, used to separate elements in a list.

About the Categories in the Category Pane

The categories that appear in the Category pane vary, depending on the dialog from which you accessed Expression Builder.

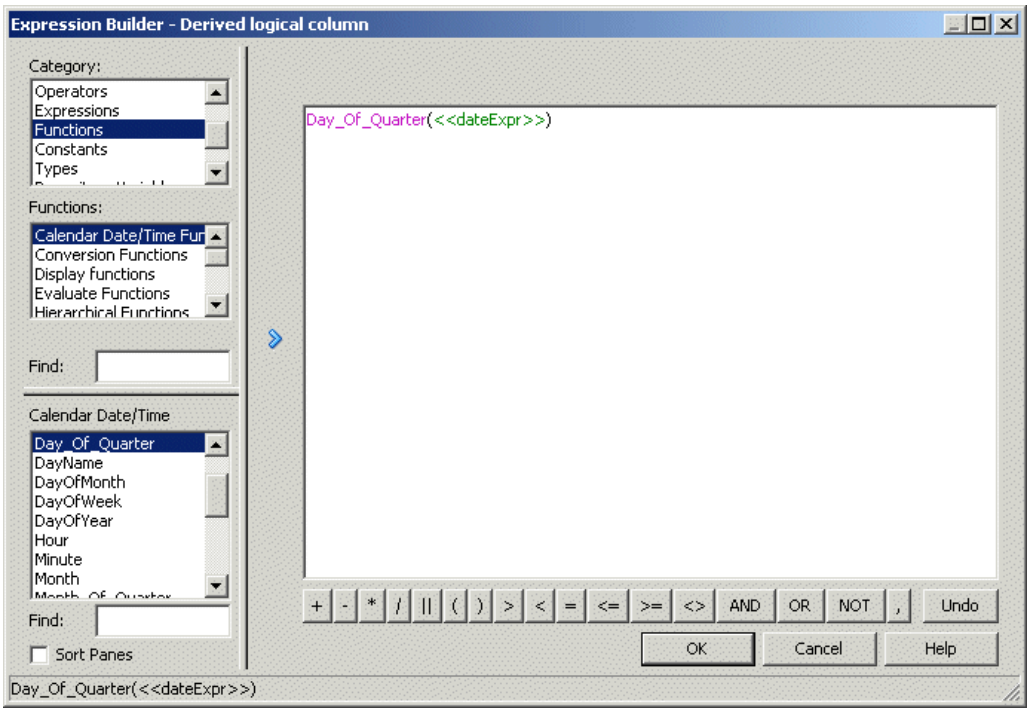
Category Name	Description
Aggregate Content	Contains the available aggregate functions. Aggregate sources must use one of the functions listed here to specify the level of their content.
Time Dimensions	Contains the time dimensions configured in the business model. If no time dimensions exist in a business model, or if time dimensions aren't pertinent to a particular Expression Builder, the Time Dimensions category isn't displayed. When you select the Time Dimensions category, each configured time dimension appears in the middle pane, and each level for the selected dimension appears in the lower pane.

Category Name	Description
Logical Tables	Contains the logical tables configured in the business model. If logical tables aren't pertinent to a particular Expression Builder, the Logical Tables category isn't displayed. When you select the Logical Tables category, each logical table in the business model appears in the middle pane, and each column for the selected logical table appears in the lower pane.
Value Based Dimensions	Contains the dimensions with parent-child hierarchies configured in the business model. If no dimensions with parent-child hierarchies exist in a business model, or if dimensions with parent-child hierarchies aren't pertinent to a particular Expression Builder, the Value Based Dimensions category isn't displayed. When you select the Value Based Dimensions category, the configured dimensions with parent-child hierarchies appear in the middle pane. No lower pane exists for this category.
Logical Levels	Contains the related logical levels. If level-based dimensions aren't pertinent to a particular Expression Builder, the Logical Levels category isn't displayed. When you select the Logical Levels category, you can then select the appropriate logical dimension (level-based) in the middle pane, and the level itself in the lower pane.
Physical Tables	Contains the related physical tables. If physical tables aren't pertinent to a particular Expression Builder, the Physical Tables category isn't displayed.
Operators	Contains the available SQL logical operators.
Expressions	Contains the available expressions.
Functions	Contains the available functions. The functions that appear depend on the object you selected.
Constants	Contains the available constants.
Types	Contains the available data types.
Repository Variables	Contains the available repository variables. If no repository variables are defined, this category doesn't appear.
Session Variables	Contains the available system session and non-system session variables. If no session variables are defined, this category doesn't appear.

Set Up an Expression

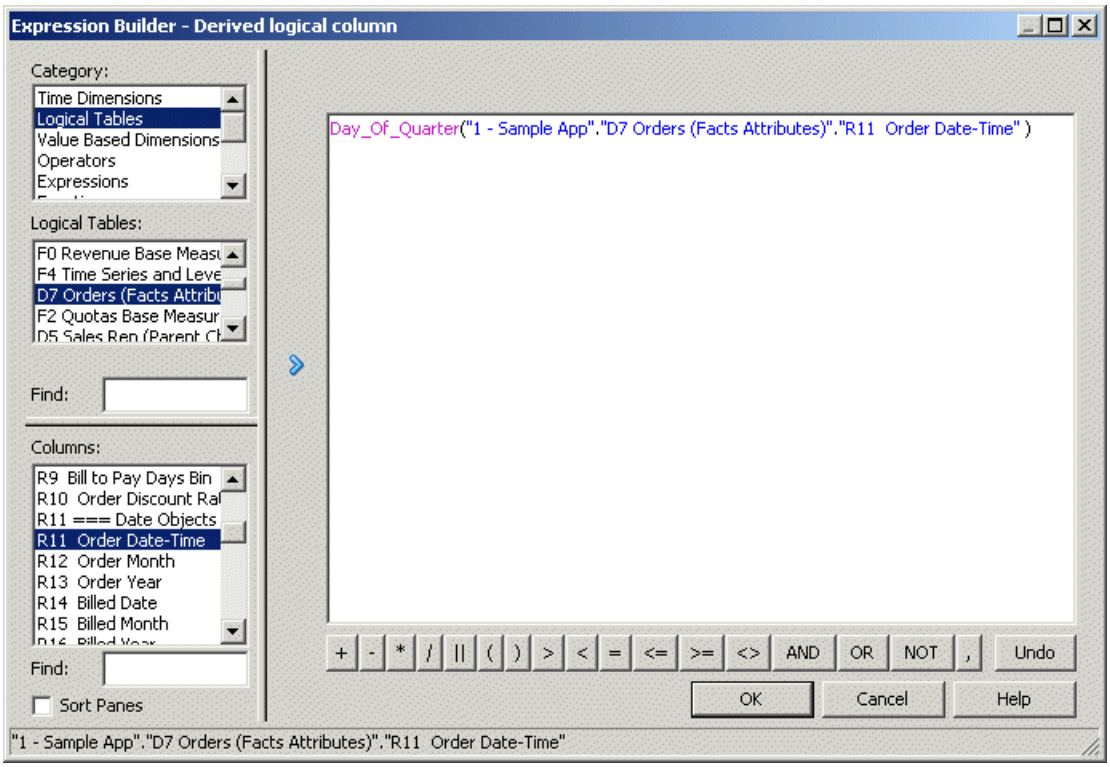
You can view the Expression Builder dialog for a derived logical column.

To set up an expression, select Functions from the Category pane, select a function type from Functions pane, then select a function from the lower pane.



Double-click the function you want to use to paste it in the edit pane. Then, in the edit pane, click once between the parentheses of the function to select that area as the insertion point for adding the argument of the function.

To paste a logical column at the insertion point, select Logical Tables from the Category pane, select the table you want to use in the Logical Tables pane, and then double-click the logical column in the lower pane to paste the logical column at the insertion point as the argument of the function in the edit pane. The image shows where the expression appears in the edit pane.



Navigate Within Expression Builder

Use these steps to navigate within Expression Builder.

1. In the Category pane, select the appropriate category for the expression you want to build.

   The available expression types for the selected category appear in the middle pane.
2. Select the appropriate item for the expression you want to build.

   The available building blocks for the selected item appear in the lower pane.
3. Double-click a building block to display it in the edit pane.
4. To insert an operator into the expression, click an operator on the Expression Builder toolbar.

Build an Expression

Use these steps to build an expression in Expression Builder.

1. Navigate to the individual building blocks you want in the expression.

   The Syntax bar at the bottom of the Expression Builder dialog shows the syntax for the expression.

   For example: `BETWEEN <<Upper Bound>> AND <<Lower Bound>>`
2. Add the building blocks to the edit pane.
3. Edit the building blocks to reflect the expression you want.
4. Use the Expression Builder toolbar to insert operators into the expression.

5. Repeat the preceding steps until the expression is complete, and then click **OK**.

   The Administration Tool displays a message for any syntax errors in the expression. When the expression is syntactically correct, the Administration Tool adds the expression to the dialog from which you accessed Expression Builder.

If the parameter `PREVENT_DIVIDE_BY_ZERO` is set to `YES` in `NQSConfig.INI`, the Oracle BI Server prevents errors in divide-by-zero situations, even for Answers column calculations. The Oracle BI Server creates a divide-by-zero prevention expression using `nullif()` or a similar function when it writes the physical SQL. Because of this, you don't have to use `CASE` statements to avoid divide-by-zero errors, as long as `PREVENT_DIVIDE_BY_ZERO` is set to `YES` (the default value).

About the INDEXCOL Conversion Function

The INDEXCOL function enables you to build a derived logical column.

Selecting `INDEXCOL` automatically generates the following function template:

```
IndexCol( <<integer literal>>, <<expr1>> [, <<expr2>>, ?-] )
```

You can also use a session variable, an arithmetic expression, or a `CASE WHEN` statement, when an evaluation is possible without reference to back-end data, as the argument `integer literal`.

See *Logical SQL Reference Guide for Oracle Business Intelligence Enterprise Edition*.

Use Administration Tool Utilities

The Oracle BI Administration Tool provides several utilities and wizards that perform functions like renaming objects, persisting aggregates, and externalizing strings.

This section contains the following topics:

- Use the Replace Column or Table Wizard
- Use the Event Tables Utility
- Use the Externalize Strings Utility
- Use the Rename Wizard
- Use the Update Physical Layer Wizard
- Generate Documentation of Repository Mappings
- Generate a Metadata Dictionary
- Provide Access to Metadata Dictionary Information
- Remove Unused Physical Objects
- Persist Aggregates
- Use the Convert Presentation Folders Utility
- Generate a List of Logical Column Types
- Compare Logical Column Types
- Fix Upgrade IDs
- Set Permissions In Bulk

Use the Replace Column or Table Wizard

The Replace Column or Table Wizard automates the process of replacing physical columns or tables in logical table sources.

For example, if you've purchased Oracle BI Applications, you can update your logical table sources to map to a different database type. You can also use this utility to change logical table source mappings from a development table to a production table.

You can use the Replace Column or Table Wizard to replace a single column in the same table, or to replace an entire table. If you replace a table, you must map all the columns in the table.

If you select an invalid logical table source, or in other words, one that can't be used for replacement, a message appears explaining why that source can't be used, and the check box for that source is disabled.

Invalid logical table sources don't appear in the list when **Hide unusable logical table sources in Replace wizard** has been selected in the General tab of the Options dialog. The **Info** button is displayed when a logical table source maps to a column that doesn't appear in the list. Click **Info** to see details for the reason the physical objects weren't replaced in the logical table source or sources.

The Select Sources screen only appears if there are multiple logical table sources that map to the physical table you selected.

1. In the Oracle BI Administration Tool, select **Tools**, then select **Utilities**.
2. Select **Replace Column or Table in Logical Table Sources** and click **Execute**.
3. In the Select Object screen, select **Replace whole table**.
4. Select the physical table to replace.
5. Select the physical table to use as a replacement for the original table, and then, click **Next**.
6. Select the logical table sources to change the physical table mapping.
7. Optional: Select **Show Qualified Names** to see the full context for each source.
8. Click **Next** after you've selected logical table sources.
9. In Replace Wizard - Select Columns, specify the individual column mappings between the selected physical tables.

   If column names in the two selected tables match, default column mappings appear in the bottom pane.
10. When you've finished mapping columns between the selected physical tables, click **Next**.
11. Optional: In online mode, click **Next** to check out the necessary objects.
12. Click **Finish**.

Use the Event Tables Utility

The Event Tables utility lets you identify a table as an event polling table.

An event polling table is a way to notify the Oracle BI Server that one or more physical tables have been updated.

Each row that's added to an event table describes a single update event. The cache system reads rows from, or polls, the event table, extracts the physical table information from the rows, and purges cache entries that reference those physical tables.

See Cache Event Processing with an Event Polling Table in *Administering Oracle Analytics Server*.

1. In the Administration Tool, select **Tools**, then select **Utilities**.
2. Select **Event Tables** and click **Execute**.

Use the Externalize Strings Utility

You can use the Externalize Strings utility to localize the names of Presentation layer subject areas, tables, hierarchies, columns and their descriptions.

You can save these text strings to an external file with ANSI, Unicode, and UTF-8 encoding options. You can also choose to save strings to a set of XML files with Unicode encryption.

Before you can use the Externalize Strings utility, you must externalize strings in the Presentation layer, consider the following:

- You can right-click any Presentation layer object, such as a subject area, presentation table, or presentation column, and choose **Externalize Display Names** , select **Generate Custom Names** or **Externalize Descriptions** and select **Generate Custom Descriptions** to externalize strings. When you select **Generate Custom Names** and then run the Externalize Strings utility, the translation key also appears in the Externalize Strings dialog.
- Choosing one of these right-click externalization options automatically selects the **Custom display name** or **Custom description** options in the Properties dialog for the selected object and all of its child objects.

  For example, if you right-click a subject area and choose one of the externalization options, the externalization flag is set on all presentation tables, columns, hierarchies, and levels within that subject area.
- Running the Externalize Strings utility only externalizes those strings that have been selected for externalization in the Presentation layer.

See Localize Metadata Names in the Repository in *Administering Oracle Analytics Server*

- To run the `externalizestrings` utility, do one of the following:
  - From the Oracle BI Administration Tool, select **Tools**, select **Utilities**, select **Externalize Strings** and then click **Execute**.
  - Use the `externalizestrings` command-line utility located in *`BI_DOMAIN`*`/bitools/bin`, and see the required syntax displayed within the `externalizestrings` utility.

Use the Rename Wizard

You can use the Rename Wizard to rename tables and columns in the Presentation layer and Business Model and Mapping layer.

The Rename Wizard provides a way to transform physical names to user-friendly names. Renaming objects in the Business Model and Mapping layer rather than the Presentation layer is a best practice for maintainability. Using friendly names for logical objects rather than presentation objects ensures reuse in multiple subject areas and ensures that the names persist even when you need to delete and re-create subject areas to incorporate changes to your business model.

- You must enable the **Edit presentation names** Administration Tool option before you can select objects from the Presentation layer.
- You can only select individual presentation columns with the **Use Logical Column Name** property not selected is set to *false*.
- If you select Presentation Column, then only presentation columns without the **Use Logical Column Name** property are renamed.

The renaming rules are applied in the order in which they appear in the list. Select a rule that you've added and click **Up** or **Down** to change the order in which to apply the rules.

For example, to rename the logical columns GlobalGROUP, GlobalSales, and GlobalCustomerName to Group, Sales, and Customer Name. You can apply the following rules in the given order:

```
Insert space before each first uppercase letter, unless on the first position
or there is a space already
All text lowercase
First letter of each word capital
Change each occurrence of "Global " to "" (not case sensitive)
```

1. In the Administration Tool, select **Tools**, then select **Utilities**. Then, select **Rename Wizard** and click **Execute**.
2. In Select Objects, from the **Presentation** or **Business Model and Mapping** layer that contains the objects, select an object and click **Add**.
3. Click **Add Hierarchy** to add all objects associated with the selected object, and then click **Next**
4. In Select Types, choose the object types you want to rename such as Subject Area, Logical Table, or Logical Column, and then click **Next**.
5. In Select Rules screen, choose the renaming rules and click **Add**.
6. Select **Change specified text** to rename particular words or phrases, and click **Next**.
7. In online mode, click **Next** to check out the necessary objects.
8. Click **Finish** to rename the objects.

Use the Update Physical Layer Wizard

You can use the Update Physical Layer Wizard to update database objects in the Physical layer of a repository, based on their current definitions in the back-end database.

The Update Physical Layer wizard is only available for repositories open in online mode.

When the wizard processes the update, the Presentation Services connects to each back-end database. The objects in the Physical layer are compared with those in the back-end database. Explanatory text alerts you to differences between objects as defined in the database in the Physical layer and as defined the back-end database, such as data type-length mismatches and objects that are no longer found in the back-end database. For example, if an object exists in the database in the Physical layer of the repository but not in the back-end database, the following text is displayed:

```
Object does not exist in the database and will be deleted.
```

The wizard doesn't add columns or tables to the repository that exist in the back-end database, but not in the repository. Additionally, the wizard doesn't update column key assignments. It checks that there is a column in the repository that matches the column in the database, and then, if the values don't match, the wizard updates the type and length of the column in the repository.

The connection pool settings for each database need to match the connection pool settings used when the objects were last imported into the Physical layer from the back-end database. See Create or Change Connection Pools.

1. In the Administration Tool, select **Tools**, then select **Utilities**.
2. Select **Update Physical Layer** and click **Execute**.

   The databases in the Physical layer of the repository are listed in the left pane of the wizard.
3. In Select Database, select the databases to update in the left pane, and then click **Add**.

   Click **Remove** to remove a database from the selection list.
4. Click **Next**.
5. In Select Connection Pool, select the connection pool for each database to update, and then click **Next**.

   You might need to set or confirm values for variables before continuing.
6. In Update, review the information about each update and select the updates to perform.
7. In online mode, you must check out objects before you can make changes to them.
8. Click **Next** to check out the necessary objects.
9. Click **Finish**.

   The wizard updates the objects in the Physical layer, and then closes automatically. Objects that don't exist in the database are deleted.
10. From the **File** menu, select **Save** to save the updated objects in the Physical layer.

Generate Documentation of Repository Mappings

The Repository Documentation utility documents the mapping from the presentation columns to the corresponding logical and physical columns.

The documentation includes conditional expressions associated with the columns. You can save the documentation in comma separated (CSV), XML, or tab delimited (TXT) format.

You can use the Repository Documentation utility to extract metadata to a flat file and load the file into Excel and RDBMS. You can query the resulting file to answer questions such as "If I delete physical column X, what logical columns are affected?" or "How many places in the business model refer to the physical table W_SRVREQ_F?" You can establish dependency relationships among elements in the repository.

Excel only allows data sets of 1,000,000 rows. You might exceed the row limitation in a large repository. Run the Repository Documentation utility on a subset of the repository by extracting relevant business models into a new project. See Set Up and Use the Multiuser Development Environment.

The Repository Documentation utility creates a comma-separated values file or a tab-separated values file that shows the connections between the Presentation and Physical layers in the current repository. You can import the file into a repository as a Physical layer. The file doesn't include information about repository variables and marketing objects.

1. In the Administration Tool, select **Tools**, then select **Utilities**.
2. Select **Repository Documentation** and click **Execute**.
3. In the Save As dialog, choose the directory where you want to save the file.
4. Type a name for the file.
5. Choose a type of file and an Encoding value, and then click **Save**. Current encoding options are ANSI, Unicode, and UTF-8.

Generate a Metadata Dictionary

You can generate a metadata dictionary to help users obtain more information about metrics or attributes for repository objects.

Users might need to resolve issues caused by confusing metadata object names, or to obtain more details when an attribute is derived in a complicated way. By generating a metadata dictionary, users can gain an understanding of the repository and its objects.

A metadata dictionary is a static set of XML documents. Each XML document describes a metadata object, such as a column, including its properties and relationships with other metadata objects. Your users can view the XML documents in the Oracle BI Presentation Services user interface, or in a browser.

Use the Administration Tool to generate a metadata dictionary for your repository. Because the dictionary isn't updated as repository changes are made, you must generate the dictionary periodically to update the content.

You must host the metadata dictionary files on a Web server such as Oracle HTTP Server or Apache HTTP Server. When you generate the dictionary, you can set the output to the final location on the Web server, or to a temporary location. If you generate the dictionary in a temporary location, you must then copy the files to the location on the Web server.

Some large repositories can contain tens of thousands of objects. Generating a dictionary for a large repository can take a significant period of time.

You can't store the dictionary in a directory with multi-byte characters. If you receive a system error about creating the necessary directories for the dictionary, then you must choose another directory.

When choosing a destination for your dictionary:

- Select a local or network location. When the dictionary is generated, a subdirectory with the same name as the repository is created in that location. The dictionary directories and files are created in that subdirectory.

  For example, if you select `J:\BI_DataDictionary` and your repository name is *demo1.rpd*, the dictionary files, including the style sheets are located in `J:\BI_DataDictionary\demo1`.

- If you want to use an IIS virtual directory, you can create or select a virtual directory in IIS before you generate the dictionary. When you generate the dictionary, choose the physical directory associated with the IIS virtual directory.

The location where users can view the metadata dictionary files is dependent on the host name and port number of your Web server, along with the directory location where you store the files.

See Provide Access to Metadata Dictionary Information.

1. In the Administration Tool, open your repository in offline mode. You can't generate a metadata dictionary in online mode.
2. Select **Tools**, then select **Utilities**.
3. Select **Generate Metadata Dictionary** and click **Execute**.
4. In the Choose Directory dialog, click **Browse** to locate and select the location where you want to store the dictionary.
5. Click **OK**.
6. Copy the files over to your Web server and ensure they're available.
7. Edit the `instanceconfig.xml` configuration file to enable the metadata dictionary in the Oracle BI Presentation Services user interface, as well as grant the appropriate privilege to your users, groups, or application roles.

After you generate a metadata dictionary, style sheets and index files are created for that dictionary. The related style sheets (XSL files) are created and stored in a directory named `xsl` within the repository directory.

A name index and tree index are created and stored in the `[drive]:\[path]\[repository name]` root directory. The index files are associated with each other so that you can quickly switch views.

To learn about viewing metadata dictionary information from the Oracle BI Presentation Services user interface, see *User's Guide for Oracle Business Intelligence Enterprise Edition*.

Provide Access to Metadata Dictionary Information

When creating analyses, content designers might need more information about subject areas, folders, columns, or levels such as relationships to other metadata objects to guide them.

You can provide content designers with this information by allowing them access to the metadata dictionary for the repository.

The metadata dictionary describes the metrics that are contained within the repository and the attributes of repository objects. The metadata dictionary output is a static set of XML documents.

See Manage Presentation Services Privileges Using Application Roles in *Managing Security for Oracle Analytics Server*.

Ensure that the metadata dictionary has been generated and the files have been stored in an appropriate location. See Generate a Metadata Dictionary.

1. Set the `DictionaryURLPrefix` element within the `ServerInstance` element in the `instanceconfig.xml` file to one of the following values. The value that you specify depends on the web servers in use.
   - The prefix for the name of the directory in which you've stored the XML files. The directory must have been specified as a shared directory for the web server, and the web server must be the same one that's used by Oracle Analytics Server.

     For example, suppose that you stored the XML files for the metadata dictionary in a directory called *demo1* under the metadictionary directory. Suppose that the metadictionary directory is specified as a shared directory for the web server, which is also used by Oracle Analytics Server. Then you specify the following value for the `DictionaryURLPrefix` element:

     ```
     <DictionaryURLPrefix>demo1</DictionaryURLPrefix>
     ```

     See the documentation for the web server for information about sharing directories.
   - The URL that points to the directory in which you've stored the XML files. Use a value such as this when the files for the metadata dictionary are stored in the directory structure for a web server that isn't being used by Oracle Analytics Server. For example:

     ```
     <DictionaryURLPrefix>http://10.10.10.10/metadictionary/demo1</
     DictionaryURLPrefix>
     ```

   The following shows an example setting in the `instanceconfig.xml` file:

   ```
   <WebConfig>
     <ServerInstance>
       <SubjectAreaMetadata>
         <DictionaryURLPrefix>demo1</DictionaryURLPrefix>
       </SubjectAreaMetadata>
     </ServerInstance>
   </WebConfig>
   ```
2. Grant the Access to Metadata Dictionary privilege to the appropriate content designers.

See View Metadata Information from the Subject Areas Pane in *Visualizing Data in Oracle Analytics Server*.

Remove Unused Physical Objects

Use the procedure to remove objects that you no longer need in your repository.

Large repositories use more memory on the server and are harder to maintain. Additionally, development activities take longer on a large repository. You can remove databases, initialization blocks, physical catalogs, and variables.

1. In the Administration Tool, select **Tools**, then select **Utilities**.
2. Select **Remove Unused Physical Objects** and click **Execute**.
3. In the Remove Unused Physical Objects dialog, from the **Type** list, select the type of object.
4. In the list of objects, verify that only the objects that you want to remove are selected.

Below the list of objects, the number of selected objects and the total number of objects appears.

5. To remove the selected objects, click **Yes**.

Persist Aggregates

You can use the Aggregate Persistence Wizard to create the SQL file used to create aggregate tables and map them into the metadata.

See Use the Aggregate Persistence Wizard to Generate the Aggregate Specification.

Use the Convert Presentation Folders Utility

You can designate child presentation tables using the Child Presentation Tables tab in the Presentation Table dialog to give the appearance of nested folders in Answers and BI Composer.

You could add one level of nesting in Answers by adding a hyphen at the beginning of a presentation table name, or by adding an arrow (->) at the beginning of a presentation table description. If you used these methods, Oracle recommends that you run the Convert Presentation Folders utility to convert your metadata to the new structure.

1. Open your repository in the Administration Tool in offline mode. Don't run the Convert Presentation Folders utility in online mode.
2. Select **Tools**, then select **Utilities**.
3. Select **Convert Presentation Folders** and click **Execute**.

   The hyphens and arrows disappear from presentation table names and descriptions, and the affected tables are listed as child tables for the appropriate parent object.

Generate a List of Logical Column Types

You can use the Generate Logical Column Type Document utility to generate a complete list of logical columns and their corresponding types.

The output is stored in XML format. You can select ANSI, Unicode, or UTF-8 encoding options.

This utility is often used with the Compare Logical Column Types utility. See Compare Logical Column Types.

1. In the Administration Tool, select **Tools**, then select **Utilities**.
2. Select **Generate Logical Column Type Document** and click **Execute**.
3. In the Save As dialog, choose the directory where you want to save the file.
4. Type a name for the file. The file must have an XML extension.
5. Choose an **Encoding** value, and then click **Save**.

Use the biservergentypexml Utility to Generate a List of Logical Column Types

You can generate a list of logical columns and their corresponding types using the `biservergentypexml` utility.

The `biservergentypexml` utility is similar to the Generate Logical Column Type Document utility in the Administration Tool. This utility is available on both Windows and Linux systems. You can only use `biservergentypexml` with binary repositories in Oracle BI repository file format.

Provide the full path names to your repository file and XML output file if they're located in a different directory.

The location of the `biservergentypexml` utility is:

BI_DOMAIN/bitools/bin

Syntax

The `biservergentypexml` utility takes the following parameters:

```
biservergentypexml -R repository_name [-P repository_password]
-O output_XML_file_name {-8 | -U | -A}
```

Where:

repository_name is the name and path of the repository from which you want to generate a list of logical column types.

repository_password is the password for the repository from which you want to generate a list of logical column types.

The *repository_password* argument is optional. If you don't provide the password argument, you're prompted to enter the password when you run the command. To minimize the risk of security breaches, Oracle recommends that you don't provide password arguments either on the command line or in scripts. For scripting purposes, you can pass the password through standard input.

output_XML_file_name is the name and path of the XML file where you want to store the output generated by the utility.

- `8` specifies UTF-8 encoding for the output file.
- `U` specifies Unicode encoding for the output file.
- `A` specifies ANSI encoding for the output file.

Example

The following example creates a UTF-8 encoded output XML file called log_col_types.xml that includes logical column type information from *my_repos.rpd*.

```
biservergentypexml -R my_repos.rpd -O log_col_types.xml -8
Give password: my_rpd_password
```

Compare Logical Column Types

In Oracle Analytics Server logical column types can change over the course of MUD development, resulting in unexpected logical column types.

When this occurs, you can generate a list of logical columns and their types using the Generate Logical Column Type Document utility in the Administration Tool or `biservergentypexml`, and then use the Compare Logical Column Types utility for subsequent MUD versions to ensure that the logical column types match as expected. For example, you could generate a logical column type list for repository version 20, and then use the Compare Logical Column Types utility to compare the list against repository version 30.

To use this utility, you must have already generated a list of logical column types with which you want to compare the current repository. The Compare Logical Column Types utility only compares logical columns that exist in both the repository and the XML file; newly created logical columns and deleted columns are ignored.

1. In the Administration Tool, select **Tools**, then select **Utilities**.
2. Select **Compare Logical Column Types** and click **Execute**.
3. In the Select XML File dialog, select the generated list of logical column types with which you want to compare the column types in the current repository.
4. Click **Open**.

Fix Upgrade IDs

In cases where you're comparing or merging repositories, the upgrade IDs sometimes don't function correctly.

You can use the Fix Upgrade IDs utility to correct issues with upgrade IDs.

You can use upgrade IDs to compare or merge repositories. They identify when two object in two repositories are supposed to be the same object. However, in some cases, the upgrade IDs don't work correctly. For example, when two or more objects have the same upgrade ID, when objects are missing upgrade IDs, and when hidden internal object have upgrade IDs set.

1. In the Administration Tool, select **Tools**, then select **Utilities**.
2. Select **Fix Upgrade IDs** and click **Execute**.

   If you ran the utility on a repository open in read-only mode, then the utility reports how many invalid upgrade IDs are in the repository. To fix the upgrade IDs, you must open the repository in non-ready-only mode and rerun the utility.

   If you ran the utility on a repository that isn't t read-only, then the utility fixes the invalid upgrade IDs and displays a message stating how many upgrade IDs it fixed.

Set Permissions In Bulk

You can use the Set Permissions in Bulk utility when you want to assign the same object, data filters, and query limits permissions to several users or roles at the same time.

1. In the Oracle BI Administration Tool, select **Tools**, then select **Utilities**.
2. Select **Set Permissions in Bulk** and click **Execute**.
3. In the Set Permissions in Bulk dialog, specify the items to set permissions for Object Permissions, Data Filters, and Query Limits.
4. Select the users and roles for which you want to bulk assign permissions and click **Add** to move them to the selected table.
5. Click **OK**.
6. In the dialog, select the tab corresponding to the item to bulk assign permissions, for example, Data Filters, and specify the permissions to assign to the users and roles you chose.
7. After you've specified the permissions, click **OK**.

Use the Calculation Wizard

You can use the Calculation Wizard to create new calculation columns that compare two existing columns, and also to create metrics in bulk.

It has a built-in mechanism to handle divide-by-zero and null cases, as well as other difficult situations. The Calculation Wizard provides an automated way to calculate the sales by quarter, the percentage of revenue, minimum and maximum values, and so on.

In the Calculation Wizard, the New Calculation page select from the following options:

- Change (CurrentX - ComparisonX)

  Subtract the value of the comparison column from the source column.

  For example, for the Change calculation, you can choose whether to return `NULL` or some other value when the comparison column is `NULL`.
- Percent Change (100.0 * (CurrentX - ComparisonX) / ComparisonX)

  Subtract the value of the comparison column from the source column and express as a percentage.
- Index (1.0 * CurrentX / ComparisonX)

  Divide the source column by the comparison column.
- Percent (100.0 * (CurrentX / ComparisonX))

  Divide the source column by the comparison column and express as a percentage.

1. In the Business Model and Mapping layer, right-click any logical fact or dimension column of data type numeric measure column, and then select **Calculation Wizard**.
2. Click **Next**
3. In the Select Columns, choose the columns that you want to compare with the source column.
4. Click **Remove** to take items out of the selected list.
5. Click **Next**.
6. In the New Calculations screen, choose the calculations to perform for specific columns.
7. Click **Next** when you've finished creating calculations.
8. In online mode, check out the objects to change.

9. Click **Next** to check out the necessary objects.

   The Finish screen displays a summary of the calculations that are created.

10. Click **Back** or select a particular step from the navigation panel to make changes.

11. Click **Finish**. The new calculation columns are created.

Associate S_NQ_ACCT Record with the Query Log

Your administrator can associate the records in the usage tracking table with the Oracle BI Server query log to help you troubleshoot Logical SQL query issues or to find queries related to a specific subject matter area.

The Oracle BI Server calculates a hash code from the text of the Logical SQL query and the text of the physical SQL queries. The physical SQL hash code, of any SQL queries run from the Oracle BI Server, is recorded in the **ACTION** column in `V$SQL`. Your administrator can join the **ACTION** column with the **PHYSICAL_HASH_ID** column in the `S_NQ_DB_ACCT` table.

When usage tracking is enabled, every Logical SQL request submitted to the Oracle BI Server is recorded in the `S_NQ_ACCT` table. See Set Up Direct Insertion to Collect Information for Usage Tracking in the *Administering Oracle Analytics Server*.

Set the `ENABLE_HASH_CODE_IN_SQL_COMMENTS` parameter to `YES` in the `NQSConfig.ini` file to create an unique `Hash_ID` with each Logical SQL comment associated with a Logical SQL query.

You should use `DISABLE_HASH_CODE` after resolving the query issue.

You can associate the physical SQL hash code that's recorded in the `query.log` with the same hash code value that's stored in the `ACTION` column of the `V$SQL` performance view in the Oracle Database.

Your administrator can associate the physical SQL queries in the `V$SQL` view by doing the following:

- Getting the physical query hash code from the ACTION column of the V$SQL view.
- Querying the physical query usage tracking table, `S_NQ_DB_ACCT`, filtering on the `PHYSICAL_HASH_ID` column using the hash code value obtained from the `ACTION` column of the `V$SQL` view.
- Querying the logical query usage tracking table, `S_NQ_ACCT`, joining the `S_NQ_ACCT.ID` column with the `LOGICAL_QUERY_ID` column from the `S_NQ_DB_ACCT` table.

You can obtain various properties of the corresponding BI logical request from the columns in the `S_NQ_ACCT` table including the **SUBJECT_AREA_NAME** column.

The relevant columns for associating the logical request record from `S_NQ_ACCT` table with the query log and the catalog are:

- `QUERY_TEXT` represents the text of the logical SQL query, truncated to 4000 bytes. For the complete text of the SQL query, use the `QUERY_BLOB` columns or in the query log file.

  For example:

```
select product.productid, product.qtysold, supplier.companyname, supplier.qtysold,
(1.0 * product.qtysold) / supplier.qtysold from SnowflakeSales
```

- `HASH_ID` represents the hash code of the Logical SQL query. You can use this identifier to search the query log for all occurrences of the same query.

  For example:

  a3a04491 as the `HASH_ID` value

- ID represents a unique identifier for the logical request. You can join the ID column with the `LOGICAL_QUERY_ID` column in the `S_NQ_DB_ACCT` table to get the physical SQL query details.

19

Use Variables in the Oracle BI Repository

This chapter describes how to use varin the Oracle BI repository to streamline administrative tasks and dynamically modify metadata content to adjust to a changing data environment. You can use the Variable Manager in the Administration Tool to define two classes of variables: repository variables and session variables. Values in repository and session variables aren't secure, because object permissions don't apply to variables. Anybody who knows or can guess the name of the variable can use it in an expression in Answers or in a Logical SQL query. Because of this, don't put sensitive data like passwords in session or repository variables.

Initialization blocks are used to initialize dynamic repository variables, system session variables, and nonsystem session variables.

This chapter contains the following topics:

- Work with Repository Variables
- Work with Session Variables
- Work with Multi-Source Session Variables
- List Repository Variables Command
- Update Repository Variables Command

Work with Repository Variables

Learn about repository variables and how to create repository variables in these topics.

This section provides information about working with repository variables, and contains the following topics:

- About Repository Variables
- Create Repository Variables

About Repository Variables

A repository variable has a single value at any point in time.

Use repository variables instead of literals or constants in the Administration Tool Expression Builder. The Oracle BI Server substitutes the value of the repository variable with the variable in the metadata.

This section contains the following topics:

- About Static Repository Variables
- About Dynamic Repository Variables

About Static Repository Variables

The value of a static repository variable is initialized in the Variable dialog.

The value of the static repository variable persists and doesn't change until an administrator decides to change it. For example, suppose you want to create an expression to group times of day into different day segments. If Prime Time were one of those segments and corresponded to the hours between 5:00 PM and 10:00 PM, you could create a `CASE` statement like the following:

```
CASE WHEN "Hour" >= 17 AND "Hour" < 23 THEN 'Prime Time' WHEN... ELSE...END
```

Hour is a logical column, mapped to a timestamp physical column using the date-and-time `Hour(<<timeExpr>>)` function.

Rather than entering the numbers 17 and 23 into this expression as constants, you could use a static repository variable named `prime_begin`, initialize the variable to a value of `17`, and then create another variable named `prime_end` and initialize it to a value of `23`.

Static repository variables must have default initializers that are either numeric or character values. You can use the Expression Builder to insert a constant as the default initializer, such as Date, Time, and TimeStamp. You can't use any other value or expression as the default initializer for a static repository variable.

In previous releases, the Administration Tool didn't limit the values of default initializers for static repository variables. Because of this, if your repository has been upgraded from a previous release, you may see warnings in the Consistency Checker similar to the following:

```
The variable, 'Current Month' does not have a constant default initializer.
```

If you see warnings similar to this, update the relevant static repository variables so that the default initializers have constant values.

About Dynamic Repository Variables

You initialize dynamic repository variables in the same way as static variables, but the values are refreshed by data returned from queries.

When defining a dynamic repository variable, you create an initialization block or use a preexisting one that contains a SQL query. You also set up a schedule that the Oracle BI Server uses to run the query and periodically refresh the value of the variable. When the value of a dynamic repository variable changes, all cache entries associated with a business model that reference the value of that variable are purged automatically.

Each query can refresh several variables, one variable for each column in the query. In Oracle BI Server, create a schedule to run the queries.

Dynamic repository variables are useful for defining the content of logical table sources. If, for example, you've two sources for information about orders, one source contains recent orders and the other source contains historical data, you need update the repository to use the recent orders and move the historical order data to a different view.

You describe the content of the sources in the Logical Table Source. Without using dynamic repository variables, you would describe the content of the source containing recent data with an expression such as:

```
Orders.OrderDates."Order Date" >= TIMESTAMP '2001-06-02 00:00:00'
```

This content statement becomes invalid as new data is added to the recent source and older data is moved to the historical source. To accurately reflect the new content of the recent source, you would have to modify the fragmentation content description manually. You can set up dynamic repository values to do automatically modify the content.

Another suggested use for dynamic repository values is in `WHERE` clause filters of logical table sources.

A common use of these variables is to set filters for use in Oracle BI Server. For example, to filter a column on the value of the dynamic repository variable `CurrentMonth`, set the filter to the variable `CurrentMonth`.

Create Repository Variables

Use these steps to create repository variables.

Use unique names for all variables. The names of system session variables are reserved, so you can't use system session variable names for other types of variables.

When you create a dynamic repository variable to override selection steps in a hierarchy column, choose an initialization block with its initialization string written in JSON syntax. See Initialization Strings Used in Variables to Override Selection Steps. To create a new initialization block, click **New**; see Create Initialization Blocks.

Static repository variables must have a default value defined in the **Default initializer** field. Static repository variables are constants that don't change values. If you initialize a variable using a character string, enclose the string in single quotes (').

1. In the Administration Tool, select **Manage**, then select **Variables**.
2. In the Variable Manager, From **Action**, select **New** , then select **Repository** , and select **Variable**.
3. In the Repository Variable dialog, in **Name**, type a name for the variable.
4. From **Type**, select one of the following:
   - **Static**
   - **Dynamic**
5. If you selected **Dynamic**, from the **Initialization Block** list select an existing initialization block to refresh the value on a continuing basis.
6. In **Default initializer**, type the value for the repository variable, or click the **Expression Builder** button to define an expression to use as the variable value.
7. Click **OK**.

Use Repository Variables in Expression Builder

After creating variables, you can use them in the Expression Builder.

Use variables as arguments of the function `VALUEOF()`. This happens automatically when you double-click the variables to paste them into the expression.

You can't use variables to represent columns or other repository objects.

For example, the following `CASE` statement is identical to the one explained in the preceding example, except that variables have been substituted for the constants:

```
CASE WHEN "Hour" >= VALUEOF("prime_begin")AND "Hour" < VALUEOF("prime_end") THEN
'Prime Time' WHEN ... ELSE...END
```

- In Expression Builder, click the Repository Variables folder in the left pane to display all repository variables (both static and dynamic) in the middle pane by name.
- To use a repository variable in an expression, select it and double-click. Expression Builder pastes it into the expression at the active cursor insertion point.

Work with Session Variables

Learn about session variables and how to create them.

This section provides information about working with session variables, and contains the following topics:

- About Session Variables
- Create Session Variables

About Session Variables

Session variables obtain their values from initialization blocks.

Unlike dynamic repository variables, however, the initialization of session variables isn't scheduled. When a user begins a session, the Oracle BI Server creates new instances of session variables and initializes them.

Unlike a repository variable, there are as many instances of a session variable as there are active sessions on the Oracle BI Server. Each instance of a session variable could be initialized to a different value.

Session variables are primarily used when authenticating users against external sources such as database tables or LDAP servers. If a user is authenticated successfully, session variables can be used to set filters and permissions for that session. When using session variables to set up security, see Manage Session Variables in *Managing Security for Oracle Analytics Server*.

Note: Oracle Analytics doesn’t support the variables :user and :password in data source connection credentials.

This section contains the following topics:

- About System Session Variables
- About Nonsystem Session Variable

About System Session Variables

Oracle BI Server and Oracle BI Presentation Services use system session variables for specific purposes.

System session variables have reserved names that can't be used for other kinds of variables such as static or dynamic repository variables and non-system session variables.

When you use these variables for Oracle BI Presentation Services, preface their names with `NQ_SESSION`. For example, to filter a column on the value of the variable `LOGLEVEL`, set the filter to the variable `NQ_SESSION.LOGLEVEL`.

The table describes the available system session variables.

Variable	Description
USER	Holds the value the user enters. The `USER` value always matches the `PROXY` variable that's the *act as* value. When the user logs in to act as some other user, the value of the `USER` session variable matches the `USERID` the user is acting as.
USERGUID	Contains the global unique identifier (GUID) of the user, populated from the LDAP or other profile for the user.
GROUP	Contains the groups that the user belongs to. Legacy groups are mapped to application roles automatically. When a user belongs to multiple groups, include the group names in the same column, separated by semicolons, for example, *GroupA;GroupB;GroupC*. If you must use a semicolon as part of a group name, precede the semicolon with a backslash character (\).
ROLES	Contains the application roles that the user belongs to. When a user belongs to multiple roles, include the role names in the same column, separated by semicolons, for example, *RoleA;RoleB;RoleC*. If a semicolon must be included as part of a role name, precede the semicolon with a backslash character (\).
ROLEGUIDS	Contains the global unique identifiers (GUIDs) for the application roles to which the user belongs. GUIDs for application roles are the same as the application role names.
PERMISSIONS	Contains the permissions held by the user such as *oracle.bi.server.manageRepositories*.
PROXY	Holds the name of the proxy user that is authorized to act for another user. See *Managing Security for Oracle Analytics Server* for more information about the PROXY system session variable.
DISPLAYNAME	Used for Oracle BI Server. It contains the name that's displayed to the user in the greeting in the Oracle BI Presentation Services user interface. It's also saved as the author field for catalog objects. `DISPLAYNAME` variable is populated from the LDAP or other user profile.
LOGLEVEL	The `LOGLEVEL` value is a number between 0 and 5. `LOGLEVEL` specifies the logging level that the Oracle BI Server uses for user queries. `LOGLEVEL` overrides a variable defined in the Users object in the Administration Tool. If the administrator user, defined upon install, has a Logging level defined as 4 and the session variable `LOGLEVEL` defined in the repository has a value of 0 (zero), the value of 0 applies.
DESCRIPTION	Contains a description of the user as populated from the LDAP or other user profile.
USERLOCALE	Contains the locale of the user as populated from the LDAP or other user profile.
DISABLE_CACHE_HIT	Used to enable or disable Oracle BI Server result cache hits. This variable has a possible value of 0 or 1.
DISABLE_CACHE_SEED	Used to enable or disable Oracle BI Server result cache seeding. This variable has a possible value of 0 or 1.

Variable	Description
DISABLE_SUBREQUEST_CACHE	Used to enable or disable Oracle BI Server subrequest cache hits and seeding. This variable has a possible value of 0 or 1.
SELECT_PHYSICAL	Identifies the query as a `SELECT_PHYSICAL` query.
DISABLE_PLAN_CACHE_HIT	Used to enable or disable Oracle BI Server plan cache hits. This variable has a possible value of 0 or 1.
DISABLE_PLAN_CACHE_SEED	Used to enable or disable Oracle BI Server plan cache seeding. This variable has a possible value of 0 or 1.
TIMEZONE	Contains the time zone of the user as populated from the LDAP or other user profile.
WEBLANGUAGE	Used for Oracle BI Presentation Services. Holds the user interface display language. Users can select a language on the sign-in page for Oracle Analytics Server, or they can change the language setting on the Preferences tab of the My Account dialog after signing in.
AUTHINITBLOCKONLY	Determines if the initialization blocks required for authentication are run. This variable has a value of Yes. The value is case-insensitive.
PORTALPATH	Used for Oracle BI Server. It identifies the default dashboard the user sees when logging in, the user can override this preference after signing onto Oracle Analytics Server.
REQUESTKEY	Used for Oracle BI Presentation Services. Any users with the same nonblank request key share the same cache entries. This tells Oracle BI Presentation Services that these users have identical content filters and security. Sharing cache entries is a way to minimize unnecessary communication with the Oracle BI Presentation Services.
SKIN	Determines certain elements of the look and feel of the Oracle BI Presentation Services user interface. The user can alter some elements of the user interface by picking a style when logged on. The `SKIN` variable points to a folder that contains the non-alterable elements, for example, figures such as GIF files. Such directories begin with sk_. For example, if a folder were called *sk_companyx*, the `SKIN` variable would be set to *companyx*.

About Nonsystem Session Variables

A common use for nonsystem session variables is setting user filters.

For example, you could define a nonsystem variable called SalesRegion that's initialized to the name of the user's sales region.

You can set a security filter for all members of a group that allow the group to view only the data pertinent to their region.

When you use these variables for Oracle BI Server, preface their names with `NQ_SESSION`. For example, to filter a column on the value of the variable `SalesRegion`, set the filter to the variable `NQ_SESSION.SalesRegion`.

Create Session Variables

Use these steps to create session variables.

Create unique names for all variables. The names of system session variables are reserved. You can't use system session variable names for other types of variables.

The **Enable any user to set the value** option lets non-administrators set the variable for sampling.

The `NQSSetSessionValues()` stored procedure isn't supported for use through the Issue SQL page in Oracle BI Presentation Services Administration. You must select the **Enable any user to set the value** option to set a value for the variable.

When **Security Sensitive** is selected, the Oracle BI Server looks at the parent database object of each column or table that's referenced in the logical request projection list. If the database object has the **Virtual Private Database** option selected, the Oracle BI Server matches a list of security-sensitive variables to each prospective cache hit. Cache hits would only occur on cache entries that included and matched all security-sensitive variables.

If you're creating a session variable to override a hierarchy column's selection steps, then you must choose an initialization block with its initialization string written in JSON syntax. See Initialization Strings Used in Variables to Override Selection Steps and Create Initialization Blocks.

See Set Up an Expression.

1. In the Administration Tool, select **Manage**, then select **Variables**.
2. In the Variable Manager dialog, from the **Action** menu, select **New**, select **Session** , and then select **Variable**.
3. In the Session Variable dialog, in **Name**, type a variable name.
4. Optional: Select **Enable any user to set the value** to set the session variable after the initialization block has populated the value, at user login, by calling the ODBC stored procedure `NQSSetSessionValue()`.
5. Optional: Select **Security Sensitive** to identify the variable as sensitive to security when using a row-level database security strategy, such as a Virtual Private Database (VPD).
6. From the **Initialization Block** list, select an initialization block to use to refresh the value on a continuing basis or click **New** to create a new initialization block.
7. In **Default Initializer**, type the value, or click the **Expression Builder** button to use Expression Builder.
8. Click **OK**.

Work with Initialization Blocks

Initialization blocks are used to initialize dynamic repository variables, system session variables, and nonsystem session variables.

For example, the `NQ_SYSTEM` initialization block is used to refresh system session variables.

This section contains the following topics:

- About Using Initialization Blocks with Variables
- Create Initialization Blocks
- Variable Order in Initialization Blocks
- Associate Variables with Initialization Blocks
- Establish Execution Precedence
- When Processing of Session Variable Initialization Blocks Can't Be Deferred

- Enable and Disable Initialization Blocks

About Using Initialization Blocks with Variables

An initialization block contains the SQL statement that's run to initialize or refresh the variables associated with that block.

The SQL statement must reference physical tables that can be accessed using the connection pool specified in the **Connection Pool** field in the Initialization Block dialog.

If you want the query for an initialization block to have database-specific SQL, you can select a database type for that query. If a SQL initialization string for that database type was defined when the initialization block is instantiated, the string is used, otherwise, a default initialization SQL string is used.

By default, when you open the Initialization Block dialog for editing in online mode, the initialization block object is automatically checked out. While the initialization block is checked out, the Oracle BI Server can continue to refresh the value of dynamic variables updated by this initialization block, depending on the refresh interval rate. When you check in the initialization block, the values of the dynamic variables are reset to the values shown in the Default initializer. If you don't want to reset the value, use the **Undo Check Out** option.

This section contains the following topics:

- Initialize Dynamic Repository Variables
- Initialize Session Variables
- About Row-Wise Initialization

Initialize Dynamic Repository Variables

The values of dynamic repository variables are set by queries defined in the Default initialization string field of the Initialization Block dialog.

You can set up a schedule that the Oracle BI Server follows to run the query and periodically refresh the value of the variable. If you stop and restart the Oracle BI Server, the server automatically runs the SQL statements in repository variable initialization blocks, reinitializing the repository variables.

The Oracle BI Server logs all SQL queries issued to retrieve repository variable information in `obis1_query.log` located in the `DOMAIN_Home/servers/obis1/logs` when the logging level for the administrator account, set upon installation, is set to 2 or higher. You should set the logging level to 2 for the administrator to provide the most useful level of information.

Initialize Session Variables

As with dynamic repository variables, session variables obtain their values from initialization blocks. Unlike dynamic repository variables, session variables aren't updated at scheduled time intervals.

Instead, the Oracle BI Server creates new instances of those variables whenever a user begins a new session. The values remain unchanged for the duration of the session.

Processing of session variable initialization blocks during session logon can be deferred until their associated session variables are actually accessed within the session, see Create Initialization Blocks.

The Oracle BI Server logs all SQL queries issued to retrieve session variable information if the logging level is set to 2 or higher in the Identity Manager User object, or the `LOGLEVEL` system session variable is set to 2 or higher in the Variable Manager.

Qeries and errors are stored in the `obis1_query.log` located in the *`DOMAIN_Home`*`/servers/obis1/log` directory.

See Manage the Query Log in *Administering Oracle Analytics Server*.

About Row-Wise Initialization

You can use the row-wise initialization option to create session variables dynamically and set their values when a session begins.

The names and values of the session variables reside in an external database that you access through a connection pool. The variables receive their values from the initialization string that you type in the Initialization Block dialog.

For example, suppose you want to create session variables using values contained in a table named `RW_SESSION_VARS`. The table contains three columns:

- `USERID`, containing values that represent the unique identifiers of the users
- `NAME`, containing values that represent session variable names
- `VALUE`, containing values that represent session variable values

The table shows the example.

USERID	NAME	VALUE
JOHN	LEVEL	4
JOHN	STATUS	FULL-TIME
JANE	LEVEL	8
JANE	STATUS	FULL-TIME
JANE	GRADE	AAA

To use row-wise initialization, create an initialization block and select the **Row-wise initialization** option, see Create Initialization Blocks. For this example, provide the following SQL statement for the initialization string:

```
SELECT NAME, VALUE
FROM RW_SESSION_VARS
WHERE USERID='VALUEOF(NQ_SESSION.USERID)'
```

`NQ_SESSION.USERID` has already been initialized using another initialization block.

The following session variables would be created:

- When John connects to the Oracle BI Server, his session contains two session variables from row-wise initialization: `LEVEL`, containing the value `4`, and `STATUS`, containing the value `FULL_TIME`.

- When Jane connects to the Oracle BI Server, her session contains three session variables from row-wise initialization: `LEVEL`, containing the value `8`; `STATUS`, containing the value `FULL-TIME`; and `GRADE`, containing the value `AAA`.

Initialize a Variable with a List of Values

You can use the row-wise initialization option to initialize a variable with a list of values. You can then use the SQL IN operator to test for values in a specified list.

This information and example pertain to Logical SQL. If you're using Physical SQL to initialize a variable with a list of values, then you must use the `VALUELISTOF` function. For example, to get the customers assigned to the user names in the variable *LIST_OF_USERS*, use the following statement:

For example, you type the following SQL statement for the initialization string:

```
SELECT 'LIST_OF_USERS', USERID
FROM RW_SESSION_VARS
WHERE NAME='STATUS' AND VALUE='FULL-TIME'
```

This SQL statement populates the variable `LIST_OF_USERS` with a list, separated by colons, of the values `JOHN` and `JANE`, for example, `JOHN:JANE`. You can then use this variable in a filter, as shown in the following `WHERE` clause:

```
WHERE TABLE.USER_NAME = valueof(NQ_SESSION.LIST_OF_USERS)
```

The variable `LIST_OF_USERS` contains a list of values, that is, one or more values. This logical `WHERE` clause expands into a physical `IN` clause, as shown in the following statement:

```
WHERE TABLE.USER_NAME IN ('JOHN', 'JANE')

Select 'LIST_OF_CUSTOMERS', Customer_Name from RW_CUSTOMERS where
RW.CUSTOMERS.USER_NAME in (VALUELISTOF(NQ_SESSION.LIST_OF_USERS))
```

To filter by specific values in the list, use `ValueNameof` similar to the following example. The first value is 0, not 1.

```
Select 'LIST_OF_CUSTOMERS', Customer_Name from RW_CUSTOMERS where
RW.CUSTOMERS.USER_NAME in '(ValueNameOf(0,NQ_SESSION.LIST_OF_USERS))
```

Create Initialization Blocks

Learn about initialization blocks in these topics.

To create initialization blocks, perform the steps in the following sections:

- Assign a Name and Schedule to Initialization Blocks
- Select and Test the Data Source and Connection Pool

Create Session Variable Initialization Blocks

Use these steps to create a session variable initialization block.

When using Database as the data source type, you can use a default initialization string or a database-specific SQL statement, with a valid connection pool. You can test the connection before saving the changes.

When using LDAP Server as the data source type, select an existing LDAP Server or define a new server with the hostname, port number, password, and other LDAP specific configuration information.

When using a Custom Authenticator as the data source type, select an existing custom authenticator, or select a new authenticator and supply the required configuration properties.

1. In the Oracle BI Administration Tool, click the **Manage** menu, and select **Variables**.
2. In the Variable Manager, from the **Action** menu, select **Session**, and then select **Initialization Block**.
3. In the Session Variable Initialization Block, type a name for the initialization block.
4. Optional: Specify when the initialization block runs, select **Disabled**.
5. Click **Edit Data Source**, select a **Data Source Type** and complete the remaining fields specific to the selected data source type.
6. Click **Edit the Target Variable**, and edit or define the variable to use with the initialization block.
7. Click **Edit the Execution Precedence**, and **Add** or **Remove** initialization blocks that run before this initialization block.
8. Optional: Click **Test** to verify that your initialization block performs as expected.
9. Click **OK**.

Assign a Name and Schedule to Initialization Blocks

For repository variables, you can specify the day, date, and time for the start date, as well as a refresh interval.

The session initialization block options are:

- **Disabled**. If you select this option, the initialization block is disabled.

  You can also right-click an existing initialization block in the Variable Manager and choose **Disable** or **Enable**. This option enables you to change this property without opening the initialization block dialog.

- **Allow deferred execution**. If you select this option, processing of the initialization block is deferred until an associated session variable is accessed for the first time during the session.

  This option prevents processing of all session variable initialization blocks during the session logon stage, giving a shorter logon time. Session variables that aren't needed during the session don't have their initialization blocks run. This saves the resources which would've been used to run these unnecessary initialization blocks.

  The deferred run of an initialization block also triggers the processing of all unprocessed predecessor initialization blocks. All associated variables of the initialization block and its unprocessed predecessors are updated with the values returned from the deferred execution.

  The **Allow deferred execution** option is unavailable in some circumstances; see When Processing of Session Variable Initialization Blocks Can't Be Deferred.

- **Required for authentication**. Oracle doesn't recommend that you use this setting because initialization blocks for authentication is a deprecated feature.

  If you select this option, this initialization block must succeed for users to log in. Users are denied access to Oracle Analytics Server if the initialization block fails to run. Failure

to run can occur if the wrong credentials have been defined in the initialization block, or if there's an error in the default initialization string.

The initialization block success requirement is waived for internal processes, like Delivers that use impersonation, if a single user session variable has been associated with the initialization block. In this case, the trusted internal process can connect regardless of whether the initialization block succeeds or fails.

1. In the Administration Tool, select **Manage**, then select **Variables** to assign a name and schedule to initialization blocks.
2. In the Variable Manager, expand **Session** or **Respository**, and then select **Initialization Block**.
3. In the [Repository|Session] Variable Initialization Block dialog, type a name for the block. The `NQ_SYSTEM` initialization block name is reserved.
4. (Repository initialization blocks only) In the **Schedule** area, select a start date and time and the refresh interval.
5. (Session init blocks only) Select an option.

The next step is to select the data source and connection pool.

Select and Test the Data Source and Connection Pool

If you select Database as the data source type for an initialization block, the values returned by the database for the columns in your SQL statement are assigned to variables that you associate with the initialization block.

For session variable initialization blocks, you can also select **LDAP Server** or **Custom Authenticator**.
It's recommended that you create a dedicated connection pool for initialization blocks where you select **Database** as the data source type. In addition, if an initialization block fails for a particular connection pool during Oracle BI Server start-up, no more initialization blocks using that connection pool are processed. Instead, the connection pool is blocklisted and subsequent initialization blocks for that connection pool are skipped.

See:

- Initialization Strings Used in Variables to Override Selection Steps
- Examples of Initialization Strings
- Test Initialization Blocks

If you select Database as the data source type, and don't select the Use OBI EE Server option.

The SQL statement used to refresh the variable must reference physical tables accessed through the connection pool specified in the **Connection Pool** field. You don't have to include the tables in the Physical layer of the metadata. At run time, if an initialization string for the database type has been defined, the initialization string is used. If the initialization string for the database type wasn't defined, the default initialization SQL for the database type is used. You can overwrite the default string.

When you create SQL and submit it directly to the database, for example, when using database-specific SQL in initialization blocks, the SQL statement bypasses the Oracle BI Server. The order of the columns in the SQL statement and the order of the

variables associated with the initialization block determine which columns are assigned to each variable.

You should test this SQL using the **Test** button in the *[Repository|Session]* Variable Initialization Block Data Source dialog. If the SQL statement contains an error, the database returns an error message.

If you select Database as the data source type, and select the Use OBI EE Server option.

The SQL statement you use to refresh the variable might be written for a specific database. However, it still works with other data sources because the SQL statement is processed by the Oracle BI Server. The Oracle BI Server can also provide functions such as `PI` that might not be available in the data source, and the SQL statement works with other data sources supported by the Oracle BI Server, for example, ADF, Oracle, and XML files. When you select the **Use OBI EE Server** option, there's no need for a connection pool, because the SQL statement is sent to the Oracle BI Server and not directly to the underlying database.

You can only test this SQL statement using the **Test** button in the *[Repository|Session]* Variable Initialization Block Data Source dialog when in online mode. If the SQL statement contains an error, the database returns an error message.

1. In the Oracle BI Administration Tool, select **Manage**, then select **Variables**.
2. In Variable Manager, select the initialization block to edit.
3. Click **Edit Data Source** next to the **Connection Pool** field.
4. From the **Data Source Type** list, select one of the following types.
   - **Database:** For repository and session variables.
   - **LDAP Server:** For session variables.
   - **Custom Authenticator:** For session variables.
5. If you selected **Database** for your data source type, perform one of the following steps:
   - Select **Default initialization string** or **Use database specific SQL**, and then perform the following steps:
     a. Click **Browse** next to the **Connection Pool** field to select the connection pool associated with the database where the target information is located. If you don't select a connection pool before typing the initialization string, you receive a message prompting you to select the connection pool.
     b. In the Select Connection Pool dialog, select the connection pool and click **Select**. You must select a connection pool before typing an initialization string.

        By default, the first connection pool under the database object in the Physical layer isn't available for selection. This behavior ensures that you can't use the same connection pool for initialization blocks that you use for queries.

        You can change this behavior so that the first connection pool is available for selection by selecting **Allow first Connection Pool for Init Blocks** in the Options dialog, although this isn't recommended.
     c. If you selected **Use database specific SQL**, then in the **Database** pane, expand and select the database. Then, enter its associated string.

        Otherwise, in the **Default initialization string** box, type the SQL initialization string needed to populate the variables.

If you're editing an initialization block to be used by a variable to override a hierarchy column's selection steps, then in the **Default initialization string** box, type the JSON initialization string.

d. (Optional) Click **Test** to test the data source connectivity for the SQL statement.

e. Click **OK** to return to the Initialization Block dialog.

- Select **Use OBI EE Server**, and then perform the following steps:

  a. In the box, enter the SQL initialization string needed to populate the variables.

     The string you enter here is processed by the Oracle BI Server, and therefore as long as it's supported by the Oracle BI Server, the string works with different data sources.

  b. Click **OK** to return to the Initialization Block dialog.

6. If you selected **LDAP Server** for your data source type, perform the following steps:

   a. Click **Browse** to select an existing LDAP Server, or click **New** to open the General tab of the LDAP Server dialog and create an LDAP Server.

   b. Click **OK** to return to the Initialization Block dialog.

      The LDAP server name and the associated domain identifier appear in the **Name** and **Domain identifier** columns.

7. If you selected **Custom Authenticator** for your data source type, perform the following steps:

   a. Click **Browse** to select an existing custom authenticator, or click **New** to create one.

   b. Click **OK** to return to the Initialization Block dialog.

8. Click **OK**.

Initialization Strings Used in Variables to Override Selection Steps

For analyses that contain hierarchical columns, selection steps can be overridden with session variables or repository variables.

Session and repository variables intended for this purpose must contain valid JSON syntax, rather than SQL syntax, in their initialization strings.

Using JSON, you must define type, column, and members with the following syntax.

```
{
  "type": "Hierarchy",
  "column": {
    "subject_area":"your_subject_area",
    "hier_id":"your_hier_id",
    "dim_id":"your_dim_id",
    "table_name":"your_table_name"
  },
  "members": [
    {
      "level_id":"your_level_id",
      "values": [
        your_value,
```

```
        your_value
      ]
    },
    {
      "level_id":"your_level_id",
      "values": [
        your_value
      ]
    }
  ]
}
```

Where:

`"type"` indicates hierarchy type.

`"column"` indicates the hierarchy column's information such as subject area and table name.

`"dim_id"` is the logical dimension name.

`"members"` indicates which hierarchy level and which member ID.

`"level_id"` is the presentation level name.

Example of Standard Hierarchy Syntax

```
{
  "type": "Hierarchy",
  "column": {
    "subject_area": "A - Sample Sales",
    "hier_id": "H2 Offices",
    "dim_id": "H3 Offices",
    "table_name": "Offices"
  },
  "members": [
    {
      "level_id": "Company",
      "values": [
        10001,
        10002
      ]
    },
    {
      "level_id": "Organization",
      "values": [
        1005
      ]
    }
  ]
}
```

Example of Parent-Child Hierarchy Syntax

```
{
  "type":"Hierarchy",
  "column":{
    "subject_area":"A - Sample Sales",
    "hier_id":"Sales Rep Hierarchy",
    "dim_id":"H5 Sales Rep",
    "table_name":"Sales Person"
  },
```

```
  "members":[
    {
      "level_id":"Grand Total",
      "values":[
        27,
        24,
        18,
        16
      ]
    }
  ]
}
```

Examples of Initialization Strings

These examples show you how to initialize strings.

A SQL Statement When Site Uses Delivers

```
SELECT username, groupname, dbname, schemaname FROM users
WHERE username=':USER'
NQS_PASSWORD_CLAUSE(and pwd=':PASSWORD')NQS_PASSWORD_CLAUSE
```

This SQL contains two constraints in the `WHERE` clause:

`':USER'` , use the colon and single quotes, is the ID the user types when logging in.

`':PASSWORD'` , use the colon and single quotes, is the password the user enters. This is another system variable whose presence is always assumed when the `USER` system session variable is used. You don't need to set up the `PASSWORD` variable, and you can use this variable in a database connection pool to allow pass through login using the user ID and password of the user. You can also use this variable in a SQL statement.

When using external table authentication with Delivers, the portion of the SQL statement that makes up the `:PASSWORD` constraint must be embedded between `NQS_PASSWORD_CLAUSE` clauses.

The query returns data only if the user ID and password match values found in the specified table. You should test the SQL statement outside of the Oracle BI Server, substituting valid values for the `USER` and `PASSWORD` variables and removing the `NQS_PASSWORD_CLAUSE` clause.

A SQL Statement When Site Doesn't Use Delivers

```
SELECT username, groupname, dbname, schemaname FROM users
WHERE username=':USER'
AND pwd=':PASSWORD'
```

This SQL statement contains two constraints in the `WHERE` clause:

`':USER'` (note the colon and the single quotes) is the ID the user types when logging in.

`':PASSWORD'` , use the colon and single quotes, is the password the user enters. This is another system variable whose presence is always assumed when the `USER` system session variable is used. You don't need to set up the `PASSWORD` variable, and you can use this variable in a database connection pool to allow pass through login using the user ID and password of the user. You can also use this variable in a SQL statement.

The query returns data only if the user ID and password match values found in the specified table. You should test the SQL statement outside of the Oracle BI Server, substituting valid values for the `USER` and `PASSWORD` variables.

A SQL Statement Joining Tables From Multiple Data Sources - When Using the 'OBI EE Server' Setting

```
select WUSER.name, wuser_detail.email
from "db-11g/orcl"."NAME"."WUSER',
"sqlexpress"."master"."dbo"."wuser_detail"
where username=:USER:
```

The above query example in the initialization block uses a join query with multiple tables from different data sources, for example, Oracle and XML Files. The query works because when you select the **Use OBI EE Server** option, the query is rewritten by the Oracle BI Server for the specified data sources.

Test Initialization Blocks

You should test the SQL statement using the Test button or a SQL tool such as the Oracle Analytics Server Client utility.

If you use a SQL tool, be sure to use the same DSN or one set up identically to the DSN in the specified connection pool.

In online mode, Initialization Block tests don't work with connection pools set to use `:USER` and `:PASSWORD` as the user name and password. In offline mode, the Set values for variables dialog is displayed so that you can populate `:USER` and `:PASSWORD`.

1. In the Administration Tool, select **Manage**, then select **Variables**.
2. In the Variable Manager dialog, double-click the initialization block.
3. In the [Repository|Session] Variable Initialization Block dialog, click **Edit Data Source**.
4. In the [Repository|Session] Variable Initialization Block Data Source dialog, click **Test**.

   The **Test** button is disabled when the **Use OBI EE Server** option is selected in offline mode.
5. In the Set value for the variables dialog, verify the information is correct, and then click **OK**.
6. In the View Data from Table dialog, type the number of rows and the starting row for your query, and then click **Query**.

   The Results dialog lists the variables and their values.

The next step is to associate variables with the initialization block.

Variable Order in Initialization Blocks

The column order in the SQL statement and variable order associated with the initialization block determines the column value that's assigned to each variable.

When you associate variables with an initialization block, the value returned in the first column is assigned to the first variable in the list.

When you open a repository in online mode, the value shown in the **Default initialization string** field of the Initialization Block dialog is the current value of that variable as known to the Oracle BI Server. The number of associated variables could differ from the number of

columns that are retrieved. If there are fewer variables than columns, extra column values are ignored. If there are more variables than columns, the additional variables aren't refreshed. The variables retain their original values. You can run any legal SQL using an initialization block, including SQL that writes to the database or alters database structures, when user ID associated with the connection pool has permissions to perform these actions.

If you stop and restart the Oracle BI Server, the server automatically runs the SQL statement in the repository variable initialization blocks, re-initializing the repository variables.

For session variable initialization blocks, you can select Row-wise initialization. The **Use caching** option is automatically selected when you select the **Row-wise initialization** option. Selecting the **Use caching** option directs the Oracle BI Server to store the results of the query in a main memory cache. See About Row-Wise Initialization.

The Oracle BI Server uses the cached results for subsequent sessions. This can reduce session startup time. However, the cached results might not contain the most current session variable values. If every new session needs the most current set of session variables and their corresponding values, you should clear this option.

See About Using Initialization Blocks with Variables.

Associate Variables with Initialization Blocks

Use this procedure to associate repository or session variables with initialization blocks.

See:

- About Using Initialization Blocks with Variables
- Create Repository Variables
- Create Session Variables

For the Custom Authenticator data source type (session variables only), the variable `USER` is required.

If you select **Row-wise initialization**, the Use caching option is available. See About Row-Wise Initialization

1. In the Administration Tool, select **Manage**, then select **Variables** to associate variables with initialization block.
2. In the Variable Manager dialog, double-click the initialization block to edit the repository initialization blocks or session initialization blocks.
3. Click **Edit Data Target**.
4. In the *Repository*|*Session* Variable Initialization Block Variable Target dialog, do one of the following:
   - Click **New**, and in the Variable dialog, create a new variable to associate with the initialization block.
   - Click **Link**, in the Browse dialog, select the variable to associate the variable with the initialization block, and then and click **OK**.
   - For session variable initialization blocks only, select **Row-wise initialization**.

5. To remove a variable from association with this block, select the variable and click **Remove**.
6. Click **OK**.

Establish Execution Precedence

When a repository has multiple initialization blocks, you can set the order (establish the precedence) in which the blocks are initialized.

First, you open the block that you want to be run last and then add the initialization blocks that you want to be run before the block you've open. For example, suppose a repository has two initialization blocks, A and B. You open initialization block B, and then specify that block A runs before block B. This causes block A to run according to block B's schedule, in addition to its own.

When you select the **Use OBI EE Server** option for an initialization block:

- Execution precedence doesn't apply, because during user login, an initialization block with the **Use OBI EE Server** option selected is run after initialization blocks with the **Use OBI EE Server** option not selected.
- The **Required for authentication** option is dimmed, because this type of initialization block is run after authentication.

1. In the Administration Tool, select **Manage**, then select **Variables**.
2. In the Variable Manager dialog, double-click the last initialization block that you want to be initialized.
3. In the [Repository|Session] Variable Initialization Block dialog, click **Edit Execution Precedence**.
4. In the [Repository|Session] Variable Initialization Block Execution Precedence dialog, click **Add**.

   Add is only available if there are initialization blocks that haven't yet been selected.
5. In the Browse dialog, select the blocks that should be initialized before the block that you've open, and then click **OK**.
6. To remove a block, in the [Repository|Session] Variable Initialization Block Execution Precedence dialog, select the block you want to remove and click **Remove**.
7. Click **OK**.
8. If you want the initialization block to be required, in the [Repository|Session] Variable Initialization Block dialog, select the **Required for authentication** option.
9. Click **OK**.

When Processing of Session Variable Initialization Blocks Can't Be Deferred

Processing of session variable initialization blocks can't be deferred in some circumstances.

When the processing of session variable initialization blocks can't be deferred, a message is displayed that explains why.

See Assign a Name and Schedule to Initialization Blocks.

The following list summarizes the scenarios in which processing of session variable initialization blocks can't be deferred:

- The **Row-wise initialization** option is selected in the Session Variable Initialization Block Variable Target dialog and the variables haven't been declared explicitly with default values.

  Example message: `"The execution of init block 'A_blk' cannot be deferred as it is using row-wise initialization."`

- The **Required for authentication** option is selected in the Session Variable Initialization Block dialog.

  Example message: `"The execution of init block 'A_blk' cannot be deferred as it is required for authentication."`

- The **Data Source Type** isn't Database.

  Example message: `"The execution of init block 'A_blk' cannot be deferred as it does not have a connection pool."`

- The initialization block is used by session variables named `PROXY` or `USER`.

  Example message: `"The execution of init block 'A_blk' cannot be deferred as it is used by session variable 'PROXY'."`

- The initialization block is used by session variables where the **Security Sensitive** option is selected in the Session Variable dialog.

  Example message: `"The execution of init block 'A_blk' cannot be deferred as it is used by session variable 'A' which is security sensitive."`

- The initialization block is a predecessor to another initialization block which doesn't have the **Allow deferred execution** option selected.

  Example message:`"One of the successors for init block 'A_blk' does not have "Allow deferred execution" flag set. Init block 'B_blk' does not have "Allowed deferred execution" flag set.`

Enable and Disable Initialization Blocks

You can use the Variable Manager in the Oracle BI Administration Tool to enable and disable initialization blocks.

1. In the Administration Tool, select **Manage**, then select **Variables**.
2. In the Variable Manager, select **Initialization Blocks** under **Repository** or **Session**.
3. In the right pane, right-click the initialization block you want to enable or disable.
4. Choose **Enable** or **Disable** from the right-click menu.
5. Close the Variable Manager and save the repository.

Work with Multi-Source Session Variables

Oracle Analytics Server supports session variables that are populated from multiple data sources.

While the main focus of this section is on the definition and usage of multi-source session variables, you may also select the `VALUEOF` the component session variables in logical queries and data filters.

You can use these multi-source session variables in logical queries or in repository data filters, and contain the union of values from the different data sources.

There is no restriction on the number of values that the multi-source session variable can hold. To create a multi-source session variable, you first create row-wise initialization blocks for each source.

You explicitly define session variables for each source. Use the following format for the session variable names:

- *<ms_variable_name>____<source>*

You must use exactly four underscore characters as the separator.

This automatically creates a single multi-source session variable, named:

- *<ms_variable_name>*

The component session variable names, *<ms_variable_name>____<source>*, appear separately in the Variable Manager in the Oracle BI Administration Tool, but the Expression Builder displays only the single multi-source session variable name, *<ms_variable_name>*.

If any of the row-wise initialization blocks returns null results, this is logged in the Oracle BI Server log, `nqserver.log`. You can add values to the multi-source session variable from other component initialization blocks that succeed in returning values. The multi-source session variable fails only if all of the component initialization blocks return null values.

You can set processing precedence and deferred processing with multi-source session variables, similar to regular session variables.

Example to Illustrate the Creation and Usage of Multi-Source Session Variables

Use these examples to learn how to create a multi-source session variable.

The following example illustrates how to create and use a multi-source session variable:

1. In the Variable Manager in the Administration Tool, select **Action**, select **New**, select **Session**, and then select **Initialization Block**.
2. Create a row-wise initialization block called `mvcountry_sebl_init` with the following SQL for **Default initialization string**:

   ```
   select distinct 'MVCOUNTRY____SEBL', country from siebel_table
   ```

3. Create a second row-wise initialization block called `mvcountry_orcl_init` with the following SQL for **Default initialization string**:

   ```
   select distinct 'MVCOUNTRY____ORCL', country from oracle_table
   ```

4. Still in the Variable Manager, select **Action**, then **New**, then **Session**, and then **Variable**.
5. Create a session variable called MVCOUNTRY____SEBL, making sure to include four underscores between the variable name and the source name. For Initialization Block, select `mvcountry_sebl_init`.
6. Create a second session variable called MVCOUNTRY____ORCL, making sure to include four underscores between the variable name and the source name. For Initialization Block, select `mvcountry_orcl_init`.

While the component session variables appear in the Variable Manager, the *MVCOUNTRY* multi-source session variable that has been created appears in Expression Builder.

Using the Multi-Source Session Variable in a Logical Query

You can now use the MVCOUNTRY multi-source session variable in a logical query.

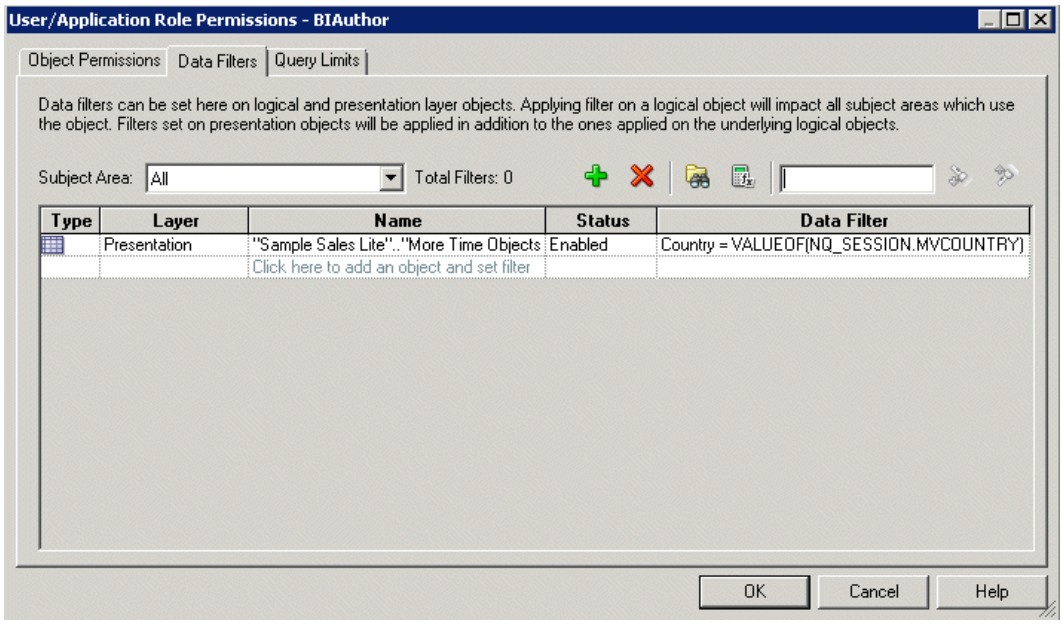
For example:

```
select lastName, firstName, country from employee
where country=VALUEOF(NQ_SESSION.MVCOUNTRY)
```

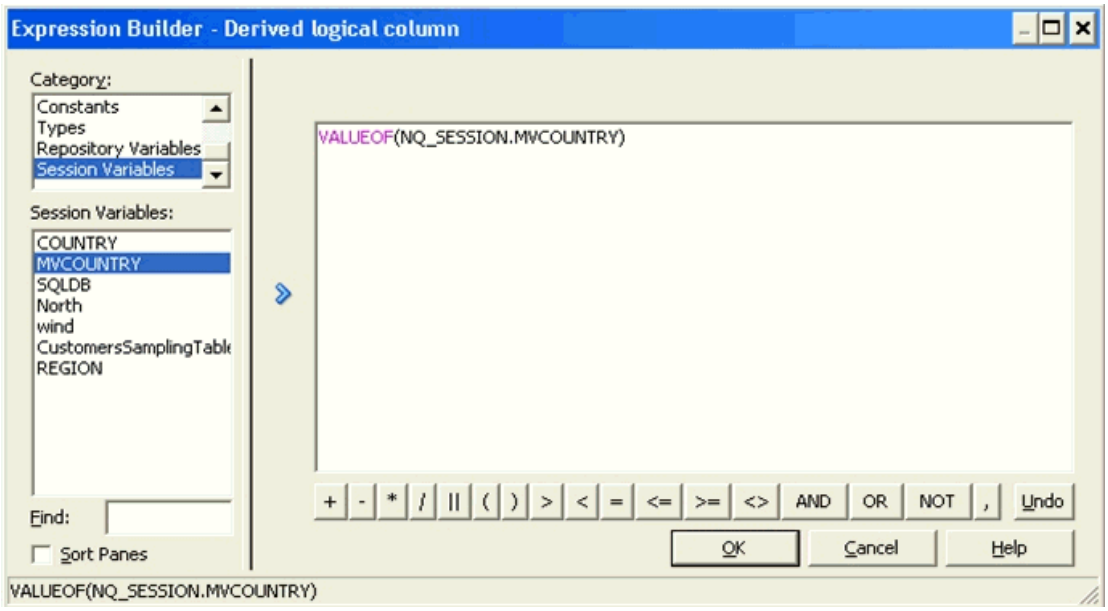
Using the Multi-Source Session Variable in a Data Filter

Perform the following steps to use the MVCOUNTRY multi-source session variable in a data filter:

1. In the Administration Tool, select **Manage**, then select **Identity**.
2. In the Identity Manager dialog, in the tree pane, select **BI Repository**.
3. In the right pane, select the Application Roles tab, then double-click the application role for which you want to set data filters.
4. In the Application Role dialog, click **Permissions**.
5. In the User/Application Role Permissions dialog, click the Data Filters tab.
6. In the Data Filters tab, create the data filter expression:
   - `Country=VALUEOF(NQ_SESSION.MVCOUNTRY)`



The Expression Builder, as shown in the image that follows, displays only the MVCOUNTRY multi-source session variable, and not the regular session variables that were used during the creation of the multi-source session variable.



List Repository Variables Command

Use the list connection pool command `listrpdvariable` to create a list of repository variables in JSON format for a specific service instance.

Use the `listrpdvariable` and the `updaterpdvariables` utility when you need to update more than one variable.

You run the utility through a launcher script, `datamodel.sh` on Linux, and `datamodel.cmd` on Windows.

If the domain is installed in default folder then the location of the launcher script looks like the following:

Oracle_Home`/user_projects/domains/`*Domain_Name*`/bitools/bin/datamodel.sh` or `datamodel.cmd` on Windows.

If the client install doesn't have domain names, the launcher script location is as follows:

Oracle_Home`\bi\bitools\bin\datamodel.cmd`

See What You Need to Know Before Using the Command.

Syntax

The `listrpdvariables` command takes the following parameters:

```
listrpdvariables -SI <service_instance> -U <cred_username> [-P
<cred_password>] [-S <hostname>] [-N <port_number>] [-V <comma or new line
separated FILE containing selected variables names>] [-O
<outputFile.json>] [-SSL] [-H]
```

Where

`SI` specifies the name of the service instance.

`U` specifies a valid user's name to be used for authentication.

`P` specifies the password corresponding to the user's name that you specified for `U`. If you don't supply the password, then you're prompted for the password when the command is run. Oracle recommends that you include a password in the command only if you're using automated scripting to run the command.

`S` specifies the host name. Only include this option when you're running the command from a client installation.

`N` specifies the port number. Only include this option when you're running the command from a client installation.

`V` is an optional argument that specifies the repository variable names that you want to list. You must separate the variable names with commas. If you don't pass the `V` argument or pass the `V` argument without listing any variable names, then by default all repository variables are returned.

`O` specifies the output file name with the `.json` suffix.

`SSL` specifies to use SSL to connect to the Oracle WebLogic Server to run the command. Only include this option when you're running the command from a client installation.

`H` displays the usage information and exits the command. Use `-H` or run `.sh` without any parameters to display the help content.

Example

```
datamodel.sh listrpdvariables -SI ssi -U weblogic -P password -
slc01.example.com -N 7777 -V selectedvar.csv -O listrpdvar.json
```

Sample JSON List Repository Variable Output

```
{    "Title":"List Rpd Variables",
    "Rpd-Variables":[
        {
            "uid":"80000000-3335-155c-991a-0af2537d0000",
            "variable":"RPD_ST_VARIABLE",
            "value":"'rpdStatic Variable'"
        },
        {
            "uid":"c0000000-33c0-155c-991a-0af2537d0000",
            "variable":"DYNAMIC_REPO_VAR",
            "value":"'dynamic repo var'"
        }
    ]
}
```

Sample JSON Output

> **Note:**
>
> If there is no match, meaning none of the variable names included in the `V` argument matched the repository variables in the repository, then the JSON output is an empty array list.

```
{
    "Title":"List Rpd Variables",
    "Rpd-Variables":[
    ]
}
```

Update Repository Variables Command

Use the updaterpdvariables command to upload a JSON input file or a modified JSON file containing variable information to a specific server instance.

Use this and the `listrpdvariable` utility when you need to update more than one variable.

You can create and upload a JSON input file that contains new repository variables, names and values.

You can also upload an updated JSON file containing modified repository variables, names or values. Use the `listrpdvariable` command to create a JSON file containing a list of repository variables for a specific service instance. Modify the variable information in this file and then upload it to the service instance using the `updaterpdvariables` command.

> **Note:**
>
> You must not modify the `uid` values for variables in the file. See Overview of User and Application Role Commands.

You run the utility through a launcher script, `datamodel.sh` on Linux and `datamodel.cmd` on Windows. If the domain is installed in default folder then the location of the launcher script looks like the following:

Oracle_Home`/user_projects/domains/`*Domain_Name*`/bitools/bin/datamodel.sh` or `datamodel.cmd` on Windows

If the client install doesn't have domain names, the launcher script location is as follows:

Oracle_Home`\bi\bitools\bin\datamodel.cmd`

See What You Need to Know Before Using the Command.

Syntax

```
updaterpdvariables -C <rpdVariablesList.json> -SI <service_instance> -U
<cred_username> [-P <cred_password>] [-S <hostname>] [-N <port_number>] [-
SSL] [-H]
```

Where

`C` specifies the name of the JSON file that you want to upload. Note this file must not contain modified `uid` values for variables. See the *Creating a JSON Input File* section and the *JSON Input Repository Variable File* example that follows below.

`SI` specifies the name of the service instance.

`U` specifies a valid user's name to be used for authentication.

`P` specifies the password corresponding to the user's name that you specified for `U`. If you don't supply the password, then you're prompted for the password when the command is run. Oracle recommends that you include a password in the command only if you're using automated scripting to run the command.

`S` specifies the host name. Only include this option when you're running the command from a client installation.

`N` specifies the port number. Only include this option when you're running the command from a client installation.

`SSL` specifies to use SSL to connect to the Oracle WebLogic Server to run the command. Only include this option when you're running the command from a client installation.

`H` displays the usage information and exits the command. Use `-H` or run `.sh` without any parameters to display the help content.

Example

```
datamodel.sh updaterpdvariables -SI ssi -U weblogic -P password -S
slc01.example.com -N 7777 -C listrpdvar.json
```

Creating a JSON Input File

Use the JSON file that was generated when you ran the `listrpdvariable` command as a model for a JSON input file. Using the outputted JSON file as a model ensures that the new file's syntax is valid. See List Repository Variables Command.

When writing the input file, note the following information:

uid – This element can be any text.

variable – This element is the new variable's name.

value – This element is the new variable's value. Use singular quotes inside of double quotes. For example '"VALUE"'.

JSON Input Repository Variable File Example

```
{
    "Title":"List Rpd Variables",
    "Rpd-Variables":[
      {
        "uid":"80000000-3335-155c-991a-0af2537d0000",
        "variable":"RPD_ST_VARIABLE",
        "value":"'rpdStatic Variable My value'"
      },
      {
        "uid":"c0000000-33c0-155c-991a-0af2537d0000",
        "variable":"DYNAMIC_REPO_VAR_NEW_NAME",
        "value":"'dynamic repo var'"
      },
      {
        "uid":"New1",
        "variable":"NEW_VAR_NAME",
        "value":"'new value for new variable'"
      }
    ]
}
```

20

Manage the Repository Lifecycle in a Multiuser Development Environment

This chapter provides best practice information for managing the lifecycle of the Oracle BI repository when you're using a multiuser development environment.
Building your repository using the multiuser development environment enables you to do the following:

- Build large, interrelated semantic models
- Independently build multiple, independent semantic models to run in the same Oracle BI Server and Presentation Services server
- Develop several branches on different schedules, in parallel, while fixing urgent bugs or enhancement requests on the production version
- Incrementally design and test at the individual and team levels
- Enable individual developers to design and test manageable subsets without impacting each other, yet share their changes with other developers in a controlled, incremental fashion
- Migrate changes to test and production systems in bulk, or incrementally

This section covers the development lifecycle of the Oracle Analytics Server repository. It doesn't cover the development lifecycle of the Oracle BI Presentation Catalog used by Presentation Services. This section also doesn't cover how to use the multiuser development environment for Independent Software Vendor (ISV) organizations building portable Oracle Analytics Server applications for sale as products.

See MUD Case Study: Eden Corporation for detailed examples of how the multiuser development environment is used in a typical business scenario.

This chapter contains the following topics:

- Plan Your Multiuser Development Deployment
- Multiuser Development Architecture
- Understand the Multiuser Development Environment
- MUD Tips and Best Practices
- Troubleshoot Multiuser Development

Plan Your Multiuser Development Deployment

Review the tasks you need to perform as part of the planning phase before beginning multiuser development.

This section contains the following topics:

- About Business Organization and Governance Process Best Practices
- About Technical Team Roles and Responsibilities

About Business Organization and Governance Process Best Practices

You need to provide a strong, effective governance process to make decisions about shared resources and to resolve conflicts among the many stake-holders.

As in any business process, you must have a strong business sponsor, and the steering committee staffed with strong business people who can negotiate effectively and make good decisions that don't change over time. Having an effective governance process is the single most important factor in achieving successful multiuser development with Oracle Analytics Server.

Before you begin your multiuser development project, you must first lay out the business value, priorities, road map, and requirements, as well as lower level details of the design, as described in the table below.

Task	Description
Strategic requirements	• Determine which business processes to measure • Determine which data sources and subject areas to access
Business requirements for repository objects	• Select and define metrics, dimensions, and hierarchies • Identify objects that are shared between development teams • Resolve conflicts between teams • Define Presentation layer subject areas
Security requirements	• Define Application Roles and corresponding privileges for your user base • Define which repository developers can access which metadata and data
Development	• Determine the styles of multiuser development to use • Define areas to break down into MUD projects • Determine the owners for metadata objects
Project management	• Set initiatives - purpose, goals, requirements, scope, schedule, budget • Define phases - scope, schedule • Allocate resources - hardware, software, databases, developers • Decide on a strategy for development branching • Prioritize and schedule production updates from different development teams
Operations	• Negotiate service level agreements • Coordinate schedules for updates and downtime

About Technical Team Roles and Responsibilities

These topics describe the hands-on roles involved in repository development and its lifecycle.

Depending on the size of your company and team, one person might perform several roles.

Repository development roles include:

- MUD administrator, one for each development team, plus backup
  - Assigns repository password

  - Sets up and maintains MUD projects
  - Manages the master repository shared directories
  - Manages branches and branch merges
  - Manages repository migrations
  - Manages test and production connection pools
  - Manages independent semantic models, has metadata read/write privileges for all models
- Repository developer, many per development team
  - Knows the repository password
  - Owns, operates, and maintains a personal development sandbox that includes all necessary Oracle Analytics Server components
  - Manages user and application role provisioning on their sandbox stack
  - Creates functional and data authorization content in the repository
  - Performs unit testing
  - Performs check-outs, merges, and publishing, as required
- Production Operations staff
  - Knows the repository password, for managing connection pools and applying patches
  - Applies updated repositories, and applies XML patch updates to the running BI Server's repository
  - Can log in to production computers and read/write the Oracle Analytics Server directories or run programs
  - Manages the production file system, including the repository directory, logs, and configuration files
  - Manages the production servers including the Administration Server, Managed Servers with Java components, and system components like Oracle BI Server and Oracle BI Presentation Services
  - Manages production security, including provisioning users, groups, and application roles
  - Manages and migrates application roles in production
  - Manages production connection pools, in the case where the MUD administrator doesn't have security privileges for production connection information

People in other roles outside the repository development team are also involved. These include people administering the test environment and running the tests, and also the Oracle BI Presentation Catalog developers.

Multiuser Development Architecture

By reviewing these topics, you can get an understanding of the multiuser development environment architecture.

This section contains the following topics:

- About Multiuser Development Concepts

- About Multiuser Development Styles
- Multiuser Development Sandbox Architecture
- Multiuser Development and Lifecycle Management Architecture

About Multiuser Development Concepts

Learn the fundamental concepts related to developing and deploying systems for multiuser development.

Oracle BI Repository

The Oracle BI Repository is the fundamental artifact under development. It defines all the metadata used by the Oracle BI Server for interpreting user requests, applying role-based security, generating queries to data sources, and post-processing the results. Repositories used in multiuser development environments must be in binary Oracle BI repository file (RPD) format, not MDS XML format.

Application Roles and the Policy Store

A secondary artifact under development is the set of application roles. User object permissions, data access filters, and query limits (governors) are defined against these application roles in the repository logic. Oracle BI Presentation Services also uses application roles for assigning its privileges and permissions.

You can use the default policy store embedded in Oracle WebLogic Server, or you can use a separate external policy store. If you're using the embedded policy store, you define application roles using the Console in Oracle Analytics Server, which persists them in the Policy Store in Oracle WebLogic Server. You can then use the Administration Tool in online mode to add application roles from your policy store to your repository at design time. At run time, the Oracle BI Server uses the application roles provisioned to each user to apply the correct security privileges to user requests.

Sandboxes, Projects, and Branches

An instance of the repository is usually edited by only one repository developer at a time. Multiple developers work in parallel on subset instances of the repository, called *projects*. The developers work in separate sandbox environments, and merge their changes into a master repository instance frequently to distribute changes and pick up changes made by others in the team. This approach enables the creation of very large enterprise applications. It also enables independent semantic models to be developed by separate teams and merged into the master repository for production hosting in a single Oracle BI Server cluster. Finally, it enables branching and merging so that teams can work on major projects in parallel, and can even make emergency fixes to the main code line in production without disrupting ongoing development projects.

You use the Simple install type when installing a development sandbox.

Single, Shared Repository

Presentation Services connects to just one repository that has been uploaded to the Oracle BI Server. The metadata for all semantic models must reside in this single repository, even if the semantic models share no objects, see About Multiuser Development Styles.

Repository Password

The repository file is protected by the repository password. The Oracle BI Server needs this password to open and load the repository at startup. It stores the repository password in the secure Credential Store. You must also enter this password when you open the repository in the Administration Tool or other utilities and line commands. User logon credentials are stored in the identity store, not the credential store.

Oracle BI Presentation Catalog

The Oracle BI Presentation Catalog is an important artifact that contains the metadata that defines reports, dashboards, and other reporting layer objects. See *Using Oracle Analytics Publisher in Oracle Analytics Server*.

Migration

The completed repository is migrated to test and production systems using Fusion Middleware Control. Downtime isn't necessary because you can refresh clustered production Oracle BI Servers with a rolling restart.

Deployment Parameters During Migration

Some repository parameters must change when migrating a repository between development, test, and production systems, such as connection pool settings. These parameters must change because they're based on the deployment, not the application logic. You can automate these updates using the Oracle BI Server XML API (`biserverxmlexec -B`). During multiuser development, developers merging in content are automatically prevented from overwriting the master repository test connection pool and database parameters with their local unit test parameters.

Application Role (Policy Store) Migration

There are several options for migrating application roles between development, test, and production systems. For simplicity, this document assumes you'll re-key a small number of application role names by hand. For full information about migrating application roles, and other migration considerations, see Moving Oracle Business Intelligence to a Target Environment in *Administering Oracle Fusion Middleware*.

Users and the Identity Store

As a best practice, users aren't represented by metadata objects in the repository at design time. Also, the repository doesn't manage or store their credentials. Instead, users must always be provisioned to application roles in the run-time environment to receive privileges. Their credentials, as well as their mapping to application roles through groups, are managed in an external Identity Store, see Set Up Security With Users, Groups, and Application Roles in *Managing Security for Oracle Analytics Server*.

About Multiuser Development Styles

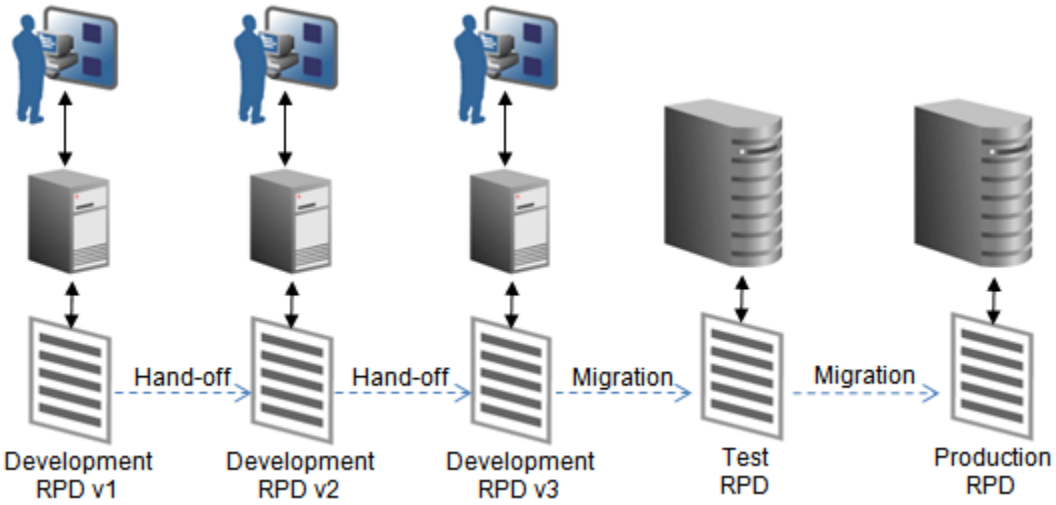
You can learn about multiuser development styles and view potential workflows or architectures.

Choose your style of development based on the size of your team, the number of teams and parallel initiatives, and your requirements for security and availability. The table shows the multiuser development styles.

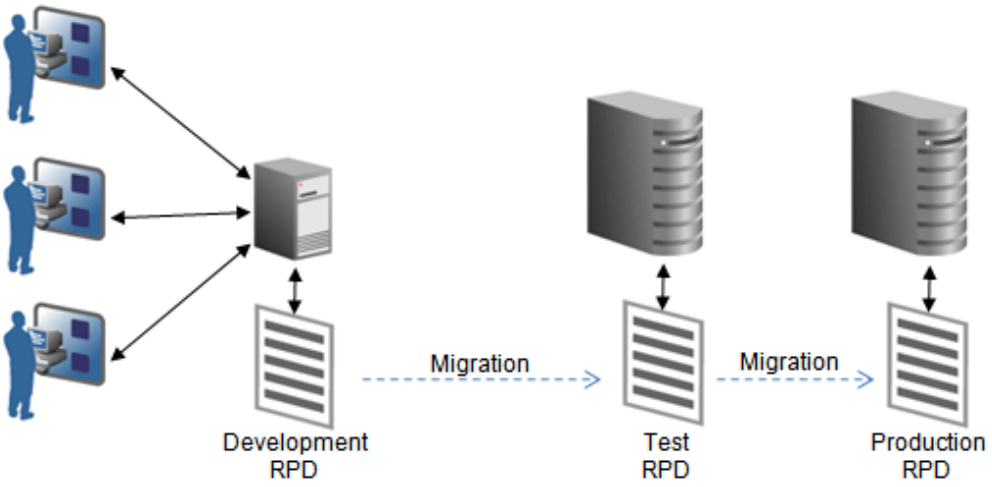
Style	Description
Serial Development	You can use this method if you've a small number of developers and low concurrency. Development users share a repository file through e-mail, a shared directory, or on a shared development system, and only one of them makes changes at a time. They must coordinate with each other on the development schedule.
Serial Development with Patch Files	As a variation on serial development, you can share a base binary repository, and ship changes only between users using patch files.
Shared Online Development	The best practice is for only one developer at a time to develop metadata in online mode against a single Oracle BI Server and its repository. However, multiple online users are an option for development situations where communication among the team members is frequent, a higher risk of conflicts is acceptable, and minimum administrative overhead is a goal.
MUD	The Multiuser Development feature enables over one hundred development users to work in parallel on a shared, enterprise repository. Each user can develop and unit test in a separate sandbox environment, using only manageable-sized subsets of the metadata. When a unit of work is complete, they can automatically merge and publish it into the branch, where other users can pick up those changes and integrate them with their own metadata. When a project phase is ready for promotion, the MUD administrator migrates it to the test environment, and eventually, production. The MUD administrator manages branches and sub-branches to enable parallel development of independent initiatives or fixes, and merges them into the main branch to incrementally migrate them to test and production environments. The MUD administrator also manages fine-grained projects which are the manageable-sized repository subsets individual developers check out to their local sandbox environments. See Understand the Multiuser Development Environment.
MUD with Multiple, Independent Semantic Models	You might need two or more independent semantic models, rather than a single, integrated, enterprise-wide model. The multiple model requirement can be for security requirements, or when unrelated divisions of a business share a common infrastructure. The MUD administrator creates a branch for each model, which enables parallel development and integrated testing for each team's semantic model. When an independent semantic model's branch is ready for promotion to production, the MUD administrator simply merges the branch into main. The MUD administrator can set security on the branches so that each developer can only see the semantic model to which they're assigned, and so that only the MUD administrator and selected production operations staff can access the integrated main model.

Style	Description
MUD with Delegated Administration	When the independent semantic models are developed by different organizations on different schedules, a centralized MUD administrator might not provide the desired level of local control. In this case, you can provide a dedicated MUD administrator for each independent semantic model's branch. The branch administrator operates in the same way as an ordinary MUD administrator. In this scenario, the MUD super-administrator defines a branch for each organization, checks out the subset repository, and provides it to the branch administrator. When the model is ready for promotion to production, the branch administrator passes the repository back to the super-administrator, who merges it into the main branch for promotion, and then migrates the combined repository to production.

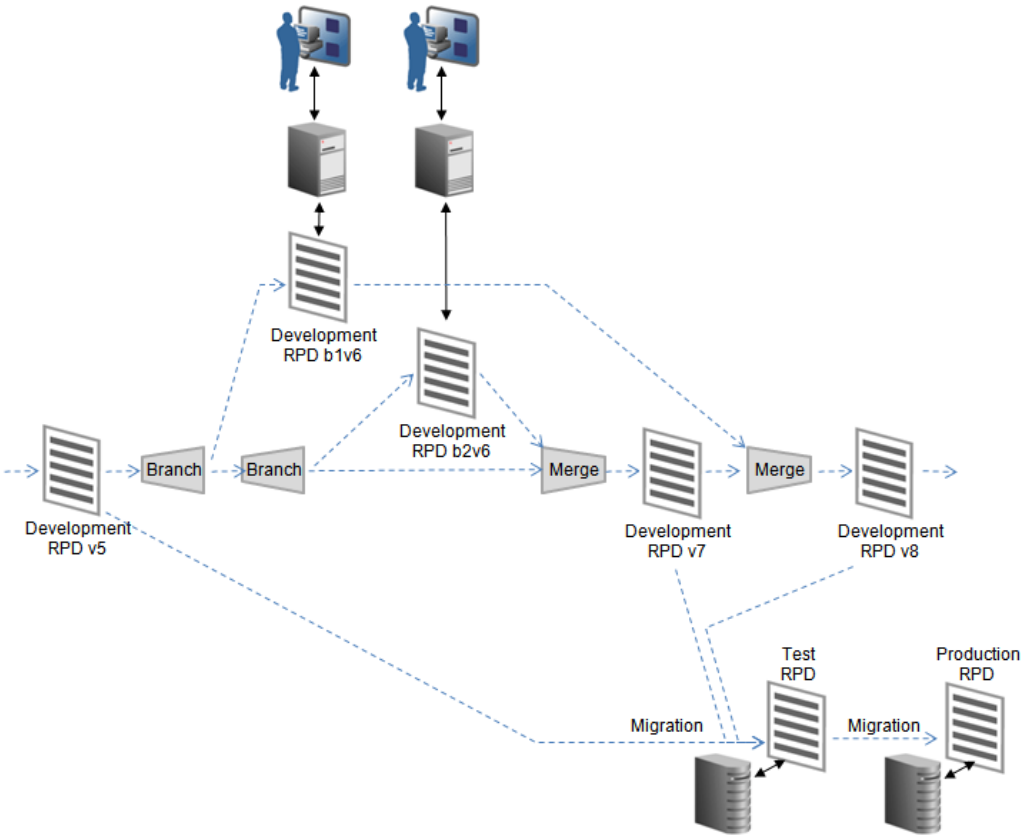
The image shows the serial development style of multiuser development.



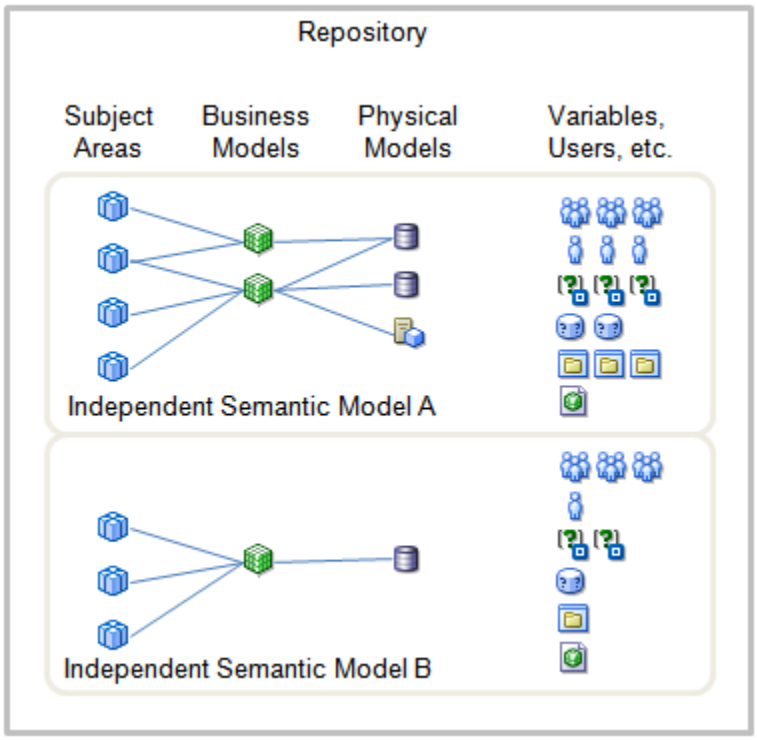
The image shows the shared online development style of multiuser development.



The image shows true multiuser development with branching.



The image shows the architecture for a repository with multiple, independent semantic models.



The table shows which multiuser development styles meet various requirements for security and availability.

Requirement	Serial	Shared Online	MUD with Single Semantic Model	MUD with Multiple Semantic Models	MUD with Delegated Administration
No administrator	Yes	No	No	No	No
Up to five concurrent developers	No	Yes	Yes	Yes	Yes
More than five concurrent developers	No	No	Yes	Yes	Yes
Work on manageable subsets of a large repository, such as Oracle BI Applications	No	No	Yes	Yes	Yes
Built-in checkout, merge, and rollback	No	No	Yes	Yes	Yes
Host independent semantic models in single repository	Yes	Yes	No	Yes	Yes

Requirement	Serial	Shared Online	MUD with Single Semantic Model	MUD with Multiple Semantic Models	MUD with Delegated Administration
Incremental migration of units of work to production	No	No	Yes	Yes	Yes
Developers of independent semantic models can't see each others' metadata	No	No	No	Yes	Yes. Requires secure MUD Directory. An overall MUD administrator must still have access to all metadata from all teams.
Each independent semantic model has its own MUD administrator	No	No	No	No	Yes

Multiuser Development Sandbox Architecture

When using MUD, each developer works on their own, fully dedicated sandbox system.

You should set up your sandbox to contain all the components you need for development and unit testing.

You need to decide whether to use a Linux or Windows server for Oracle Analytics Server. You can use these guidelines:

- If you choose the Windows-only option, make sure your system has enough memory. You need additional resources if you choose to host your database on the same hardware.
- If you choose the Linux option, you need a Windows system to run the Oracle BI Administration Tool. Use the Oracle Analytics Server Simple install type on the Linux system, and use the Client install type on the Windows system to install the Administration Tool.

  In online mode, the Oracle BI Server loads the repository from its local repository directory on the Linux system in:

  ORACLE_INSTANCE`/bifoundation/OracleBIServerComponent/coreapplication_obis`*n*`/repository`

  The Administration Tool on Windows also points to a local /repository directory by default, but you can use any directory for offline development.

You need to install a development database. You can set up the database as dedicated database, personal database, or share the database among multiple repository developers.

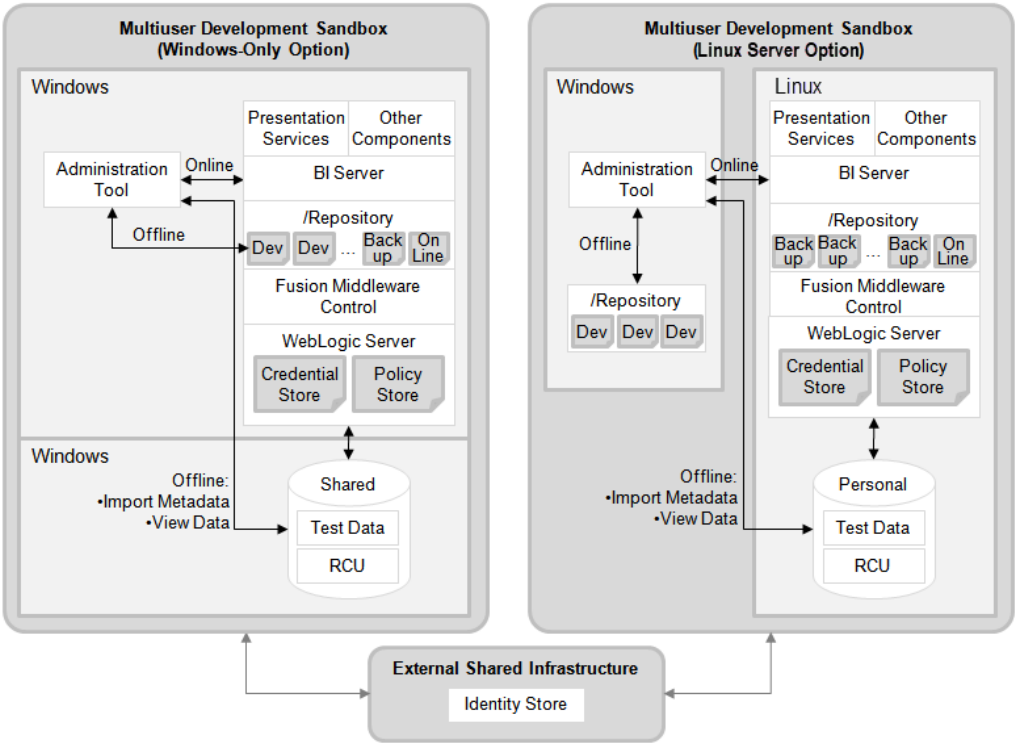
Consider the following about the development database:

- **Platform:** You can choose to host your development database on your sandbox computer if you provide enough memory, or you can host it on a centralized, shared server. Both scenarios are shown in the image that follows.

- **RCU:** The database must contain the schemas required by Oracle Analytics Server. You load these schemas using the Repository Creation Utility (RCU). These schemas enable support for Oracle BI Scheduler, provide sample tables for Usage Tracking, and enable many other features. The Oracle WebLogic Server Managed Servers for Oracle Analytics Server, and all the services that depend on it, require access to a running database with the required RCU schemas in order to operate.
- **Data Source Schemas:** You also need data source schemas for the metadata under development. You can optionally include some data source schemas in your RCU database, or they can be in other databases. In addition, consider the following:
  - **Test Data:** You should load the data source schemas with test data. If users are testing read-only metadata, You can share the schemas among multiple development sandboxes. You can put the schemas on the development sandbox computer when the computer has enough memory.
  - **Multiple Sources:** Your environment can support multiple data sources needed by your initiative such as other relational sources, Essbase, Microsoft Analysis Services, and others. You can share the data sources or put the data sources on dedicated, local or remote servers.
  - **Connectivity:** You must set up connectivity from your Administration Tool and Oracle Analytics Server stack to each data source. This configuration can include installing the required drivers or clients, setting up ODBC DSNs, setting up native connectivity, and other steps. See Import Metadata and Working with Data Sources and Set Up Data Sources on Linux.

    For Oracle Database connectivity, Oracle Analytics Server requires an instance of `TNSnames.ora` in `BI_DOMAIN/bidata/components/core/serviceinstances/ssi/oracledb`.

The image shows the architecture of the multiuser development sandbox.



Note:

Most developers prefer to disable caching in the development sandbox. This makes it easier to validate and debug physical queries using the log. When the cache is enabled, the physical SQL might not appear in the log, because the request might get fulfilled by the cache. In this release, you must disable caching using Fusion Middleware Control. See Using Fusion Middleware Control to Enable and Disable Query Caching in *Administering Oracle Analytics Server*.

Multiuser Development and Lifecycle Management Architecture

The overall MUD architecture contains developer sandbox systems, test, and production systems.

There are several additional major components:

- The Windows MUD administration system is maintained by the MUD administrator:
  - It provides one shared network MUD directory for the main branch, and additional shared network MUD directories for each side branch. The Windows permissions on each shared directory only allow access to the developers for that branch. Each shared directory stores the master repository for that branch, as well as various control and history files for MUD functions.
  - It has a client installation of Oracle Analytics Server. The Administration Tool and Oracle BI Server utilities are used for creating and managing MUD

projects, performing merges, creating patches, and other MUD administrator tasks. Other processes such as the policy store and credential store are typically not used on this platform.

  - You can use a 32-bit or 64-bit system because no Java components, system components, or other infrastructure are used on this computer.

- One or more test systems

  These systems are Linux- or Windows-based, and are used for running integrated tests of merged content. They run the full Oracle Analytics Server stack. These systems are frequently clustered.

- Oracle BI Presentation Catalog system

  You could have a system with a full Oracle Analytics Server stack for developing Oracle BI Presentation Catalog content.

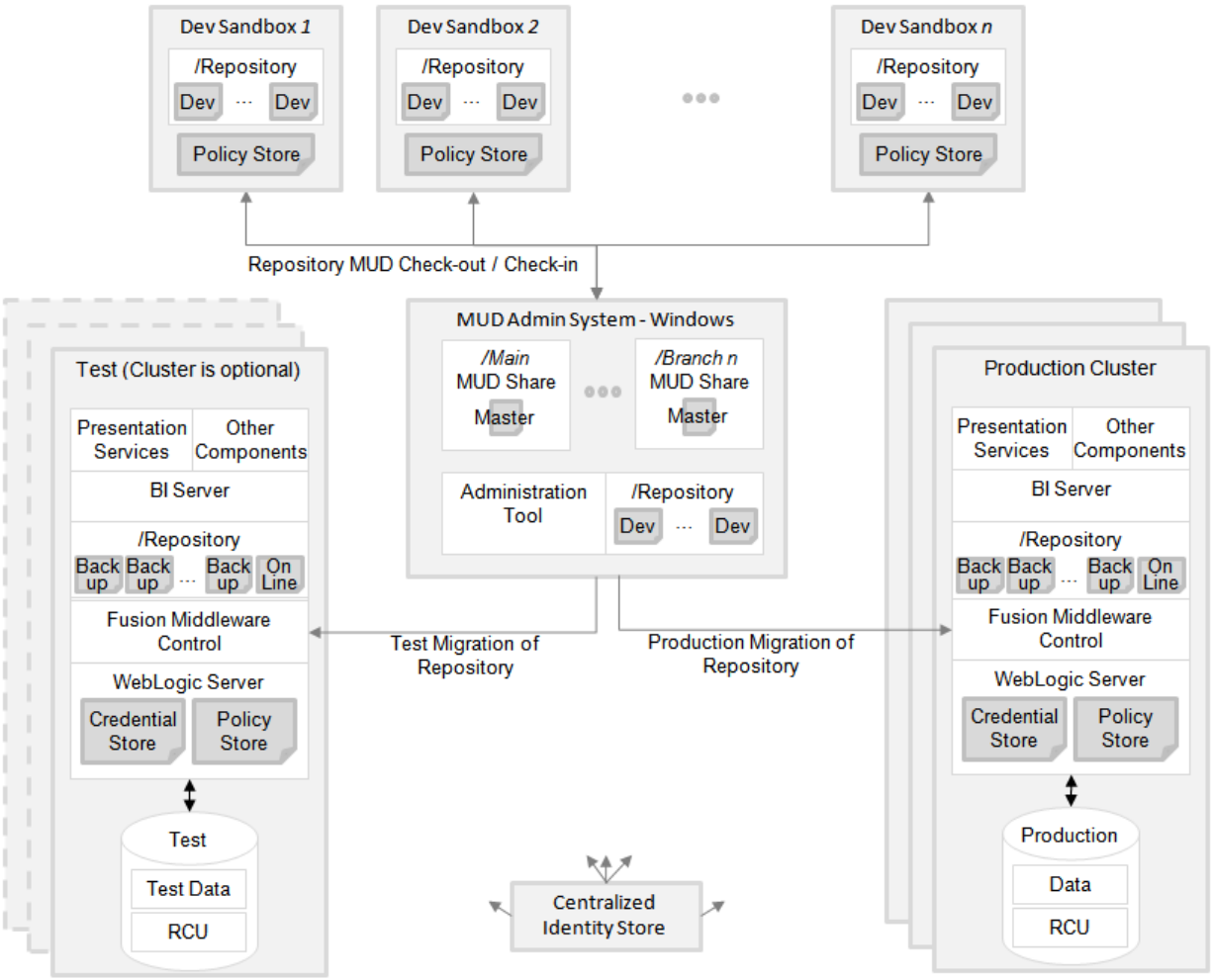
- Clustered production system

  You use a clustered production system on one of the supported platforms.

- External identity store.

  Oracle assumes that you're using an external identity store such as the Oracle Internet Directory.

The image shows a sample deployment architecture for the repository lifecycle using the multiuser development environment.



Understand the Multiuser Development Environment

MUD is a set of features that enables teams of developers to work in parallel on projects of any size, despite the complex interrelationships and dependencies in the repository model.

With MUD, you can:

- Divide the repository file into subsets
  - Enables users to work with manageable subsets when the repository is very large
  - Enables independent testing for each subset by each developer or team
  - Makes it easier to manage merges later after checking out a branch subset
  - Enables you to separate independent semantic models into secure branches for development
- Incrementally develop, test, and migrate
- Merge subsets and branches, handling conflicts between user changes
- Apply Oracle updates to a packaged BI Application you've modified

- Merge separately developed applications into a single repository
- Access history logging and audit information
- Roll back to historical repository states

The multiuser development feature also provides the following other useful capabilities:

- Coordinates merging into the master, including tracking original repository files
- Provides locking for reliable updates
- Logs changes
- Automatically backs up repositories before each potentially destructive operation

This section contains the following topics:

- About Multiuser Development Environment Task Flow
- About Multiuser Development Projects
- How to Create Branches
- Which Merge Utility Should I Use?

About Multiuser Development Environment Task Flow

Learn the basics for working with multiple users in a development environment.

The basic flow of working with multiple users is as follows:

- A developer defines the starting Physical layer, as well as, the basic facts and subject areas. This provides some basic objects to anchor the MUD projects.
- The MUD administrator defines projects and puts the Oracle BI repository (RPD) into the main branch MUD directory.

  The MUD directory where the master repository is stored can't be the same as the Oracle BI Server local repository directory.
- A developer can check out one or more projects, do development work, and then merge the changes into the master by publishing to the MUD directory.
- Other developers check out and do development on the same or other projects. Use projects to create subsets of the development work and not as a tool for enforcing security. Because the publishing step uses a three-way merge, users can check out, develop, and publish their changes in any order. Even property changes to a single object from multiple users are merged. If conflicts do occur between users, the three-way merge feature provides a way for the developer to choose which objects to keep. Communication between users is a key to avoiding and resolving conflicts, and you should have your governance process assign ownership of major objects in order to avoid such conflicts.
- When a development phase is complete, the MUD administrator can migrate the content to a test system. There might be several iterations back through check out, bug fix, publish, and retest. When the repository passes the testing phase, the MUD administrator can migrate it to the production environment.
- The MUD administrator can create and manage multiple development branches as large MUD projects. A branch can be secured to ensure that only one development team can work on it. A branch can even be treated recursively as a main, with its own, delegated MUD administrator.

About Multiuser Development Projects

The multiuser development feature is built around a metadata object called a project. The project is the unit of check-out from the master repository, and the subsequent merge and publish.

When a master repository becomes very large, use a project is a manageable subset that a developer can check out to work on. Projects are self-consistent, so that you can run the consistency checker against the project and then test the project on the Oracle BI Server with a client such as Answers at run time. When you're satisfied with the results, you can merge the project into the master repository so that the project becomes part of the larger application. History is logged and repository backups are automatically created at key points.

The MUD features in the Administration Tool streamline the flow for fine-grained developer projects. Superset projects streamline the management and merging of branches.

The project subset contains a set of metadata objects. You define a project to include a minimum set of objects explicitly, but many others are included implicitly. Having objects implicitly added to projects simplifies your project management task.

The following objects are explicitly specified by the MUD administrator as members of a project:

- Logical fact tables
- Presentation layer subject areas
- Application roles
- Users

  The best practice is to assign users to application roles in Oracle BI repository logic.
- Initialization blocks
- Variables

All other objects are implicitly included in a project and are found by the Administration Tool during the check-out process. For example:

- Descendents of the explicitly defined objects. For example, when a logical fact table is included explicitly, all its logical columns are included in the project implicitly.
- Logical dimension tables that join to the selected logical fact tables, and the join objects themselves.
- Logical table sources for the included logical fact and dimension tables.
- Physical tables that map to the logical tables included in the project, and the joins between them.
- Marketing target levels and list catalogs.

Objects that are in the list of explicitly defined objects are sometimes included implicitly. For example, if a logical column contains an expression that includes a variable, the variable is implicitly included in the project, even if the MUD administrator doesn'texplicitly add it.

It's normal for projects to overlap. An object that appears in one project can also appear in another project. For example, some customers prefer to create an overall project for each independent semantic model, as well as smaller projects within each independent model for checking out individual units of development work. You can also check out multiple projects simultaneously to work on a larger set of metadata.

See Set Up Projects.

How to Create Branches

Learn how to create main branches, side branches, and delegated administration branches.

This section contains the following topics:

- How to Create a Main Branch
- How to Create a Side Branch
- How to Create a Delegated Administration Branch

How to Create a Main Branch

The master repository is usually source-controlled in the main branch, out of which all branches and all development projects check out.

The main branch can stage the repository in production. To migrate content to production, you merge it into the main branch, and then migrate the main repository to the production system.

To fix a production bug, a developer should check out the source from the main branch. The developer then fixes the bug, and then merges it back into the main branch for migration to test and production. The parallel development in side branches isn't affected.

To create the main branch as the MUD administrator, you must first create a shared directory and copy the master repository file to the shared directory. You can use Windows or Linux for the directory and make the Linux share available by Windows users.

Set the security on the share to only allow access by the appropriate developers. Depending on your requirements, you might only allow developers to access the side branch master repositories, not the main branch master repository.

If this is a new project, a developer needs to populate the repository with initial content to split into branch projects.

How to Create a Side Branch

The best practice for branching is to start with a superset MUD project, and then use the MUD check-out, merge, and publish features. Individual users or sub-branches can use the finer-grained projects and check out of the branch master.

Using MUD for this functionality provides automatic back-ups at the check-out points, tracks original repositories to ensure correct merges, uses more optimistic merge assumptions that require less user intervention, and provides history and roll-backs.

Follow these guidelines to create the project:

- If little or no metadata has been designed in the repository, the best practice is to add content that can anchor the project. Adding content makes it easier to ensure the project extracts the physical content you need to support the logical fact tables. You should

create one or more logical fact tables with some representative columns. Map the columns to the physical tables and joins needed to support the fact tables. Finally, create the project and define the objects that belong to it.

- If content already exists, create the project and define the objects needed in that branch. The branch can overlap with other projects, if necessary.
- It's also possible to create an empty project for check-out. However, the developer who checks it out must ensure that all the physical objects that need to be implicitly added to the project are mapped to the logical fact table before changes are published. Similarly, the developer must ensure dimensions are joined before changes are published to ensure their inclusion, and must explicitly add any subject areas, variables, initialization blocks, application roles, and users. This method is more prone to errors than seeding the project before defining it.
- You must secure connection pools for environments such as your production environment. Verify that the connection pool settings in the master repository are acceptable for the developers to access. Developers can change the settings to match their local test databases. When changes are published, connection pool and database settings aren't merged, to prevent overwriting the settings in the master repository.

  Use the Oracle BI Server XML API to automate connection pool changes required during migrations to production and other environments. See Moving from Test to Production Environments in *XML Schema Reference for Oracle Business Intelligence Enterprise Edition*.

Every branch should have its own MUD directory. Set the permissions so that only the developers working on that branch have access to it.

- You can use branch permissions combined with project subsets to prevent developers from seeing metadata that belongs to other teams. Design the projects carefully so that they only extract metadata related to one team. This goal is easiest to achieve if the teams use different business models, subject areas, physical models, variables, initialization blocks, and application roles.
- It's also a best practice to use a consistent system of naming and numbering your branches.

Perform integrated testing on the branch:

When all changes of planned content are published for the phase, the branch project is ready to undergo integrated testing. Migrate the branch master repository file to the test environment. When a bug is found, the assigned developer checks out the appropriate projects, fixes the bug, and tests the metadata. After the changes are published, migrate the branch repository to the test environment again. Developers can test the branch project without impacting, or being impacted by, development work in other branches.

1. In the main master repository, create a project that extracts all content required for the branch.
2. Create a shared MUD directory for the branch.
3. From **File**, select **Multiuser** , and then select **Checkout** to a local repository directory, or another directory.
4. Copy the repository to the branch MUD directory, where it serves as the master repository.
5. Define fine-grained MUD projects for developers to check out from the branch. Inform the developers that the branch is ready for development.

6. Based on your project plan, your developers perform a final merge and publish of their changes when they have completed development and unit tests.
7. Test the branch.
8. Remove the branch master repository from the branch shared directory so that users can't change it and copy it back into your local repository directory, and merge it into the main using the Administration Tool. The main repository is now ready for migration to integrated test and production.
9. The MUD administrator checks out the branch again and places the branch repository in the shared MUD directory for the next phase of development. During the check-out, any changes from other branches, or bug fixes from the main branch, are picked up by the branch repository.

How to Create a Delegated Administration Branch

You can use a branch to delegate local control of a metadata subset to the organization that's developing and maintaining it. To do this, you assign a branch MUD administrator to the branch, who performs the same roles as the main MUD administrator.

This approach works best with an independent semantic model, so that you can ensure that there is no metadata overlapping with other groups.

The delegated branch MUD administrator performs the same tasks as the main branch administrator, including defining projects for further branches and creating fine-grained projects for developers.

1. Set permissions on the main MUD directory so that only the main MUD administrator, and the main MUD administrator backup, have access.
2. Create a branch MUD project, branch MUD directory, and checked-out branch master repository as described in the previous section.
3. Set security on the branch MUD directory so that the main MUD administrator and the delegated branch MUD administrator have access.
4. The branch administrator defines projects for further branches, as well as fine-grained projects for developers. If required, the branch administrator deploys additional branches off the delegated branch for development initiatives, with permissions set to allow developers to check out of these repositories.
5. Developers fix production bugs by checking out of the delegated branch MUD directories, because individual developers aren't allowed access to the main branch.
6. When developers publish all their changes, the branch administrator checks their branches into the delegated branch for integrated testing.
7. To promote a delegated branch to production after integrated testing is complete, the main MUD administrator performs the following two steps:
    a. Removes the branch master repository from the delegated branch repository shared directory and checks it back into the main branch using the Administration Tool.
    b. Migrates the main branch master repository to production.
8. Typically, the main MUD administrator checks out the branch again and places the branch repository in the delegated branch shared MUD directory for the next phase of development. The branch administrator then checks out next-level branches and places their repositories into the branch shared MUD directories, so that developers can check out their fine-grained projects and begin their work.

Which Merge Utility Should I Use?

There are several different merge tools that are optimized for various situations and environments.

When deciding which merge approach and utility to use, consider the platform on which you must perform the task. You should also consider your other requirements, such as whether you need to merge changes you made to a semantic model, or whether you need to combine two semantic models from different development efforts.

> **Important:**
>
> When using tools such as `nqcmd`, `biserverxmlcli`, and `comparerpd`, you must edit the input to match the format expected by SQL, for example, don't include a single quote in your XML content.

The table shows which merge approaches and tools meet various requirements.

Requirement	Merge Approach	Tools Used	Platform
• Merge a checked-out MUD project back into master repository • Merge a checked-out MUD branch project back into the main branch master repository	Three-way merge	MUD merge	Windows
• Combine non-MUD branches and changes back into the main branch	Three-way merge	In the Merge Repository Wizard use Full Merge.	Windows
• Apply an Oracle update XML patch to customized, deployed BI Application • Apply an update XML patch you created from development to a deployed repository	Three-way merge	In the Merge Repository Wizard with Patch Merge selected use the **Patchrpd** utility.	• Windows • All
• Combine disjoint logical content with potential ID conflicts	Two-way merge	In the Merge Repository Wizard select **blank original**.	Windows
• Combine disjoint content guaranteed in advance by the developer to have no conflicts (all platforms)	Insert-Update-Delete	`biserverxmlexec -B` `biserverxmlcli` (online) Copy/Paste XML	All
• Combine disjoint content guaranteed in advance by the developer to have no conflicts (Windows only)	Insert-Update-Delete	• Copy/Paste Administration Tool tool objects • Administration Tool Import from Repository (deprecated)	Windows

See Merge Repositories.

MUD Tips and Best Practices

Learn tips and best practices for working in a multiuser development environment.

This section contains the following topics:

- Best Practices for Branching
- Best Practices for Setting Up Projects
- Best Practices for Three-Way Merges
- Best Practices for MUD Merges
- Best Practices for Two-Way Merges
- Best Practices for Production Migration
- Best Practices for Application Roles and Users

Best Practices for Branching

Use these guidelines for creating side branches.

- The MUD directory where the master repository is stored can't be the same as the Oracle BI Server local repository directory.
- A branch should be a checked-out MUD project. This automates and streamlines many of the tasks of merging the branch back into the main branch, such as using the correct original repository.
- Always put the checked-out branch master repository into its own MUD directory. Then, let developers check out their fine-grained projects from the branch master repository. When branch development, publishing, and testing are complete, remove the master from the branch repository directory and publish it back to the main branch master repository using the Administration Tool. Then, check it out again and place the new version in the branch MUD directory for development of the next phase.
- Use Windows permissions on the branch MUD directory to control which developers have access to it.
- Set multiuser development options by creating an .opt file in the branch MUD directory. As a best practice, define specific administrators, and set Mandatory Consistency Check and Equalize During Merge to Yes, see Set Multiuser Development Options.
- Plan your branches based on the increments of functionality you want to deliver to production. Each branch should contain an increment you want to migrate as a unit.
- If you accidentally merge branches in the wrong order, you can roll them back using the MUD history, see View and Delete History for Multiuser Development.

Best Practices for Setting Up Projects

Use these guidelines for setting up projects.

- Break your repository building into fine-grained projects that are as small as possible while still useful to improve performance and ease of management.
- Break your logical fact tables down into smaller partitions to enable smaller, separate projects.
- For each side branch, overlay a larger project to extract the branch's contents. This enables the project to manage the checkout and merge of the branch, including tracking of the original repository. Individual developers can check out their development projects from the checked-out branch project. Be sure that all development projects are published back to the side branch before merging it back into the main branch.

- When you add new content to a repository, be sure it's part of your project before you check it in. If you create and publish objects that aren't part of a project, they won't be in your extract the next time you check the project out. You or the MUD Administrator must then edit the entire repository, or at least several other projects that do happen to include your new content, and then add the objects to the project at that time.
- Sometimes, you might need to extract several projects at the same time to get all the content you need.
- Presentation layer objects, including subject areas, presentation tables, and presentation hierarchies, are objects that you explicitly include in the project. The security settings of the Administration Tool user have no impact on which subject areas, presentation tables, or presentation columns are included in a project when checking it out. Instead, the set of Presentation layer objects determines the scope of the project.

See Set Up Projects.

Best Practices for Three-Way Merges

Use these guidelines when performing three-way merges.

- Ensure that you've the original repository from which both the modified and current repositories were built.
- Typically, you should open the development repository as current, then use the main repository as modified, and the starting point of the branch as original.
- Unit test before merging.
- As a best practice, select **Equalize during merge** and **Check consistency of the merged RPD** in the Merge Repository Wizard. See Equalize Objects.

Best Practices for MUD Merges

Use these guidelines when performing MUD merges.

- Unit test before merging.
- Unit test after merging, but before publishing. Keep in mind that you're holding the lock on the master repository, so keep it brief.
- Verify that your full name is correct in the **Multiuser** tab. Doing so assists in logging and in checking who holds the locks.
- When publishing changes, be sure to write useful comments in the Lock Information screen. You or other administrators can use the comments later to help identify historical repositories when you need to perform rollbacks or other tasks.
- When the MUD administrator is editing the master Oracle BI repository, it must be inaccessible to checkout users. To accomplish this, you can temporarily remove it from the shared directory and place it in another directory, or you can rename it before editing. Make sure to restore it when the edits are complete.

  You can open the repository in offline mode to avoid locking users out by the Windows file system. You should only use this method when you can finish all work in one session.

- Merge frequently. The list of conflicts and decisions needed in a small merge is easy to understand. When the merge is too large, the number of changes make it much harder to understand, and it's much harder to avoid human errors. If you need to roll back, the number of changes discarded is also much bigger. Performance is also better for small merges.
- If performance of merges is a problem, consider breaking the project down into several, finer-grained projects. Merge frequently, so the number of changes in each merge is smaller and faster.
- Because local connection pool changes are overridden by the connection pool settings in the master repository each time changes are published, the local test settings must be reapplied at each checkout if they're different from the master. It's best to automate application of your local connection pool settings using the Oracle BI Server XML API. See Moving from Test to Production Environments in *XML Schema Reference for Oracle Business Intelligence Enterprise Edition*.
- The most successful large teams have formal process requirements and expectations for the communications and tasks between the repository developers and the MUD administrator. For example, they might have a form for a request for the MUD administrator to create a project. Many teams also have service level agreements and lead times, such as 24 hours to create a project.
- Set the option to force a consistency check during MUD merges. A clean consistency check ensures that the Oracle BI Server can load the model correctly, similar to the way a compiler checks to see if code can be generated correctly. Even if the merge seems to succeed, an inconsistent repository may have trouble with starting the Oracle BI Server, online repository editing check-ins, and subsequent merges. See Set Multiuser Development Options.
- Set the option to force an equalize before merge. This reduces the number of duplicate objects, since it's common for developers to import the same physical tables, for example. See Set Multiuser Development Options.

  See Equalize Objects for the importance of equalizing objects.
- Don't delete or change content needed by others, unless you're the owner and have coordinated with the other developers. If you delete a column you don't need in your project that action usually causes it to be deleted from the master when you merge, even if other users depend on it.
- Presentation object aliases receive special treatment in merges. Their purpose is to hold historical names of objects, so that when names change, old reports don't break. If you changed any names during development, new aliases were added. During merge, you've the option whether to keep any new aliases you've created, or not. You also have the option to keep any or all past aliases, because the historical reports might still exist.

See About the Multiuser Development Merge Process.

Best Practices for Two-Way Merges

Use two-way merge when you need to combine two repositories that were developed separately into a single repository.

This situation usually occurs when you need to host two semantic models in a single repository.

Follow these guidelines when performing two-way merges:

- Make sure that the top-level objects in each repository have different names, so there are no unintentional renames or object merges. Check the following objects:
  - Business models
  - Subject areas
  - Physical databases
  - Variables
  - Initialization block
  - Application roles
  - Users
  - Marketing objects
- Equalize before merging. Doing so honors the fully qualified names over which you've control, and assigns upgrade IDs to ensure there are no conflicts between the two repositories. See Equalize Objects.
- In the Administration Tool, perform a full merge with a blank repository as the original file.

  To create a blank repository, open a new repository, and save it without importing a source or creating any objects. Although this repository contains default security objects, these don't impact your merges.
- Don't use features like Import from Repository or copy/paste in the Administration Tool to move metadata incrementally. These approaches don't correctly merge changes.

  You should also use caution when using the `biserverxmlcli` and `biserverxmlexec -B` utilities. Be sure to fully understand the information about managing IDs described in *XML Schema Reference for Oracle Business Intelligence Enterprise Edition*.

  Using these features might produce the results you expect most of the time, but this is just good luck. The rest of the time, values of the upgrade IDs in the metadata objects clash, effectively causing overwrites of random objects. However, you might not notice the problem until much later when you do regression testing. Because upgrade IDs are assigned sequentially, and are only unique within one repository, clashes are very likely.

See Perform Full Repository Merges Without a Common Parent.

Best Practices for Production Migration

Use these guidelines when moving from test to production.

- When updating metadata on the production cluster, perform a rolling restart to restart one Oracle BI Server at a time, so that users don't experience down time while changes are being loaded. You can use the BI Systems Management API to programmatically start and stop Oracle BI Servers, you can run the start and stop script commands (see Start and Stop Your System in *Administering Oracle Analytics Server*), or you can restart each Oracle BI Server manually in Fusion Middleware Control.
- It isn't recommended to alter metadata in online mode in production using the Oracle BI Administration Tool.

- You shouldn't update metadata in online mode in production using the `biserverxmlcli` utility.

Best Practices for Application Roles and Users

Use these guidelines when working with application roles and users.

- Don't build data access security around user objects in the repository. Instead, define repository permissions, filters and governors based on application roles.
- The set of application roles should be defined by the governance committee. The business team knows what the business roles are, who is allowed to see which data, and which roles are the same from one application to the next. Therefore, the governance committee is in a position to define and name the application roles and decide which roles can be shared across applications.
- When you create a new Application Role, be sure to add it to a project so that you can check it out again after you merge. Also, if you create a placeholder application role in the Administration Tool in offline mode, make sure to add it to the policy store later.
- You can find whether the application roles used by a repository are provisioned in the system by opening your repository in the Administration Tool in online mode and running the consistency checker. It's recommended that you perform this check each time you migrate the repository and application roles to a new system.
- If you only need to migrate a small number of application roles between environments, you can enter them manually in Fusion Middleware Control on the target system if you're using the embedded policy store in Oracle WebLogic Server.

Troubleshoot Multiuser Development

These are some common problems and how to resolve them in this section.

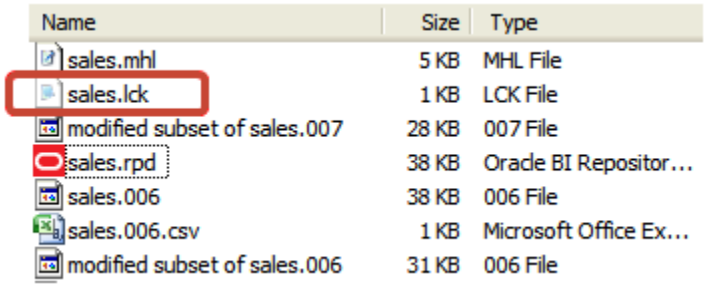
Orphan Lock Held on Master Oracle BI Repository

If a user sets a lock by issuing the command to publish changes to the network, it isn't cleared until publishing is complete. If the user forgets and leaves for a two-week vacation, the MUD administrator can release the lock.

The lock is stored in a hidden system file in the master directory. If you can't see the lock file, in Windows Explorer, select **Tools**, then select **Folder Options**. In the View menu, ensure that the option **Show hidden files and folders** is selected.

The lock file has the same name of the master Oracle BI repository with a `.lck` extension. Delete the lock file to release the lock on the repository.

The image shows a repository lock file.



Object Deleted By Other User

If another MUD developer deletes an object that you need, you can choose one of the following options:

- Roll back to an earlier version, and reapply all the changes since then. The easiest way to roll back is generally to replay history in the history log. To rollback changes, choose **File**, select **Multiuser**, and then select **History**. Select an entry, use **Actions**, and select **View**.

  See View and Delete History for Multiuser Development.

- Re-create the deleted objects, and equalize so that future merges treat it as the same object.

Project Missing Needed Physical Tables and Joins After Checkout

Physical objects don't explicitly belong to a project. Instead, the physical objects mapped to the logical fact tables in your project are extracted at the time of check out.

To get needed physical objects into your local extract, check out an additional project that does have mappings to the physical objects you need. If there's no such project, then the entire repository must be edited to create mappings to a logical fact table in your project. The MUD administrator can take the repository off line to make that change. Then, your next check out should include the physical objects.

Objects Added in the Last Session Missing from Checked Out Repository

If recently added objects are missing from your checked out repository, you might have forgotten to add the objects to your project before you merged and published. Only objects in your project, or inferred from your project like dimensions and physical objects, are included in your extracted repository.

To resolve this issue, ask the MUD administrator to add the objects to your project in the master repository, and then check out again.

Object Renamed by Appending #1

This situation occurs when two objects are merged with the same fully qualified name, but with different internal upgrade IDs. The merge logic in this situation determines that the objects are semantically different objects, and changes the name of the object to preserve uniqueness.

To resolve this issue, run the `equalizerpds` utility, which reassigns upgrade IDs so that objects with the same fully qualified names in the two repositories have the same upgrade IDs. Then, try the merge again. The two objects should be merged instead of causing a rename.

See Equalize Objects.

Rolling Back to Previous Versions

The multiuser development environment stores backup copies of Oracle BI repositories at appropriate checkpoints. Each time a potentially destructive operation is performed, a new backup is stored in the master directory. It has the name of the Oracle BI repository, and the extension is a three-digit incrementing integer. Individual developers can also make copies of their Oracle BI repository files at any time during development.

In the developer's sandbox, the original version of a checked-out project is stored with the name `original`*`rpd_name`*`.rpd`. This version is automatically used if the developer discards changes.

You can also view and roll back to an older version by following these steps:

1. Open the Oracle BI Server, but not a repository.
2. From the **File** menu, select **Multiuser**, and then select **History**.
3. Select the version of interest, choose **Actions**, select **View**, and then select **Repository**.
4. Select **File**, then select **Copy As** to save that version to a new name.
5. Use the older version to replace the latest version, or replace the master repository with the older version.

Manually Updating the Master MUD Repository

During the course of Oracle BI repository development in a Multiuser Development (MUD) environment, it might be necessary to make manual changes to the master repository. Because of the highly controlled nature of the MUD process, you need to be careful when performing any manual steps because there's accounting information stored in the MUD history log (`.mhl`) file. To manually work on the master repository, you must work on the repository in a separate directory from your MUD directory. Then, you must replace both the master Oracle BI repository and the latest versioned repository in the MUD directory.

For example, follow these steps to manually update a repository named *master.rpd*:

1. Copy the master repository (`master.rpd`) out of the MUD directory into a local directory.
2. Use the Oracle BI Server to make the changes necessary to the local copy of the master repository, *master.rpd*.
3. When manual edits are complete, copy *master.rpd* to the MUD directory as *master.rpd*. For example:

   ```
   copy c:\local\master.rpd c:\mud\master.rpd
   ```

4. In the MUD directory, identify the latest repository with a version number, for example, `master.7011`.
5. Copy `master.rpd` to the MUD directory and overwrite the latest versioned repository. For example:

   ```
   copy c:\local\master.rpd c:\mud\master.7011
   ```

Replacing the Latest Version

This example explains how to copy an older version to replace the latest version. Assume you're at version 1000 and want to roll back to version 900. In this situation, you've three files: repository.900, repository.1000, and repository.rpd, the current version. To perform the roll back, make a copy of repository.900 and rename it to repository.1001. This lets you keep repository.1000 in your version history. Then, copy repository.900 to repository.rpd.

21

MUD Case Study: Eden Corporation

Review a fictional case study that shows how the Business Intelligence multiuser development environment might be used for a particular business case.
This chapter contains the following topics:

- About the Eden Corporation Fictional Case Study
- Phase I - Initiate Multiuser Development (MUD)
- Phase II - Branch, Fix, and Patch
- Phase III - Independent Semantic Model Development

About the Eden Corporation Fictional Case Study

Depicts a fictional corporation to describe Oracle Analytics Server initiative examples.

Eden Corporation, a fictional company, recently purchased Oracle Analytics Server. They have two divisions that are licensed and plan to use the product.

Because of this, the company has two separate initiatives:

- Initiative S

  The Sales Division wants to use the dashboard and analysis of revenue versus plan. They want to deploy an initial phase to production quickly to meet an immediate need. Then, they want to roll out more functionality in Phases II and III. Initiative S is large enough for two developers.

- Initiative H

  The Human Resources Division (HR) needs to create a dashboard and analysis of HR data. Initiative H is a smaller initiative with only one developer. They plan to deliver their application to production between Initiative S Phases II and III.

The Sales developers and the HR developers aren't allowed to see each others' data or metadata. The metadata administrator is the only person who has security privileges for all the metadata.

As in all organizations, there is a steady stream of urgent requests and occasional bugs from production. The developers need to deliver fixes for these within days, even though the longer-term initiatives S and H are in development at the same time.

About the Technical Team Roles and Responsibilities

Eden Corporation has staffed the team as follows:

- Adam Straight - MUD Administrator
- Sally Andre - Developer for Sales Division, Revenue project
- Scott Baker - Developer for Sales Division, Quota project
- Helen Rowe - Developer for HR Division

About the Eden Corporation Development Phases

Eden Corporation plans to deploy RPDs to production based on the following timeline:

- January - Sales Phase I (projects Revenue and Quota)
- February - Sales Phase II (add project Target, extend projects Revenue and Quota)
- March - HR (one project used)
- April - Sales Phase III (extend all three projects)

About the Eden Corporation Topology

Eden Corporation plans to use the following systems for their multiuser development environment:

- MUD Administrator - NT computer with a share
- Sally Andre - NT computer for Administration Tool client, and Linux computer to run the Oracle Analytics Server stack
- Scott Baker - high-powered NT computer
- Helen Rowe - either of the above
- Test - Linux computer
- Production - Clustered Linux computers

About the Repository Architecture

Because of Eden Corporation's business structure and initiatives, they need to have two independent semantic models in their repository: one for Sales and one for HR. Because they are hosting multiple independent semantic models, the team itemized the names of top-level objects to prevent duplicate names. Each of these models can have multiple projects.

Planning the Repository Structure

Eden Corporation knows that it's important to plan the structure of their repository file to support the multiuser development needs of their organization. They assigned owners to major objects, so the developers know who to go to when conflicts arise, and which objects they shouldn't modify on their own.

The tables show the high-level repository objects in *main.rpd* for both Initiative S and Initiative H, mapped to projects and owners. Adam is the overall owner of both Initiative S and Initiative H.

Object Type	Object	Owner	ProjRevenue	ProjQuota	ProjTarget
physical database	Sample App Data	Sally	Yes	Yes	Yes
business model	Sales	Sally	n/a	n/a	n/a
logical fact table 1	F10 Billed Rev	Sally	Yes	Yes	No
logical fact table 2	F30 Facts Targets	Scott	No	No	Yes
logical fact table 3	F50 Facts Quotas	Scott	No	Yes	No
logical dimension	(various)	Sally	Yes	Yes	Yes

Object Type	Object	Owner	ProjRevenue	ProjQuota	ProjTarget
subject area (1)	Sales Quota	Scott	No	Yes	No
subject area (2)	Sales Revenue	Sally	Yes	No	No
subject area (3)	Sales Target	Scott	No	No	Yes
variable	S_Last_Load	Sally	Yes	Yes	Yes
initialization block	S_Last_Load	Sally	Yes	Yes	Yes
application role (1)	Sales Management	Sally	Yes	Yes	Yes
application role (2)	Sales Rep	Sally	Yes	Yes	Yes

Object Type	Object	Owner	ProjHR
physical database	Human Resources Data	Helen	Yes
business model	HR	Helen	n/a
logical fact table (1)	Payroll Facts	Helen	Yes
logical fact table (2)	Medical Ins Facts	Helen	Yes
logical dimension	(various)	Helen	Yes
subject area (1)	HR Payroll	Helen	Yes
subject area (2)	HR Medical	Helen	Yes
variable	H_Last_Load	Helen	Yes
initialization block	H_Last_Load	Helen	Yes
application role (1)	HR Management	Helen	Yes
application role (2)	HR Rep	Helen	Yes

Phase I - Initiate Multiuser Development (MUD)

In the first phase of the fictional company example, both Sally Andre and Scott Baker develop in parallel.

Sally creates the starter content that Adam Straight divides into projects. He creates the MUD directory so that Sally and Scott can check out and perform their development. After unit testing, they merge and publish their changes, and then Adam migrates the repository to the test environment. After a bug fix cycle, Adam promotes the repository to production.

The following sections describe Phase I development:

- Start Initiative S
- Set Up MUD Projects
- First Developer Checks Out
- Second Developer Checks Out
- First Developer Publishes Changes to the Master MUD Repository
- Second Developer Publishes Changes to the Master MUD Repository
- MUD Administrator Test Migration Activities
- Phase I Test

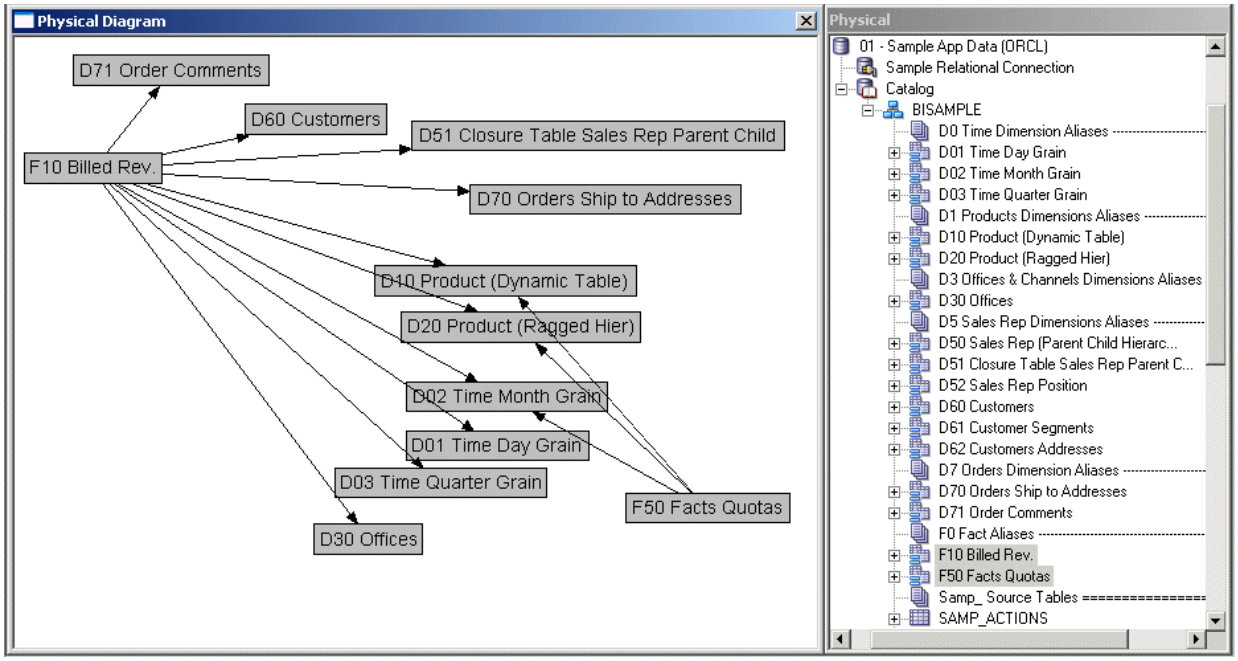
- Phase I Migration to Production
- Phase I Summary

Start Initiative S

In the fictional MUD project, Sally Andre starts off Initiative S from an empty Oracle BI repository.

Because it's easier to divide the repository into MUD projects if you define some logical stars and subject areas first, she begins by developing the physical model needed for Phase I. She makes sure the physical model includes meaningful name aliases for the physical tables and joins. She includes connection pool details for her own local test data sources.

The image shows the physical model for Initiative S.



Sally drags the Physical layer to the Business Model and Mapping layer to create some starter content. She removes unneeded tables, and ensures that the star joins are correct. She also ensures that all the physical tables that are needed during development have mappings from the starter logical tables, so that they're included in the projects when they're checked out. For Sally, these steps create two logical fact tables, F10 Revenue and F50 Quotas, that can act as the basis for the projects.

Sally also needs to have some subject areas to map to the projects in the business model. She could drag the entire business model, but a convenient way to accomplish this is to instead right-click the business model and select **Create Subject Areas for Logical Stars and Snowflakes**. This feature creates a subject area from each logical fact table.

Sally doesn't need to be concerned about the contents of the subject areas yet. All that matters is that each subject area maps to the logical fact table for the same project.

However, she does name the subject areas based on the plan agreed to in the governance meeting: Sales Quota and Sales Revenue.

Sally now has enough content for the MUD administrator to create the first two projects based on the Revenue and Quota fact tables. To review, Sally has made sure that she meets the following criteria at a minimum:

1. At least one logical fact table according to the governance plan, to anchor the projects. The columns of the logical fact tables need not be complete or even properly named, but they do need to be complete enough to map all the physical content.
2. Enough logical dimensions so that the repository passes the consistency check.
3. Physical content that maps to one or more logical fact tables, so they're included in projects.
4. The subject areas needed according to the governance plan.

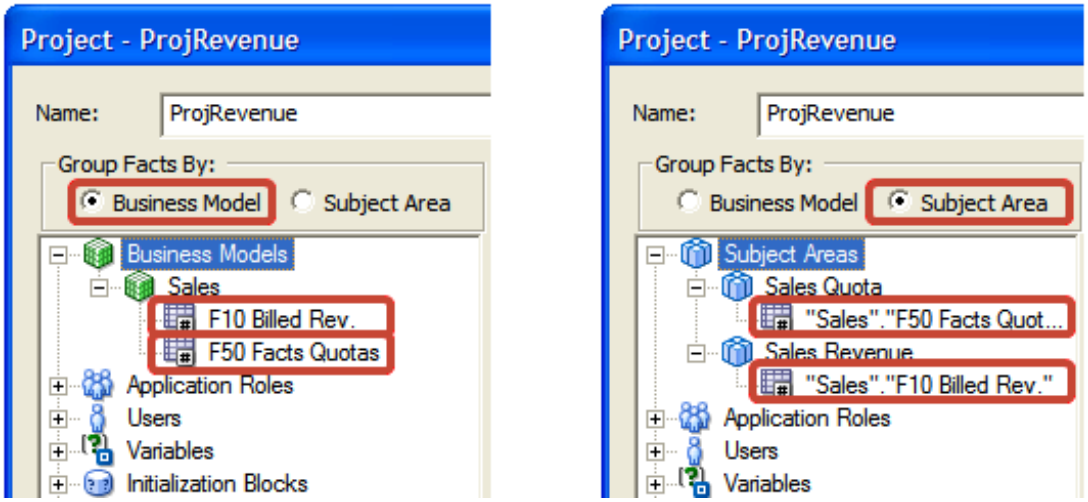
Set Up MUD Projects

In the fictional company example, the MUD administrator for Eden Corporation, Adam Straight, now handles the next few steps to create the projects and get them ready for checkout.

Adam Straight creates the MUD directory,*RPD_main*, where the master Oracle BI repository is stored. This master Oracle BI repository contains the superset of content for the developers. The users check their projects out of the master, and merge their project back into the master when they want to share their changes. Sally copies her started Oracle BI repository to the master folder so that Adam can create the first two projects, *ProjRevenue* and *ProjQuota*.

Adam opens the master Oracle BI repository in the Oracle BI Administration Tool and selects **Projects**. Then, in the Project Manager, he selects **New Project**. Adam names the project *ProjRevenue* and proceeds to pick the logical fact tables at the center of the project. The top object in the list expands to show the logical fact tables, but he has a choice of seeing them grouped by the Business Model to which they belong, or by Subject Area.

The image shows the different ways Adam can view the logical fact tables.



Adam decides to group facts by Business Model for convenience, although he could've used the Subject Area grouping to select the same fact table. He adds the fact table, plus the default application roles and subject areas specified for this project. Because there are no custom-defined application roles, users, variables, or initialization blocks yet, he can't yet add them to the project. Adam repeats this process for ProjQuota, the second project.

Some of the explicit objects are the same in both projects, because both projects share application roles. Similarly, many of the implicit project objects are shared, particularly dimension tables in both the logical and physical models. Projects are a convenience for creating small subsets that are easy to work with. Project aren't for security. It's critical in the governance process that the owner of each top-level object is assigned and documented for the whole team, because this enables the developers to avoid conflicts.

Adam included the logical fact table F10 Bill Rev in the project, even though it's owned by Sally Andre, not by Scott Baker, the owner of this project. He did this because Scott needs to create a measure that derives from measures in both fact tables, Sales percent of quota. Again, the point is to provide the user with the subset of content they need to implement their requirements, not just the objects they own.

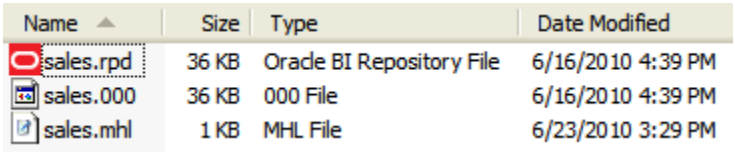
Adam saves the master Oracle BI repository to the shared drive, RPD_Main, as *sales.rpd*. It's now ready for users to check out projects and begin working in parallel.

First Developer Checks Out

An administrator configures their Administration Tool clients for the master repository, check out their projects, and begin working.

The developer, Sally starts by setting up her Administration Tool client to use the master repository. She selects **Tools**, select **Options**, and then selects the Multiuser tab. She sets up the pointer to the master repository directory in the Multiuser tab. She also enters her full name, used in logs and locks. She checks out her project and begins work.

In the Master Repository directory, two new files have been created the *sales.000* and *sales.mhl* files. The image shows the new files.



Name	Size	Type	Date Modified
sales.rpd	36 KB	Oracle BI Repository File	6/16/2010 4:39 PM
sales.000	36 KB	000 File	6/16/2010 4:39 PM
sales.mhl	1 KB	MHL File	6/23/2010 3:29 PM

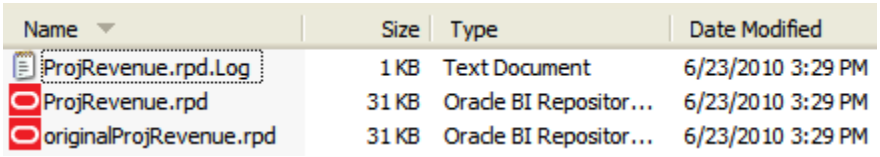
The *sales.000* file is an automatic backup created for *sales.rpd* file that was created when Sally checked out the repository. The backup file is used if a roll back is needed resulting from a problem. The *sales.mhl* file tracks her checkout status and parameters, including project, computer, and user.

Three files have been created in Sally's local repository directory:

- The *originalProjRevenue.rpd* file is the project subset repository at the time of checkout. The *originalProjRevenue.rpd* used later as the original in the three-way merge process, and also when Sally discards her changes.

- The *ProjRevenue.rpd* file contains only the self-consistent subset project, *ProjRevenue*. The *ProjRevenue.rpd* file that's open for editing.
- The *ProjRevenue.rpd.Log* file is the log file for the editing session in the Administration Tool. You can view *ProjRevenue.rpd.Log* files contents in the Administration Tool using **File**, select **Multiuser** , and then select **History**.

  The image shows the three files in the local repository directory.



Sally begins to work on the model for her application in offline mode. She doesn't need to change her connection pool settings because she used her own test data source connection pool details when she created the starter content.

Sally starts by opening her fact table and deleting the unused keys based on the modeling best practice. Then, she adds `SUM` aggregation rules to three measures, *Discnt_Value*, *Revenue*, and *Units*. She also changes the name of *Discnt_Value* to *Discount Amount*, *Units* to *Units Sold*, and *Revenue* to *Sales Revenue*.

Sally also needs to add a new column to the D10 Product table, an upper-case version of the Prod_Dsc column called *PRODUCT DESCRIPTION*. The column uses the following expression: `Upper("Sales"."D10 Product (Dynamic Table)"."Prod_Dsc")`. Sally adds dimension hierarchies, creates a variable called *Constant One*, and initializes it to the value 1. She uses the variable to create a new measure, Constant One. Finally, she saves her work.

Sally starts her sandbox stack so that she can add application roles, and then test her repository using Answers. She follows these steps to start her components in the right order and to configure her system environment:

1. Start the database containing the RCU schema, using its standard controls. This database is the local sandbox developer database.
2. Start the sandbox Oracle WebLogic Server Administration Server. For example, on Windows, from **Start**, select **Programs**, select **Oracle WebLogic**. In Oracle WebLogic, select **User Projects**, select **bifoundation_domain**, and then select **Start Admin Server for WebLogic Server Domain** and enter the user and password created during installation when prompted.

   If you used an *Enterprise or Software-Only* install type, you must also start the Oracle WebLogic Server Managed Server using the Oracle WebLogic Server Administration Console. Typically, you use the Simple install type when installing development sandboxes.
3. Log in to the local sandbox Fusion Middleware Control and upload the repository file, making sure to enter the correct repository password. You upload the local subset repository, in Sally's case, the MUD checked-out repository, *ProjRevenue.rpd*, not the master repository.
4. Also in Fusion Middleware Control, turn off Oracle BI Server caching, so that interpreting the query log is simpler.

5. Still in Fusion Middleware Control, start the system components from the Business Intelligence Overview page.

   See Use Tools to manage and Configure the System in *Administering Oracle Analytics Server* provides more information about steps 2 - 5.

Because Sally's Oracle BI Server is on a Linux system, she must set up ODBC connectivity on her Windows computer so that her Administration Tool client can access the Oracle BI Server there.

Sally manually adds an Oracle BI Server ODBC DSN pointing to the Oracle BI Server on the Linux computer. See Integrating Other Clients with Oracle Business Intelligence in *Integrator's Guide for Oracle Business Intelligence Enterprise Edition* for information about how to create an ODBC DSN for the Oracle BI Server.

Sally is using the Oracle WebLogic Server embedded policy store and needs to add two application roles, *Sales Management* and *Sales Rep*. To add the roles, she opens a Web browser on her Windows computer and logs in to Fusion Middleware Control, pointing to her stack on Linux. She uses Fusion Middleware Control to create the new roles, maps it to the appropriate users, groups, or other roles, and grants the appropriate permissions to the role. (See *Managing Security for Oracle Analytics Server*.)

Sally needs to add the new application roles to her repository, and then use them for object permissions and data access filters. Sally uses the following steps:

1. Sally opens the Administration Tool and selects **File**, then **Open**, and then **Online**. She picks the local Windows ODBC DSN that connects to her local stack, enters her repository password, and also enters the default user name and password for administering her stack that she created upon install.
2. Next, Sally selects **Manage**, and then selects **Identity** to open the Identity Manager. She clicks **BI Repository** in the navigation tree and then clicks the Application Roles tab. She sees the five default application roles, as well as the new ones she just created.
3. Sally double-clicks the Sales Rep application role, and then clicks **Permissions**. On the Data Filters tab, she adds a data filter with an expression that only allows users who belong to this role to see sales that they themselves have made. On the Object Permissions tab, she sets Read, Read/Write, or No Access permissions that allow Sales Rep users to see revenue, but not quota or cost information. On the Query Limits tab, she keeps the defaults for Max Rows and Max Time, and doesn't set any time restrictions. She clicks **OK** to return to the Identity Manager.
4. Next, Sally double-clicks the Sales Management application role and sets up Data Filters, Object Permissions, and Query Limits appropriate for this role, based on the decisions of the governance committee.
5. Finally, Sally exits the Identity Manager.
6. Sally commits the changes she made in online mode by using the **Check In Changes** menu option. This action propagates the online mode changes to her local subset, *ProjRevenue.rpd*, but doesn't commit them to the master MUD repository. Sally publishes her changes to the master repository in a later step.

For the new variable and application roles to be in Sally's project the next time she checks it out, she must add them to the project before she checks in her changes. To do this, she performs the same steps that Adam did when he created the projects: She selects **Manage**, and then selects **Projects**. Sally selects her project, selects the new objects, and clicks **Add**.

Second Developer Checks Out

In the fictional company example, Sally Andre is working on the ProjRevenue project and Scott Baker is getting started on ProjQuota.

Scott sets up his Administration Tool options for MUD, checks out his project, and starts working.

Scott prefers to work in online mode. Doing this tightens the development/unit test loop, because he is modifying the repository while it's running in the Oracle BI Server. Every time he clicks **Check In Changes** in the Administration Tool, his changes are applied to the running server. He can then immediately move to Answers and test the changes there. When he adds, deletes, renames, or reorganizes Presentation layer objects, he must reload metadata in the Answers criteria tab to refresh the tree visible there.

Scott starts his local stack, and uploads his checked-out subset repository. He restarts the Oracle BI Server, opens the Administration Tool, and opens his repository in online mode.

Scott must change the connection pool settings to point to his local test database, because the master repository contains Sally's settings. In the merge process, these connection pool changes are overridden by the connection pools already in the master repository. The next time Scott checks out, he needs to apply his local test connection pool changes again.

Scott can also use the Oracle BI Server XML API to automate connection pool changes required during migrations to production and other environments. (See Moving from Test to Production Environments in *XML Schema Reference for Oracle Business Intelligence Enterprise Edition*.)

Scott's next task is to clean up his logical fact table by removing keys. He also gives a measure a `SUM` aggregation rule and a business-friendly name, *Quota Amount*.

Scott doesn't change anything in the F10 logical table because it's owned by Sally. After she merges and publishes her changes to the master Oracle BI repository, he does the same. Scott checks out again, picking up Sally's changes.

Scott adds a new measure called *Sales percent of quota* to the F50 table. The measure derives from both fact tables with the following expression:

```
"Sales"."F10 Billed Rev."."Revenue" / "Sales"."F50 Facts Quotas"."Quota Amount"
```

Even if Sally changes the name of Revenue in her project, the merge identifies it as the same object and uses the new name in Scott's expression. The merge logic can identify the name change because the upgrade ID of the object is the same as the original.

Scott forgets what he learned in the Governance Committee meeting, that all the dimensions are owned by Sally. He has a requirement for an all-capitals version of the D10 Product.Prod_Dsc column called *PRODUCT DESCRIPTION*. He creates a column identical to the one Sally created. This mistake is detected and resolved through the merge process during the publishing step in a few moments.

Scott doesn't need to upload his repository and restart his system because he is working in online mode. Instead, he unit tests his work immediately after committing his changes using **Check In Changes**. Meanwhile, Sally has finished testing her changes.

First Developer Publishes Changes to the Master MUD Repository

Sally has finished creating and unit testing her first batch of changes, so she saves her work and prepares to merge it into the master repository.

She chooses **File** , selects **Multiuser** , and then select **Publish to Network**. If she forgot to add any new objects to a project, a detailed warning is displayed so that she can add the objects to her project and try the merge again. Otherwise, the objects aren't extracted the next time she checks out the project.

Next, the Administration Tool locks the master repository so that Sally can merge her changes without any chance of corruption from other users' merges.

For logging purposes, Sally uses the comment field to provide a description of the changes she's publishing. Publishing frequently, or performing a subset refresh, also makes it easier to keep track of changes, and easier to audit the history later. A best practice in Administration Tool modeling is to work incrementally to simplify testing and reduce the complexity of each task.

Sally's changes cause no conflicts, so they don't appear in the Define Merge Strategy step that's displayed next. However, aliases for presentation objects are a special case where you can choose to keep either the modified (your local version) or current (the master), or merge the two. The aliases were automatically created when Sally changed the column names, so that reports written to the old names wouldn't break when she put the new names into production. Because Eden Corporation has no reports yet, Sally keeps the aliases empty by selecting Current. She does this for "Sales Revenue," "Units Sold," and "Discount Amount."

Sometimes, there can be a series of aliases if names change more than once. Because there might be a set of reports using the older names, you can select **Merge Choices** in the Define Merge Strategy screen to keep any aliases already in Current as well as the new ones in Modified.

When the merge step is complete, the master sales.rpd is overwritten with the changes from Sally. A merge log is also stored.

Second Developer Publishes Changes to the Master MUD Repository

In the fictional company example, Scott has completed his development work for this phase, he selects **Refresh Subset** to perform a subset refresh to merge his changes with the latest version of the master repository.

The Define Merge Strategy screen asks whether to keep the alias created on the presentation column **Quota Amount**. Like Sally, Scott chooses to keep the current repository value, which doesn't use the alias.

After the subset refresh, Scott unit tests again briefly. Upon inspection, he also notices his mistake of creating the same PRODUCT DESCRIPTION column that Sally did. Because Scott's column was created separately, its internal upgrade ID is different than the one in Sally's. Therefore, even though the name is the same, the merge logic knows it's a different column, and renames it rather than overwriting it by appending #1 (PRODUCT DESCRIPTION#1).

Scott deletes the extra column, connects his logic to Sally's PRODUCT DESCRIPTION column, tests again briefly, and publishes his changes to the network master repository.

If Scott had deleted or modified a different user's object, the error might have been more difficult to resolve. It might have required re-creating and equalizing the object, or rolling back to a backup version of the repository and re-creating his own changes.

MUD Administrator Test Migration Activities

To prepare the repository for the test environment, the MUD administrator, Adam Straight, must now perform several tasks directly on the master repository.

The MUD administrator, Adam, opens the *sales.rpd* repository in offline mode. As soon as he does this, other users are locked out, and get Windows permissions errors if they try to check out projects. If Adam needs to open and close the file several times, he needs to remove the Oracle BI repository from the shared directory while modifying it so that other users can't check out repository objects between his changes.

Adam changes the connection pool settings to match the test environment. When Administration Tool users check out projects, connection pool parameters aren't included in the checkout. The master repository in the MUD directory contains the test connection pools, but each individual developer might need different settings for connecting to their own test databases. At merge and publish, the connection pools in the master repository aren't overwritten by developer changes, so that they can continue to point at the shared test databases.

Adam must ensure that the new application roles are migrated to the test system. Because there are only two application roles, he decides to reenter them in Fusion Middleware Control on the test system. Adam also provisions some test users or groups to the new application roles so the security filters, permissions, and query limits can be tested.

Adam uploads the repository to the test system and restarts the Oracle BI Server. Using his local Administration Tool, he connects to the test Oracle BI Server in online mode and runs the consistency checker. If any application roles referenced by this repository are missing or incorrect, the consistency checker lists the errors.

Phase I Test

Learn about testing in a multiuser development environment by reviewing fictional company example.

The test team can now test the repository. During testing, the test team discovers a bug: "Sales"."F50 Facts Quotas"."Sales percent of quota" was erroneously created with the expression quota/sales instead of sales/quota. The test team writes a bug report, and Scott Baker is assigned to fix the bug.

Scott opens the Administration Tool, checks out ProjQuota, makes the change, changes the connection pool to point to his local test database, and tests on his own sandbox. Then he publishes the changes to the shared MUD directory. He informs Adam that the bug is fixed and that the repository is ready for him to send to test again.

Adam notes that the connection pools are still pointed at the correct test system, because the MUD feature isolates the master repository from connection pool changes in checked out projects. Adam uploads the repository, and restarts the Oracle BI Server.

The test team tests to completion, and the repository is cleared for production.

Phase I Migrate to Production

After the repository has passed the testing phase, you must update the repository's database connection parameters before uploading the repository to production.

You must also migrate and provision the application roles.

Based on the plan provided by the governance team, the production operations team knows the new application roles needed. They create them as Adam did for the test environment. They also provision users or groups to those application roles, based on the security specification from the governance team.

Before migrating to production, Adam has to change the connection pool parameters to the values needed for the production database. In Eden Corporation, Adam has the privilege to see the production connection pools, but the repository developers don't. Therefore, Adam can't change from the test to production connection pools and leave the repository in the master directory, because the developers have Windows permissions to read and write to it. Instead, he creates an XML patch of the connection pools needed for Production. Then, he copies sales.rpd to a secure directory and applies the patch, and then tests to be sure it really does connect to the production data sources. He then uploads the repository to the production system, and starts the production cluster of servers.

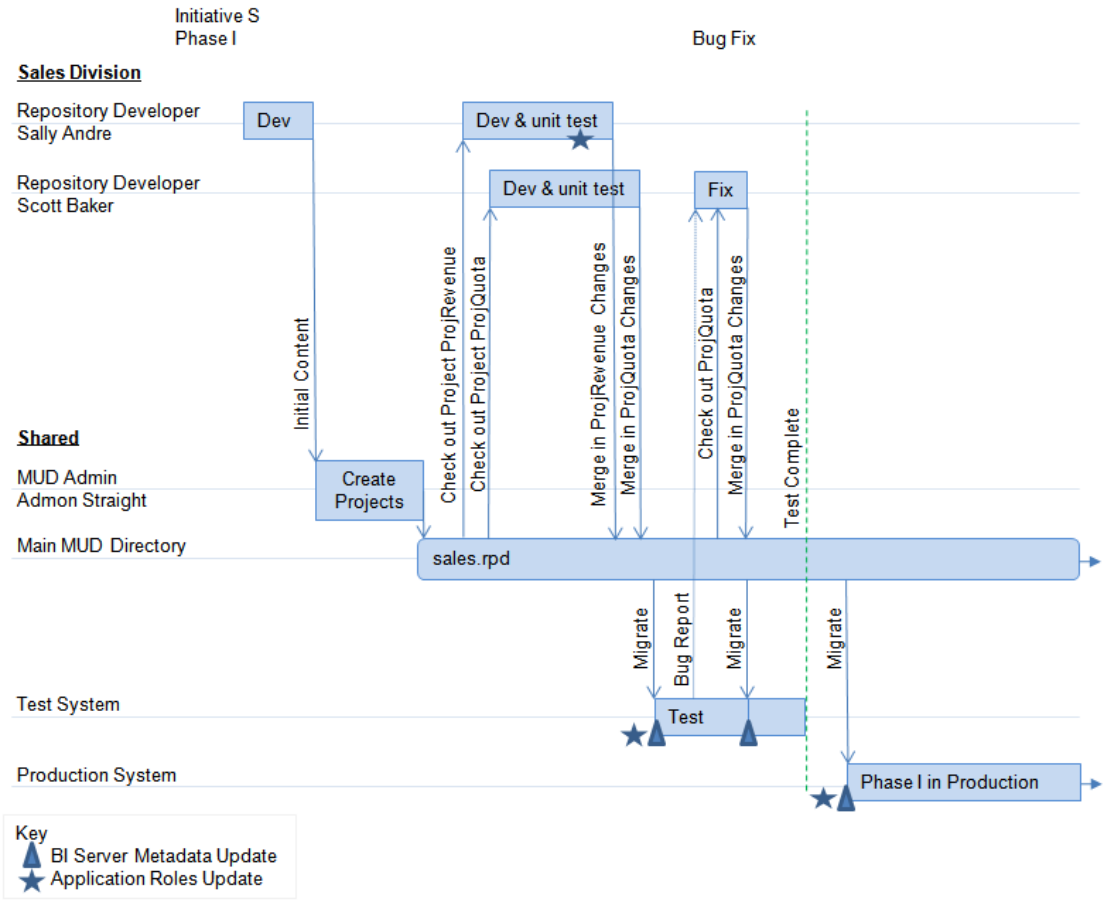
Adam can use the Oracle BI Server XML API to automate connection pool changes required during migrations to production and other environments. See Moving from Test to Production Environment in *XML Schema Reference for Oracle Business Intelligence Enterprise Edition*.

Because the master repository still points to the test databases, the Administration Tool users can still be allowed to see it. Meanwhile, new versions of the production repository can be built at any time by applying the connection pool changes in the XML patch file.

Production validations are now performed. Similar to the migration to the test system, an important validation is to run the consistency checker in online mode to ensure that the application roles are all correct. When this validation is complete, Phase I is in production.

Phase I Summary

The image shows the parallel activities for Phase I for the fictional company example.



Phase II - Branch, Fix, and Patch

In Phase II of the fictional company example, development continues on a new Phase II branch, while a Main branch tracks the production application.

To manage this work, Adam adds a branch project, and set up a second master repository shared directly, one for Main, and one for the new Phase II branch.

Sally adds more content to ProjRevenue. While she works on that, Scott adds brand new content. After Scott merges and publishes, Adam creates the new project, ProjTarget, and move Scott's new content into it. Meanwhile, they must handle any bugs that occur in production, which is still on the main sales.rpd branch.

The following sections describe Phase II development:

- Set Up the Second Branch
- Developers Check Out Projects

- Patch Fix for the Main Branch
- Finish and Merge Phase II Branch
- Phase II Summary

Set Up the Second Branch

In the fictional company case study, Adam begins by creating another MUD directory to hold the master for the new branch. He sets the Windows share security so that Sally and Scott can read or write to it.

Next, Adam places the main repository into the main MUD directory. He adds a new project for the branch, which encompasses all the existing functionality. Then, he closes the repository, and checks out the branch project in his local Administration Tool repository folder. He copies it to the branch MUD directory, where it now serves as the master for the branch.

Developers Check Out Projects

Sally and Scott check out their projects again, and begin developing Sales Initiative Phase II in parallel with each other, and in parallel with Phase I being in production.

Because Scott is adding new content for a new project, he needs to check out one or more other projects to provide the shared objects that he needs to map or join in the new content. He chooses to check out ProjQuota.

Patch Fix for the Main Branch

A fictional company, Eden, is used to show how to create a new measure.

While Sally and Scott are developing Phase II, an urgent CEO request is escalated to them. The CEO wants the key sales managers to see a new measure called *Sales Quota Variance* on their dashboards within two days.

Scott closes his work on the new project on the Phase II branch. The new project remains checked out. He checks out the project to contain the new measure, *ProjQuote*, from the main branch master repository, *sales.rpd*. Scott creates the new measure and corresponding presentation column, tests the measure locally, and publishes the changes back to the main branch.

Scott reopens the checked-out Phase II repository from his local drive and continues development.

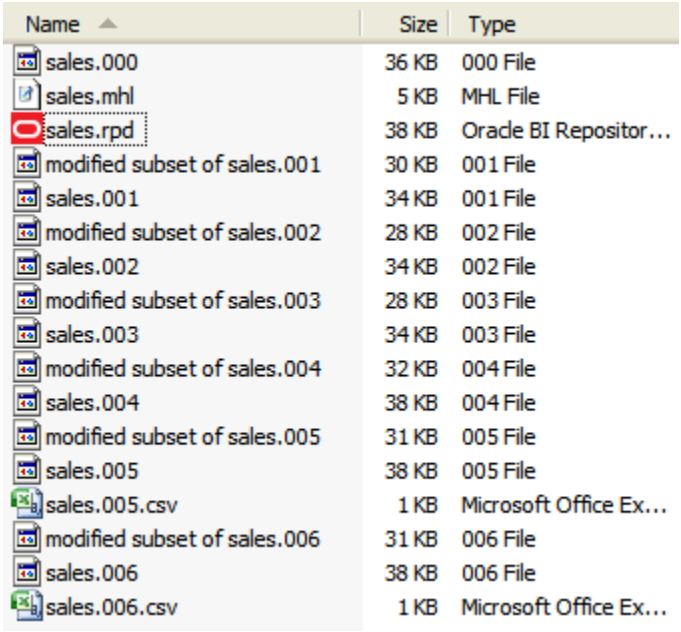
Adam sends the updated *sales.rpd* to the test environment, where the test team validates the fix.

Adam prepares to send the fixed repository to Production. He sends a patch of the repository change to the testers.

To create the patch, Adam compares the modified repository to the one that's currently running in production. The repository running in production is the same as the main repository before the new changes were merged in and is one of the backup repositories in the MUD directory. The current repository running in production is the backup called *sales.006*, the same one Adam identified as the original for the upcoming branch merge. He copies this to *sales.006.rpd* so the Administration Tool

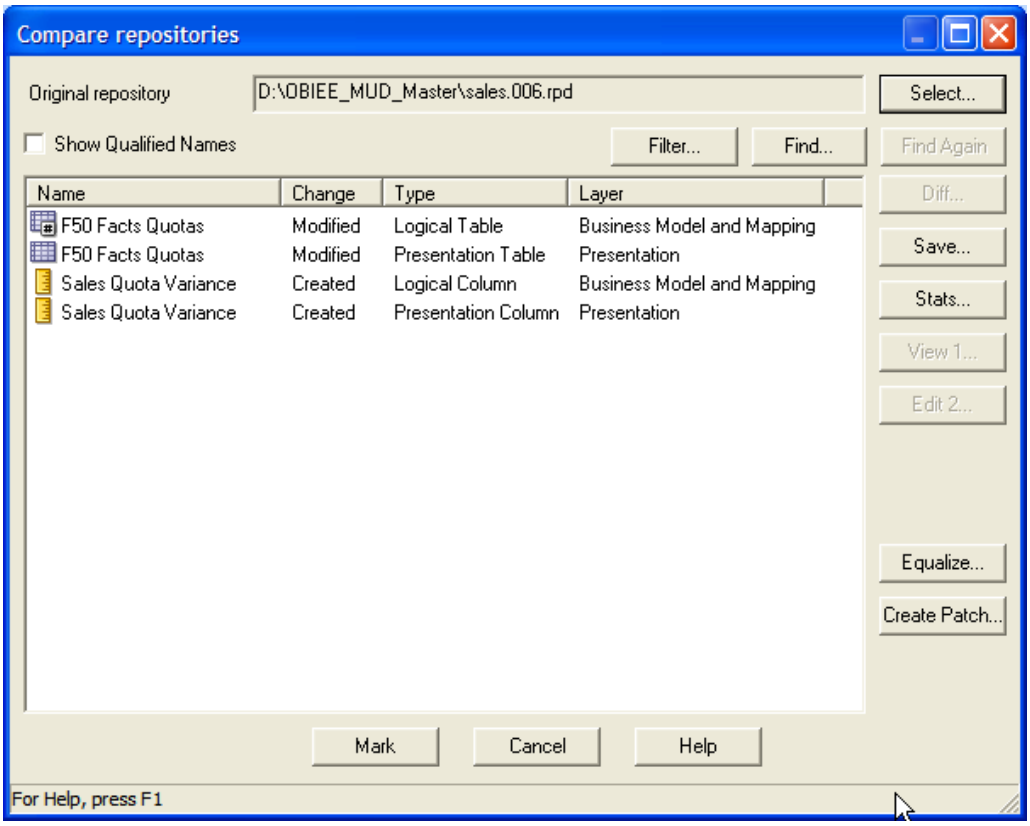
can see and open the file. He can't simply rename it, because it may be needed for another merge later.

The image shows the files in the MUD directory, including *sales.rpd* and *sales.006*.



Adam opens the repository containing the update, `sales.rpd`. He selects **Compare**, and chooses the *sales.006.rpd* as the old version to compare. Compare repositories shows the differences between versions that Adam can include in the patch.

The image shows the Compare repositories dialog.



Adam clicks **Create Patch** and saves the result as `Patch_variance.xml`. The patch contains just the objects needed to apply the two new columns, and their associated interconnections. Adam knows that complex patches might delete objects or overwrite objects to merge in new property values.

Adam's patch appears as follows:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Repository xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <DECLARE>
  <LogicalTable name="F50 Facts Quotas" parentName="&quot;Sales&quot;"
  parentId="2000:68667" parentUid="2160843965" id="2035:69454" uid="2160843966"
  x="718" y="288">
    <Description/>
    <Columns>
      <RefLogicalColumn id="2006:69460" uid="2160844041"
      qualifiedName="&quot;Sales&quot;.&quot;F50 Facts Quotas&quot;.&quot;Quota
      Amount&quot;"/>
      <RefLogicalColumn id="2006:69786" uid="2160845070" qualifiedName=
      "&quot;Sales&quot;.&quot;F50 Facts Quotas&quot;.&quot;
      Sales percent of quota&quot;"/>
      <RefLogicalColumn id="2006:70033" uid="2160845342" qualifiedName=
      "&quot;Sales&quot;.&quot;F50 Facts Quotas&quot;.&quot;
      Sales Quota Variance&quot;"/>
    </Columns>
    <TableSources>
      <RefLogicalTableSource id="2037:69456" uid="2160844747"
      qualifiedName="&quot;Sales&quot;.&quot;F50 Facts Quotas&quot;.&quot;
      F50 Facts Quotas&quot;"/>
    </TableSources>
```

```
  </LogicalTable>
  <LogicalColumn name="Sales Quota Variance" parentName=
  "&quot;Sales&quot;.&quot;F50 Facts Quotas&quot;" parentId="2035:69454"
  parentUid="2160843966" id="2006:70033" uid="2160845342" isDerived="true"
  isWriteable="false">
    <Description><![CDATA[quota - sales]]></Description>
    <Expr><![CDATA["Sales"."F50 Facts Quotas"."Quota Amount" - "Sales".
    "F10 Billed Rev."."Sales Revenue" ]]></Expr>
  </LogicalColumn>
  <PresentationTable name="F50 Facts Quotas" parentName=
  "&quot;Sales Quota&quot;.&quot;&quot;"
  parentId="4004:69706" parentUid="2160844968" id="4008:69707"
  uid="2160844969" hasDispName="false" hasDispDescription="false">
    <Description/>
    <Columns>
      <RefPresentationColumn id="4010:69711" uid="2160844973" qualifiedName=
      "&quot;Sales Quota&quot;..&quot;F50 Facts Quotas&quot;.&quot;
      Quota Amount&quot;"/>
      <RefPresentationColumn id="4010:70032" uid="2160845338" qualifiedName=
      "&quot;Sales Quota&quot;..&quot;F50 Facts Quotas&quot;.&quot;
      Sales percent of quota&quot;"/>
      <RefPresentationColumn id="4010:70036" uid="2160845345" qualifiedName=
      "&quot;Sales Quota&quot;..&quot;F50 Facts Quotas&quot;.&quot;
      Sales Quota Variance&quot;"/>
    </Columns>
  </PresentationTable>
  <PresentationColumn name="Sales Quota Variance" parentName="
  &quot;Sales Quota&quot;..&quot;F50 Facts Quotas&quot;" parentId=
  "4008:69707" parentUid="2160844969" id="4010:70036" uid="2160845345"
  hasDispName="false" hasDispDescription="false" overrideLogicalName="false">
    <Description><![CDATA[quota - sales]]></Description>
    <RefLogicalColumn id="2006:70033" uid="2160845342" qualifiedName=
    "&quot;Sales&quot;.&quot;F50 Facts Quotas&quot;.
    &quot;Sales Quota Variance&quot;"/>
  </PresentationColumn>
  </DECLARE>
</Repository>
```

Adam doesn't need to make any connection pool changes before applying this patch. The correct connection pool settings are already in the repository running in production. The patch doesn't affect this logic, so the connection pool remains correct without an intervention.

Adam must have this patch migrated and applied to the production system. There are several ways to accomplish this:

- Patch main repository offline and upload

  Adam can apply the patch to a copy of the production repository locally on his Windows computer by using the Administration Tool to perform a patch merge. Then, he can upload the repository to the production system, like Sally did earlier in her sandbox. Because the production system is clustered, he must restart all the Oracle BI Servers after uploading the repository. Adam can restart manually through Fusion Middleware Control, one server at a time. If he performs a rolling restart in this way, end users don't see any unavailability. Alternatively, Adam or one of the operations staff can write a script using the BI Systems Management API to automate a rolling restart.

- Patch production repository in place using **patchrpd** utility

  The operations staff can log onto a production system directly, and apply the XML patch using the `patchrpd` utility. If any conflict occurs, the utility cancels the update and exits

without making changes. If the update is successful, the operations staff can then perform a rolling restart, as described in the previous paragraph.

- Patch running system using **biserverxmlcli** utility

  This method isn't recommended for production systems.

If you've privileges to log on to a production Oracle BI Server using the Administration Tool in online mode, you can use **Copy As** to copy it to your local drive.

Finish and Merge Phase II Branch

Sally and Scott complete their changes in the new branch and publish them.

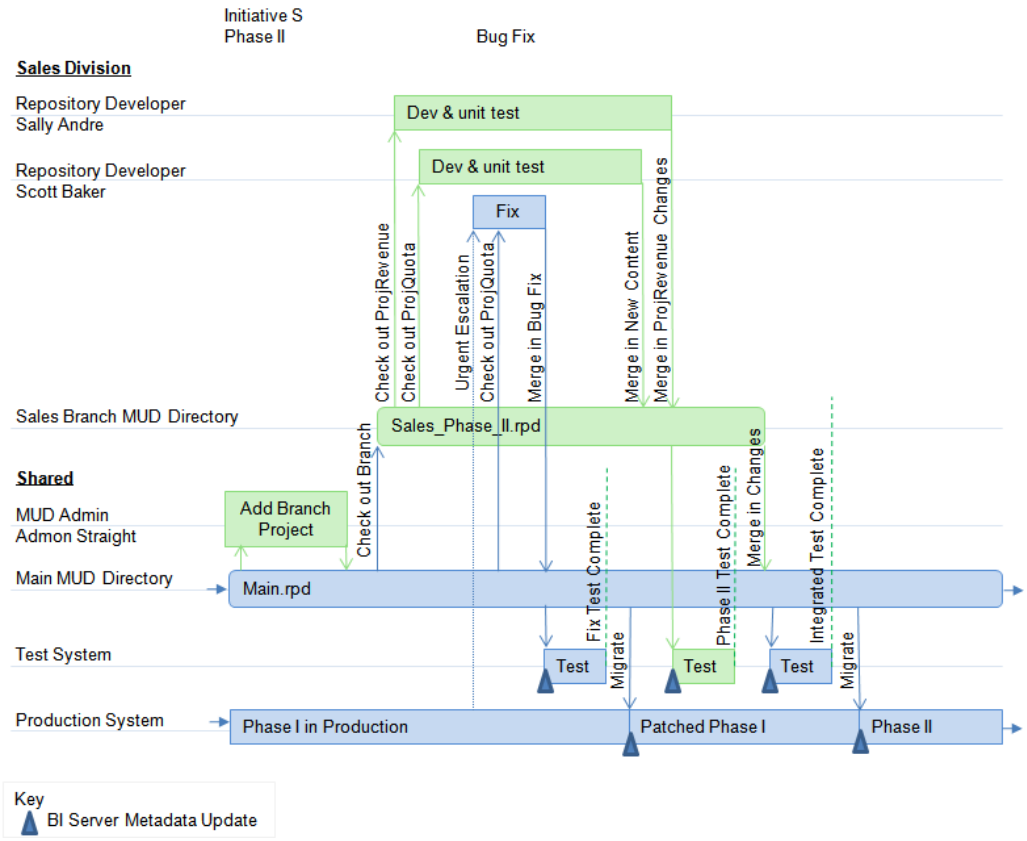
Adam adds Scott's new content to a new project, *projTarget*. He performs the same steps as before to send the branch repository to the testing team.

When testing is complete, merge the branch back into the main branch using MUD merge. The result is a merge of the production patch with the newly developed content to place into production later.

The *sales.rpd* contains all the changes, and the branch is no longer needed. *Sales.rpd* is sent to integrated test, to ensure the merged content doesn't cause any bugs in the existing content. When integrated testing is complete, Adam creates another patch containing the changes, and has the operations staff apply it to the running production system. Sales Initiative Phase II is now in production.

Phase II Summary

The image shows the parallel activities for Phase II.



Phase III - Independent Semantic Model Development

In the next phase of the fictional company example, Sally and Scott begin development of Phase III of the Sales initiative.

Meanwhile, Helen Rowe builds the first phase of the HR initiative and brings this new independent semantic model into production.

The following sections describe Phase III development:

- Security Considerations for Multiple Independent Semantic Models
- HR Semantic Model Developer Builds Content
- Phase III Summary

Security Considerations for Multiple Independent Semantic Models

In the fictional company example, Helen's application has highly sensitive personal information, such as salaries and medical information.

Meanwhile, the Sales application has legally sensitive financial information. Due to corporate security compliance, these two teams aren't allowed to see each other's data or metadata. They also have little content they could share, other than generic dimensions like time dimensions. Finally, they have different business drivers, budgets, and schedules.

For these reasons, the Eden Corporation governance committee decided to use independent semantic models in the repository, one repository for Sales, and the other for HR. This approach requires the two teams to ensure that there aren't any shared objects. No conflicts can exists between the two repositories. The easiest way to ensure this is to make sure that the names for all top-level objects don't conflict. You must also use different variables and application roles.

Some governance committees ensure that top-level objects don't conflict by requiring developers to put a prefix specific to each semantic model before the name of each top-level object, such as S_ for Sales and H_ for HR. This practice makes it easy to see which objects belong to which organizations. Other committees prefer to keep a master list of top-level objects, and require new applications to submit top-level object names for review to ensure there are no conflicts. Two-way merges can also catch any mistakes before overwrites can damage content or cause unexpected object name changes.

Another security requirement is the need to apply security to the separate MUD directories so that only the correct developers have access to each repository. Sally and Scott can only see and check out from the Sales MUD directory, and Helen can only see and check out from the HR MUD directory. The Main directory continues to exist, since it must hold the merged master that's actually in production, but now only Adam has privileges to see or modify that directory.

At Eden Corporation, a final security requirement is to disable the ability for independent semantic model developers to access the running repository in online mode after the merge. There is only a single repository password, so a developer who has the password and access to the repository can see and modify all its contents in offline mode. However, in online mode, the developer also needs a data access user name and password to log on to the Oracle BI Server. To enforce this security requirement, Adam must ensure that the developers have no privileges to log on to the production or test system in this way. The production operations staff can change the repository password to one that only they know, you must perform the

change on a Windows computer because repository passwords are changed using the Administration Tool.

HR Semantic Model Developer Builds Content

Because Helen is working alone on her secure, independent semantic model, she doesn't need to check out a project. She needs to start building her content from a new, blank repository on her local computer.

Helen follows the usual steps of building and unit testing content incrementally. When she is done with unit testing, Helen has a complete, free-standing repository. She sends the repository to Adam. He manually updates the master repository or performs a two-way merge in a separate location. Adam uses one of the following merge methods:

- Manually updates the repository by:
  - Adam equalizes the two repositories to reassign IDs honoring the different names given to the top-level objects. This practice ensures that there are no conflicts during the merge.
  - Adam copies the master repository out of the MUD directory and into a local directory, performs the required manual updates to add the contents of Helen's repository into the master repository, and then copies the master repository back into the master directory.
- Performs a two-way merge in a separate location
  - Adam equalizes the two repositories to reassign IDs honoring the different names given to the top-level objects. This practice ensures that there are no conflicts during the merge.
  - Adam copies the master repository out of the MUD directory and into a local directory, performs a two-way merge by using the Merge Repository Wizard, and then copies the master repository back into the master directory.

After the merge, Adam creates a new project for managing the content going forward, hr_payroll. He adds Helen's content to the project. Adam then checks it out of main and posts it to the HR Branch MUD directory. Using a project checkout makes managing IDs and merges easier later.

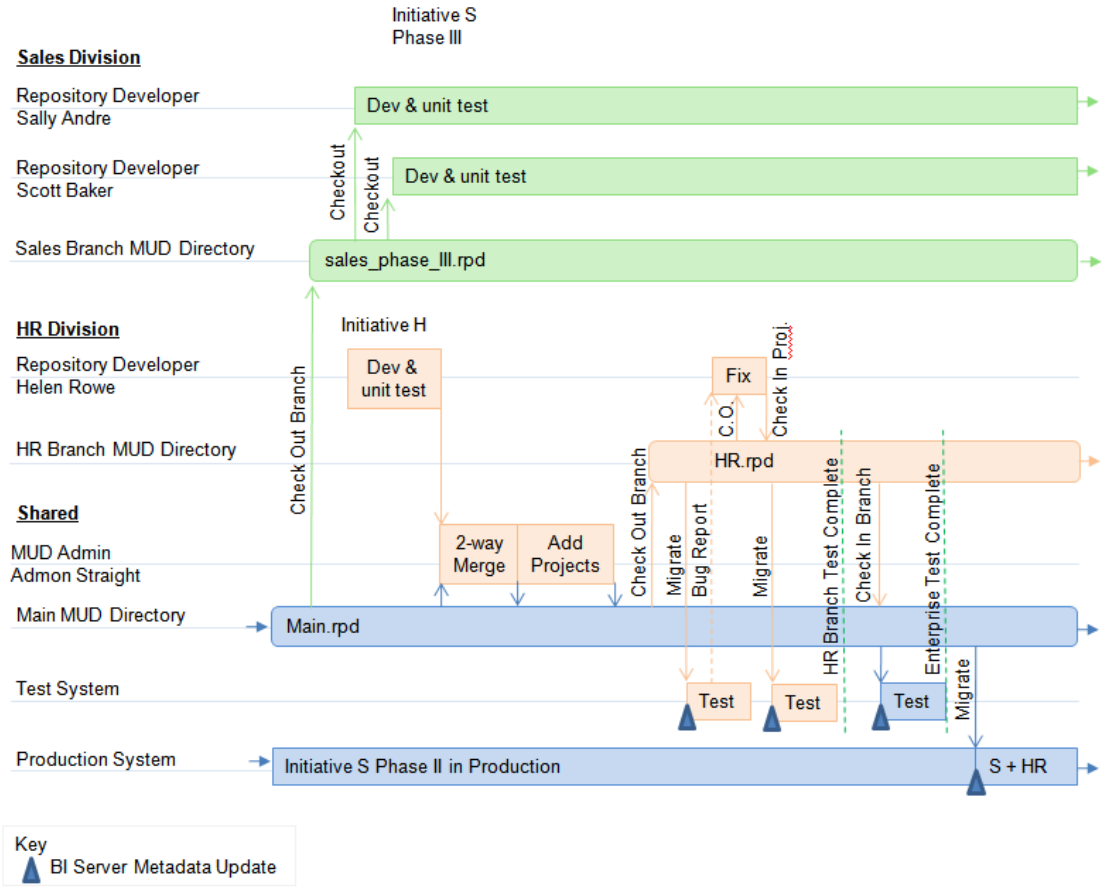
Adam adjusts connection pool parameters, and migrates the repository to the test computer. When a bug is found, Helen checks out the hr_payroll project, fixes it, unit tests it, and publishes it. Helen checks her functional project out of the checked-out branch project. Adam migrates it to the test system for further testing. When testing is complete, he merges the completed HR branch repository back into the main branch, and sends the integrated repository to integration testing on the test system.

When the integrated repository completes testing, it's ready for migration to production. Again, the options are complete repository migration, or applying a patch to the production environment using `patchrpd`. Both methods require a rolling restart.

After this step, the production repository contains content for both Initiative S and Initiative H.

Phase III Summary

The image shows the parallel activities for Phase III for the fictional company example.



22

Merge Rules

You can learn about the merge rules and behavior of the Oracle BI repository merge process. When you use the Merge Repository Wizard in the Administration Tool, the `patchrpd` utility, or publish changes to the network in a multiuser development environment, sophisticated rules determine how objects are merged. Some decisions about merging objects are made automatically by the system, while other decisions appear as prompts in the Define Merge Strategy screen.

This chapter contains the following topics:

- About the Merge Process
- Merge Rules and Behavior for Full Merges
- Merge Rules and Behavior for Multiuser Development Merges
- Merge Rules and Behavior for Patch Merges

About the Merge Process

The repository allows three types of merges.

- Full merges, sometimes called upgrade merges, are used during development processes, when there are two different repositories that need to be merged. The Administration Tool provides a three-way merge feature that lets you merge two repositories that have both been derived from a third, original repository. Full merges can also be used to import objects from one repository into another; see Perform Full Repository Merges.
- Patch merges are used when you're applying the differential between two versions of the same repository. For example, you might want to use a patch merge to apply changes from the development version of a repository to your production repository, or to upgrade your Oracle BI Applications repository; see Perform Patch Merges.
- Multiuser development merges are used when you're publishing changes to projects using a multiuser development environment; see About the Multiuser Development Merge Process.

The merge process involves three versions of a repository: the *original* repository, *modified* repository, and *current* repository. The original repository is the original unmodified file, the parent repository, while the modified and current repository are the two changed files you want to merge. The current repository is the one currently open in the Administration Tool.

The original, modified, and current repository may mean different things, depending on your situation. For example:

- In a development-to-production scenario, you've an original parent file, a current file that contains the latest development changes, and a modified file that's the deployed copy of the original file.
- In an Oracle BI Applications repository upgrade scenario, the current file is the latest version of the repository shipped by Oracle, and the original file is the original repository

shipped by Oracle. The modified file is the file that contains the changes you made to the original file.

You can use patch merge with both of these situations. In a patch merge, you open the current file and select the original file, then generate the patch. To apply the patch, you open the modified file and select the original file, then apply the patch.

Merge Rules and Behavior for Full Merges

Learn about the rules that apply to full merges.

For full merges, the following rules are applied:

- Oracle assumes that you want to keep the changes in the modified repository, for example, if an object is added to or deleted from the modified repository, the object is added or deleted without prompting.
- If an object is added to or deleted from the current repository, the Merge Repository Wizard asks if you want to keep the changes.

  The Merge Repository Wizard tries to ensure that you've the minimum set of objects necessary to service your queries. During a merge, it's possible that objects were introduced by the current repository that aren't needed in the merged repository. To address unwanted objects issue, the Merge Repository Wizard asks whether new Presentation layer objects in the current repository are needed in the final merged result. If you choose to keep the new presentation objects, the dependent logical and physical objects are also added. If you choose not to keep the new presentation objects, then the dependent logical and physical objects aren't kept. The Merge Repository Wizard discards these objects to ensure that the merged repository doesn't get populated with unused objects.
- If an object is added to or deleted from both repositories, the object is added or deleted without prompting. If the same object was added with slight differences in its properties, the Merge Repository Wizard asks which version of the properties you want to keep.
- If an object has been modified only in the current repository, or only in the modified repository, the change is kept. If the same object is modified in both the current and modified repository, and the changes are different, then there's a merge conflict. When conflicts occur, the Merge Repository Wizard asks you to choose which change you want to keep.
- Making a decision about one object can influence a whole set of decisions, depending on the object relationships involved. For example, if you choose to keep a presentation column that was added to the current repository, then the merge process keeps the associated presentation table and subject area, along with the logical column, physical column, and other associated objects upon which the presentation column was based. If you choose not to keep a subject area that was added to the current repository, you aren't prompted to choose whether to keep its associated objects. Adding a join might require the addition of its base tables, while changing an expression might add physical columns.
- Object relationships are interconnected through their properties such as strings and numbers, and an internal value of a property that represent other repository objects. A change to one object might cause a corresponding change to an interrelated object. For example, assume you change the data source of *Init Block B* from a connection pool to *Custom Authenticator A*. In addition to the data source property change to the initialization block object, a corresponding property change

occurs in the custom authenticator object because the value of the initialization block property for *Custom Authenticator A* is now *Init Block B*. The merge process requires synchronized decisions for these properties. If you select **Current** as the decision for the data source property of *Init Block B*, then the decision for the initialization block property of *Custom Authenticator A* is also **Current**.

Special Merge Algorithms for Logical Table Sources and Other Objects

There are special rules for certain types of objects and situations that are in addition to the general rules governing how objects are merged and which situations require prompting.

This section contains the following topics:

- Merge Objects that Use the Vector Merge Algorithm
- Merge Logical Table Sources
- Merge Security Filters
- Infer the Use Logical Column Property for Presentation Columns
- Merge Aliases

Merge Objects that Use the Vector Merge Algorithm

Some objects such as levels, application roles, and object permissions, use a vector merge algorithm that determines the parent/child relationships between objects.

Objects that use the vector merge algorithm include:

- Levels in a dimension, levels associated with a logical column, and child levels
- Dimensions and tables in a logical display folder
- Aggregation content in a logical table source
- Security objects like user and application role membership and permissions
- Initialization block LDAP server settings and processing precedence

The Oracle BI Server determines the initial state of object relationships in each repository during the merge process. For example, the following list shows the different possibilities for object permissions and how they relate to users and application roles:

- M - Missing. The application role, user, or object isn't present in the repository.
- D - Default. Permissions are inherited from the parent application role.
- Y - Yes. The permission is explicitly granted to the user or application role.
- N - No. The permission is explicitly denied to the user or application role.

The Merge Repository Wizard determines the appropriate relationship for the merged repository depending on the state of the object permission relationships in each repository. For example:

- For an original repository with a result of Y, a modified repository with a result of N, and a current repository with a result of M, the Merge Repository Wizard determines a result of N for the merged repository.
- For an original repository with a result of N, a modified repository with a result of Y, and a current repository with a result of M, the Merge Repository Wizard determines a result of Y for the merged repository.

Vector Merge Example: Merging Application Roles

The following list shows the different possibilities for user and application role relationships:

- M - Missing. The application role or user isn't present in the repository.
- Y - Yes. The application role or user is a member of the application role.
- N - No. The application role or user isn't a member of the application role.

The table shows the merged result for different combinations of object relationships in the merging repositories.

Original Repository	Modified Repository	Current Repository	Result
M	M	M	N This situation can happen if neither the application role nor the user are present in the original repository, but the user is present in the modified repository and the application role is present in the current repository. In this case, no membership can be assumed.
M	M	Y	Y
M	M	N	N
M	Y	M	Y
M M in original for this case implies that either the user or application role isn't present. The missing object added in both can't be considered the same object	Y	Y	Y
M	Y	N	Y
M	N	M	N
M	N	Y	Y
M	N	N	N
Y	M	M	Y
Y	M	Y	Y
Y	M	N	N
Y	Y	M	Y
Y	Y	Y	Y
Y	Y	N	N

Original Repository	Modified Repository	Current Repository	Result
Y	N	M	N
Y	N	Y	N
Y	N	N	N
N	M	M	N
N	M	Y	Y
N	M	N	N
N	Y	M	Y
N	Y	Y	Y
N	Y	N	Y
N	N	M	N
N	N	Y	Y
N	N	N	N

Merge Logical Table Sources

Special rules govern how to merge column mappings in logical table source objects. Each column mapping is merged individually.

For each column, if the mapping has changed in either the modified or current repository, the change is kept. If the mapping has changed in both repositories, the Oracle BI Server attempts to merge the mappings automatically.

The deletion of a column isn't considered to be a change in its mapping. If a column isn't present in the modified repository, then the mapping in the current repository is used instead.

If there are differences in aggregation content, then the aggregation content specified by level has priority. In other words, if the aggregation content in one repository is by level and the aggregation content in another repository is by column, then the aggregation content by level is retained.

Merge Security Filters

If a filter for an application role has changed in only one repository, then the change is kept. If the filter has changed in both repositories, the Oracle BI Server attempts to merge the filters automatically.

If an object is required for merging a particular filter such as a presentation column and isn' present, then that filter is considered invalid and doesn't appear in the merged repository. This rule doesn't apply to variables. If a variable is required for merging a particular filter, the Oracle BI Server ensures that the variable is retained in the merged repository.

Infer the Use Logical Column Property for Presentation Columns

Presentation columns have both a Name property and a Use Logical Column Name property.

In some cases, these properties can come into conflict. For example, the table shows a scenario where this situation could occur.

Repository	Presentation Column Name	Logical Column Name	Use Logical Column Name
Original	Sales	GroupSales	No
Current	Sales	Sales	Yes
Modified	GroupSales	GroupSales	Yes

If the regular merge rules for the objects in the table are applied, the merged repository contains a presentation column called *GroupSales* and a logical column called *Sales*, with the `Use Logical Column Name` property set to *Yes*. This result is incorrect because the name of the presentation column is different from the name of the logical column.

To avoid this situation, the Oracle BI Server infers the value of the `Use Logical Column Name` property. Using the inference logic, the merged repository has a presentation column called *GroupSales*, a logical column called *Sales*, and the `Use Logical Column Name` property set to *No*.

Merge Aliases

During the full merge process, users aren't prompted to make decisions about aliases.

Aliases from the current and modified repositories are merged automatically.

In multiuser development merges, however, users are prompted to choose whether to keep aliases from the current repository, keep aliases from the modified repository, or merge choices to keep aliases from both repositories.

- If object names change because of the merge process, then the previous names are added as aliases.
- Any aliases that aren't associated with presentation objects are deleted.

Merge Rules and Behavior for Multiuser Development Merges

The rules for multiuser development merges are very similar to the full merge rules, with important differences.

- Changes to security settings aren't retained when you perform a MUD merge to prevent developers from overwriting passwords and other important objects in the master repository.
- The database and connection pool properties in the master repository take precedence over the same properties on the developer's computer. This precedence are applied without a prompt during a multiuser development merge.
- Inserts (created objects) are applied automatically. Because a subset of the master repository is being used as the original repository, most objects in the master repository appear to be new. This would result in many unnecessary prompts that the developer would have to manually approve. Therefore, new objects are created without a prompt during a multiuser development merge.
- Conflicts that aren't inserts but are resolved because of the automatic inserts are applied without a prompt during a multiuser development merge.

To change security settings or database features in a multiuser development environment, you must edit the master repository directly. To do this, remove the master repository from the multiuser development directory, edit it in offline mode, then move it back.

Merge Rules and Behavior for Patch Merges

The rules for patch merges are also similar to the full merge rules, except that the behavior for deleting objects is different.

For example, if an object is deleted in the current repository, the default behavior for patch merges is to always ask the user whether the object should be discarded or retained. This is different from full merges, which often accept deletions from the current repository without prompting.

Use Patchrpd to Automate the Patch Process

You can use the -U and -V options in the `patchrpd` command-line utility to automate the patching process.

The `-U` option enables the patching process to complete by accepting default decisions for conflicts, while the `-V` option enables you to specify an output file for recording all merge conflicts so that you can examine them and possibly take action later.

Follow these guidelines to use `patchrpd` to automate the patch process:

- **Apply patching automatically using default rules.** Include the -U option in `patchrpd` to always apply the default decisions for conflicts. For example, if both repositories (current and modified) change the name of an object, the default decision is to keep the name in the modified repository, to avoid overwriting user customizations. If you don't include this parameter, `patchrpd` displays a warning and exits if a conflict is detected.
- **Record conflicts in an output decision file.** Include the -V option to cause `patchrpd` to generate a decision file that shows all the conflicts from the merge. The decision file lists the decisions that would have been displayed in the Define Merge Strategy screen of the Merge Wizard if the merge had been performed in the Administration Tool. The decision file provides a record of all items that can be influenced by user input.
- **Modify the decision file to change the result.** After you run `patchrpd`, there are two ways to make changes to the resulting repository:
  - You can use the **Load Decision File** button in the Define Merge Strategy screen of the Merge Wizard to load the merge decisions, and then change the decisions if needed. You can then complete the merge in the Administration Tool. Alternatively, you can save the modified decision file using the **Save Decisions to File** button, and then re-run `patchrpd` with the decision file as an input using the -D option to reapply the patch with the new decisions.
  - You can edit the decision file by hand, and then re-run `patchrpd` with the decision file as an input using the -D option to reapply the patch with the new decisions.

Follow these guidelines to set up `patchrpd` to match the Administration Tool's merge functionality:

- **Administration Tool's full merge -** Use the -A option, apply changes on the whole repository, and -M option, use *upgrade* mode.

- **Administration Tool's patch merge with the "use subset patching" option selected -** Don't include the -M option. Excluding the -M option means that the default merge mode is set to *patch*.
- **Administration Tool's patch merge with the "use subset patching" option not selected -** Use the -A option and don't include the -M option. Excluding the -M option means that the default merge mode is set to *patch*.

See Use patchrpd to Apply a Patch.

23
Delete Unwanted Objects from the Repository

You can learn how to use the command-line pruning utility, `prunerpd`, to delete unwanted objects in the Oracle BI repository.
You can only use `prunerpd` with binary repositories in Oracle BI repository file format.

This chapter contains the following topics:

- About the Object Pruning Utility
- Use the Object Pruning Utility
- Deletion Rules for the Object Pruning Utility

About the Object Pruning Utility

If you've a large number of extraneous or unwanted objects in your repository, you can delete the unwanted objects using the `prunerpd` command-line utility.

You can use `prunerpd` on both Windows and Linux systems.

You can delete unwanted repository objects such as databases, tables, columns, initialization blocks, and variables. The pruning utility doesn't remove objects from the Oracle BI Presentation Catalog.

Deleting objects from the repository has a cascading effect. For example, if a physical column is deleted, then any mapped logical columns are deleted, as well as any associated presentation columns, see Deletion Rules for the Object Pruning Utility.

Use the Object Pruning Utility

You must create the input file that contains the list of repository objects to delete, and then, run the object pruning utility at the command line, passing the input file as an argument.

This section contains the following topics:

- Create the Input File
- Run the prunerpd Utility

Create the Input File

The prune utility accepts the list of repository objects you want to delete as a text file.

The utility can accept multiple input files at a time. Object names in the input file must match the fully qualified name that's used in the repository. Wildcards such as "*" and "?" aren't supported in the object name. The syntax rules for the input file are shown in the table.

Object Type	Example	Action
Database	D "Paint"	Deletes the database named "Paint."

Object Type	Example	Action
Table	• T "W_AGREE_D" • T "DB"."Catalog"."Schema"."Table"	• Deletes the table or alias named "W_AGREE_D" from the Physical layer. • Deletes the table or alias named "Table" from the schema named "Schema," contained in the catalog named "Catalog," located in the database named "DB," from the Physical layer.
Column	C "W_AGREE_MD"."AGREE_CD"	Deletes the column named "AGREE_CD" located in a table or alias named "W_AGREE_D" from the Physical layer.
Initialization block	I "External Metadata Strings"	Deletes the initialization block named "External Metadata Strings."
Variable	V CURR_USER	Deletes the variable named "CURR_USER."

For example, a text file that contains instructions to delete a database named *Stock Quotes* and a physical column named **S_NQ_ACCT"."USER_NAME** would include the following entry:

```
D "Stock Quotes" C "S_NQ_ACCT"."USER_NAME"
```

Use white space as a delimiter in the input file: a single space, tab, or multiple spaces.

Run the prunerpd Utility

Learn how to use the `prunerpd` utility.

The location of the `prunerpd` utility is:

BI_DOMAIN`/bitools/bin`

Syntax

The `prunerpd` utility accepts the following parameters:

```
prunerpd -s source_rpd [-p rpd_password] -f input_file -o output_rpd -l
output_log_file -e error_log_file [-8]
```

Where:

source_rpd is the name and location of the target repository file.

rpd_password is the repository password for the source repository.
The password argument is optional. If you don't provide a password argument, you're prompted to enter a password when you run the command. To minimize the risk of security breaches, Oracle recommends that you don't provide a password argument either on the command line or in scripts. The password argument is supported for backward compatibility only, and is removed in a future release. For scripting purposes, you can send the password through standard input.

input_file is the input file name, in text format, that contains the list of repository objects to remove. Separate multiple file names by spaces. Enclose spaces within a filename with double quotes (" ").

output_rpd is the name and location of the output repository file, also known as the pruned repository.

output_log_file is the name and location of the output log file. All actions performed on the repository are written to this file, including descriptions. The output log file is in XML format. Other messages such as progress indicators are sent to the standard output stream.

error_log_file is the name and location of the error log file. The pruning utility writes exceptions and errors to this log. The error log file is in XML format. Other errors are sent to the standard output error stream.

`-8` specifies UTF-8 encoding.

Use `-H` or run `.sh` without any parameters to display the help content.

Example

```
prunerpd -s C:/OBI/Server/Repository/BIApps.rpd
-f "C:/Remove Oracle EBS Objects.txt"
-o "C:/OBI/Server/Repository/BIApps Pruned.rpd"
-l "C:/temp/BIApps Prunning.log" -e "C:/temp/ BIApps Prunning.err"
Give password: my_repos_password
```

Deletion Rules for the Object Pruning Utility

Deleting repository objects has a cascading effect.

24

Data Types Supported by Oracle Analytics Server

The topics list and describe the data types supported by Oracle Analytics Server, and contains information about the data type limitations, other Oracle BI Server limitations, and floating point limitations.

You can learn how to use the Oracle BI Server `nqcmd` utility to run the `NQSGetSQLDataTypes` procedure to obtain information about the data types.

When you import metadata from a data source into the repository's physical layer, each column is assigned a data type. The data type is associated with a specific storage format, constraints, and a valid range of values.

This chapter contains the following topics:

- Supported Data Types
- Use the NQSGetSQLDataTypes Procedure to Access Data Type Information
- Data Type Limitations
- Other Oracle BI Server Limitations
- Data Type Mapping in Oracle Database and Oracle Analytics Server

Supported Data Types

Learn about the data types by category, for example, numeric data and date data.

See Supported Base Data Types in *Administering Oracle Analytics Server*.

Textual Data

Oracle Analytics Server supports three character data types.

The textual data types are:

- CHAR
- LONGVARCHAR
- VARCHAR

Numeric Data

Learn about the supported numeric data types.

Oracle Analytics Server supports the following numeric data types:

- BIGINT
- DECIMAL

- DOUBLE
- FLOAT
- INTEGER
- NUMERIC
- REAL
- SMALLINT
- TINYINT

Date and Time Data

Oracle Analytics Server supports these data and time data types.

- DATE
- TIME
- TIMESTAMP

Binary Data

Oracle Analytics Server supports numerous binary data types.

The supported binary data types are:

- BIT
- BINARY
- LONGVARBINARY
- VARBINARY

Use the NQSGetSQLDataTypes Procedure to Access Data Type Information

To access a list of data types supported by Oracle Analytics Server, use the Oracle BI Server nqcmd utility to run the NQSGetSQLDataTypes procedure.

For example: `call NQSGetSQLDataTypes(0);`

When you run this procedure, the results contain a list of supported data types and information specific to each data type such as case sensitivity and the ability to search.

See Use nqcmd to Test and Refine the Repository.

Data Type Limitations

The table provides a description of the supported data types and their limitations.

An administrator or repository builder can use this information to evaluate whether a particular data type is suitable for a given column or set of values, and to determine whether the data type is capable of representing all the required values.

For example, the `INTEGER` column in the Oracle database supports a very large range of values, up to 38 decimal digits, but the `INTEGER` data type in Oracle Analytics Server is a 32-bit binary integer type that's capable of holding up to nine digits without encountering data overflow (truncation) issues. If the column holds values in the range of [-2,147,483,648, 2,147,483,647], then you should use the `INTEGER` data type. However, if the column stores values larger than this range, then you should use another data type such as `NUMERIC` or `VARCHAR`.

Choose the smallest, in bytes, data type that's capable of representing the column's expected range of values. Choosing a data type in this way reduces the amount of memory and disk space consumed by the Oracle BI Server for cache files, temp files, and so on.

Data Type	Limitations
BIG INT	JDBC and the Oracle BI Administration Tool don't support this type; therefore, Oracle Analytics Server doesn't fully support the BIG INT type. The BIG INT type is intended to be same as the C int64 data type.
BINARY	Oracle Analytics Server doesn't fully support the BINARY type. Oracle Analytics Server supports only the fetching of columns whose data type is BINARY. The BI Server doesn't support the BINARY type in bind parameters or insert statements.
BIT	Oracle Analytics Server doesn't fully support the BIT type. Instead, you should use either the INT or CHAR type to represent Boolean data.
CHAR	The CHAR type's values are always padded with ending spaces that can equal up to the length specified by the data type. The CHAR type supports Unicode values. On the Windows platform, the storage is two bytes per character. On all Linux 64-bit platforms, the storage is four bytes per character.
DATE	The DATE type represents only year, month, and day components. DATE type doesn't represent hours, minutes, or seconds like the Oracle DATE data type.
DECIMAL	The DECIMAL type is the same as the NUMERIC type.
DOUBLE	The DOUBLE type is the same as the IEEE 754 64-bit double-precision binary floating-point data type. The internal storage is eight bytes. The significand occupies 53 bits (including the sign bit). Therefore, the precision is limited to approximately 16 decimal digits. The exponent occupies 11 bits. The range of the exponent is approximately ±307 as a base 10 decimal value. See Floating Point Limitations.
INTEGER	The INTEGER type is a signed binary integer data type occupying four bytes. The maximum value that can be represented is 2,147,483,647, and the minimum value is -2,147,483,648.
FLOAT	The FLOAT type is the same as the IEEE 754 32-bit single-precision binary floating-point data type. The internal storage is four bytes. The significand occupies 24 bits (including the sign bit). Therefore, the precision is limited to approximately 7 decimal digits. The exponent occupies eight bits. The range of the exponent is approximately ±38 as a base 10 decimal value. See Floating Point Limitations.
LONGVARBINARY	The LONGVARBINARY type supports up to 32,678 bytes.

Data Type	Limitations
LONGVARCHAR	The LONGVARCHAR type supports up to 32,678 bytes. Both the LONGVARCHAR type and the VARCHAR type support Unicode values.
NUMERIC	The NUMERIC type is a true decimal data type occupying 22 bytes. The internal representation and limitations are the same as the Oracle NUMBER data type. The NUMERIC type supports positive numbers in the range of 1 x 10^-130 to 9.999...9 x 10^125 with up to 38 significant digits. The precision and scale aren't stored in the repository. The scale is assumed to be 10.
REAL	The REAL type has the same description and limitations as the FLOAT type.
SMALLINT	The SMALLINT type is represented as the INTEGER type internally in the BI Server and has the same limitations as the INTEGER data type.
TIME	The TIME type represents only hour, minute, and second components.
TIMESTAMP	The TIMESTAMP type represents year, month, day, hour, minute, and second components. For some data sources on some platforms, it can also support fractions of a second.
TINYINT	The TINYINT type is represented as an INTEGER internally in BI Server. The TINYINT type and INTEGER type have the same limitations.
VARBINARY	The VARBINARY type is interchangeable with the LONGVARBINARY type. The VARBINARY type and the LONGVARBINARY type have the same limitations.
VARCHAR	The VARCHAR type is interchangeable with the LONGVARCHAR type. The VARCHAR type and LONGCARCHAR type have the same limitations. The Oracle BI Administration Tool allows users to enter a maximum character length of 2,147,483,647. However, the actual maximum length supported is 32,678.

Floating Point Limitations

You can't represent some numbers exactly with binary floating point data types such as FLOAT and DOUBLE.

When converting decimal numbers to and from binary floating point representations, often there are rounding errors because of the representational limitations of binary floating point formats. For example, a decimal number such as 1.365 might be represented as 1.364999999999999 when converted to the DOUBLE type. When this number is rounded to 3 digits after the decimal point, the result is 1.365. However, if the number is rounded to 2 decimal digits, then the result is 1.36 and not 1.37.

Oracle BI Server supports the NUMERIC type for RDBMS and TimesTen data sources. To avoid the limitations of the FLOAT and DOUBLE types, Oracle suggests that you update the FLOAT and DOUBLE data types to the NUMERIC type. There is no workaround to fix the inherent limitations with binary floating point data types, other than switching to the NUMERIC data type.

Other Oracle BI Server Limitations

Learn about other data type limitations such as table name and column name length.

In addition to the data type limitations, Oracle BI Server has the following limitations:

- The default maximum length of all fields in Oracle BI Server is 32,678 bytes. This default limit can be changed by setting the environment variable `OBIS_MAX_FIELD_SIZE`.
- The default maximum length of all SQL identifiers, for example, table names and column names, is 128 characters.

See Data Type Limitations.

Data Type Mapping in Oracle Database and Oracle Analytics Server

When you import metadata from an Oracle database, the Oracle BI Administration Tool uses the mapping in the following table to determine the corresponding data type in the BI Server for each imported column.

The mapping of data types from the Oracle Database to BI Server might differ depending on the database.

Oracle Database Data Type	BI Server Data Type
CHAR	CHAR
NCHAR	CHAR
VARCHAR2	VARCHAR
NVARCHAR2	VARCHAR
NUMBER	NUMERIC if ENABLE_NUMERIC_DATA_TYPE = YES; otherwise, DOUBLE
NUMBER (precision, scale)	INT if scale = 0 and 1 <= precision <= 9; otherwise, same as NUMBER
BINARY_FLOAT	FLOAT
BINARY_DOUBLE	DOUBLE
DATE	DATETIME
TIMESTAMP	TIMESTAMP
TIMESTAMP WITH TIME ZONE	TIMESTAMP
TIMESTAMP WITH LOCAL TIME ZONE	TIMESTAMP
BLOB	LONGVARBINARY
CLOB	LONGVARCHAR
NCLOB	LONGVARCHAR
BFILE	Not supported
LONG	LONGVARCHAR

Oracle Database Data Type	BI Server Data Type
LONG RAW	Not supported
ROWID	CHAR
XML Type	LONGVARBINARY
UriType	Not supported

25

Exchange Metadata with Databases to Enhance Query Performance

This chapter explains how to use Oracle Database to enhance the data warehouse performance and functionality of queries that run on the Oracle BI Server.
This chapter contains the following topics:

- About Exchanging Metadata with Databases
- Generate the Import File
- Use Materialized Views in the Oracle Database with Oracle Analytics Server

About Exchanging Metadata with Databases

By exchanging Oracle Analytics Server metadata from the Oracle BI Server with your Oracle Database, you enable the database to accelerate the performance of data warehouse queries.

Use the Oracle BI Server `sametaexport` utility to exchange the metadata. When you run this utility to generate metadata for Oracle Database, the utility is called the Oracle Database Metadata Generator.

The Oracle BI Server export utility works with the following tools:

- In the Oracle Database, the SQL Access Advisor creates materialized views and index recommendations for optimizing performance.

The `sametaexport` utility generates the information necessary for the SQL Access Advisor tool to pre-aggregate the relational data and improve query performance.

Generate the Import File

The Oracle Database Metadata Generator creates the files needed to import metadata from the Oracle BI Server into the SQL Access Advisor.

This section contains the following topics that are common to the generator:

- Run the Generator
- About the Metadata Input File
- About the Output Files
- Troubleshoot Errors from the Generator
- Metadata Conversion Rules and Error Messages

Run the Generator

The Oracle Database Metadata Generator is invoked from the command line or embedded in a batch file.

The command-line executable is named `sametaexport`.

The `sametaexport` utility is available on both Windows and Linux systems. However, you can only use `sametaexport` with binary repositories in repository format.

The location of the `sametaexport` utility is:

ORACLE_HOME`/bi/bifoundation/server/bin`

```
sametaexport -r "PathAndRepositoryFileName" [-p repository_password]
-f "InputFileNameAndPath" [options]
```

The table contains descriptions of the parameters in the command-line executable file.

Parameter	Definition	Additional Information
-r	Repository file name and full path	Quotation marks are required for the file name and path only if the file path is in long format or has spaces. Use the full path if the file isn't in the current directory.
-p	Repository password	The password for the given repository. The password argument is optional. If you don't provide a password argument, you're prompted to enter a password when you run the command. To minimize the risk of security breaches, Oracle recommends that you don't provide a password argument either on the command line or in scripts. For scripting purposes, you can pass the password through standard input.
-f	Input file name and full path	Quotation marks are required for the file name and path only if the file path is in long format or has spaces. Use the full path if the file isn't in the current directory. You specify input files so that you don't have to type all the required information at the command line, and so that you can type international characters. See About the Metadata Input File.

You can include some additional parameters in the input file or at the command line to change various defaults for the Oracle Database Metadata Generator. Parameters specified in the input file take precedence over parameters specified at the command line. You must include these parameters only if you want to change the default values.

The tables describe these optional parameters.

Parameter Definition	Additional Information	Input File Usage Example	Command Line Usage Example
Use schema name from Oracle BI repository	When set to `YES`, the table schema names are used as they're used in the repository. The default value is `YES`.	`USE_SCHEMA_NAME_FROM_RPD = NO`	`-schemafrom rpd NO`

Parameter Definition	Additional Information	Input File Usage Example	Command Line Usage Example
Default schema name	The default schema name is used as the table schema name if the value of `-schemafromrpd` is set to `NO`, or if the repository schema name can't be determined. The default value is `SIEBEL`.	`DEFAULT_SCHEMA_NAME = ORACLE`	`-defaultschema ORACLE`
Oracle schema name	The metadata from Oracle Database Metadata Generator is created under this schema. The default value is `SIEBEL`.	`ORA_DIM_SCHEMA_NAME = ORACLE`	`-orclschema ORACLE`
Logging enabled	Indicates whether to keep a log of the metadata export process. Valid values are `ON`, `OFF`, and `DEBUG`. The default value is `ON`.	`LOGGING = DEBUG`	`-logging DEBUG`
Log file name	The path to the log file. If you provide an invalid path, an error occurs. If you don't provide this parameter, the default log file path is used. The default path is: *`ORACLE_INSTANCE`*`\diagnostics\logs\OracleBIServerComponent\coreapplication_obis`*`n`*`\OraDimExp.log`	`LOG_FILE_NAME = C:\bea_default\instances\instance1\diagnostics\logs\generator\logfile.log`	`-logfile C:\bea_default\instances\instance1\diagnostics\logs\generator\logfile.log`

Parameter Definition	Additional Information	Input File Usage Example	Command Line Usage Example
Distinct count supported	When set to `YES`, allows measure containing the `DISTINCT_COUNT` aggregation to be exported. The recommended setting and default value is `NO`.	`DISTINCT_COUNT_SUPPORTED = YES`	`-distinct YES`
Statistical functions supported	When set to `YES`, allows measures containing the aggregation `STDDEV` to be exported. The recommended setting and default value is `NO`.	`STATISTICAL_FUNCTIONS_SUPPORTED = YES`	`-stat YES`
Use schema name	When set to `YES`, the Cube Views metadata attributes have columns from tables under a schema name, which are then specified in the parameters. When set to `NO`, the schema names for these tables are empty. The default value is `YES`.	`USE_SCHEMA_NAME = NO`	`-useschema NO`
Use schema name from Oracle BI repository	When set to `YES`, the table schema names are used as they're used in the repository. The default value is `YES`.	`USE_SCHEMA_NAME_FROM_RPD = NO`	`-schemafromrpd NO`

Parameter Definition	Additional Information	Input File Usage Example	Command Line Usage Example
Default schema name	The default schema name is used as the table schema name if the value of `-schemafromrpd` is set to `NO`, or if the repository schema name can't be determined. The default value is `SIEBEL`.	`DEFAULT_SCHEMA_NAME = ORACLE`	`-defaultschema ORACLE`
Cube views schema name	The name of the schema under which the Cube Views metadata is created. The default value is `SIEBEL`.	`CUBE_VIEWS_SCHEMA_NAME = ORACLE`	`-cubeschema ORACLE`
Log file name	The path to the log file. If you provide an invalid path, an error occurs. If you don't provide this parameter, the default log file path is used. The default path is: *ORACLE_INSTANCE*`\diagnostics\logs\OracleBIServerComponent\coreapplication_obis`*n*`\CubeViews.log`	`LOG_FILE_NAME = C:\bea_default\instances\instance1\diagnostics\logs\generator\logfile.log`	`-logfile C:\bea_default\instances\instance1\diagnostics\logs\generator\logfile.log`
Log failures	When set to `YES`, the log file lists the metadata that was invalidated under a certain rule. The default value is `YES`.	`LOG_FAILURES = NO`	`-logfail NO`
Log success	When set to `YES`, the log file lists the metadata that's been checked under each rule and has passed the check. The default value is `NO`.	`LOG_SUCCESS = YES`	`-logsuccess YES`

About the Metadata Input File

The table describes the parameters in the metadata input file.

The input file is a text file that contains the parameters that are described in the following table.

Input File Name	Description
BUSINESS_MODEL	The name of the business model in the logical layer of the Oracle BI repository that contains the metadata to export. If the business model isn't found in the repository, then an error message is displayed. You can only specify one business model name in the input file. To generate metadata for multiple business models, create another input file and run the Oracle Database Metadata Generator again.

Input File Name	Description
PHYSICAL_DATABASE	The name of the database in the physical layer of the Oracle BI repository that contains the metadata to export. When the business model derives from multiple databases, then it eliminates metadata from all databases other than the one specified here. When the physical database isn't found in the repository, an error message is displayed.
RUN_AS_USER	The user name of the database user whose visibility must be duplicated for the metadata export. This parameter can't be empty. This user must exist as a user reference in the repository.
OUTPUT_FOLDER	The full path and file name of the folder where the SQL file placed. If the folder doesn't exist when you run the Oracle Database Metadata Generator, then the folder is created in the process. See About the Output Files.

The following text shows a sample metadata input file:

```
BUSINESS_MODEL = "1 - Sample App"
PHYSICAL_DATABASE = "1 - Sample App Data"
RUN_AS_USER = "Administrator"
OUTPUT_FOLDER = "C:\OracleBI"
```

About the Output Files

Each Generator creates different types of output files.

The following list describes the output files:

- The Oracle Database Metadata Generator creates a SQL file that's encoded in UTF-8 and stored in the specified output folder. The file name is based on the name of the business model you specified in the input file, such as *my_business_model.sql*.

Troubleshoot Errors from the Generator

Error messages indicate that the Generator was unable to complete some or all of its tasks.

After starting the Generator, you might observe the following error messages:

- `Unable to write to Log file: log_file_name.`

  The log file specified in the input file or at the command line might contain the wrong path, the user might not have write permissions to that folder, or the disk could be out-of-space.

- `Run_as_user, user_name, is invalid.`

  The user name is incorrect.

- `Repository, repository_name.rpd, is invalid or corrupt.`

  The repository name might be incorrect, it might not exist in the given path, or the user might not have permission to read it.

- `Physical Database, database_name, is invalid.`

  The physical database name doesn't match a valid physical database object in the repository.

- `Business Model, model_name, is invalid.`

  The business model name doesn't match a valid business model object in the repository.

- `Authentication information provided is invalid.`

  The repository password provided at the command line is incorrect.

- `Path: "path_name" is invalid.`

  The path or file name is incorrect, or the current user doesn't have read access.

Metadata Conversion Rules and Error Messages

When the Generator creates the output files, it also maps the metadata objects in the Oracle BI repository to similar objects in the metadata of the Oracle Database.

This section explains the rules used to identify metadata that can't be translated (converted) into either SQL or XML format. These rules are necessary because Oracle Database and IBM Cube Views don't support some of the metadata constructs that are allowed by Oracle Analytics Server.

Dimensional metadata in the SQL or XML file is generated at the logical fact table source level. If a logical fact table source has an invalid logical dimension table source, then the logical dimension table source is invalidated. If the logical fact table source is invalid, then all the logical dimension table sources that are mapped to it are also be invalidated. Invalid repository metadata elements aren't converted to cubes in the SQL or XML file.

When a rule is violated, the Generator writes the error messages and the metadata that violated the rule to the log file.

Conversion Rules for Oracle Databases

Learn the rules for converting Oracle BI metadata into objects.

The following list provides the rules for converting Oracle Analytics Server metadata into objects in the Oracle Database:

- Attributes that contain expressions in the logical table can't be exported.
- Tables joined using complex joins aren't considered.
- Tables that are opaque views aren't considered.
- Columns used as part of a key in one level can't be used as part of another level key.

  Oracle Database prohibits the use of columns as keys in multiple levels. This prohibition requires the Oracle Database Metadata Generator to eliminate one of the two joins, usually the join that's encountered first. Therefore, the other joins are lost, which prevents them from being exported.

Use Materialized Views in the Oracle Database with Oracle Analytics Server

Learn how to export metadata from Oracle Analytics Server into the SQL Access Advisor and create materialized views using the Oracle Database Metadata Generator.

This section contains the following topics:

- About Using the SQL Access Advisor with Materialized Views
- Deploy Metadata for Oracle Database

About Using the SQL Access Advisor with Materialized Views

You can use the SQL Access Advisor with Materialized Views to enhance the data warehouse performance and the functionality of a database.

The Oracle Database Metadata Generator enables the SQL Access Advisor to store metadata about the logical relationships of the data that resides in the database. Additionally, it accelerates data warehouse queries by using more efficient Oracle materialized views. These materialized views preaggregate the relational data and improve query performance. Once the metadata is stored in the SQL Access Advisor, the database administrator can optimize the database objects and improve query performance.

When processing queries, Oracle Database routes queries to tables that hold materialized views when possible. Because these tables of materialized views are smaller than the underlying base tables and the data has been pre aggregated, the queries that are rerouted to them might run faster.

Oracle Database Metadata Generator works as a metadata bridge to convert the proprietary metadata into a SQL file that contains PL/SQL commands to generate dimensions in the SQL Access Advisor. After converting metadata into a SQL file, you use a tool such as SQL*Plus to import the translated metadata into the SQL Access Advisor and store it in metadata catalog tables. After importing the metadata, you create materialized views, which are used to optimize incoming application queries.

Deploy Metadata for Oracle Database

Become familiar with the Oracle Database and its tools before attempting to deploy metadata in the Oracle Database.

See SQL Access Advisor in *Oracle Database Performance Tuning Guide*.

Ensure that you complete the steps in Run the Generator before deploying metadata. To deploy cube metadata, perform the tasks described in the following sections:

- Run the SQL File for Oracle Database
- Define Constraints for the Existence of Joins
- Create the Query Workload
- Create Materialized Views

Run the SQL File for Oracle Database

Before running the SQL file for importing into the SQL Access Advisor, ensure that you're familiar with Oracle Database import tools. See the Oracle Database documentation set for information.

Use a tool such as SQL*Plus to run the SQL file that the Oracle Database Metadata Generator generated. You might see error messages if the dimensions already exist or if the database schema differs from that in the Oracle BI repository file. When the script runs successfully, you can see the dimensions that were created by using the database web console or the Oracle Enterprise Manager Database Control. In the Oracle Enterprise Manager Database Control, expand the following nodes: Network, Databases, database-name, Warehouse, Summary Management, Dimensions, System.

After you run the SQL file, be aware of the following:

- No incremental metadata changes are allowed. Schema changes require that you manually delete cube model metadata in the Oracle Database and convert the metadata again. For example, if you must make a change to a dimension in a cube in the Oracle BI repository, you must delete the cube model in the Oracle Database, regenerate the SQL file from the Oracle BI repository, and import it into the SQL Access Advisor.
- You can't delete metadata using the Oracle Database Metadata Generator. Instead, you must manually delete the cube model using the Oracle Enterprise Manager Database Control.

Define Constraints for the Existence of Joins

You must ensure that Oracle Database knows about the joins between the dimension tables and the fact tables.

See your Oracle Database documentation.

To do so, you create constraints in SQL*Plus or the Oracle Enterprise Manager Database Control. In the Oracle Enterprise Manager Database Control, you select the table on which you must create a constraint, then select the Constraint tab.

You create a different type of constraint for each kind of table, as follows:

- For dimension tables, create a `UNIQUE` key constraint.
- For fact tables, create a `FOREIGN` key constraint and specify the referenced schema and referenced table. In the Constraint Definition area, include the foreign key columns in the fact table and the corresponding unique keys in the dimension table. An attempt to create a foreign key on a fact table can fail if the foreign key column data doesn't match the unique key column data on the dimension table.

Create the Query Workload

A query workload is a sample set of physical queries to optimize.

See the Oracle Database documentation set for detailed information about creating the query workload.

Before you create the workload, you generate a Trace file with information on the slowest-running queries.

You can generate the Trace file of the slowest-running queries using a tool that's appropriate to your database version, as described in the following list:

- **Usage Tracking**: Use this capability in Oracle Analytics Server to log queries and how long they take to run. Long-running queries can then be run as a script and used with the Trace feature in the Oracle Database to capture the Oracle Database SQL code for these queries.
- **Oracle Database Trace**: Use this tool to identify the slowest physical query. You can enable the Trace feature either within Oracle Enterprise Manager Database Control or by entering SQL commands with the `DBMS_MONITOR` package. Once you enable the Trace feature, you use a script to create a Trace file to capture the SQL code for queries in a query workload table.
- **Oracle Enterprise Manager**: Use this tool to track slow-running queries.

The capabilities that are described in the following sections are available in Oracle Database, rather than as part of Oracle Analytics Server.

1. Use the following guidelines when reviewing the Trace file:
   - When you've traced many statements at once, such as in batch processes, quickly discard any statements that have acceptable query processing times. Focus on those statements that take the longest times to run.
   - Check the Query column for block visits for read consistency, including all query and subquery processing. Inefficient statements are often associated with a large number of block visits. The Current column indicates visits not related to read consistency, including segment headers and blocks that are updated.
   - Check the Disk column for the number of blocks that were read from disk. Because disk reads are slower than memory reads, the value likely is significantly lower than the sum of the Query and Current columns. If it isn't, check for issues with the buffer cache.
   - Locking problems and inefficient PL/SQL loops can lead to high CPU time values even when the number of block visits is low.
   - Watch for multiple parse calls for a single statement, because this indicates a library cache issue.
2. After identifying the problem statements in the file, check the execution plan to learn why each problem statement occurred.
3. To load queries into the workload:
   a. After you use the Trace utility to learn the names of the slowest physical queries, insert them into the `USER_WORKLOAD` table.
   b. Use `INSERT` statements to populate the `QUERY` column with the SQL statements for the slowest physical queries and the `OWNER` column with the appropriate owner names.

User Workload

The table describes the columns of the `USER_WORKLOAD` table.

Column	Data Type	Required	Description
QUERY	Any LONG or VARCHAR type (all character types)	YES	SQL statement for the query.
OWNER	VARCHAR2 (30)	YES	User who last ran the query.
APPLICATION	VARCHAR2 (30)	NO	Application name for the query.
FREQUENCY	NUMBER	NO	Number of times that the query was run.
LASTUSE	DATE	NO	Last date on which the query was run.
PRIORITY	NUMBER	NO	User-supplied ranking of the query.
RESPONSETIME	NUMBER	NO	Processing time of the query in seconds.
RESULTSIZE	NUMBER	NO	Total number of bytes that the query selected.
SQL_ADDR	NUMBER	NO	Cache address of the query.
SQL_HASH	NUMBER	NO	Cache hash value of the query.

Create Materialized Views

After you populate the query workload table, use the appropriate tool for the Oracle Database version to create materialized views.

The SQL Access Advisor generates recommendations on improving the performance of the specified fact tables. The SQL Access Advisor can also help determine appropriate indexing schemes. The SQL Access Advisor displays the SQL code it uses to create the appropriate materialized views. Before enabling SQL Access Advisor to create the materialized views, review the following:

- The creation of a materialized view can fail if the SQL code includes a `CAST` statement.
- The `CREATE MATERIALIZED VIEW` statement mustn't specify the same query that you provided as a workload table. If the statement does specify the same query, then the materialized views can't reflect the true performance gain. However, if the query is run frequently, then creating a materialized view might still be worthwhile.
- Add a forward slash (/) to the end of the `CREATE MATERIALIZED VIEW` statement after the SQL statement. If the forward slash isn't included, the SQL*Plus worksheet doesn't recognize it as a valid statement.

26

XML Schema Files for ADF Mapping Customizations

You can review the app_segment_rule.xsd and mapping_rules.xsd XML schema files. XML files based on these XML validation schemas define mapping rules that control the physical to logical mapping behavior for ADF data source objects.
This chapter contains information about the following XML schema files:

- app_segment_rule.xsd XML Schema File
- app_segment_rules_*.xml Example
- mapping_rules.xsd XML Schema File
- mapping_rules_*.xml Example

app_segment_rule.xsd XML Schema File

The app_segment_rule.xsd XML schema file contains mapping rules that define the physical table to logical table mapping.

The following is an annotated version of the `app_segment_rule.xsd` XML schema file.

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:complexType name="alias_t">
  <xs:attribute name="aliasTableName" type="xs:string" use="required" />
 </xs:complexType>
 <xs:complexType name="aliasTableList_t">
  <xs:sequence>
   <xs:element name="AliasTable" type="alias_t" minOccurs="1" maxOccurs=
   "unbounded" />
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="subjectArea_t">
  <xs:attribute name="subjectAreaName" type="xs:string" use="required" />
  <xs:attribute name="table" type="xs:string" use="optional" />
 </xs:complexType>
 <xs:complexType name="subjectAreaList_t">
  <xs:sequence>
   <xs:element name="OTBISubjectArea" type="subjectArea_t" minOccurs="1"
   maxOccurs="unbounded" />
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="columnmapping_t">

<!-- voColumnName specifies the ADF view object column name -->

  <xs:attribute name="voColumnName" type="xs:string" use="required" />

<!-- logicalDimColumnName specifies the logical column name -->

  <xs:attribute name="logicalDimColumnName" type="xs:string" use="required" />
```

```
 </xs:complexType>
 <xs:complexType name="columnMappingsList_t">
  <xs:sequence>

<!-- ColumnMapping specifies the set of column mappings -->

   <xs:element name="ColumnMapping" type="columnmapping_t" minOccurs="1"
   maxOccurs="unbounded" />
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="tableName_t">
  <xs:attribute name="name" type="xs:string" />
 </xs:complexType>
 <xs:complexType name="relatedLogicalTablesList_t">
  <xs:sequence>
   <xs:element name="LogicalTable" type="tableName_t" minOccurs="1"
   maxOccurs="unbounded" />
  </xs:sequence>
 </xs:complexType>

<!-- Base VO to which DFF alias needs to be joined. -->

 <xs:complexType name="joinWith_t">
  <xs:attribute name="voName" type="xs:string" use="required"/>
 </xs:complexType>

<!-- Logical Dimension to which DFF alias needs to be mapped. -->

 <xs:complexType name="mapTo_t">
  <xs:attribute name="logTabName" type="xs:string" use="required"/>
 </xs:complexType>

<!-- Specifies DFF alias name. -->

 <xs:complexType name="createAlias_t">
  <xs:sequence>
   <xs:element name="JoinWith" type="joinWith_t" minOccurs="1"
   maxOccurs="unbounded"/>
   <xs:element name="MapTo" type="mapTo_t" minOccurs="1" maxOccurs="unbounded"/>
  </xs:sequence>

<!-- Specifies DFF alias name. -->

  <xs:attribute name="name" type="xs:string" use="required"/>
 </xs:complexType>

<!-- Specifies set of LTSes. -->

 <xs:complexType name="ltsObject_t">
  <xs:attribute name="ltsName" type="xs:string"/>
 </xs:complexType>
  <xs:complexType name="ltsObjects_t">
  <xs:sequence>
   <xs:element name="Lts" type="ltsObject_t" minOccurs="0"
maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="labelToDimMap_t">
  <xs:sequence>

<!-- For mapping the _ and _c suffixed value and value set columns in the code,
```

```
combine flattened fact view objectS corresponding to the segment mapped to the
logical table -->

   <xs:element name="FlattenedFlexVOColumnMappings" type="columnMappingsList_t"
   minOccurs="0" maxOccurs="1" />

<!-- TableSpecificColumnMappingOverrides is for future use -->

   <xs:element name="TableSpecificColumnMappingOverrides"
   type="columnMappingsList_t" minOccurs="0" maxOccurs="1" />

<!-- TableSpecificOTBISubjectAreas specifies the Oracle Transactional Business
Intelligence subjects areas. The unqualified segment Logical Tables Dim - GL
Segment 1 - 10 are dragged to these subject areas. For other use cases, specify
your own repository subject areas. -->

   <xs:element name="TableSpecificOTBISubjectAreas" type="subjectAreaList_t"
   minOccurs="0" maxOccurs="1" />

<!-- DFFBaseVOAliases specifies the alias view objects of the base view object
mapped to the same logical table. For each of these view objects, the
Administration Tool creates an alias of the DFF view object and joins it to the
base view object alias. -->

   <xs:element name="DFFBaseVOAliases" type="aliasTableList_t" minOccurs="0"
   maxOccurs="1" />
  </xs:sequence>

<!-- RelatedLogicalTables specifies the list of logical tables to which the DFF
view object must be mapped in addition to the logical table mentioned in the
logicalDimTableName attribute. -->

   <xs:element name="RelatedLogicalTables" type="relatedLogicalTablesList_t"
   minOccurs="0" maxOccurs="1" />

<!-- Use this tag to create an alias of the VO, join it with several other
VOs, and map to a specified set of logical tables-->
   <xs:element name="CreateAlias" type="createAlias_t"
   minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>

<!-- segmentLabelName is for future use -->

  <xs:attribute name="segmentLabelName" type="xs:string" use="optional" />

<!-- logicalDimTableName specifies the logical table to which the ADF view object
is mapped. ADF database properties contain the mapping between the ADF view object
("BIObject_'ADF VO Name'") and logical table mentioned here. -->

  <xs:attribute name="logicalDimTableName" type="xs:string" use="required" />

<!-- In role playing dimensions, the logical table specified in the
roleMasterLogicalTableName attribute is used as the master logical table. Whenever
the master logical table is mapped to a view object, the alias of this view object
is created and joined to the table referenced in the flattenedFlexVORoleAlias
attribute. The alias table name is "{ADF_VO_Name}_{Role_Playing_Logical_Table_
Name}". Logical columns are mapped to the columns of this alias table. -->

  <xs:attribute name="roleMasterLogicalTableName" type="xs:string"
  use="optional" />
```

```
<!-- flattenedFlexVORoleAlias specifies the flattened view object alias that
needs
to be joined with the role playing view object alias table. -->

  <xs:attribute name="flattenedFlexVORoleAlias" type="xs:string"
use="optional" />
 </xs:complexType>

<!-- The root element for the mapping rule document -->

 <xs:complexType name="mappingRules_t">
  <xs:sequence>

<!-- List of LTSes to be Disabled -->
   <xs:element name="LTSToBeDisabled" type="ltsObjects_t" minOccurs="0"
maxOccurs="1"/>

<!-- GlobalColumnMappings is for future use -->

   <xs:element name="GlobalColumnMappings" type="columnMappingsList_t"
   minOccurs="0" maxOccurs="1" />

<!-- GlobalOTBISubjectAreas specifies the Oracle Transactional Business
Intelligence subject areas. The unqualified segment Logical Tables Dim - GL
Segment 1 - 10 are dragged to these subject areas. The rules specified here apply
to all logical tables. For other use cases, specify your own subject areas. -->

   <xs:element name="GlobalOTBISubjectAreas" type="subjectAreaList_t"
   minOccurs="0" maxOccurs="1" />

<!-- LabelToDimensionMappings specifies the rules for a logical table -->

   <xs:element name="LabelToDimensionMappings" type="labelToDimMap_t"
   minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>

<!-- appName is the ADF Module Name from where the ADF view objects are
imported. It is also used as a key for reading the properties from the ADF
database. It is
sometimes known as the Category or Module Name. -->

  <xs:attribute name="appName" type="xs:string" use="required" />

<!-- businessModelName specifies the business model in the Business Model and
Mapping layer. The ADF view objects are mapped to the logical tables in this
business model. The default business model is "Core." -->

  <xs:attribute name="businessModelName" type="xs:string" default="Core"
   use="optional" />

<!-- flattenedFlexLogicalTable specifies the logical table to which the flattened
fact view object for the KFF needs to map -->

  <xs:attribute name="flattenedFlexLogicalTable" type="xs:string"
  use="optional" />

<!-- Determines whether VOs under this AM needs to be submitted for
ETL extension -->

  <xs:attribute name="isETLSupported" type="xs:boolean" use="optional"/>
```

```
<!-- Specifies the unqualified Dimension Prefix for the AM -->

  <xs:attribute name="unqualifiedDimensionPrefix" type="xs:string"
  use="optional"/>

 </xs:complexType>

<!-- This is the actual element declaration for the entire Document. -->

 <xs:element name="document">
  <xs:complexType>
   <xs:sequence>

<!--  MappingRules specifies the set of rules. -->

    <xs:element name="MappingRules" type="mappingRules_t" minOccurs="0"
    maxOccurs="unbounded" />
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

app_segment_rules_*.xml Example

You can review an example of an app_segment_rules_*.xml file, based on the validation rules contained in app_segment_rules.xsd.

The app_segment_rules_*.xml file has been shortened for this example.

```
<?xml version="1.0" encoding="UTF-8" ?>
<document>
<!--  Begin Finance -->
<MappingRules appName="FscmTopModelAM.AccountBIAM" businessModelName="Core"
flattenedFlexLogicalTable="Dim - GL Account">
 <GlobalOTBISubjectAreas>
  <OTBISubjectArea subjectAreaName="General Ledger - Journals Real Time" />
  <OTBISubjectArea subjectAreaName="General Ledger - Transactional Balances Real Time" />
  <OTBISubjectArea subjectAreaName="Payables Invoices - Prepayment Invoice Distributions Real
  Time" />
  <OTBISubjectArea subjectAreaName="Payables Invoices - Transactions Real Time" />
  <OTBISubjectArea subjectAreaName="Payables Invoices - Trial Balance Real Time" />
...
 </GlobalOTBISubjectAreas>
 <LabelToDimensionMappings logicalDimTableName="Dim - Balancing Segment">
  <FlattenedFlexVOColumnMappings>
   <ColumnMapping voColumnName="_" logicalDimColumnName="Balancing Segment Code" />
   <ColumnMapping voColumnName="_c" logicalDimColumnName="Balancing Segment Value Set
   Code" />
  </FlattenedFlexVOColumnMappings>
  <TableSpecificOTBISubjectAreas>
   <OTBISubjectArea subjectAreaName="General Ledger - Journals Real Time" table="- Balancing
   Segment" />
   <OTBISubjectArea subjectAreaName="General Ledger - Transactional Balances Real Time"
   table="Balancing Segment" />
   <OTBISubjectArea subjectAreaName="Payables Invoices - Prepayment Invoice Distributions
   Real Time" table="- Balancing Segment" />
   <OTBISubjectArea subjectAreaName="Payables Invoices - Transactions Real Time" table=
   "- Balancing Segment" />
   <OTBISubjectArea subjectAreaName="Payables Invoices - Trial Balance Real Time"
   table="Balancing Segment" />
...
  </TableSpecificOTBISubjectAreas>
 </LabelToDimensionMappings>
 <LabelToDimensionMappings logicalDimTableName="Dim - Cost Center">
```

```
  <FlattenedFlexVOColumnMappings>
   <ColumnMapping voColumnName="_" logicalDimColumnName="Cost Center Segment Code" />
   <ColumnMapping voColumnName="_c" logicalDimColumnName="Cost Center Segment Value Set
Code"
   />
  </FlattenedFlexVOColumnMappings>
  <TableSpecificOTBISubjectAreas>
   <OTBISubjectArea subjectAreaName="General Ledger - Journals Real Time" table="- Cost
   Center Segment" />
   <OTBISubjectArea subjectAreaName="General Ledger - Transactional Balances Real Time"
   table="Cost Center Segment" />
   <OTBISubjectArea subjectAreaName="Payables Invoices - Prepayment Invoice Distributions
   Real Time" table="- Cost Center" />
   <OTBISubjectArea subjectAreaName="Payables Invoices - Transactions Real Time" table="-
   Cost Center" />
   <OTBISubjectArea subjectAreaName="Payables Invoices - Trial Balance Real Time"
table="Cost
 Center" />
...
  </TableSpecificOTBISubjectAreas>
 </LabelToDimensionMappings>
 <LabelToDimensionMappings logicalDimTableName="Dim - Natural Account Segment">
  <FlattenedFlexVOColumnMappings>
   <ColumnMapping voColumnName="_" logicalDimColumnName="Natural Account Segment Code" />
   <ColumnMapping voColumnName="_c" logicalDimColumnName="Natural Account Segment Value Set
   Code" />
  </FlattenedFlexVOColumnMappings>
  <TableSpecificOTBISubjectAreas>
   <OTBISubjectArea subjectAreaName="General Ledger - Journals Real Time" table="- Natural
   Account Segment" />
   <OTBISubjectArea subjectAreaName="General Ledger - Transactional Balances Real Time"
   table="Natural Account Segment" />
   <OTBISubjectArea subjectAreaName="Payables Invoices - Prepayment Invoice Distributions
   Real Time" table="- Natural Account" />
   <OTBISubjectArea subjectAreaName="Payables Invoices - Transactions Real Time" table=
   "- Natural Account" />
   <OTBISubjectArea subjectAreaName="Payables Invoices - Trial Balance Real Time"
   table="Natural Account" />
...
  </TableSpecificOTBISubjectAreas>
 </LabelToDimensionMappings>
 <LabelToDimensionMappings logicalDimTableName="Dim - GL Segment1">
  <FlattenedFlexVOColumnMappings>
   <ColumnMapping voColumnName="_" logicalDimColumnName="Account Segment1 Code" />
   <ColumnMapping voColumnName="_c" logicalDimColumnName="Account Segment1 Value Set
Code" />
  </FlattenedFlexVOColumnMappings>
 </LabelToDimensionMappings>
 <LabelToDimensionMappings logicalDimTableName="Dim - GL Segment2">
  <FlattenedFlexVOColumnMappings>
   <ColumnMapping voColumnName="_" logicalDimColumnName="Account Segment2 Code" />
   <ColumnMapping voColumnName="_c" logicalDimColumnName="Account Segment2 Value Set
Code" />
  </FlattenedFlexVOColumnMappings>
 </LabelToDimensionMappings>
...
 <LabelToDimensionMappings logicalDimTableName="Dim - AP Account Balancing Segment"
 roleMasterLogicalTableName="Dim - Balancing Segment"
flattenedFlexVORoleAlias="Dim_FLEX_BI_
 Account_VI_APAccount">
  <TableSpecificOTBISubjectAreas>
   <OTBISubjectArea subjectAreaName="Payables Invoices - Prepayment Invoice Distributions
   Real Time" table="- Distribution Balancing Segment Value" />
   <OTBISubjectArea subjectAreaName="Payables Invoices - Transactions Real Time" table="-
   Distribution Balancing Segment Value" />
  </TableSpecificOTBISubjectAreas>
 </LabelToDimensionMappings>
 <LabelToDimensionMappings logicalDimTableName="Dim - AP Account Cost Center"
 roleMasterLogicalTableName="Dim - Cost Center" flattenedFlexVORoleAlias="Dim_FLEX_BI_
```

```
 Account_VI_APAccount">
  <TableSpecificOTBISubjectAreas>
   <OTBISubjectArea subjectAreaName="Payables Invoices - Prepayment Invoice Distributions
   Real Time" table="- Distribution Cost Center Segment Value" />
   <OTBISubjectArea subjectAreaName="Payables Invoices - Transactions Real Time" table="-
   Distribution Cost Center Segment Value" />
  </TableSpecificOTBISubjectAreas>
 </LabelToDimensionMappings>
...
</MappingRules>
<MappingRules appName="FscmTopModelAM.LocationBIAM" businessModelName="Core"
flattenedFlexLogicalTable="Dim - Asset Location">
 <LabelToDimensionMappings logicalDimTableName="Segment1">
  <FlattenedFlexVOColumnMappings>
   <ColumnMapping voColumnName="_" logicalDimColumnName="Segment1" />
  </FlattenedFlexVOColumnMappings>
 </LabelToDimensionMappings>
 <LabelToDimensionMappings logicalDimTableName="Segment2">
  <FlattenedFlexVOColumnMappings>
   <ColumnMapping voColumnName="_" logicalDimColumnName="Segment2" />
  </FlattenedFlexVOColumnMappings>
 </LabelToDimensionMappings>
...
</MappingRules>
<MappingRules appName="FscmTopModelAM.CategoryBIAM" businessModelName="Core"
flattenedFlexLogicalTable="Dim - Asset Category">
 <LabelToDimensionMappings logicalDimTableName="Segment1">
  <FlattenedFlexVOColumnMappings>
   <ColumnMapping voColumnName="_" logicalDimColumnName="Segment1" />
  </FlattenedFlexVOColumnMappings>
 </LabelToDimensionMappings>
 <LabelToDimensionMappings logicalDimTableName="Segment2">
  <FlattenedFlexVOColumnMappings>
   <ColumnMapping voColumnName="_" logicalDimColumnName="Segment2" />
  </FlattenedFlexVOColumnMappings>
 </LabelToDimensionMappings>
...
 <LabelToDimensionMappings logicalDimTableName="Major Category">
  <FlattenedFlexVOColumnMappings>
   <ColumnMapping voColumnName="_" logicalDimColumnName="Major Category" />
  </FlattenedFlexVOColumnMappings>
 </LabelToDimensionMappings>
 <LabelToDimensionMappings logicalDimTableName="Minor Category">
  <FlattenedFlexVOColumnMappings>
   <ColumnMapping voColumnName="_" logicalDimColumnName="Minor Category" />
  </FlattenedFlexVOColumnMappings>
 </LabelToDimensionMappings>
</MappingRules>
...
<MappingRules appName="FscmTopModelAM.ExternalTransactionBIAM" businessModelName="Core"
flattenedFlexLogicalTable="">
<!-- Use the below tag if you want any one of the following use cases for the "base logical
table" your DFF is mapped to in ATG -->
<!-- (a) need to expose the base logical table DFF attributes to specific presentation
subject area and table -->
<!-- (b) you have additional VO aliases mapped to the base logical table in addition to the
base VO itself -->
<!-- (c) you have additional "related logical tables" that you want automapped when the base
logical table is mapped in ATG -->
 <LabelToDimensionMappings logicalDimTableName="Dim - CE External Cash Transaction Details">
<!-- Use the below tag to indicate the presentation subject area and table where the DFF
attributes need to be exposed -->
<!-- Repeat the OTBISubjectArea tag to expose them in multiple subject areas or tables -->
  <TableSpecificOTBISubjectAreas>
   <OTBISubjectArea subjectAreaName="Cash Management - External Cash Transactions Real Time"
   table="External Cash Transaction Detail" />
  </TableSpecificOTBISubjectAreas>
<!-- Use the below tag to give the VO alias names, if any, that are mapped to the base
logical table in addition to the base VO -->
```

```
<!-- Repeat the AliasTable tag if you have multiple VO aliases -->
  <DFFBaseVOAliases>
   <AliasTable aliasTableName="Fact_ExternalTransactionsPVO_JournalEntryLine" />
  </DFFBaseVOAliases>
 </LabelToDimensionMappings>
</MappingRules>
...
<MappingRules appName="FscmTopModelAM.CalendarTypeDFFBIAM" businessModelName="Core"
flattenedFlexLogicalTable="">
...
 <LabelToDimensionMappings logicalDimTableName="Dim - Date Fixed Assets Calendar">
...
  <TableSpecificOTBISubjectAreas>
   <OTBISubjectArea subjectAreaName="Fixed Assets - Asset Transactions Real Time"
   table="Transaction Effective Date" />
  </TableSpecificOTBISubjectAreas>
 </LabelToDimensionMappings>
<!-- Use the below tag if you have any "role-playing logical tables" for the base logical
table. -->
 <LabelToDimensionMappings logicalDimTableName="Dim - Date Placed in Service Fixed Asset
 Calendar" roleMasterLogicalTableName="Dim - Date Fixed Assets Calendar"
 flattenedFlexVORoleAlias="Dim_CalendarDayPVO_DatePlacedInService">
<!-- Use the below tag to indicate the presentation subject area and table where the DFF
attributes from the role-playig logical table need to be exposed -->
  <TableSpecificOTBISubjectAreas>
   <OTBISubjectArea subjectAreaName="Fixed Assets - Asset Financial Information Real Time"
   table="Date Placed in Service" />
  </TableSpecificOTBISubjectAreas>
 </LabelToDimensionMappings>
 <LabelToDimensionMappings logicalDimTableName="Dim - Transaction Date Fixed Asset
Calendar"
 roleMasterLogicalTableName="Dim - Date Fixed Assets Calendar"
flattenedFlexVORoleAlias="Dim_
 CalendarDayPVO_TransactionDate">
<!-- Use the below tag to indicate the presentation subject area and table where the DFF
attributes from the role-playing logical table need to be exposed -->
  <TableSpecificOTBISubjectAreas>
   <OTBISubjectArea subjectAreaName="Fixed Assets - Asset Assignments Real Time"
   table="Transaction Date" />
   <OTBISubjectArea subjectAreaName="Fixed Assets - Asset Financial Information Real Time"
   table="Transaction Date" />
   <OTBISubjectArea subjectAreaName="Fixed Assets - Asset Retirements and Reinstatements
Real
   Time" table="Transaction Date" />
   <OTBISubjectArea subjectAreaName="Fixed Assets - Asset Transactions Real Time"
   table="Transaction Date" />
   <OTBISubjectArea subjectAreaName="Fixed Assets - Asset Transfer Real Time"
   table="Transaction Date" />
  </TableSpecificOTBISubjectAreas>
 </LabelToDimensionMappings>
...
</MappingRules>
...
<!-- Rule file for AP_TERMS_B -->
<MappingRules appName="FscmTopModelAM.PaymentTermHeaderDffBIAM" businessModelName="Core"
flattenedFlexLogicalTable="">
 <LabelToDimensionMappings logicalDimTableName="Dim - AP Terms">
  <TableSpecificOTBISubjectAreas>
   <OTBISubjectArea subjectAreaName="Payables Invoices - Transactions Real Time" table=
   "- Header Information" />
  </TableSpecificOTBISubjectAreas>
 </LabelToDimensionMappings>
</MappingRules>
<!-- Rule file for AP_PAYMENT_SCHEDULES -->
<MappingRules appName="FscmTopModelAM.InstallmentsDffBIAM" businessModelName="Core"
flattenedFlexLogicalTable="">
 <LabelToDimensionMappings logicalDimTableName="Dim - AP Payment Schedule Details">
  <TableSpecificOTBISubjectAreas>
   <OTBISubjectArea subjectAreaName="Payables Invoices - Installments Real Time"
```

```
   table="Invoices Installment Details" />
  </TableSpecificOTBISubjectAreas>
  <DFFBaseVOAliases>
   <AliasTable aliasTableName="Fact_InvoicePaymentSchedulePVO_Disbursement" />
  </DFFBaseVOAliases>
 </LabelToDimensionMappings>
</MappingRules>
...
<!-- Rule file for DFF AP_HOLDS -->
<MappingRules appName="FscmTopModelAM.HoldsDffBIAM" businessModelName="Core"
flattenedFlexLogicalTable="">
 <LabelToDimensionMappings logicalDimTableName="Dim - AP Hold Details">
  <TableSpecificOTBISubjectAreas>
   <OTBISubjectArea subjectAreaName="Payables Invoices - Holds Real Time" table="Invoices
   Hold Details" />
  </TableSpecificOTBISubjectAreas>
 </LabelToDimensionMappings>
</MappingRules>
<!-- End Finance -->

<!-- Rule file for JobAM EFF -->
</MappingRules>
<MappingRules appName="HcmTopModelAnalyticsGlobalAM.JobAM.PER_JOBS_LEG_EFF"
businessModelName="Core" flattenedFlexLogicalTable="">
 <LabelToDimensionMappings logicalDimTableName="Dim - Job">
  <TableSpecificOTBISubjectAreas>
   <OTBISubjectArea subjectAreaName="Workforce Management - Worker Assignment Real Time"
table="Job" />
  </TableSpecificOTBISubjectAreas>
 </LabelToDimensionMappings>
</MappingRules>
...
</document>
```

mapping_rules.xsd XML Schema File

The XML schema file mapping_rules.xsd contains mapping rules that define the physical column to logical column mapping.

This section provides an annotated version of the `mapping_rules.xsd` XML schema file.

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<!-- This XML validation schema defines data extensibility rules that determine
how objects imported at the Physical layer are mapped to the Business
Model and Mapping layer. -->

 <xs:simpleType name="yes_no_t">
  <xs:restriction base="xs:string">
   <xs:enumeration value="yes" />
   <xs:enumeration value="no" />
  </xs:restriction>
 </xs:simpleType>

<!-- Description of one mapping -->

 <xs:complexType name="mapping_t">

<!-- The source attribute specifies the physical ADF view object column -->

  <xs:attribute name="source" type="xs:string" use="required" />

<!-- The target attribute specifies the target logical column suffix or complete
```

```
name -->

  <xs:attribute name="target" type="xs:string" use="required" />

<!-- The addprefix attribute specifies whether or not the prefix specified in the
"rule" must be prefixed with the target to get the logical table column name
before performing the mapping -->

  <xs:attribute name="addprefix" type="yes_no_t" default="no" use="optional" />
 </xs:complexType>
 <xs:complexType name="rule_t">
  <xs:sequence>

<!-- List of mappings that define the rule -->

   <xs:element name="mapping" type="mapping_t" minOccurs="1"
maxOccurs="unbounded"
   />
  </xs:sequence>

<!-- The name attribute specifies the rule name. This name is used by the
instance
to specify which rule to apply. -->

  <xs:attribute name="name" type="xs:string" use="required" />
 </xs:complexType>
 <xs:complexType name="rules_t">
  <xs:sequence>

<!-- Each rule defines the source (physical) to target (logical) mapping. It
is used by the Import Metadata Wizard during the Physical to Logical Mapping
phase. Each mapping in a rule can specify whether the default prefix for an
instance should be added. -->

   <xs:element name="rule" type="rule_t" minOccurs="1" maxOccurs="unbounded" />
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="apply_t">

<!-- The rule attribute specifies which rule to apply for the logical table
mentioned in the instance. -->

  <xs:attribute name="rule" type="xs:string" use="required" />

<!-- The prefix attribute specifies the string that might or might not be
prepended when performing the mapping between the ADF view object column and the
logical table column. -->

  <xs:attribute name="prefix" type="xs:string" default="" use="optional" />
 </xs:complexType>
 <xs:complexType name="instance_t">
  <xs:sequence>

<!-- apply specifies what rule to apply to the logical table -->

   <xs:element name="apply" type="apply_t" minOccurs="1"
maxOccurs="unbounded" />
  </xs:sequence>

<!-- The logicaltable attribute specifies the logical table to which the rule is
applied -->
```

```
  <xs:attribute name="logicaltable" type="xs:string" use="required" />
 </xs:complexType>
 <xs:complexType name="instances_t">
  <xs:sequence>

<!-- Each instance is associated with a single logical table. It applies any
number of rules, with an optional prefix. The rules are applied in order. Note
that the prefix includes a space at the end. It merely concatenates with the
target logical column to determine the right logical column to map to. -->

   <xs:element name="instance" type="instance_t" minOccurs="1"
maxOccurs="unbounded" />
  </xs:sequence>

<!-- appName is the ADF Module Name from which ADF view objects are imported -->

   <xs:attribute name="appName" type="xs:string" />
 </xs:complexType>

<!-- The document element is the element declaration for the entire document -->

 <xs:element name="document">
  <xs:complexType>
   <xs:sequence>

<!-- The rules element specifies the set of rules -->

    <xs:element name="rules" type="rules_t" minOccurs="1" maxOccurs="1" />

<!-- The instances element specifies the set of instances related to a single ADF
module. -->

    <xs:element name="instances" type="instances_t" minOccurs="1" maxOccurs="1" />
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

mapping_rules_*.xml Example

Review the example of a mapping_rules_*.xml file based on the validation rules contained in mapping_rules.xsd.

The actual file has been shortened.

```
<?xml version="1.0" encoding="UTF-8" ?>
<document>
<rules>

<!-- Each Rule defines the source (physical) to target (logical) mapping. It is
used by the Import Metadata Wizard during the Physical to Logical Mapping phase.
Each mapping in a rule can specify whether it should add the default prefix for an
instance. -->

 <rule name="GL Segment Tree Rule">
  <mapping source="TreeCode" target="Tree Code" addprefix="no" />
  <mapping source="TreeVersionId" target="Tree Version ID" addprefix="no" />
  <mapping source="VersionName" target="Tree Version Name" addprefix="no" />
  <mapping source="Dep0Pk2Value" target="Segment Value Set Code" addprefix="yes" />
  <mapping source="Dep0Value" target="Segment Code" addprefix="yes" />
  <mapping source="Dep0Description" target="Segment Description" addprefix="yes" />
```

```
  <mapping source="Dep1Value" target="Level 1 Code" addprefix="yes" />
  <mapping source="Dep1Description" target="Level 1 Description" addprefix="yes" />
  <mapping source="Dep2Value" target="Level 2 Code" addprefix="yes" />
  <mapping source="Dep2Description" target="Level 2 Description" addprefix="yes" />
  <mapping source="Dep3Value" target="Level 3 Code" addprefix="yes" />
  <mapping source="Dep3Description" target="Level 3 Description" addprefix="yes" />
...
  <mapping source="Distance" target="Fixed Hierarchy Level" addprefix="no" />
  <mapping source="VersionEffectiveStartDate" target="Effective From" addprefix="no" />
  <mapping source="VersionEffectiveEndDate" target="Effective To" addprefix="no" />
 </rule>
 <rule name="GL Segment Non-Tree Rule">
  <mapping source="Value" target="Segment Code" addprefix="yes" />
  <mapping source="ValueSetCode" target="Segment Value Set Code" addprefix="yes" />
  <mapping source="Description" target="Segment Description" addprefix="yes" />
 </rule>
 <rule name="Qualified Segment Tree Rule">
  <mapping source="TreeCode" target="Tree Code" addprefix="no" />
  <mapping source="TreeVersionId" target="Tree Version ID" addprefix="no" />
  <mapping source="VersionName" target="Tree Version Name" addprefix="no" />
  <mapping source="Dep0Pk2Value" target="Value Set Code" addprefix="yes" />
  <mapping source="Dep0Value" target="Code" addprefix="yes" />
  <mapping source="Dep0Description" target="Description" addprefix="yes" />
  <mapping source="Dep1Value" target="Level 1 Code" addprefix="yes" />
  <mapping source="Dep1Description" target="Level 1 Description" addprefix="yes" />
  <mapping source="Dep2Value" target="Level 2 Code" addprefix="yes" />
  <mapping source="Dep2Description" target="Level 2 Description" addprefix="yes" />
  <mapping source="Dep3Value" target="Level 3 Code" addprefix="yes" />
  <mapping source="Dep3Description" target="Level 3 Description" addprefix="yes" />
...
  <mapping source="Distance" target="Fixed Hierarchy Level" addprefix="no" />
  <mapping source="VersionEffectiveStartDate" target="Effective From" addprefix="no" />
  <mapping source="VersionEffectiveEndDate" target="Effective To" addprefix="no" />
  <mapping source="Dep0FlexValueAttribute6" target="Group Account Number"
addprefix="no" />
 </rule>
 <rule name="Qualified Segment Non-Tree Rule">
  <mapping source="Value" target="Code" addprefix="yes" />
  <mapping source="ValueSetCode" target="Value Set Code" addprefix="yes" />
  <mapping source="Description" target="Description" addprefix="yes" />
  <mapping source="FlexValueAttribute6" target="Group Account Number" addprefix="no" />
 </rule>
 <rule name="Role Playing KFF Tree Rule 1">
  <mapping source="TreeCode" target="Tree Code" addprefix="no" />
  <mapping source="TreeVersionId" target="Tree Version ID" addprefix="no" />
  <mapping source="Dep0Description" target="Description" addprefix="yes" />
  <mapping source="Dep0Pk2Value" target="Value Set Code" addprefix="yes" />
  <mapping source="Dep0Value" target="Code" addprefix="yes" />
  <mapping source="Value" target="Code" addprefix="yes" />
  <mapping source="ValueSetCode" target="Value Set Code" addprefix="yes" />
  <mapping source="Description" target="Description" addprefix="yes" />
 </rule>
 <rule name="Role Playing KFF Tree Rule 2">
  <mapping source="TreeCode" target="Tree Code" addprefix="no" />
  <mapping source="TreeVersionId" target="Tree Version ID" addprefix="no" />
  <mapping source="Dep0Description" target="Desscription" addprefix="yes" />
  <mapping source="Dep0Pk2Value" target="Value Set Code" addprefix="yes" />
  <mapping source="Dep0Value" target="Code" addprefix="yes" />
  <mapping source="Value" target="Code" addprefix="yes" />
  <mapping source="ValueSetCode" target="Value Set Code" addprefix="yes" />
  <mapping source="Description" target="Desscription" addprefix="yes" />
 </rule>
...
</rules>
<instances appName="FscmTopModelAM.AccountBIAM">

<!-- Each instance is associated with the a single logical table. It applies any number of
rules, with an optional prefix. The rules are applied in order. Note that the prefix
includes
```

```
a space at the end. It merely concatenates with the target logical column to determine the
right logical column to map to. -->

 <instance logicaltable="Dim - Balancing Segment">
  <apply rule="Qualified Segment Tree Rule" prefix="Balancing Segment" />
  <apply rule="Qualified Segment Non-Tree Rule" prefix="Balancing Segment" />
 </instance>
 <instance logicaltable="Dim - Cost Center">
  <apply rule="Qualified Segment Tree Rule" prefix="Cost Center" />
  <apply rule="Qualified Segment Non-Tree Rule" prefix="Cost Center" />
 </instance>
 <instance logicaltable="Dim - Natural Account Segment">
  <apply rule="Qualified Segment Tree Rule" prefix="Account" />
  <apply rule="Qualified Segment Non-Tree Rule" prefix="Account" />
 </instance>
 <instance logicaltable="Dim - GL Segment1">
  <apply rule="GL Segment Tree Rule" />
  <apply rule="GL Segment Non-Tree Rule" />
 </instance>
 <instance logicaltable="Dim - GL Segment2">
  <apply rule="GL Segment Tree Rule" />
  <apply rule="GL Segment Non-Tree Rule" />
 </instance>
 <instance logicaltable="Dim - GL Segment3">
  <apply rule="GL Segment Tree Rule" />
  <apply rule="GL Segment Non-Tree Rule" />
 </instance>
...
 <instance logicaltable="Dim - AP Account Balancing Segment">
  <apply rule="Role Playing KFF Tree Rule 1" prefix="Balancing Segment" />
 </instance>
 <instance logicaltable="Dim - AP Asset Account Balancing Segment">
  <apply rule="Role Playing KFF Tree Rule 1" prefix="Balancing Segment" />
 </instance>
 <instance logicaltable="Dim - AP Pay Account Balancing Segment">
  <apply rule="Role Playing KFF Tree Rule 1" prefix="Balancing Segment" />
 </instance>
 <instance logicaltable="Dim - Asset From Balancing Segment">
  <apply rule="Role Playing KFF Tree Rule 3" prefix="Asset From Balancing Segment" />
 </instance>
 <instance logicaltable="Dim - Asset To Balancing Segment">
  <apply rule="Role Playing KFF Tree Rule 3" prefix="Asset To Balancing Segment" />
 </instance>
 <instance logicaltable="Dim - Asset Balancing Segment">
  <apply rule="Role Playing KFF Tree Rule 3" prefix="Asset Balancing Segment" />
 </instance>
 <instance logicaltable="Dim - Distribution Balancing Segment">
  <apply rule="Role Playing KFF Tree Rule 3" prefix="Distribution Balancing Segment" />
 </instance>
 <instance logicaltable="Dim - SLA Balancing Segment">
  <apply rule="Role Playing KFF Tree Rule 3" prefix="Balancing Segment" />
 </instance>
 <instance logicaltable="Dim - CE Asset Balancing Segment Value">
  <apply rule="Role Playing KFF Tree Rule 1" prefix="Balancing Segment" />
 </instance>
 <instance logicaltable="Dim - CE Offset Balancing Segment Value">
  <apply rule="Role Playing KFF Tree Rule 1" prefix="Balancing Segment" />
 </instance>
...
</instances>
</document>
```

27

Administration Tool Keyboard Shortcuts

Learn about the keyboard shortcut information for the Oracle BI Server, including menu items and their corresponding keyboard shortcuts, keyboard shortcuts for navigating dialogs, and Physical Diagram and Business Model Diagram keyboard shortcuts.
This chapter contains the following topics:

- Menu Keyboard Shortcuts
- Dialog Keyboard Shortcuts
- Physical Diagram and Business Model Diagram Keyboard Shortcuts

Menu Keyboard Shortcuts

Learn the keyboard shortcuts available when using the Oracle BI Administration Tool menu options.

File Menu Shortcuts

- The shortcut for New is Ctrl + N.
- The shortcut for Open, and then Offline is Ctrl + F.
- The shortcut for Open, and then Online is Ctrl + L.
- The shortcut for Save is Ctrl + S.
- The shortcut for Check Global Consistency is Ctrl + K.

Edit Menu Shortcuts

- The shortcut for Cut is Ctrl + X.
- The shortcut for Copy is Ctrl + C.
- The shortcut for Paste is Ctrl + V.
- The shortcut for Delete is Delete.

View Menu Shortcut

- The shortcut for Refresh is F5.

Tools Menu Shortcuts

- The shortcut for Show Consistency Checker is Ctrl + E.
- The shortcut for Query Repository is Ctrl + Q.

General Menu Shortcuts

The following table lists the general keyboard shortcuts available in the Administration Tool menus. You can use the Window menu options to change the focus from the menus to the navigation panes.

Action	Keyboard Shortcut
Quit the application	Alt+ F4
Move cursor to the menu option	Alt + Underlined letter
Open application's control menu	Alt+ Spacebar
View the shortcut menu for the selected item	Shift + F10
Move through the menu bar	Left arrow key Right arrow key
Open a menu option	Down arrow key
Move through a menu list	Up arrow Down arrow
Close the current menu	Esc
Select or deselect items in a check box or list	Spacebar
Make noncontinguous selections	Ctrl + Up arrow + Spacebar

Dialog Keyboard Shortcuts

The table lists the keyboard shortcuts available in Oracle BI Administration Tool dialogs.

Action	Keyboard Shortcut
Move forward through options	Tab
Move backward through options	Shift + Tab
Select or deselect an item in a list	Shift + Up arrow Shift + Down arrow
Close the current dialog	Esc
Go to the top of a list	Home
Go to the bottom of a list	End
Refresh	F5
For dialogs with up arrow buttons: Move selected item up in the list	Select a list item and then type: Alt + Up arrow
For dialogs with down arrow buttons: Move selected item down in the list	Select a list item and then type: Alt + Down arrow
For dialogs with plus (add) buttons: Insert item from list	Alt + Insert
For dialogs with x (delete) buttons: Delete item from list	Alt + Delete

Action	Keyboard Shortcut
For dialogs with pencil (edit) buttons: Edit item from list	Alt + Enter
Browse dialog: Move focus between trees located in left pane	F5, F6, Shift + Tab, Tab
When a table row has a child row (grid): Expand the child row from the cell displaying the plus icon This situation occurs in the Define Merge Strategy page of the Merge Repository Wizard.	Move the focus to the cell displaying the plus icon and then type: Spacebar
When a table row has a check box: Select or deselect the check box This situation occurs in the Define Merge Strategy page of the Merge Repository Wizard.	Move the focus to the cell displaying the check box and then type: Spacebar

Physical Diagram and Business Model Diagram Keyboard Shortcuts

The table lists the keyboard shortcuts available in the Physical and Business Model Diagrams.

The Physical and Business Model Diagram toolbar options are also available from the Diagram menu.

Action	Keyboard Shortcut
Pan around the diagram when no diagram objects are selected	Arrow keys
Select a diagram object: use the arrow keys to move an object under the pointer, then press the spacebar to select the object	Spacebar + Arrow keys
Open the property dialog for a selected diagram object	Enter
Cancel current operation	Esc
Resume default mode (Select) after using Pan or Marquee Zoom	Esc
Deselect an object	Use one of the following methods: Esc Press the spacebar when the mouse cursor isn't over an object

Action	Keyboard Shortcut
Zoom in	+
Zoom out	-
Select the pan tool	P You can use the arrow keys to pan around the diagram.
Revert to auto-layout	S
Create a new join	J This shortcut selects the New Join option.
Create a new table	N This shortcut selects the New Table option. After using this shortcut, you can use the arrow keys and spacebar to pan around the diagram and open the Logical Table dialog for the new table.
Select the Marquee Zoom tool	Z This shortcut selects the Marquee Zoom tool.
Zoom to fit all objects in the current view	F
Show all tables in Expanded View, with columns visible	1
Show all tables in Collapsed View, with columns hidden and only the table name displayed	2

Part III
Reference

This part provides reference information.

Appendices:

- Expression Editor Reference

28

Expression Editor Reference

This section describes the expression elements that you can use in the Expression Editor.

Topics:

- SQL Operators
- Conditional Expressions
- Functions
- Constants
- Types

SQL Operators

SQL operators are used to specify comparisons between expressions.

You can use various types of SQL operators.

Operator	Example	Description	Syntax
BETWEEN	`"COSTS"."UNIT_COST" BETWEEN 100.0 AND 5000.0`	Determines if a value is between two non-inclusive bounds. `BETWEEN` can be preceded with `NOT` to negate the condition.	`BETWEEN [LowerBound] AND [UpperBound]`
IN	`"COSTS"."UNIT_COST" IN(200, 600, 'A')`	Determines if a value is present in a set of values.	`IN ([Comma Separated List])`
IS NULL	`"PRODUCTS"."PROD_NAME" IS NULL`	Determines if a value is null.	`IS NULL`
LIKE	`"PRODUCTS"."PROD_NAME" LIKE 'prod%'`	Determines if a value matches all or part of a string. Often used with wildcard characters to indicate any character string match of zero or more characters (%) or any single character match (_).	`LIKE`
+	`(FEDERAL_REVENUE + LOCAL_REVENUE) - TOTAL_EXPENDITURE`	Plus sign for addition.	`+`

Operator	Example	Description	Syntax
-	`(FEDERAL_REVENUE + LOCAL_REVENUE) - TOTAL_EXPENDITURE`	Minus sign for subtraction.	`-`
* *or* X	`SUPPORT_SERVICES_EXPENDITURE * 1.5`	Multiply sign for multiplication.	`*` `X`
/	`CAPITAL_OUTLAY_EXPENDITURE/ 1.05`	Divide by sign for division.	`/`
%		Percentage	`%`
||	`STATE|| CAST(YEAR AS CHAR(4))`	Character string concatenation.	`||`
(	`(FEDERAL_REVENUE + LOCAL_REVENUE) - TOTAL_EXPENDITURE`	Open parenthesis.	`(`
)	`(FEDERAL_REVENUE + LOCAL_REVENUE) - TOTAL_EXPENDITURE`	Close parenthesis.	`)`
>	`YEAR > 2000 and YEAR < 2016 and YEAR <> 2013`	Greater than sign, indicating values higher than the comparison.	`>`
<	`YEAR > 2000 and YEAR < 2016 and YEAR <> 2013`	Less than sign, indicating values lower than the comparison.	`<`
=		Equal sign, indicating the same value.	`=`
>=		Greater than or equal to sign, indicating values the same or higher than the comparison.	`>=`
<=		Less than or equal to sign, indicating values the same or lower than the comparison.	`<=`
<>	`YEAR > 2000 and YEAR < 2016 and YEAR <> 2013`	Not equal to, indicating values higher or lower, but different.	`<>`
,	`STATE in ('ALABAMA','CALIFORNIA')`	Comma, used to separate elements in a list.	`,`

Conditional Expressions

You use conditional expressions to create expressions that convert values.

The conditional expressions described in this section are building blocks for creating expressions that convert a value from one form to another.

Follow these rules:

- In `CASE` statements, `AND` has precedence over `OR`.
- Strings must be in single quotes.

Expression	Example	Description	Syntax

CASE (If)	`CASE`
	`WHEN score-par < 0 THEN 'Under Par'`
	`WHEN score-par = 0 THEN 'Par'`
	`WHEN score-par = 1 THEN 'Bogey'`
	`WHEN score-par = 2 THEN 'Double Bogey'`
	`ELSE 'Triple Bogey or Worse'`
	`END`
		Evaluates each `WHEN` condition and if satisfied, assigns the value in the corresponding `THEN` expression.
		If none of the `WHEN` conditions are satisfied, it assigns the default value specified in the `ELSE` expression. If no `ELSE` expression is specified, the system automatically adds an `ELSE NULL`.
		Note: See *Best Practices for using CASE statements in Analyses and Visualizations*.
			`CASE WHEN request_condition1 THEN expr1 ELSE expr2 END`

CASE (Switch)	`CASE Score-par`
	`WHEN -5 THEN 'Birdie on Par 6'`
	`WHEN -4 THEN 'Must be Tiger'`
	`WHEN -3 THEN 'Three under par'`
	`WHEN -2 THEN 'Two under par'`
	`WHEN -1 THEN 'Birdie'`
	`WHEN 0 THEN 'Par'`
	`WHEN 1 THEN 'Bogey'`
	`WHEN 2 THEN 'Double Bogey'`
	`ELSE 'Triple Bogey or Worse'`
	`END`
		Also referred to as `CASE (Lookup)`. The value of the first expression is examined, then the `WHEN` expressions. If the first expression matches any `WHEN` expression, it assigns the value in the corresponding `THEN` expression.
		If none of the `WHEN` expressions match, it assigns the default value specified in the `ELSE` expression. If no `ELSE` expression is specified, the system automatically adds an `ELSE NULL`.
		If the first expression matches an expression in multiple `WHEN` clauses, only the expression following the first match is assigned.
		Note See *Best Practices for using CASE statements in Analyses and Visualizations*.
			`CASE expr1 WHEN expr2 THEN expr3 ELSE expr4 END`

Expression	Example	Description	Syntax
IfCase > ELSE	-	-	`ELSE [expr]`
IfCase > IFNULL	-	-	`IFNULL([expr], [value])`
IfCase > NULLIF	-	-	`NULLIF([expr], [expr])`
IfCase > WHEN	-	-	`WHEN [Condition] THEN [expr]`
IfCase > CASE	-	-	`CASE WHEN [Condition] THEN [expr] END`
SwitchCase > ELSE	-	-	`ELSE [expr]`
SwitchCase >IFNULL	-	-	`IFNULL([expr], [value])`
SwitchCase > NULLIF	-	-	`NULLIF([expr], [expr])`
SwitchCase > WHEN	-	-	`WHEN [Condition] THEN [expr]`

Functions

There are various types of functions that you can use in expressions.

Topics:

- Aggregate Functions
- Analytics Functions
- Conversion Functions
- Date and Time Functions
- #unique_756
- Display Functions
- Evaluate Functions
- Mathematical Functions
- Running Aggregate Functions
- Spatial Functions
- String Functions
- System Functions
- Time Series Functions

Aggregate Functions

Aggregate functions perform operations on multiple values to create summary results.

The following list describes the aggregation rules that are available for columns and measure columns. The list also includes functions that you can use when creating calculated items for analyses.

- **Default** — Applies the default aggregation rule as in the Oracle BI Repository or by the original author of the analysis. Not available for calculated items in analyses.
- **Server Determined** — Applies the aggregation rule that's determined by the Oracle BI Server (such as the rule that is defined in the Oracle BI Repository). The aggregation is performed within Oracle BI Server for simple rules such as Sum, Min, and Max. Not available for measure columns in the Layout pane or for calculated items in analyses.
- **Sum** — Calculates the sum obtained by adding up all values in the result set. Use this for items that have numeric values.
- **Min** — Calculates the minimum value (lowest numeric value) of the rows in the result set. Use this for items that have numeric values.
- **Max** — Calculates the maximum value (highest numeric value) of the rows in the result set. Use this for items that have numeric values.
- **Average** — Calculates the average (mean) value of an item in the result set. Use this for items that have numeric values. Averages on tables and pivot tables are rounded to the nearest whole number.
- **First** — In the result set, selects the first occurrence of the item for measures. For calculated items, selects the first member according to the display in the Selected list. Not available in the Edit Column Formula dialog box.
- **Last** — In the result set, selects the last occurrence of the item. For calculated items, selects the last member according to the display in the Selected list. Not available in the Edit Column Formula dialog box.
- **Count** — Calculates the number of rows in the result set that have a non-null value for the item. The item is typically a column name, in which case the number of rows with non-null values for that column are returned.
- **Count Distinct** — Adds distinct processing to the Count function, which means that each distinct occurrence of the item is counted only once.
- **None** — Applies no aggregation. Not available for calculated items in analyses.
- **Server Complex Aggregate** — Applies the aggregation rule that is determined by the Oracle BI Server (such as the rule that is defined in the Oracle BI Repository). The aggregation is performed by the Oracle BI Server, rather than within Presentation Services. Not available for calculated items in analyses.
- **Report-Based Total (when applicable)** — If not selected, specifies that the Oracle BI Server should calculate the total based on the entire result set, before applying any filters to the measures. Not available in the Edit Column Formula dialog box or for calculated items in analyses. Only available for attribute columns.

Function	Example	Description	Syntax
AGGREGATE AT	`AGGREGATE(sales AT year)`	Aggregates columns based on the level or levels in the data model hierarchy you specify. • *measure* is the name of a measure column. • `level` is the level at which you want to aggregate. You can optionally specify more than one level. You can't specify a level from a dimension that contains levels that are being used as the measure level for the measure you specified in the first argument. For example, you can't write the function as `AGGREGATE(yearly_sales AT month)` if *month* is from the same time dimension used as the measure level for *yearly_sales*.	`AGGREGATE(measure AT level [, level1, levelN])`
AGGREGATE BY	`AGGREGATE(sales BY month, region)`	Aggregates a measure based on one or more dimension columns. • *measure* is the name of a measure column to aggregate. • `column` is the dimension column at which you want to aggregate. You can aggregate measures based more than one column.	`AGGREGATE(measure BY column [, column1, columnN])`
AVG	`Avg(Sales)`	Calculates the average (mean) of a numeric set of values.	`AVG(expr)`
AVGDISTINCT		Calculates the average (mean) of all distinct values of an expression.	`AVG(DISTINCT expr)`
BIN	`BIN(revenue BY productid, year WHERE productid > 2 INTO 4 BINS RETURNING RANGE_LOW)`	Classifies a given numeric expression into a specified number of equal width buckets. The function can return either the bin number or one of the two end points of the bin interval. numeric_expr is the measure or numeric attribute to bin. BY grain_expr1,..., grain_exprN is a list of expressions that define the grain at which the numeric_expr is calculated. BY is required for measure expressions and is optional for attribute expressions. WHERE a filter to apply to the numeric_expr before the numeric values are assigned to bins INTO number_of_bins BINS is the number of bins to return BETWEEN min_value AND max_value is the min and max values used for the end points of the outermost bins RETURNING NUMBER indicates that the return value should be the bin number (1, 2, 3, 4, etc.). This is the default. RETURNING RANGE_LOW indicates the lower value of the bin interval RETURNING RANGE_HIGH indicates the higher value of the bin interval	`BIN(numeric_expr [BY grain_expr1, ..., grain_exprN] [WHERE condition] INTO number_of_bins BINS [BETWEEN min_value AND max_value] [RETURNING {NUMBER | RANGE_LOW | RANGE_HIGH}])`

Function	Example	Description	Syntax
BottomN		Ranks the lowest n values of the expression argument from 1 to n, 1 corresponding to the lowest numerical value. *expr* is any expression that evaluates to a numerical value. integer is any positive integer. Represents the bottom number of rankings displayed in the result set, 1 being the lowest rank.	`BottomN(expr, integer)`
COUNT	`COUNT(Products)`	Determines the number of items with a non-null value.	`COUNT(expr)`
COUNTDISTINCT		Adds distinct processing to the COUNT function. *expr* is any expression.	`COUNT(DISTINCT expr)`
COUNT*	`SELECT COUNT(*) FROM Facts`	Counts the number of rows.	`COUNT(*)`
First	`First(Sales)`	Selects the first non-null returned value of the expression argument. The `First` function operates at the most detailed level specified in your explicitly defined dimension.	`First([NumericExpression)]`
Last	`Last(Sales)`	Selects the last non-null returned value of the expression.	`Last([NumericExpression)]`
MAVG		Calculates a moving average (mean) for the last n rows of data in the result set, inclusive of the current row. *expr* is any expression that evaluates to a numerical value. integer is any positive integer. Represents the average of the last n rows of data.	`MAVG(expr, integer)`
MAX	`MAX(Revenue)`	Calculates the maximum value (highest numeric value) of the rows satisfying the numeric expression argument.	`MAX(expr)`
MEDIAN	`MEDIAN(Sales)`	Calculates the median (middle) value of the rows satisfying the numeric expression argument. When there are an even number of rows, the median is the mean of the two middle rows. This function always returns a double.	`MEDIAN(expr)`
MIN	`MIN(Revenue)`	Calculates the minimum value (lowest numeric value) of the rows satisfying the numeric expression argument.	`MIN(expr)`
NTILE		Determines the rank of a value in terms of a user-specified range. It returns integers to represent any range of ranks. NTILE with numTiles=100 returns what is commonly called the "percentile" (with numbers ranging from 1 to 100, with 100 representing the high end of the sort). *expr* is any expression that evaluates to a numerical value. numTiles is a positive, nonnull integer that represents the number of tiles.	`NTILE(expr, numTiles)`

Function	Example	Description	Syntax
PERCENTILE		Calculates a percentile rank for each value satisfying the numeric expression argument. The percentile rank ranges are between 0 (0th percentile) to 1 (100th percentile). *expr* is any expression that evaluates to a numerical value.	`PERCENTILE(expr)`
RANK	`RANK(chronological_key, null, year_key_columns)`	Calculates the rank for each value satisfying the numeric expression argument. The highest number is assigned a rank of 1, and each successive rank is assigned the next consecutive integer (2, 3, 4,...). If certain values are equal, they'reare assigned the same rank (for example, 1, 1, 1, 4, 5, 5, 7...). *expr* is any expression that evaluates to a numerical value.	`RANK(expr)`
STDDEV	`STDDEV(Sales)` `STDDEV(DISTINCT Sales)`	Returns the standard deviation for a set of values. The return type is always a double.	`STDDEV(expr)`
STDDEV_POP	`STDDEV_POP(Sales)` `STDDEV_POP(DISTINCT Sales)`	Returns the standard deviation for a set of values using the computational formula for population variance and standard deviation.	`STDDEV_POP([NumericExpression])`
SUM	`SUM(Revenue)`	Calculates the sum obtained by adding up all values satisfying the numeric expression argument.	`SUM(expr)`
SUMDISTINCT		Calculates the sum obtained by adding all of the distinct values satisfying the numeric expression argument. *expr* is any expression that evaluates to a numerical value.	`SUM(DISTINCT expr)`
TOPN		Ranks the highest n values of the expression argument from 1 to n, 1 corresponding to the highest numerical value. *expr* is any expression that evaluates to a numerical value. integer is any positive integer. Represents the top number of rankings displayed in the result set, 1 being the highest rank.	`TOPN(expr, integer)`

Analytics Functions

Analytics functions allow you to explore data using models such as trendline and cluster.

Function	Example	Description	Syntax
TRENDLINE	`TRENDLINE(revenue, (calendar_year, calendar_quarter, calendar_month) BY (product), 'LINEAR', 'VALUE')`	Oracle recommends that you apply a Trendline using the **Add Statistics** property when viewing a visualization. See Adjust Visualization Properties. Fits a linear, polynomial, or exponential model, and returns the fitted values or model. The *numeric_expr* represents the Y value for the trend and the *series* (time columns) represent the X value.	`TRENDLINE(numeric_expr, ([series]) BY ([partitionBy]), model_type, result_type)`
CLUSTER	`CLUSTER((product, company), (billed_quantity, revenue), 'clusterName', 'algorithm=k-means;numClusters=%1;maxIter=%2;useRandomSeed=FALSE;enablePartitioning=TRUE', 5, 10)`	Collects a set of records into groups based on one or more input expressions using K-Means or Hierarchical Clustering.	`CLUSTER((dimension_expr1 , ... dimension_exprN), (expr1, ... exprN), output_column_name, options, [runtime_binded_options])`
OUTLIER	`OUTLIER((product, company), (billed_quantity, revenue), 'isOutlier', 'algorithm=kmeans')`	Classifies a record as Outlier based on one or more input expressions using K-Means or Hierarchical Clustering or Multi-Variate Outlier detection Algorithms.	`OUTLIER((dimension_expr1 , ... dimension_exprN), (expr1, ... exprN), output_column_name, options, [runtime_binded_options])`
REGR	`REGR(revenue, (discount_amount), (product_type, brand), 'fitted', '')`	Fits a linear model and returns the fitted values or model. This function can be used to fit a linear curve on two measures.	`REGR(y_axis_measure_expr, (x_axis_expr), (category_expr1, ..., category_exprN), output_column_name, options, [runtime_binded_options])`

Date and Time Functions

Date and time functions manipulate data based on `DATE` and `DATETIME`.

Function	Example	Description	Syntax
CURRENT_Date	`CURRENT_DATE`	Returns the current date. The date is determined by the system in which the Oracle BI is running.	`CURRENT_DATE`

Function	Example	Description	Syntax
CURRENT_TIME	`CURRENT_TIME(3)`	Returns the current time to the specified number of digits of precision, for example: HH:MM:SS.SSS If no argument is specified, the function returns the default precision.	`CURRENT_TIME(expr)`
CURRENT_TIMESTAMP	`CURRENT_TIMESTAMP(3)`	Returns the current date/timestamp to the specified number of digits of precision.	`CURRENT_TIMESTAMP(expr)`
DAYNAME	`DAYNAME(Order_Date)`	Returns the name of the day of the week for a specified date expression.	`DAYNAME(expr)`
DAYOFMONTH	`DAYOFMONTH(Order_Date)`	Returns the number corresponding to the day of the month for a specified date expression.	`DAYOFMONTH(expr)`
DAYOFWEEK	`DAYOFWEEK(Order_Date)`	Returns a number between 1 and 7 corresponding to the day of the week for a specified date expression. For example, 1 always corresponds to Sunday, 2 corresponds to Monday, and so on through to Saturday which returns 7.	`DAYOFWEEK(expr)`
DAYOFYEAR	`DAYOFYEAR(Order_Date)`	Returns the number (between 1 and 366) corresponding to the day of the year for a specified date expression.	`DAYOFYEAR(expr)`
DAY_OF_QUARTER	`DAY_OF_QUARTER(Order_Date)`	Returns a number (between 1 and 92) corresponding to the day of the quarter for the specified date expression.	`DAY_OF_QUARTER(expr)`
HOUR	`HOUR(Order_Time)`	Returns a number (between 0 and 23) corresponding to the hour for a specified time expression. For example, 0 corresponds to 12 a.m. and 23 corresponds to 11 p.m.	`HOUR(expr)`
MINUTE	`MINUTE(Order_Time)`	Returns a number (between 0 and 59) corresponding to the minute for a specified time expression.	`MINUTE(expr)`
MONTH	`MONTH(Order_Time)`	Returns the number (between 1 and 12) corresponding to the month for a specified date expression.	`MONTH(expr)`
MONTHNAME	`MONTHNAME(Order_Time)`	Returns the name of the month for a specified date expression.	`MONTHNAME(expr)`
MONTH_OF_QUARTER	`MONTH_OF_QUARTER(Order_Date)`	Returns the number (between 1 and 3) corresponding to the month in the quarter for a specified date expression.	`MONTH_OF_QUARTER(expr)`
NOW	`NOW()`	Returns the current timestamp. The `NOW` function is equivalent to the `CURRENT_TIMESTAMP` function.	`NOW()`
QUARTER_OF_YEAR	`QUARTER_OF_YEAR(Order_Date)`	Returns the number (between 1 and 4) corresponding to the quarter of the year for a specified date expression.	`QUARTER_OF_YEAR(expr)`
SECOND	`SECOND(Order_Time)`	Returns the number (between 0 and 59) corresponding to the seconds for a specified time expression.	`SECOND(expr)`

Function	Example	Description	Syntax
TIMESTAMPADD	`TIMESTAMPADD(SQL_TSI_MONTH, 12,Time."Order Date")`	Adds a specified number of intervals to a timestamp, and returns a single timestamp. Interval options are: *SQL_TSI_SECOND*, *SQL_TSI_MINUTE*, *SQL_TSI_HOUR*, *SQL_TSI_DAY*, *SQL_TSI_WEEK*, *SQL_TSI_MONTH*, *SQL_TSI_QUARTER*, *SQL_TSI_YEAR*	`TIMESTAMPADD(interval, expr, timestamp)`
TIMESTAMPDIFF	`TIMESTAMPDIFF(SQL_TSI_MONTH, Time."Order Date",CURRENT_DATE)`	Returns the total number of specified intervals between two timestamps. Use the same intervals as TIMESTAMPADD.	`TIMESTAMPDIFF(interval, expr, timestamp2)`
WEEK_OF_QUARTER	`WEEK_OF_QUARTER(Order_Date)`	Returns a number (between 1 and 13) corresponding to the week of the quarter for the specified date expression.	`WEEK_OF_QUARTER(expr)`
WEEK_OF_YEAR	`WEEK_OF_YEAR(Order_Date)`	Returns a number (between 1 and 53) corresponding to the week of the year for the specified date expression.	`WEEK_OF_YEAR(expr)`
YEAR	`YEAR(Order_Date)`	Returns the year for the specified date expression.	`YEAR(expr)`

Conversion Functions

Conversion functions convert a value from one form to another.

Function	Example	Description	Syntax
CAST	`CAST(hiredate AS CHAR(40)) FROM employee`	Changes the data type of an expression or a null literal to another data type. For example, you can cast a *customer_name* (a data type of `CHAR` or `VARCHAR`) or *birthdate* (a datetime literal). Use `CAST` to change to a *Date* data type. Don't use `TODATE`.	`CAST(expr AS type)`
IFNULL	`IFNULL(Sales, 0)`	Tests if an expression evaluates to a null value, and if it does, assigns the specified value to the expression.	`IFNULL(expr, value)`
INDEXCOL	`SELECT INDEXCOL(VALUEOF(NQ_SESSION.GEOGRAPHY_LEVEL), Country, State, City), Revenue FROM Sales`	Uses external information to return the appropriate column for the signed-in user to see.	`INDEXCOL([integer literal], [expr1] [, [expr2], ?-])`

Function	Example	Description	Syntax
NULLIF	`SELECT e.last_name, NULLIF(e.job_id, j.job_id) "Old Job ID" FROM employees e, job_history j WHERE e.employee_id = j.employee_id ORDER BY last_name, "Old Job ID";`	Compares two expressions. If they're equal, then the function returns NULL. If they're not equal, then the function returns the first expression. You can't specify the literal NULL for the first expression.	`NULLIF([expression], [expression])`
To_DateTime	`SELECT To_DateTime ('2009-03-0301:01:00' , 'yyyy-mm-dd hh:mi:ss') FROM sales`	Converts string literals of *DateTime* format to a *DateTime* data type.	`To_DateTime([expression], [literal])`
VALUEOF	`SalesSubjectArea.Customer.Region = VALUEOF("Region Security"."REGION")`	References the value of an Oracle BI repository variable in a filter. Use *expr* variables as arguments of the `VALUEOF` function. Refer to static repository variables by name.	`VALUEOF(expr)`

Display Functions

Display functions operate on the result set of a query.

Function	Example	Description	Syntax
BottomN	`BottomN(Sales, 10)`	Returns the *n* lowest values of expression, ranked from lowest to highest.	`BottomN([NumericExpression], [integer])`
FILTER	`FILTER(Sales USING Product = 'widget')`	Computes the expression using the given preaggregate filter.	`FILTER(measure USING filter_expr)`
MAVG	`MAVG(Sales, 10)`	Calculates a moving average (mean) for the last *n* rows of data in the result set, inclusive of the current row.	`MAVG([NumericExpression], [integer])`
MSUM	`SELECT Month, Revenue, MSUM(Revenue, 3) as 3_MO_SUM FROM Sales`	Calculates a moving sum for the last *n* rows of data, inclusive of the current row. The sum for the first row is equal to the numeric expression for the first row. The sum for the second row is calculated by taking the sum of the first two rows of data, and so on. When the *n* th row is reached, the sum is calculated based on the last *n* rows of data.	`MSUM([NumericExpression], [integer])`
NTILE	`NTILE(Sales, 100)`	Determines the rank of a value in terms of a user-specified range. It returns integers to represent any range of ranks. The example shows a range from 1 to 100, with the lowest sale = 1 and the highest sale = 100.	`NTILE([NumericExpression], [integer])`
PERCENTILE	`PERCENTILE(Sales)`	Calculates a percent rank for each value satisfying the numeric expression argument. The percentile rank ranges are from 0 (1st percentile) to 1 (100th percentile), inclusive.	`PERCENTILE([NumericExpression])`

Function	Example	Description	Syntax
RANK	`RANK(Sales)`	Calculates the rank for each value satisfying the numeric expression argument. The highest number is assigned a rank of 1, and each successive rank is assigned the next consecutive integer (2, 3, 4,...). If certain values are equal, they're assigned the same rank (for example, 1, 1, 1, 4, 5, 5, 7...).	`RANK([NumericExpression])`
RCOUNT	`SELECT month, profit, RCOUNT(profit) FROM sales WHERE profit > 200`	Takes a set of records as input and counts the number of records encountered so far.	`RCOUNT([NumericExpression])`
RMAX	`SELECT month, profit, RMAX(profit) FROM sales`	Takes a set of records as input and shows the maximum value based on records encountered so far. The specified data type must be one that can be ordered.	`RMAX([NumericExpression])`
RMIN	`SELECT month, profit, RMIN(profit) FROM sales`	Takes a set of records as input and shows the minimum value based on records encountered so far. The specified data type must be one that can be ordered.	`RMIN([NumericExpression])`
RSUM	`SELECT month, revenue, RSUM(revenue) as RUNNING_SUM FROM sales`	Calculates a running sum based on records encountered so far. The sum for the first row is equal to the numeric expression for the first row. The sum for the second row is calculated by taking the sum of the first two rows of data, and so on.	`RSUM([NumericExpression])`
TOPN	`TOPN(Sales, 10)`	Returns the *n* highest values of expression, ranked from highest to lowest.	`TOPN([NumericExpression], [integer])`

Evaluate Functions

Evaluate functions are database functions that can be used to pass through expressions to get advanced calculations.

Embedded database functions can require one or more columns. These columns are referenced by %1 ... %N within the function. The actual columns must be listed after the function.

Function	Example	Description	Syntax
EVALUATE	`SELECT EVALUATE('instr( %1, %2)', address, 'Foster City') FROM employees`	Passes the specified database function with optional referenced columns as parameters to the database for evaluation.	`EVALUATE([string expression], [comma separated expressions])`

Function	Example	Description	Syntax
EVALUATE_AGGR	`EVALUATE_AGGR('REGR_SLOPE(%1, %2)', sales.quantity, market.marketkey)`	Passes the specified database function with optional referenced columns as parameters to the database for evaluation. This function is intended for aggregate functions with a `GROUP BY` clause.	`EVALUATE_AGGR('db_agg_function(%1...%N)' [AS datatype] [, column1, columnN])`

Mathematical Functions

The mathematical functions described in this section perform mathematical operations.

Function	Example	Description	Syntax
ABS	`ABS(Profit)`	Calculates the absolute value of a numeric expression. *expr* is any expression that evaluates to a numerical value.	`ABS(expr)`
ACOS	`ACOS(1)`	Calculates the arc cosine of a numeric expression. *expr* is any expression that evaluates to a numerical value.	`ACOS(expr)`
ASIN	`ASIN(1)`	Calculates the arc sine of a numeric expression. *expr* is any expression that evaluates to a numerical value.	`ASIN(expr)`
ATAN	`ATAN(1)`	Calculates the arc tangent of a numeric expression. *expr* is any expression that evaluates to a numerical value.	`ATAN(expr)`
ATAN2	`ATAN2(1, 2)`	Calculates the arc tangent of *y* /*x*, where *y* is the first numeric expression and *x* is the second numeric expression.	`ATAN2(expr1, expr2)`
CEILING	`CEILING(Profit)`	Rounds a non-integer numeric expression to the next highest integer. If the numeric expression evaluates to an integer, the `CEILING` function returns that integer.	`CEILING(expr)`
COS	`COS(1)`	Calculates the cosine of a numeric expression. *expr* is any expression that evaluates to a numerical value.	`COS(expr)`
COT	`COT(1)`	Calculates the cotangent of a numeric expression. *expr* is any expression that evaluates to a numerical value.	`COT(expr)`
DEGREES	`DEGREES(1)`	Converts an expression from radians to degrees. *expr* is any expression that evaluates to a numerical value.	`DEGREES(expr)`

Function	Example	Description	Syntax
EXP	`EXP(4)`	Sends the value to the power specified. Calculates *e* raised to the n-th power, where *e* is the base of the natural logarithm.	`EXP(expr)`
ExtractBit	`Int ExtractBit(1, 5)`	Retrieves a bit at a particular position in an integer. It returns an integer of either 0 or 1 corresponding to the position of the bit.	`ExtractBit([Source Number], [Digits])`
FLOOR	`FLOOR(Profit)`	Rounds a non-integer numeric expression to the next lowest integer. If the numeric expression evaluates to an integer, the `FLOOR` function returns that integer.	`FLOOR(expr)`
LOG	`LOG(1)`	Calculates the natural logarithm of an expression. *expr* is any expression that evaluates to a numerical value.	`LOG(expr)`
LOG10	`LOG10(1)`	Calculates the base 10 logarithm of an expression. *expr* is any expression that evaluates to a numerical value.	`LOG10(expr)`
MOD	`MOD(10, 3)`	Divides the first numeric expression by the second numeric expression and returns the remainder portion of the quotient.	`MOD(expr1, expr2)`
PI	`PI()`	Returns the constant value of pi.	`PI()`
POWER	`POWER(Profit, 2)`	Takes the first numeric expression and raises it to the power specified in the second numeric expression.	`POWER(expr1, expr2)`
RADIANS	`RADIANS(30)`	Converts an expression from degrees to radians. *expr* is any expression that evaluates to a numerical value.	`RADIANS(expr)`
RAND	`RAND()`	Returns a pseudo-random number between 0 and 1.	`RAND()`
RANDFromSeed	`RAND(2)`	Returns a pseudo-random number based on a seed value. For a given seed value, the same set of random numbers are generated.	`RAND(expr)`
ROUND	`ROUND(2.166000, 2)`	Rounds a numeric expression to *n* digits of precision. *expr* is any expression that evaluates to a numerical value. *integer* is any positive integer that represents the number of digits of precision.	`ROUND(expr, integer)`
SIGN	`SIGN(Profit)`	Returns the following: • 1 if the numeric expression evaluates to a positive number • -1 if the numeric expression evaluates to a negative number • 0 if the numeric expression evaluates to zero	`SIGN(expr)`
SIN	`SIN(1)`	Calculates the sine of a numeric expression.	`SIN(expr)`

Function	Example	Description	Syntax
SQRT	`SQRT(7)`	Calculates the square root of the numeric expression argument. The numeric expression must evaluate to a nonnegative number.	`SQRT(expr)`
TAN	`TAN(1)`	Calculates the tangent of a numeric expression. *expr* is any expression that evaluates to a numerical value.	`TAN(expr)`
TRUNCATE	`TRUNCATE(45.12345, 2)`	Truncates a decimal number to return a specified number of places from the decimal point. *expr* is any expression that evaluates to a numerical value. *integer* is any positive integer that represents the number of characters to the right of the decimal place to return.	`TRUNCATE(expr, integer)`

Running Aggregate Functions

Running aggregate functions perform operations on multiple values to create summary results.

Function	Example	Description	Syntax
MAVG		Calculates a moving average (mean) for the last *n* rows of data in the result set, inclusive of the current row. *expr* is any expression that evaluates to a numerical value. *integer* is any positive integer. Represents the average of the last *n* rows of data.	`MAVG(expr, integer)`
MSUM	`select month, revenue, MSUM(revenue, 3) as 3_MO_SUM from sales_subject_area`	Calculates a moving sum for the last n rows of data, inclusive of the current row. *expr* is any expression that evaluates to a numerical value. *integer* is any positive integer. Represents the sum of the last n rows of data.	`MSUM(expr, integer)`
RSUM	`SELECT month, revenue, RSUM(revenue) as RUNNING_SUM from sales_subject_area`	Calculates a running sum based on records encountered so far. *expr* is any expression that evaluates to a numerical value.	`RSUM(expr)`
RCOUNT	`select month, profit, RCOUNT(profit) from sales_subject_area where profit > 200`	Takes a set of records as input and counts the number of records encountered so far. *expr* is an expression of any datatype.	`RCOUNT(expr)`

Function	Example	Description	Syntax
RMAX	`SELECT month, profit,RMAX(profit) from sales_subject_area`	Takes a set of records as input and shows the maximum value based on records encountered so far. *expr* is an expression of any datatype.	`RMAX(expr)`
RMIN	`select month, profit,RMIN(profit) from sales_subject_area`	Takes a set of records as input and shows the minimum value based on records encountered so far. *expr* is an expression of any datatype.	`RMIN(expr)`

Spatial Functions

Spatial functions enable you to perform geographical analysis when you model data using Administration Tool. For example, you might calculate the distance between two geographical areas (known as shapes or polygons).

> **Note:**
>
> You can't use these spatial functions in custom calculations for visualization workbooks.

Function	Example	Description	Syntax
GeometryArea	`GeometryArea(Shape)`	Calculates the area that a shape occupies.	`GeometryArea(Shape)`
GeometryDistance	`GeometryDistance(TRIP_START, TRIP_END)`	Calculates the distance between two shapes.	`GeometryDistance(Shape 1, Shape 2)`
GeometryLength	`GeometryLength(Shape)`	Calculates the circumference of a shape.	`GeometryLength(Shape)`
GeometryRelate	`GeometryRelate(TRIP_START, TRIP_END)`	Determines whether one shape is inside another shape. Returns 'TRUE' or 'FALSE' as a string (varchar).	`GeometryRelate(Shape 1, Shape 2)`
GeometryWithinDistance	`GeometryWithinDistance(TRIP_START, TRIP_END, 500)`	Determines whether two shapes are within a specified distance of each other. Returns 'TRUE' or 'FALSE' as a string (varchar).	`GeometryWithinDistance(Shape1, Shape2, DistanceInFloat)`

String Functions

String functions perform various character manipulations. They operate on character strings.

Function	Example	Description	Syntax
ASCII	`ASCII('a')`	Converts a single character string to its corresponding ASCII code, between 0 and 255. If the character expression evaluates to multiple characters, the ASCII code corresponding to the first character in the expression is returned. *expr* is any expression that evaluates to a character string.	`ASCII(expr)`
BIT_LENGTH	`BIT_LENGTH('abcdef')`	Returns the length, in bits, of a specified string. Each Unicode character is 2 bytes in length (equal to 16 bits). *expr* is any expression that evaluates to a character string.	`BIT_LENGTH(expr)`
CHAR	`CHAR(35)`	Converts a numeric value between 0 and 255 to the character value corresponding to the ASCII code. *expr* is any expression that evaluates to a numerical value between 0 and 255.	`CHAR(expr)`
CHAR_LENGTH	`CHAR_LENGTH(Customer_Name)`	Returns the length, in number of characters, of a specified string. Leading and trailing blanks aren't counted in the length of the string. *expr* is any expression that evaluates to a character string.	`CHAR_LENGTH(expr)`
CONCAT	`SELECT DISTINCT CONCAT ('abc', 'def') FROM employee`	Concatenates two character strings. *exprs* are expressions that evaluate to character strings, separated by commas. You must use raw data, not formatted data, with `CONCAT`.	`CONCAT(expr1, expr2)`
INSERT	`SELECT INSERT('123456', 2, 3, 'abcd') FROM table`	Inserts a specified character string into a specified location in another character string. *expr1* is any expression that evaluates to a character string. Identifies the target character string. *integer1* is any positive integer that represents the number of characters from the beginning of the target string where the second string is to be inserted. *integer2* is any positive integer that represents the number of characters in the target string to be replaced by the second string. *expr2* is any expression that evaluates to a character string. Identifies the character string to be inserted into the target string.	`INSERT(expr1, integer1, integer2, expr2)`

Function	Example	Description	Syntax
LEFT	`SELECT LEFT('123456', 3) FROM table`	Returns a specified number of characters from the left of a string. *expr* is any expression that evaluates to a character string *integer* is any positive integer that represents the number of characters from the left of the string to return.	`LEFT(expr, integer)`
LENGTH	`LENGTH(Customer_Name)`	Returns the length, in number of characters, of a specified string. The length is returned excluding any trailing blank characters. *expr* is any expression that evaluates to a character string.	`LENGTH(expr)`
LOCATE	`LOCATE('d' 'abcdef')`	Returns the numeric position of a character string in another character string. If the character string isn't found in the string being searched, the function returns a value of 0. *expr1* is any expression that evaluates to a character string. Identifies the string for which to search. *expr2* is any expression that evaluates to a character string. Identifies the string to be searched.	`LOCATE(expr1, expr2)`
LOCATEN	`LOCATEN('d' 'abcdef', 3)`	Like LOCATE, returns the numeric position of a character string in another character string. LOCATEN includes an integer argument that enables you to specify a starting position to begin the search. *expr1* is any expression that evaluates to a character string. Identifies the string for which to search. *expr2* is any expression that evaluates to a character string. Identifies the string to be searched. *integer* is any positive (nonzero) integer that represents the starting position to begin to look for the character string.	`LOCATEN(expr1, expr2, integer)`
LOWER	`LOWER(Customer_Name)`	Converts a character string to lowercase. *expr* is any expression that evaluates to a character string.	`LOWER(expr)`
OCTET_LENGTH	`OCTET_LENGTH('abcdef')`	Returns the number of bytes of a specified string. *expr* is any expression that evaluates to a character string.	`OCTET_LENGTH(expr)`

Function	Example	Description	Syntax
POSITION	`POSITION('d', 'abcdef')`	Returns the numeric position of *strExpr1* in a character expression. If *strExpr1* isn't found, the function returns 0. *expr1* is any expression that evaluates to a character string. Identifies the string to search for in the target string. *expr2* is any expression that evaluates to a character string. Identifies the target string to be searched.	`POSITION(expr1 IN expr2)`
REPEAT	`REPEAT('abc', 4)`	Repeats a specified expression *n* times. *expr* is any expression that evaluates to a character string *integer* is any positive integer that represents the number of times to repeat the character string.	`REPEAT(expr, integer)`
REPLACE	`REPLACE('abcd1234', '123', 'zz')`	Replaces one or more characters from a specified character expression with one or more other characters. *expr1* is any expression that evaluates to a character string. This is the string in which characters are to be replaced. *expr2* is any expression that evaluates to a character string. This second string identifies the characters from the first string that are to be replaced. *expr3* is any expression that evaluates to a character string. This third string specifies the characters to substitute into the first string.	`REPLACE(expr1, expr2, expr3)`
RIGHT	`SELECT RIGHT('123456', 3) FROM table`	Returns a specified number of characters from the right of a string. *expr* is any expression that evaluates to a character string. *integer* is any positive integer that represents the number of characters from the right of the string to return.	`RIGHT(expr, integer)`
SPACE	`SPACE(2)`	Inserts blank spaces. *integer* is any positive integer that indicates the number of spaces to insert.	`SPACE(expr)`
SUBSTRING	`SUBSTRING('abcdef' FROM 2)`	Creates a new string starting from a fixed number of characters into the original string. *expr* is any expression that evaluates to a character string. *startPos* is any positive integer that represents the number of characters from the start of the left side of the string where the result is to begin.	`SUBSTRING([SourceString] FROM [StartPostition])`

Function	Example	Description	Syntax
SUBSTRINGN	`SUBSTRING('abcdef' FROM 2 FOR 3)`	Like SUBSTRING, creates a new string starting from a fixed number of characters into the original string. *SUBSTRINGN* includes an integer argument that enables you to specify the length of the new string, in number of characters. *expr* is any expression that evaluates to a character string. *startPos* is any positive integer that represents the number of characters from the start of the left side of the string where the result is to begin.	`SUBSTRING(expr FROM startPos FOR length)`
TrimBoth	`Trim(BOTH '_' FROM '_abcdef_')`	Strips specified leading and trailing characters from a character string. *char* is any single character. If you omit this specification (and the required single quotes), a blank character is used as the default. *expr* is any expression that evaluates to a character string.	`TRIM(BOTH char FROM expr)`
TRIMLEADING	`TRIM(LEADING '_' FROM '_abcdef')`	Strips specified leading characters from a character string. *char* is any single character. If you omit this specification (and the required single quotes), a blank character is used as the default. *expr* is any expression that evaluates to a character string.	`TRIM(LEADING char FROM expr)`
TRIMTRAILING	`TRIM(TRAILING '_' FROM 'abcdef_')`	Strips specified trailing characters from a character string. *char* is any single character. If you omit this specification (and the required single quotes), a blank character is used as the default. *expr* is any expression that evaluates to a character string.	`TRIM(TRAILING char FROM expr)`
UPPER	`UPPER(Customer_Name)`	Converts a character string to uppercase. *expr* is any expression that evaluates to a character string.	`UPPER(expr)`

System Functions

The `USER` system function returns values relating to the session. For example, the user name you signed in with.

Function	Example	Description	Syntax
DATABASE		Returns the name of the subject area to which you're logged on.	`DATABASE()`

Function	Example	Description	Syntax
USER		Returns the user name for the semantic model to which you're logged on.	`USER()`

Time Series Functions

Time series functions are aggregate functions that operate on time dimensions.

Time dimension members must be at or below the level of the function. Because of this, one or more columns that uniquely identify members at or below the given level must be projected in the query.

Function	Example	Description	Syntax
AGO	`SELECT Year_ID, AGO(sales, year, 1)`	Calculates the aggregated value of a measure from the current time to a specified time period in the past. For example, `AGO` can produce sales for every month of the current quarter and the corresponding quarter-ago sales.	`AGO(expr, time_level, offset)`
PERIODROLLING	`SELECT Month_ID, PERIODROLLING (monthly_sales, -1, 1)`	Computes the aggregate of a measure over the period starting *x* units of time and ending *y* units of time from the current time. For example, `PERIODROLLING` can compute sales for a period that starts at a quarter before and ends at a quarter after the current quarter. *measure* is the name of a measure column. *x* is an integer that specifies the offset from the current time. *y* specifies the number of time units over which the function computes. *hierarchy* is an optional argument that specifies the name of a hierarchy in a time dimension, such as *yr*, *mon*, *day*, that you want to use to compute the time window.	`PERIODROLLING(measure , x [,y])`
TODATE	`SELECT Year_ID, Month_ID, TODATE (sales, year)`	Aggregates a measure from the beginning of a specified time period to the currently displayed time. For example, this function can calculate Year to Date sales. *expr* is an expression that references at least one measure column. *time_level* is the type of time period, such as quarter, month, or year.	`TODATE(expr, time_level)`

FORECAST Function

Creates a time-series model of the specified measure over the series using Exponential Smoothing (ETS) or Seasonal ARIMA or ARIMA. This function outputs a forecast for a set of periods as specified by *numPeriods* argument.

Syntax `FORECAST(numeric_expr, ([series]), output_column_name, options, [runtime_binded_options])])`

Where:

- *numeric_expr* indicates the measure to forecast, for example, revenue data to forecast.
- *series* indicates the time grain used to build the forecast model. The series is a list of one or more time dimension columns. If you omit series, then the time grain is determined from the query.
- *output_column_name* indicates the valid column names of *forecast*, *low*, *high*, and *predictionInterval*.
- *options* indicates a string list of name/value pairs separated by a semi-colon (;). The value can include *%1 ... %N* specified in `runtime_binded_options`.
- *runtime_binded_options* indicates a comma separated list of columns and options. Values for these columns and options are evaluated and resolved during individual query execution time.

FORECAST Function Options The following table list available options to use with the `FORECAST` function.

Option Name	Values	Description
numPeriods	Integer	The number of periods to forecast
predictionInterval	0 to 100, where higher values specify higher confidence	The confidence level for the prediction.
modelType	ETS SeasonalArima ARIMA	The model to use for forecasting.
useBoxCox	TRUE FALSE	If *TRUE*, then use Box-Cox transformation.
lambdaValue	Not applicable	The Box-Cox transformation parameter. Ignore if NULL or when `useBoxCox` is *FALSE*. Otherwise the data is transformed before the model is estimated.
trendDamp	TRUE FALSE	This is a parameter for ETS model. If *TRUE*, then use damped trend. If FALSE or NULL, then use non-damped trend.
errorType	Not applicable	This is a parameter for ETS model.
trendType	N (none) A (additive) M (multiplicative) Z (automatically selected)	This is a parameter for ETS model.
seasonType	N (none) A (additive) M (multiplicative) Z (automatically selected)	This is a parameter for ETS model.

Option Name	Values	Description
modelParamIC	ic_auto ic_aicc ic_bic ic_auto (this is the default)	The information criterion (IC) used in the model selection.

Revenue Forecast by Day Example

This example selects revenue forecast by day.

```
FORECAST("A - Sample Sales"."Base Facts"."1- Revenue" Target,
("A - Sample Sales"."Time"."T00 Calendar Date"),'forecast',
'numPeriods=30;predictionInterval=70;') ForecastedRevenue
```

Revenue Forecast by Year and Quarter Example

This example selects revenue forecast by year and quarter.

```
FORECAST("A - Sample Sales"."Base Facts"."1- Revenue",
("A - Sample Sales"."Time"."T01 Year" timeYear, "A - Sample Sales"."Time"."T02
Quarter" TimeQuarter),'forecast', 'numPeriods=30;predictionInterval=70;')
ForecastedRevenue
```

Constants

You can use constants to include specific dates and times in expressions.

Available constants include Date, Time, and Timestamp.

Constant	Example	Description	Syntax
DATE	`DATE [2014-04-09]`	Inserts a specific date.	`DATE [yyyy-mm-dd]`
TIME	`TIME [12:00:00]`	Inserts a specific time.	`TIME [hh:mi:ss]`
TIMESTAMP	`TIMESTAMP [2014-04-09 12:00:00]`	Inserts a specific timestamp.	`TIMESTAMP [yyyy-mm-dd hh:mi:ss]`

Types

You can use data types, such as `CHAR`, `INT`, and `NUMERIC` in expressions.

For example, you use types when creating `CAST` expressions that change the data type of an expression or a null literal to another data type.

Variables

Variables are used in expressions.

You can use a variable in an expression.

See Advanced Techniques: Reference Stored Values in Variables.