# Oracle® Retail Data Model

Implementation and Operations Guide

Release 11.3.2

**E20363-03**

January 2013

ORACLE®

Oracle Retail Data Model Implementation and Operations Guide, Release 11.3.2

E20363-03

Copyright © 2011, 2013, Oracle and/or its affiliates. All rights reserved.

Primary Author: Thomas Van Raalte

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

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. 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 and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon 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, Opteron, the AMD logo, and the AMD Opteron 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 on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Oracle Retail Data Model contains the ARTS Data Model licensed to Oracle by ARTS.

# Contents

## 1 Introduction to Oracle Retail Data Model Customization


**What is the Oracle Retail Data Model?** ..... 1-1
Components of the Oracle Retail Data Model ..... 1-2
Oracle Products That Make Up Oracle Retail Data Model ..... 1-2
**Steps for Implementing an Oracle Retail Data Model Warehouse** ..... 1-3
**Before You Begin Customizing the Oracle Retail Data Model** ..... 1-4
Prerequisite Knowledge for Implementors ..... 1-4
Responsibilities of a Data Warehouse Governance Committee ..... 1-5
**Managing Metadata for Oracle Retail Data Model** ..... 1-6
Metadata Categories and Standards ..... 1-6
Working with a Metadata Repository ..... 1-7
Browsing the Metadata Repository Supplied With Oracle Retail Data Model ..... 1-7
Using the Metadata Generation Packages ..... 1-9
Using Oracle Warehouse Builder with the Oracle Retail Data Model ..... 1-9
**Performing Fit-Gap Analysis for Oracle Retail Data Model** ..... 1-10


## 2 Physical Model Customization


**Characteristics of the Default Physical Model** ..... 2-1
**Customizing the Oracle Retail Data Model Physical Model** ..... 2-3
Questions to Answer Before You Customize the Physical Model ..... 2-4
Conventions When Customizing the Physical Model ..... 2-4
**Foundation Layer Customization** ..... 2-5
Common Change Scenarios When Customizing the Foundation Layer of Oracle Retail Data Model 2-6
Example of Changing the Foundation Layer of the Oracle Retail Data Model ..... 2-7
**General Recommendations When Designing Physical Structures** ..... 2-8
Tablespaces in Oracle Retail Data Model ..... 2-8
Data Compression in Oracle Retail Data Model ..... 2-9
Types of Data Compression Available ..... 2-9
Basic or Standard Compression ..... 2-9
OLTP Compression ..... 2-9
Hybrid Columnar Compression (HCC) ..... 2-10
Surrogate Keys in the Physical Model ..... 2-10
Integrity Constraints in Oracle Retail Data Model ..... 2-11
Indexes and Partitioned Indexes in Oracle Retail Data Model ..... 2-11
Partitioned Tables in Oracle Retail Data Model ..... 2-12

Partitioning the Oracle Retail Data Model for Manageability ..... 2-13
Partitioning the Oracle Retail Data Model for Easier Data Access ..... 2-13
Partitioning the Oracle Retail Data Model for Join Performance ..... 2-13
Parallel Execution in Oracle Retail Data Model ..... 2-14
Enabling Parallel Execution for a Session ..... 2-16
Enabling Parallel Execution of DML Operations ..... 2-16
Enabling Parallel Execution at the Table Level ..... 2-17

## 3 Access Layer Customization

**Introduction to Customizing the Access Layer of Oracle Retail Data Model** ..... 3-1
**Derived Tables in the Oracle Retail Data Model** ..... 3-1
Creating New Derived Tables for Calculated Data ..... 3-2
Customizing Oracle Retail Data Model Data Mining Models ..... 3-2
Creating a New Data Mining Model for Oracle Retail Data Model ..... 3-2
Modifying Oracle Retail Data Model Data Mining Models ..... 3-3
Tutorial: Customizing the Customer Life Time Value Prediction Data Mining Model ..... 3-3
Tutorial Prerequisites ..... 3-4
Preparing Your Environment ..... 3-4
Generating the Model ..... 3-7
Checking the Result ..... 3-8
**Dimensional Components in the Oracle Retail Data Model** ..... 3-9
Characteristics of a Dimensional Model ..... 3-9
Characteristics of Relational Star and Snowflake Tables ..... 3-10
Declaring Relational Dimension Tables ..... 3-11
Validating Relational Dimension Tables ..... 3-11
Characteristics of the OLAP Dimensional Model ..... 3-12
Oracle OLAP Cube Views ..... 3-13
Cube Materialized Views ..... 3-14
Characteristics of the OLAP Cubes in Oracle Retail Data Model ..... 3-15
Defining New Oracle OLAP Cubes for Oracle Retail Data Model ..... 3-16
Changing an Oracle OLAP Cube in Oracle Retail Data Model ..... 3-17
Creating a Forecast Cube for Oracle Retail Data Model ..... 3-17
Choosing a Cube Partitioning Strategy for Oracle Retail Data Model ..... 3-17
Choosing a Cube Data Maintenance Method for Oracle Retail Data Model ..... 3-18
**Materialized Views in the Oracle Retail Data Model** ..... 3-19
Types of Materialized Views and Refresh options ..... 3-20
Refresh Options for Materialized Views with Aggregates ..... 3-20
Refresh Options for Materialized Views Containing Only Joins ..... 3-21
Refresh Options for Nested Materialized Views ..... 3-22
Choosing Indexes for Materialized Views ..... 3-22
Partitioning and Materialized Views ..... 3-22
Compressing Materialized Views ..... 3-24

## 4 ETL Implementation and Customization

**The Role of ETL in the Oracle Retail Data Model** ..... 4-1
**Creating Source-ETL for Oracle Retail Data Model** ..... 4-2
Source-ETL Design Considerations ..... 4-3

ETL Architecture for Oracle Retail Data Model Source-ETL ..... 4-4
Creating a Source to Target Mapping Document for the Source-ETL ..... 4-4
Designing a Plan for Rectifying Source-ETL Data Quality Problems ..... 4-5
Designing Source-ETL Workflow and Jobs Control ..... 4-5
Designing Source-ETL Exception Handling ..... 4-5
Writing Source-ETL that Loads Efficiently ..... 4-6
Using a Staging Area for Flat Files ..... 4-6
Preparing Raw Data Files for Source-ETL ..... 4-6
Source-ETL Data Loading Options ..... 4-7
Parallel Direct Path Load Source-ETL ..... 4-8
Partition Exchange Load for Oracle Retail Data Model Source-ETL ..... 4-8
**Customizing Intra-ETL for the Oracle Retail Data Model** ..... 4-9
ORDM_DERIVED_FLW ..... 4-10
ORDM_AGG_N_DEP_FLW ..... 4-11
ORDM_AGG_DEP_FLW ..... 4-12
OLAP_MAP Mapping Flow ..... 4-13
ORDM_MNNG_FLW ..... 4-14
**Performing an Initial Load of an Oracle Retail Data Model Warehouse** ..... 4-14
Executing the Default Oracle Retail Data Model Intra-ETL ..... 4-16
Executing the ORDM_INTRA_ETL_FLW Workflow from Oracle Warehouse Builder 4-17
Executing the Intra-ETL Without Using Oracle Warehouse Builder ..... 4-17
Executing the Intra-ETL by Using the PKG_INTRA_ETL_PROCESS.RUN Procedure .... 4-17
**Refreshing the Data in Oracle Retail Data Model Warehouse** ..... 4-18
Refreshing Oracle Retail Data Model Relational Tables and Views ..... 4-18
Refreshing Oracle Retail Data Model OLAP Cubes ..... 4-19
Refreshing Oracle Retail Data Model Data Mining Models ..... 4-19
**Managing Errors During Oracle Retail Data Model Intra-ETL Execution** ..... 4-20
Monitoring the Execution of the Intra-ETL Process ..... 4-20
Recovering an Intra ETL Process ..... 4-21
Troubleshooting Intra-ETL Performance ..... 4-21
Checking the Execution Plan ..... 4-22
Monitoring PARALLEL DML Executions ..... 4-22
Troubleshooting Data Mining Model Creation ..... 4-22

## 5 Report and Query Customization

**Reporting Approaches in Oracle Retail Data Model** ..... 5-1
**Customizing Oracle Retail Data Model Reports** ..... 5-2
**Writing Your Own Queries and Reports** ..... 5-3
**Optimizing Star Queries** ..... 5-4
**Troubleshooting Oracle Retail Data Model Report Performance** ..... 5-6
**Writing As Is and As Was Queries** ..... 5-6
Characteristics of an As Is Query ..... 5-7
Characteristics of an As Was Query ..... 5-7
Examples: As Is and As Was Queries ..... 5-7
**Tutorial: Creating a New Oracle Retail Data Model Dashboard** ..... 5-12
**Tutorial: Creating a New Oracle Retail Data Model Report** ..... 5-20

## 6 Metadata Collection and Reports


**Metadata Collection and Population** ..... 6-1
Load LDM/PDM Metadata (Table MD_ENTY) ..... 6-5
GIVE_ABBRV ..... 6-5
MD_DM_ALL_ENT_ATTR ..... 6-5
PL/SQL Program to Update Column Name ..... 6-6
PL/SQL program to insert initial data into MD_OIDM_ATTR_COL_NAM ..... 6-6
PL/SQL program to load data into MD_ENTY ..... 6-6
Load Program (Intra-ETL) Metadata (Table MD_PRG) ..... 6-6
Load Reports and KPI Metadata (Table MD_KPI and MD_REF_ENTY_KPI): ..... 6-7
**Metadata Reports and Dashboard** ..... 6-9


## 7 Multi-Currency Support and Configuration


**Multi-Currency Overview** ..... 7-1
**Multi-Currency Data Field Naming Conventions** ..... 7-2
**Multi-Currency Data Movement** ..... 7-3
Movement from Interface to Base Tables ..... 7-3
Handling Currency at the Base Level ..... 7-3
Movement from Base to Derived Tables ..... 7-3
Movement from Derived to Aggregate Tables ..... 7-4
Handling Data Movement from Base to Derived and Aggregate Layers ..... 7-4
**Currency Data Flow** ..... 7-4
**Currency DWC_CRNCY_CONF Table** ..... 7-4


## A Sizing and Configuring an Oracle Retail Data Model Warehouse


**Sizing an Oracle Retail Data Model Warehouse** ..... A-1
**Configuring a Balanced System for Oracle Retail Data Model** ..... A-3
Maintaining High Throughput in an Oracle Retail Data Model Warehouse ..... A-3
Configuring I/O in an Oracle Retail Data Model for Bandwidth not Capacity ..... A-3
Planning for Growth of Your Oracle Retail Data Model ..... A-4
Testing the I/O System Before Building the Oracle Retail Data Model Warehouse ..... A-4
Balanced Hardware Configuration Guidelines for Oracle Retail Data Model ..... A-4


## Index

## List of Examples


3–1 Adding a New Column to a Mining Model in Oracle Retail Data Model ..... 3-3
3–2 Refreshing Oracle Retail Data Model Nested Materialized Views ..... 3-22
4–1 Using Exchange Partition Statement with a Partitioned Table ..... 4-8
4–2 Troubleshooting an ORA-20991 Error for Oracle Retail Data Model ..... 4-23
4–3 Troubleshooting the "Message not available ... [Language=ZHS]" Error ..... 4-23
4–4 Troubleshooting an ORA-40113 Error for Oracle Retail Data Model ..... 4-23
4–5 Troubleshooting an ORA-40112 Error for Oracle Retail Data Model ..... 4-24
4–6 Troubleshooting an ORG-11130 Error for Oracle Retail Data Model ..... 4-24
5–1 Creating a Relational Query for Oracle Retail Data Model ..... 5-3
5–2 Star Transformation ..... 5-4
5–3 Troubleshooting a Report: A Table Does Not Exist ..... 5-6
5–4 Troubleshooting a Report: When the Database is Not Connected ..... 5-6
5–5 As Is Query for Sales Split by Item Subclass ..... 5-10
5–6 As Was Query for Sales Split by Item Subclass ..... 5-10

## List of Figures


2–1 Layers of an Oracle Retail Data Model Warehouse ..... 2-2
2–2 Partitioning for Join Performance ..... 2-14
2–3 Parallel Execution of a Full Partition-Wise Join Between Two Tables ..... 2-15
2–4 Partial Partition-Wise Join ..... 2-16
3–1 Oracle SQL Developer with ORDM_SYS Schema ..... 3-5
3–2 Applying Filters ..... 3-5
3–3 Review Tables to Ensure Valid Data ..... 3-7
3–4 Checking the Result ..... 3-8
3–5 Star Schema Diagram ..... 3-11
3–6 Diagram of the OLAP Dimensional Model ..... 3-13
4–1 ETL Flow Diagram ..... 4-2
4–2 ORDM Main Intra-ETL Process Flow ..... 4-10
4–3 Intra-ETL Derived Process Flow ..... 4-11
4–4 Intra-ETL Independent MV Process Flow ..... 4-12
4–5 Intra-ETL Aggregate Process Flow ..... 4-13
4–6 OLAP Map Process Flow ..... 4-14
4–7 Mining Flow Process ..... 4-14
5–1 New Dashboard Start: BIEE Home ..... 5-13
5–2 New Dashboard: Enter Name ..... 5-13
5–3 New Dashboard: Catalog View ..... 5-14
5–4 New Dashboard: Catalog View with New Reports Vertical ..... 5-15
5–5 New Dashboard: Catalog View with New Reports Horizontal ..... 5-16
5–6 New Dashboard: Select Name for Page ..... 5-17
5–7 New Dashboard: Enter New Page Name ..... 5-18
5–8 New Dashboard: Rename Page Dialog ..... 5-19
5–9 New Dashboard: Display with Two New Reports ..... 5-20
5–10 Analysis Report: Welcome Page with New Menu ..... 5-21
5–11 Analysis Report: Welcome Page with Select Subject Area Menu ..... 5-21
5–12 Analysis Report: Welcome Page with Selected Columns ..... 5-22
5–13 Analysis Report: Results View of Report ..... 5-23
5–14 New Report: Create New View ..... 5-24
5–15 New Report: Final View of New Reports ..... 5-25
7–1 Currency and Transaction Amount Data Flow in Oracle Retail Data Model ..... 7-4

# Preface

The *Oracle Retail Data Model Implementation and Operations Guide* describes best practices for implementing a data warehouse based on the Oracle Retail Data Model.

This preface contains the following topics:


- Audience
- Documentation Accessibility
- Related Oracle Resources
- Conventions


## Audience

This document is intended for business analysts, data modelers, data warehouse administrators, IT staff, and ETL developers who implement an Oracle Retail Data Model warehouse.

## 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 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.

## Related Oracle Resources

Oracle provides many resources for you when implementing the Oracle Retail Data Model.

### Oracle Retail Data Model Documentation Set

For more information on Oracle Retail Data Model, see the following documents in the Oracle Retail Data Model Release 11*g* documentation set:

- *Oracle Retail Data Model Installation Guide*

- *Oracle Retail Data Model Reference*
- *Oracle Retail Data Model Release Notes*

### Oracle Technology Network

Visit the Oracle Technology Network (OTN to access to demos, whitepapers, Oracle By Example (OBE) tutorials, updated Oracle documentation, and other collateral.

**Registering on OTN**

You must register online before using OTN, Registration is free and can be done at

`www.oracle.com/technetwork/index.html`

**Oracle Documentation on OTN**

The Oracle Documentation site on OTN provides access to Oracle documentation. After you have a user name and password for OTN, you can go directly to the documentation section of the OTN Web site at

`www.oracle.com/technetwork/indexes/documentation/index.html`

**Oracle Learning Library on OTN**

The Oracle Learning Library provides free online training content (OBEs, Demos and Tutorials). After you have a user name and password for OTN, you can go directly to the Oracle Learning Library Web site at

`www.oracle.com/technetwork/tutorials/index.html`

Then you can search for a tutorial or demo (within "All") by name.

## 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. |

# 1

# Introduction to Oracle Retail Data Model Customization

This chapter provides an introduction to customizing the Oracle Retail Data Model. It contains the following topics:


- What is the Oracle Retail Data Model?
- Steps for Implementing an Oracle Retail Data Model Warehouse
- Before You Begin Customizing the Oracle Retail Data Model
- Managing Metadata for Oracle Retail Data Model
- Performing Fit-Gap Analysis for Oracle Retail Data Model


## What is the Oracle Retail Data Model?

Oracle Retail Data Model is a standards-based, pre-built approach to retail data warehousing enabling a retail company to gain insight from their data more quickly. Oracle Retail Data Model reduces costs for both immediate and on-going operations by leveraging out-of-box Oracle based data warehouse and business intelligence solutions, making world-class database and business intelligence technology solutions available with a retail specific data model. You can use Oracle Retail Data Model in any application environment. Also, you can easily extend the model.

Using Oracle Retail Data Model you can jump-start the design and implementation of an Oracle Retail Data Model warehouse to quickly achieve a positive ROI for your data warehousing and business intelligence project with a predictable implementation effort.

Oracle Retail Data Model provides the following features:

- Enterprise wide data model for retail industry
  - Over 1,250 tables and 18,500 attributes
  - Over 1,800 industry measures and KPIs
  - Based on ARTS 6.0
- Prebuilt mining models, OLAP cubes, and reports
- Automatic data movement across the warehouse (Intra-ETL)
- Easily extensible and customizable
- Usable within any retail application
- Metadata Browser with Refresh

Oracle Retail Data Model provides much of the data modeling work that you must do for a retail business intelligence solution. The Oracle Retail Data Model logical and physical data models were designed following best practices for retail service providers. Oracle Retail Data Model is based on the ARTS 6.0 standard.


- Components of the Oracle Retail Data Model
- Oracle Products That Make Up Oracle Retail Data Model


## Components of the Oracle Retail Data Model

Oracle Retail Data Model includes the following components which are described in detail in Oracle Retail Data Model Reference:

- Logical model which is a third normal form (3NF) entity-object standards-based model.
- Physical model defined as an Oracle Database schema.
- Intra-ETL database packages and SQL scripts to extract, transform, and load (ETL) data from the Oracle Retail Data Model 3NF physical tables to the Oracle Retail Data Model derived and aggregate objects.
- Sample reports and dashboards developed using Oracle Business Intelligence Suite Enterprise Edition.
- Data Mining Models models for: Employee Analysis, Customer Analysis, Store Analysis, Item Analysis, and Product Analysis.
- DDL and installation scripts.

> **See:** *Oracle Retail Data Model Reference* for detailed information about the Oracle Retail Data Model components.
>
> *Oracle Retail Data Model Installation Guide* for detailed information on the different types of Oracle Retail Data Model installation.

## Oracle Products That Make Up Oracle Retail Data Model

Several Oracle technologies are involved in building the infrastructure for Oracle Retail Data Model:


- Oracle Database with OLAP, Data Mining and Partitioning Option
- Oracle Development Tools
- Oracle Business Intelligence Suite Enterprise Edition Presentation Tools


### Oracle Database with OLAP, Data Mining and Partitioning Option

Oracle Retail Data Model uses a complete Oracle technical stack. It leverages the following data warehousing features of the Oracle Database: compression, partitioning, advanced statistical functions, materialized views, data mining, and online analytical processing (OLAP).

### Oracle Development Tools

You can use the following Oracle tools to customize the predefined physical models provided with Oracle Retail Data Model, or to populate the target relational tables and materialized cube views.

*Table 1–1 Oracle Development Tools Used with Oracle Retail Data Model*

| Name | Use |
|---|---|
| SQL Developer or SQL*Plus | To modify, customize, and extend database objects |
| Oracle Warehouse Builder | For the process control of the intra ETL process |
| Analytic Workspace Manager | To view, create, develop, and manage OLAP dimensional objects. |

### Oracle Business Intelligence Suite Enterprise Edition Presentation Tools

Oracle Business Intelligence Suite Enterprise Edition is a comprehensive suite of enterprise BI products that delivers a full range of analysis and reporting capabilities. You can use Oracle Business Intelligence Suite Enterprise Edition Answers and Dashboard presentation tools to customize the predefined dashboard reports that are provided with Oracle Retail Data Model.

**See:** "Reporting Approaches in Oracle Retail Data Model" on page 5-1.

## Steps for Implementing an Oracle Retail Data Model Warehouse

Although Oracle Retail Data Model was designed following best practices for retailers, usually the model requires some customization to meet your business needs.

The reasons that you might customize Oracle Retail Data Model include: your business does not have a business area that is in the logical model of Oracle Retail Data Model, or you must apply a new or different business rule. Typical physical model modifications include: adding, deleting, modifying tables; or altering foreign keys, constraints, or indexes.

To implement an Oracle Retail Data Model warehouse, take the following steps:

1. Perform the organizational tasks outlined in "Before You Begin Customizing the Oracle Retail Data Model" on page 1-4.
2. Create a fit-gap analysis report by following the process outlined "Performing Fit-Gap Analysis for Oracle Retail Data Model" on page 1-10.
3. In a development environment, install a copy of the Oracle Retail Data Model.
4. In the copy of the Oracle Retail Data Model you created in Step 3, customize Oracle Retail Data Model by making the changes you documented in the fit-gap analysis report. Make the changes in the following order:
   a. Foundation layer of the physical model and the ETL to populate that layer. When customizing the physical objects, follow the guidelines in "Foundation Layer Customization" on page 2-5. When writing the ETL, follow the guidelines in "Creating Source-ETL for Oracle Retail Data Model" on page 4-2.
   b. Access layer of the physical model and the ETL to populate that layer. When designing the physical objects, follow the guidelines in Chapter 3, "Access Layer Customization." When writing the ETL, follow the guidelines in "Customizing Intra-ETL for the Oracle Retail Data Model" on page 4-9.
5. In a test environment, make a copy of your customized version of Oracle Retail Data Model. Then, following the documentation you created in Step 2, test the customized version of Oracle Retail Data Model.
6. Following your typical procedures, roll the tested customized version of Oracle Retail Data Model out into pre-production and, then, production.

**Note:** Keep 'clean' copies of the components delivered with Oracle Retail Data Model. This is important when upgrading to later versions of Oracle Retail Data Model.

# Before You Begin Customizing the Oracle Retail Data Model

Before you begin customizing Oracle Retail Data Model, ensure the following teams and committees exist:

- Data warehouse governance steering committee. This steering committee has the responsibilities outlined in "Responsibilities of a Data Warehouse Governance Committee" on page 1-5.
- Implementation team. This team consists of IT engineers who have the expertise outlined in "Prerequisite Knowledge for Implementors" on page 1-4. This team has the responsibilities outlined in "Steps for Implementing an Oracle Retail Data Model Warehouse" on page 1-3.
- Fit-gap analysis team. This team consists of business analysts who can identify the business requirements and scope of the Oracle Retail Data Model and at least some engineers in the Implementation team. Business members of this team must understand logical data modeling so that they can evaluate what changes must be made to the foundation and access layers of the physical model. This team has the responsibilities outlined in "Performing Fit-Gap Analysis for Oracle Retail Data Model" on page 1-10.

After these teams and committees are formed:

- Discuss the approach and determine the involvement and roles of every party involved in the customization (for example, business and IT).
- Agree on the scope of the project (that is, agree on what new data must be in the data warehouse and why it is needed).
- Agree on the timing and the working arrangements.

## Prerequisite Knowledge for Implementors

As outlined in "Oracle Products That Make Up Oracle Retail Data Model" on page 1-2, the Oracle Retail Data Model uses much of the Oracle stack. Consequently, to successfully implement the Oracle Retail Data Model, the implementation team needs:

- Experience performing information and data analysis and data modeling. (Experience using Oracle SQL Developer Data Modeler, is a plus.)
- An understanding of the Oracle technology stack, especially data warehouse (Database, Data Warehouse, OLAP, Data Mining, Warehouse Builder, Oracle Business Intelligence Suite Enterprise Edition)
- Hands-on experience using:
  - Oracle Database
  - PL/SQL
  - SQL DDL and DML syntax
  - Analytic Workspace Manager
  - Oracle SQL Developer Data Modeler

  - Oracle Business Intelligence Suite Enterprise Edition Administrator, Answers, and Dashboards
- One implementation team member is familiar or has training with Oracle Retail Data Model, or the team engages a partner with Oracle Retail Data Model implementation experience.

## Responsibilities of a Data Warehouse Governance Committee

Governance is of concern to any enterprise, executive team or individual with an interest in the processes, standards, and compliance. It is even more important to organizations that have invested in data warehousing.

Data warehouse governance occurs within the context of overall IT governance. It provides the necessary policies, process and procedures, which must be clearly communicated to the entire corporation, from the IT employees to the front-end operational personnel.

Before you customize Oracle Retail Data Model, setup a data warehouse governance steering committee if one does not exist. The role of this steering committed is to oversee the data warehouse to provide an environment that reaches across the enterprise and drives the best business value.

### Data Warehouse Governance Committee: Overall Responsibilities

The data warehouse governance steering committee sets the direction and is responsible for the governance framework, including the following areas:

- The entire data warehouse life cycle
- The data to process and the data to make available to end-users
- The minimum quality criteria for the data that is available to end users and how to measure and analyze these criteria against the quality of the data that is the source data for the data warehouse.
- The business goals of the organization to apply core information from the data warehouse.
- The policies, procedures, and standards for data resources and data access.
- The life cycle of data warehouse component management.

### Data Warehouse Governance Committee: Data Governance Responsibilities

The more detailed focus in data warehouse governance is data governance. Data governance tasks include:

- Approving the data modeling standards, metadata standards and other related standards. This includes determining a metadata strategy as discussed in "Managing Metadata for Oracle Retail Data Model" on page 1-6' and identifying the data modeling tools to use that support these standards.
- Determining the data retention policy.
- Designing a data access policy based on legal restrictions and data security rules.
- Designing a data backup strategy that aligns with the impact analysis to the business unit.
- Monitoring and reporting on data usage, activity, and alerts.

# Managing Metadata for Oracle Retail Data Model

Metadata is any data about data and, as such, is an important aspect of the data warehouse environment. Metadata allows the end user and the business analyst to navigate through the possibilities without knowing the context of the data or what the data represents.

Metadata management is a comprehensive, ongoing process of overseeing and actively managing metadata in a central environment which helps an enterprise to identify how data is constructed, what data exists, and what the data means. It is particularly helpful to have good metadata management when customizing Oracle Retail Data Model so that you can do impact analysis to ensure that changes do not adversely impact data integrity anywhere in your data warehouse.


- Metadata Categories and Standards
- Working with a Metadata Repository
- Browsing the Metadata Repository Supplied With Oracle Retail Data Model
- Using the Metadata Generation Packages
- Using Oracle Warehouse Builder with the Oracle Retail Data Model


## Metadata Categories and Standards

Metadata is organized into three major categories:

- **Business metadata** describes the meaning of data in a business sense. The business interpretation of data elements in the data warehouse is based on the actual table and column names in the database. Gather this mapping information and business definition and rules information into business metadata.
- **Technical metadata** represents the technical aspects of data, including attributes such as data types, lengths, lineage, results from data profiling, and so on.
- **Process execution metadata** presents statistics on the results of running the ETL process itself, including measures such as rows loaded successfully, rows rejected, amount of time to load, and so on.

Since metadata is so important in information management, many organizations attempt to standardize metadata at various levels, such as:

- Metadata Encoding and Transmission Standard (METS). A standard for encoding descriptive, administrative, and structural metadata regarding objects within a digital library.
- American National Standards Institute (ANSI). The organization that coordinates the U.S. voluntary standardization and conformity-assessment systems.
- International Organization for Standardization (ISO). The body that establishes, develops, and promotes standards for international exchange.
- Common Warehouse Metamodel (CWM). A specification, released and owned by the Object Management Group, for modeling metadata for relational, non-relational, multi-dimensional, and most other objects found in a data warehousing environment.

When you implement your metadata management solution, reference your data warehouse infrastructure environment and make the decision which standard to follow.

## Working with a Metadata Repository

You manage metadata using a Metadata Repository. At the highest level, a Metadata Repository includes three layers of information. The layers are defined in the following order:

1. A Physical layer. This metadata layer identifies the source data.
2. A Business Model and Mapping layer. This metadata layer organizes the physical layer into logical categories and records the appropriate metadata for access to the source data.
3. The Presentation layer. This metadata layer exposes the business model entities for end-user access.

The first step in creating a Metadata Repository is to scope your metadata management needs by:

- Identifying the metadata consumers. Typically, there are business consumers and technical consumers.
- Determine the business and technical metadata requirements.
- Aligning metadata requirements to specific data elements and logical data flows.

Then:

- Decide how important each part is.
- Assign responsibility to someone for each piece.
- Decide what constitutes a consistent and working set of metadata
- Determine where to store, backup, and recover the metadata.
- Ensure that each piece of metadata is available only to those people who need it.
- Quality-assure the metadata and ensure that it is complete and up to date.
- Identify the Metadata Repository to use and how to control that repository from one place

After creating the metadata definitions, review your data architecture to ensure you can acquire, integrate, and maintain the metadata.

As the data is updated in your data warehouse day by day, you need to update the Metadata Repository. To change business rules, definitions, formulas or process especially when customizing the Oracle Retail Data Model, your first step is to survey the metadata and do an impact analysis to list all of the attributes in the data warehouse environment that would be affected by a proposed change.

## Browsing the Metadata Repository Supplied With Oracle Retail Data Model

To customize the Oracle Retail Data Model model, you must understand the dependencies among Oracle Retail Data Model components, especially how the report KPIs are mapped to the physical tables and columns and how that data has been transformed through the intra-ETL.

Oracle Retail Data Model provides a tool called the "Oracle Retail Data Model Metadata" browser that helps you discover these dependencies. When you install Oracle Retail Data Model with and the sample reports, the metadata browser is delivered as a Dashboard.

> **See:** *Oracle Retail Data Model Installation Guide* for more information on installing the sample reports and deploying the Oracle Retail Data Model RPD and webcat on the Business Intelligence Suite Enterprise Edition instance.

There are four tabs (reports) in the Oracle Retail Data Model Metadata browser:

- Measure-Entity tab: Business Areas and Measures Attributes and Entities
- Entity-Measure tab: Entity to Attribute Measures
- Program-Table tab
- Table-Program tab

#### Measure-Entity tab: Business Areas and Measures Attributes and Entities

On the Measure-Entity tab the measure descriptions, computational formulas with physical columns, physical tables, and corresponding entities can be viewed by Business Area.

To browse the data, select the business area and measure description that you are interested in.

#### Entity-Measure tab: Entity to Attribute Measures

The Entity-Measure tab displays the measures supported by the entities and how they are calculated. You can discover information about particular entities and attributes.

For example, take the following steps to learn more about an entity:

1. Select the entity.
2. Click **GO**.

#### Program-Table tab

The Program-Table tab displays the input and output tables used in the selected programs. For example, take the following steps to learn more about intra-ETL mappings:

1. Select the program type (that is, intra-ETL or report) and program name for showing particular report or intra-ETL information.
2. Select **GO**.

#### Table-Program tab

The Table-Program tab lists the Programs used by a given table and whether that table is an input or output, or both, of that program.

To discover what reports use a particular table, you must move a particular table from the right pane to the left (Selected) pane.

For example, to see the reports that use a particular table, take the following steps:

1. In the right pane of the Table-Program tab, select the table.
2. Move the table to the Selected list on the left by clicking on **<** (left arrow), and click **OK**.
3. Select **GO**.

   The reports for the selected table are displayed.

## Using the Metadata Generation Packages

As described in "Browsing the Metadata Repository Supplied With Oracle Retail Data Model" on page 1-7, Oracle Retail Data Model metadata browser lets you view and analyze metadata through a set of BIEE reports to better understand relationships between metadata objects and for end-to-end impact analysis.

You use the Oracle Retail Data Model metadata browser generation packages to generate and update the Oracle Retail Data Model metadata browser.

There are four main tables and other staging tables and views in the metadata generation package. The tables are: MD_ENTY, MD_PRG, MD_KPI, and MD_REF_ENTY_KPI; these are the tables that support metadata browser reports.

For more information, see Chapter 6, "Metadata Collection and Reports".

## Using Oracle Warehouse Builder with the Oracle Retail Data Model

Oracle Warehouse Builder provides excellent metadata management features that you can use with Oracle Retail Data Model.

Before you import source metadata into Oracle Warehouse Builder, you create a module to contain metadata definitions. The type of module you create depends on the source from which you are importing metadata into your Metadata Repository. Oracle Warehouse Builder supports many different sources and targets as discussed in *Oracle Warehouse Builder Sources and Targets Guide*. For example, when connecting to an Oracle database, Oracle Warehouse Builder queries the database dictionary to extract all needed metadata on tables, views, sequences, dimensions, cubes, data types, PL/SQL packages, and so on.

Oracle Warehouse Builder also provides an interface tool called the Metadata Dependency Manager through which you can explore dependencies among data objects, as represented by the metadata in your Oracle Warehouse Builder repository. The Metadata Dependency Manager presents dependencies in the form of interactive lineage and impact diagrams. A lineage diagram traces the data flows for an object back to the sources and displays all objects along those paths. An impact diagram identifies all the objects that are derived from the selected object.

Information provided by the Metadata Dependency Manager can help you in many circumstances. For example:

1. Starting from a target object, such as a dimension, cube, or business intelligence tool report, you can identify the columns in each data source that are used in computing the results in the target.
2. You can assess the impact of design changes in an object such as a source table or a pluggable mapping that is used throughout a project.
3. You can propagate a design change, such as a change to the data type of a source table column, to downstream objects in the design.

Using end-to-end data lineage and impact analysis reduces project risk by allowing better planning for design changes, faster identification of unanticipated impacts when source systems change, and enabling more effective auditing of your business intelligence results, master data or other data integration processes.

You can also define and use SQL-based or XML-based custom metadata stores to retrieve definitions of source and target objects such as tables and views.

For data files extracted from some mainframe sources, Oracle Warehouse Builder can interpret Cobol Copybook files that describes the structure of the data file, and create its source metadata based on that.

Oracle Warehouse Builder application adapters or application connectors provide additional metadata about ERP and CRM application sources.

Oracle Warehouse Builder can deploy or execute process flows and schedules to Oracle Enterprise Manager and Oracle Workflow. In general, you can deploy a schedule in any Oracle Database location (Oracle Database 11*g* or later).

## Performing Fit-Gap Analysis for Oracle Retail Data Model

Fit-Gap analysis is the process by which data in source tables are mapped to the Oracle Retail Data Model tables and columns and gaps are identified which would require customization. You identify any required functionality that is not included in the logical model and the default schema, and other modifications that are necessary to meet your requirements.

The result of your fit-gap analysis is a customization report which is a brief explanation of the adaptations and adjustments required to customize Oracle Retail Data Model to fit your retail environment.

The fit-gap analysis team writes the customization report by taking the following steps:

1. If you have performed previous evaluations, review the documentation from the previous phases, and if necessary add team members with the required business and technical expertise.
2. Review the data and map your logical entities and data structure with the Oracle Retail Data Model logical model and schema:
   - Starting from business requirements, questions, and rules, identify any entities and attributes that are *not* in the Oracle Retail Data Model.
   - Compare the Oracle Retail Data Model to your existing application model if you have one.
   - Compare the Oracle Retail Data Model to the transactional data that you are using as a data source to the Oracle Retail Data Model warehouse.
3. Determine the differences between your needs and Oracle Retail Data Model schema. To help you with this task, produce a list of actions people may take with the system (examples rather than models), and create use cases for appraising the functionality of the Oracle Retail Data Model Warehouse. Answer the following questions about the differences you find:
   - Which differences you can live with, and which must be reconciled?
   - What can you do about the differences you cannot live with?
4. Identify the changes you must make to the default design of Oracle Retail Data Model to create the customized warehouse. Identify these changes in the following order:
   a. Physical model. Follow the guidelines outlined in Chapter 2, "Physical Model Customization".
   b. ETL mapping. Follow the guidelines outlined in Chapter 4, "ETL Implementation and Customization" to identify and design the source-ETL that you must write and any changes you must make to the intra-ETL provided with Oracle Retail Data Model.

      **Tip:** When identifying changes, ensure that the changes meet your security and metadata requirements.

5. Write the customization report, detailing what changes are required to make the Oracle Retail Data Model match your business needs. This includes any additions and changes to interfaces to existing systems.
6. Based on the customization report, update the Project Plan and perform the steps outlined in "Steps for Implementing an Oracle Retail Data Model Warehouse" on page 1-3.

# 2

# Physical Model Customization

This chapter provides general information about customizing the physical model of Oracle Retail Data Model and more detailed information about customizing the foundation layer of the physical model. This chapter contains the following topics:


- Characteristics of the Default Physical Model
- Customizing the Oracle Retail Data Model Physical Model
- Foundation Layer Customization
- General Recommendations When Designing Physical Structures


**See also:** Chapter 3, "Access Layer Customization"

## Characteristics of the Default Physical Model

The default physical model of Oracle Retail Data Model defines:

- Over 1,250 tables and 18,500 attributes
- Over 1,800 industry measures and KPIs
- 12 pre-built data mining models
- 30 OLAP dimensions and 28 pre-built OLAP cubes

The default physical model of Oracle Retail Data Model shares characteristics of a multischema "traditional" data warehouse, as described in "Layers in a "Traditional" Data Warehouse" on page 2-1, but defines all data structures in a single schema as described in "Layers in the Default Oracle Retail Data Model Warehouse" on page 2-2.

### Layers in a "Traditional" Data Warehouse

Historically, three layers are defined for a data warehouse environment:

- **Staging layer.** This layer is used when moving data from the transactional system and other data sources into the data warehouse itself. It consists of temporary loading structures and rejected data. Having a staging layer enables the speedy extraction, transformation and loading (ETL) of data from your operational systems into data warehouse without disturbing any of the business users. It is in this layer the much of the complex data transformation and data quality processing occurs. The most basic approach for the design of the staging layer is as a schema identical to the one that exists in the source operational system.

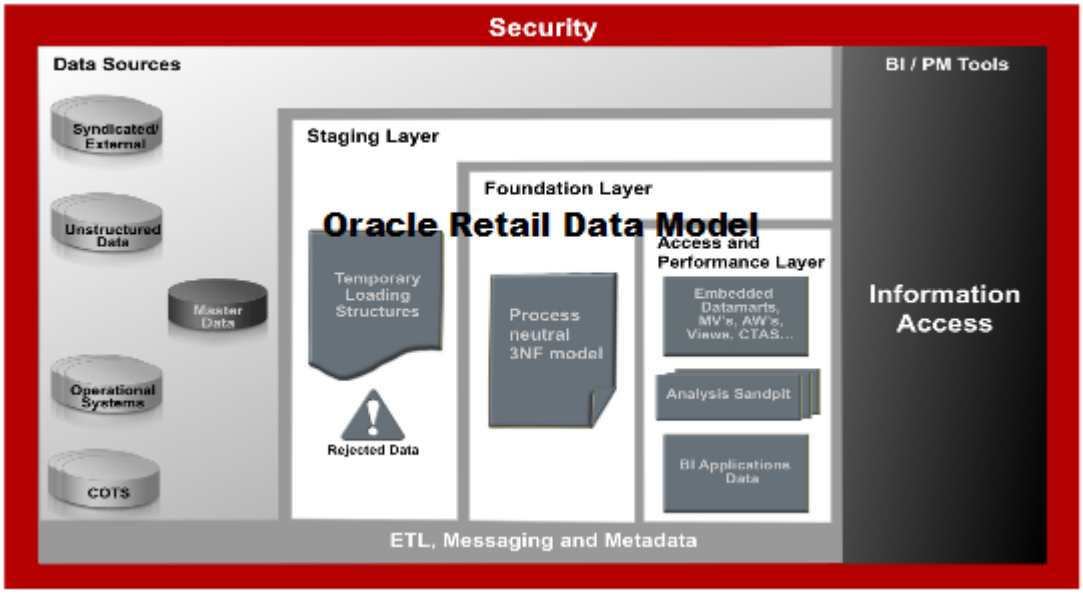
**Note:** In some implementations this layer is not necessary, because all data transformation processing is done "on the fly" as data is extracted from the source system before it is inserted directly into the foundation layer.

- **Foundation or integration layer.** This layer is traditionally implemented as a Third Normal Form (3NF) schema. A 3NF schema is a neutral schema design independent of any application, and typically has a large number of tables. It preserves a detailed record of each transaction without any data redundancy and allows for rich encoding of attributes and all relationships between data elements. Users typically require a solid understanding of the data to navigate the more elaborate structure reliably. In this layer data begins to take shape and it is not uncommon to have some end-user application access data from this layer especially if they are time sensitive, as data becomes available here before it is transformed into the Access and Performance layer.
- **Access layer.** This layer is traditionally defined as a snowflake or star schema that describes a "flattened" or dimensional view of the data.

### Layers in the Default Oracle Retail Data Model Warehouse

Oracle Retail Data Model warehouse environment also consists of three layers, as shown in Figure 2–1. Note, in the Oracle Retail Data Model the definitions of the foundation and access layers are combined in a single schema.

***Figure 2–1 Layers of an Oracle Retail Data Model Warehouse***




The layers in the Oracle Retail Data Model warehouse are:

- **Staging layer**. As in a "traditional" data warehouse environment, an Oracle Retail Data Model warehouse environment can have a staging layer. See the *Oracle Retail Data Model Release Notes* for more details on the staging layer.
- **Foundation and Access layers.** The physical objects for these layers are defined in a single schema, the `ordm_sys` schema:

- **Foundation layer.** The foundation layer of the Oracle Retail Data Model is defined by base (DWB_) and Reference (DWR_) tables that present the data in 3NF; this layer also includes the lookup and control tables defined in the ordm_sys schema (that is, the tables that have the DWL_ and DWC_ prefixes).
- **Access layer.** The Access layer of Oracle Retail Data Model is defined by derived and aggregate tables (defined with DWD_ and DWA_ prefixes), cubes (defined with a CB$ prefix), and views (that is, views defined with the _VIEW suffix). These structures provide a summarized or "flattened" perspective of the data in the foundation layer.

  This layer also contains the results of the data mining models which are stored in derived (DWD_) tables.

  **See:** *Oracle Retail Data Model Reference* for detailed information on the ordm_sys schema.

## Customizing the Oracle Retail Data Model Physical Model

The starting point for the Oracle Retail Data Model physical data model is the 3NF logical data model. The physical data model mirrors the logical model as much as possible, although some changes in the structure of the tables or columns may be necessary, and defines database objects (such as tables, cubes, and views).

To customize the default physical model of Oracle Retail Data Model take the following steps:

1. Answer the questions outlined in "Questions to Answer Before You Customize the Physical Model" on page 2-4.
2. Familiarize yourself with the characteristics of the logical and physical model of Oracle Retail Data Model as outlined in"Characteristics of the Default Physical Model" on page 2-1 and presented in detail in *Oracle Retail Data Model Reference*.
3. Modify the foundation level of your physical model of Oracle Retail Data Model, as needed. See "Common Change Scenarios When Customizing the Foundation Layer of Oracle Retail Data Model" on page 2-6 for a discussion of when customization might be necessary.

   When defining physical structures:

   - Keep the foundation layer in 3NF form.
   - Use the information presented in "General Recommendations When Designing Physical Structures" on page 2-8 to guide you when designing the physical objects.
   - Follow the conventions used when creating the default physical model of Oracle Retail Data Model as outlined in "Conventions When Customizing the Physical Model" on page 2-4.

     **Tip:** Package the changes you make to the physical data model as a patch to the ordm_sys schema.

4. Modify the access layer of your physical model of Oracle Retail Data Model as discussed in Chapter 3, "Access Layer Customization".

## Questions to Answer Before You Customize the Physical Model

When designing the physical model, remember that the logical data model is not one-to-one with the physical data model. Consider the load, query, and maintenance requirements when you are designing the physical model customizations. For example, answer the following questions before you design the physical data model:

- Identify the scope of the changes. See "Common Change Scenarios When Customizing the Foundation Layer of Oracle Retail Data Model" on page 2-6 for an overview discussion of making physical data model changes when your business needs do not result in a logical model that is the same as the Oracle Retail Data Model logical model.
- What is the result of the source data profile?
- What is the data load frequency for each table?
- How many large tables are there and which tables are these?
- How will the tables and columns be accessed? What are the common joins?
- What is your data backup strategy?

## Conventions When Customizing the Physical Model

When developing the physical model for Oracle Retail Data Model, the following conventions were used. Continue to follow these conventions as you customize the physical model.

### General Naming Conventions for Physical Objects

Follow these guidelines for naming physical objects that you define:

- When naming the physical objects follow the naming guidelines for naming objects within an Oracle Database schema. For example:
  - Table and column names must start with a letter, can use only 30 alphanumeric characters or less, cannot contain spaces or some special characters such as "!" and cannot use reserved words.
  - Table names must be unique within a schema that is shared with views and synonyms.
  - Column names must be unique within a table.
- Although it is common to use abbreviations in the physical modeling stage, as much as possible, use names for the physical objects that correspond to the names of the entities in the logical model. Use consistent abbreviations to avoid programmer and user confusion.
- When naming columns, use short names if possible. Short column names reduce the time required for SQL command parsing.
- The `ordm_sys` schema delivered with Oracle Retail Data Model uses the prefixes and suffixes shown in the following table to identify object types.

| Prefix or Suffix | Used for Name of These Objects |
|---|---|
| `CB$` | Materialized view used to support/deliver required functionality for an Oracle OLAP cube. This is an internal object built and maintained automatically by the Oracle OLAP server in the database.<br><br>**Note:** Do not report or query against this object. Instead access the corresponding `_VIEW` object. |

| Prefix or Suffix | Used for Name of These Objects |
|---|---|
| DMV_ | Materialized view created for performance reasons (that is, *not* an aggregate table or an OLAP cube). |
| DWA_ | Aggregate tables or relational materialized views (aggregate objects) |
| DWB_ | Base transaction data (3NF) tables. |
| DWC_ | Control tables. |
| DWD_ | Derived tables -- including data mining result tables. |
| DWL_ | Lookup tables. |
| DWV_ | Relational view of time dimension |
| DWR_ | Reference data tables. |
| _VIEW | A relational view of an OLAP cube, dimension, or hierarchy. |

**Note:** You should use a similar prefix and suffix, combined with an indicator for your company or project name for any new tables, views, and cubes that you define during customization. For example, if your customization project chooses a standard prefix of 'AZ', then new base tables would be created with the prefix 'AZB_', new reference tables would use the prefix 'AZR_'.

**See:** *Oracle Retail Data Model Reference* for detailed information about the objects in the default Oracle Retail Data Model.

## Foundation Layer Customization

The first step in customizing the physical model of Oracle Retail Data Model is customizing the foundation layer of the physical data model. Since, as mentioned in "Layers in the Default Oracle Retail Data Model Warehouse" on page 2-2, the foundation layer of the physical model mirrors the 3NF logical model of Oracle Retail Data Model, you might choose to customize the foundation layer to reflect differences between your logical model needs and the default logical model of Oracle Retail Data Model. Additionally, you might need to customize the physical objects in the foundation layer to improve performance (for example, you might choose to compress some foundation layer tables).

When making changes to the foundation layer, keep the following points in mind:

- When changing the foundation layer objects to reflect your logical model design, make as few changes as possible. "Common Change Scenarios When Customizing the Foundation Layer of Oracle Retail Data Model" on page 2-6 outlines the most common customization changes you will make in this regard.
- When defining new foundation layer objects or when redesigning existing foundation layer objects for improved performance, follow the "General Recommendations When Designing Physical Structures" on page 2-8 and "Conventions When Customizing the Physical Model" on page 2-4.
- Remember that changes to the foundation layer objects can also impact the access layer objects.

**Note:** Approach any attempt to change the Oracle Retail Data Model with caution. The foundation layer of the physical model of Oracle Retail Data Model has (at its core) a set of generic structures that allow it to be flexible and extensible. Before making extensive additions, deletions, or changes, ensure that you understand the full range of capabilities of Oracle Retail Data Model and that you cannot handle your requirements using the default objects in the foundation layer.

**See also:** "Example of Changing the Foundation Layer of the Oracle Retail Data Model" on page 2-7

## Common Change Scenarios When Customizing the Foundation Layer of Oracle Retail Data Model

There are several common change scenarios when customizing the foundation layer of the physical data model:

- **Additions to Existing Structures**

  If you identify business areas or processes that are not supported in the default foundation layer of the physical data model of Oracle Retail Data Model, add new tables and columns.

  Carefully study the default foundation layer of the physical data model of Oracle Retail Data Model (and the underlying logical data model) to avoid building redundant structures when making additions. If these additions add high value to your business value, communicate the additions back to the Oracle Retail Data Model Development Team for possible inclusion in future releases of Oracle Retail Data Model.

- **Deletions of Existing Structures**

  If there are areas of the model that cannot be matched to any of the business requirements of your legacy systems, it is safer to keep these structures and not populate that part of the warehouse.

  Deleting a table in the foundation layer of the physical data model can destroy relationships needed in other parts of the model or by applications based on the it. Some tables may not be needed during the initial implementation, but you may want to use these structures at a later time. If this is a possibility, keeping the structures now saves re-work later. If tables are deleted, perform a thorough analysis to identify all relationships originating from that entity.

- **Changes to Existing Structures**

  In some situations some structures in the foundation layer of the physical data model of Oracle Retail Data Model may not exactly match the corresponding structures that you use. Before implementing changes, identify the impact that the changes would have on the database design of Oracle Retail Data Model. Also identify the impact on any applications based on the new design.

**See also:** "Example of Changing the Foundation Layer of the Oracle Retail Data Model" on page 2-7

## Example of Changing the Foundation Layer of the Oracle Retail Data Model

As an example, let's examine how Oracle Retail Data Model supports the various retail services, what you might discover during fit-gap analysis, and how you might extend Oracle Retail Data Model to fit the discovered gaps.

### Entities supporting Retail services

The entities provided with the logical model of Oracle Retail Data Model that support the retail services are:

- **Customer**: An individual or organization that purchases, may purchase, or did purchase goods and or services from a store.
- **Customer Order**: The entity captures information about an order placed by a customer for merchandise or services to be provided at some future date and time.
- **SKU Item**: Stock Keeping Unit or unit identification, typically the UPC, used to track store inventory and sales. Each SKU is associated with an item, variant, product line, bundle, service, fee, or attachment. This is the lowest level of merchandise for which inventory and sales records are retained within the retail store.
- **Item Category**: Category within a subclass in the product hierarchy, as it was at a given point in time.

### The differences discovered during fit-gap analysis

Assume that during the fit-gap analysis, you discover the following need that is not supported by the logical model delivered with Oracle Retail Data Model:

Your company wants to add a new section in the retail domain. For example, if you want to add a book section in your store to other existing departments.

### Extending the physical model to support the differences

For example, to extend the physical data model, do the following:

1. Create a table named `DWR_AUTHOR` to hold the Author's information by executing the following statements:

```
CREATE TABLE DWR_AUTHOR
(
AUTHOR _ID INTEGER NOT NULL,
AUTHOR _FIRST_NAME VARCHAR2 (50) NOT NULL,
AUTHOR _LAST_NAME VARCHAR2 (50)
);
ALTER TABLE DWR_AUTHOR ADD CONSTRAINT AUTHOR_PK PRIMARY KEY (AUTHOR _ID);
```

2. Add columns in the `DWR_SKU_ITEM` table using the following statement:

```
ALTER TABLE DWR_SKU_ITEM ADD COLUMN ISBN INTEGER NULL
```

3. Create another new table named `DWR_AUTHOR_ITEM_ASGN`:

```
CREATE TABLE DWR_AUTHOR_ITEM_ASGN
(
AUTHOR _ID INTEGER NOT NULL,
SKU_ITEM_KEY NUMBER (30) NOT NULL
);
```

# General Recommendations When Designing Physical Structures

The `ordm_sys` schema delivered with Oracle Retail Data Model was designed and defined following best practices for data access and performance. Continue to use these practices when you add new physical objects. This section provides information about how decisions about the following physical design aspects were made to the default Oracle Retail Data Model:

- Tablespaces in Oracle Retail Data Model
- Data Compression in Oracle Retail Data Model
- Surrogate Keys in the Physical Model
- Integrity Constraints in Oracle Retail Data Model
- Indexes and Partitioned Indexes in Oracle Retail Data Model
- Partitioned Tables in Oracle Retail Data Model
- Parallel Execution in Oracle Retail Data Model

## Tablespaces in Oracle Retail Data Model

A tablespace consists of one or more data files, which are physical structures within the operating system you are using.

### Recommendations: Defining Tablespaces

If possible, define tablespaces so that they represent logical business units.

Use ultra large data files for a significant improvement in very large Oracle Retail Data Model warehouse.

### Changing the Tablespace and Partitions Used by Tables

You can change the tablespace and partitions used by Oracle Retail Data Model tables. What you do depends on whether the Oracle Retail Data Model table has partitions:

- For tables that do not have partitions (that is, lookup tables and reference tables), you can change the existing tablespace for a table.

  By default, Oracle Retail Data Model defines the partitioned tables as interval partitioning, which means the partitions are created only when new data arrives.

  Consequently, for Oracle Retail Data Model tables that have partitions (that is, Base, Derived, and Aggregate tables), for the new interval partitions to be generated in new tablespaces rather than current ones, issue the following statements.

  ```
  ALTER TABLE table_name MODIFY DEFAULT ATTRIBUTES
  TABLESPACE new_tablespace_name;
  ```

  When new data is inserted in the table specified by *table_name*, a new partition is automatically created in the tablespace specified as *new_tablespace_name*.

- For tables that have partitions (that is, base, derived, and aggregate tables), you can specify that new interval partitions be generated into new tablespaces.

  For Oracle Retail Data Model tables that do not have partitions, that is lookup tables and reference tables, change the existing tablespace for a table with the following statement.

  ```
  ALTER TABLE table_name MOVE TABLESPACE new_tablespace_name;
  ```

## Data Compression in Oracle Retail Data Model

A key decision that you must make is whether to compress your data. Using table compression reduces disk and memory usage, often resulting in better scale-up performance for read-only operations. Table compression can also speed up query execution by minimizing the number of round trips required to retrieve data from the disks. Compressing data however imposes a performance penalty on the load speed of the data.

### Recommendations: Data Compression

In general, choose to compress the data. The overall performance gain typically outweighs the cost of compression.

If you decide to use compression, consider sorting your data before loading it to achieve the best possible compression rate. The easiest way to sort incoming data is to load it using an `ORDER BY` clause on either your `CTAS` or `IAS` statement. Specify an `ORDER BY` a `NOT NULL` column (ideally non numeric) that has a large number of distinct values (1,000 to 10,000).

> **See also:** "Types of Data Compression Available" on page 2-9 and "Compressing Materialized Views" on page 3-24.

### Types of Data Compression Available

Oracle Database offers the following types of compression:

- Basic or Standard Compression
- OLTP Compression
- Hybrid Columnar Compression (HCC)

**Basic or Standard Compression** With standard compression Oracle Database compresses data by eliminating duplicate values in a database block. Standard compression only works for direct path operations (CTAS or IAS). If the data is modified using any kind of conventional DML operation (for example updates), the data within that database block is uncompressed to make the modifications and is written back to disk uncompressed.

By using a compression algorithm specifically designed for relational data, Oracle Database can compress data effectively and in such a way that Oracle Database incurs virtually no performance penalty for SQL queries accessing compressed tables.

Oracle Retail Data Model leverages this compress feature which reduces the amount of data being stored, reduces memory usage, and increases query performance.

You can specify table compression using the `COMPRESS` clause of the `CREATE TABLE` statement or you can enable compression for an existing table using `ALTER TABLE` statement as shown:

```
alter table <tablename> move compress;
```

**OLTP Compression** OLTP compression is a component of the Advanced Compression option. With OLTP compression, just like standard compression, Oracle Database compresses data by eliminating duplicate values in a database block. But unlike standard compression OLTP compression allows data to remain compressed during all types of data manipulation operations, including conventional DML such as `INSERT` and `UPDATE`.

**See:** *Oracle Database Administrator's Guide* for more information on OLTP table compression features.

---

**Oracle by Example:** For more information on Oracle Advanced Compression, see the "Using Table Compression to Save Storage Costs" OBE tutorial.

To access the tutorial, open the Oracle Learning Library in your browser by following the instructions in "Oracle Technology Network" on page xii; and, then, search for the tutorial by name.

---

**Hybrid Columnar Compression (HCC)** HCC is available with some storage formats and achieves its compression using a logical construct called the compression unit which is used to store a set of hybrid columnar-compressed rows. When data is loaded, a set of rows is pivoted into a columnar representation and compressed. After the column data for a set of rows has been compressed, it is fit into the compression unit. If conventional DML is issued against a table with HCC, the necessary data is uncompressed to do the modification and then written back to disk using a block-level compression algorithm.

**Tip:** If your data set is frequently modified using conventional DML, then the use of HCC is not recommended; instead, the use of OLTP compression is recommended.

HCC provides different levels of compression, focusing on query performance or compression ratio respectively. With HCC optimized for query, fewer compression algorithms are applied to the data to achieve good compression with little to no performance impact. However, compression for archive tries to optimize the compression on disk, irrespective of its potential impact on the query performance.

**See also:** The discussion on HCC in *Oracle Database Concepts*.

## Surrogate Keys in the Physical Model

The surrogate key method for primary key construction involves taking the natural key components from the source systems and mapping them through a process of assigning a unique key value to each unique combination of natural key components (including source system identifier). The resulting primary key value is completely non-intelligent and is typically a numeric data type for maximum performance and storage efficiency.

**Advantages of Surrogate keys include:**

- Ensure uniqueness: data distribution
- Independent of source systems
- Re-numbering
- Overlapping ranges
- Uses the numeric data type which is the most performant data type for primary keys and joins

**Disadvantages of Surrogate keys:**

- Requires allocation during ETL

- Complex and expensive re-processing and data quality correction
- Not used in queries – performance impact
- The operational business intelligence requires natural keys to join or trace back to source operational systems

## Integrity Constraints in Oracle Retail Data Model

Integrity constraints are used to enforce business rules associated with your database and to prevent having invalid information in the tables.

The most common types of constraints include:

- `PRIMARY KEY` constraints, this is usually defined on the surrogate key column to ensure uniqueness of the record identifiers. In general, it is recommended that you specify the `ENFORCED ENABLED RELY` mode.
- `UNIQUE` constraints, to ensure that a given column (or set of columns) is unique. For slowly changing dimensions, it is recommended that you add a unique constraint on the Business Key and the Effective From Date columns to allow tracking multiple versions (based on surrogate key) of the same Business Key record.
- `NOT NULL` constraints, to ensure that no null values are allowed. For query rewrite scenarios, it is recommended that you have an inline explicit `NOT NULL` constraint on the primary key column in addition to the primary key constraint.
- `FOREIGN KEY` constraints, to ensure that relation between tables are being honored by the data. Usually in data warehousing environments, the foreign key constraint is present in `RELY DISABLE NOVALIDATE` mode.

The Oracle Database uses constraints when optimizing SQL queries. Although constraints can be useful in many aspects of query optimization, constraints are particularly important for query rewrite of materialized views. Under some specific circumstances, constraints need space in the database. These constraints are in the form of the underlying unique index.

Unlike data in many relational database environments, data in a data warehouse is typically added or modified under controlled circumstances during the extraction, transformation, and loading (ETL) process.

## Indexes and Partitioned Indexes in Oracle Retail Data Model

Indexes are optional structures associated with tables or clusters. In addition to the classical B-tree indexes, bitmap indexes are very common in data warehousing environments

- Bitmap indexes are optimized index structures for set-oriented operations. Additionally, they are necessary for some optimized data access methods such as star transformations. Bitmap indexes are typically only a fraction of the size of the indexed data in the table.
- B-tree indexes are most effective for high-cardinality data: that is, for data with many possible values, such as `customer_name` or `phone_number`. However, fully indexing a large table with a traditional B-tree index can be prohibitively expensive in terms of disk space because the indexes can be several times larger than the data in the table. B-tree indexes can be stored specifically in a compressed manner to enable huge space savings, storing more keys in each index block, which also leads to less I/O and better performance.

#### Recommendations: Indexes and Partitioned Indexes

Make the majority of the indexes in your customized Oracle Retail Data Model bitmap indexes.

Use B-tree indexes only for unique columns or other columns with very high cardinalities (that is, columns that are almost unique). Store the B-tree indexes in a compressed manner.

Partition the indexes. Indexes are just like tables in that you can partition them, although the partitioning strategy is not dependent upon the table structure. Partitioning indexes makes it easier to manage the data warehouse during refresh and improves query performance.

Typically, specify the index on a partitioned table as local. Bitmap indexes on partitioned tables must always be local. B-tree indexes on partitioned tables can be global or local. However, in a data warehouse environment, local indexes are more common than global indexes. Use global indexes only when there is a specific requirement which cannot be met by local indexes (for example, a unique index on a nonpartitioning key, or a performance requirement).

> **See also:** "Partitioned Tables in Oracle Retail Data Model" on page 2-12, "Choosing Indexes for Materialized Views" on page 3-22, "Choosing a Cube Partitioning Strategy for Oracle Retail Data Model" on page 3-17, and "Partitioning and Materialized Views" on page 3-22.

## Partitioned Tables in Oracle Retail Data Model

Partitioning allows a table, index or index-organized table to be subdivided into smaller pieces. Each piece of the database object is called a partition. Each partition has its own name, and may optionally have its own storage characteristics. From the perspective of a database administrator, a partitioned object has multiple pieces that can be managed either collectively or individually. This gives the administrator considerable flexibility in managing partitioned objects. However, from the perspective of the application, a partitioned table is identical to a nonpartitioned table. No modifications are necessary when accessing a partitioned table using SQL DML commands.

As discussed in the following topics, partitioning can provide tremendous benefits to a wide variety of applications by improving manageability, availability, and performance:

- Partitioning the Oracle Retail Data Model for Manageability
- Partitioning the Oracle Retail Data Model for Easier Data Access
- Partitioning the Oracle Retail Data Model for Join Performance

---

**Oracle by Example:** To understand the various partitioning techniques in Oracle Database, see the "Manipulating Partitions in Oracle Database 11*g*" OBE tutorial.

To access the tutorial, open the Oracle Learning Library in your browser by following the instructions in "Oracle Technology Network" on page xii; and, then, search for the tutorial by name.

---

**See also:** "Indexes and Partitioned Indexes in Oracle Retail Data Model" on page 2-11, "Choosing a Cube Partitioning Strategy for Oracle Retail Data Model" on page 3-17, and "Partitioning and Materialized Views" on page 3-22.

### Partitioning the Oracle Retail Data Model for Manageability

Range partitioning helps improve the manageability and availability of large volumes of data (Oracle Retail Data Model uses Interval partitioning which is an extension to range partitioning). For more information, see *Oracle Database VLDB and Partitioning Guide*.

Consider the case where two year's worth of sales data or 100 terabytes (TB) is stored in a table. At the end of each day a new batch of data must be to loaded into the table and the oldest days worth of data must be removed. If the `Sales` table is range partitioned by day then the new data can be loaded using a partition exchange load. This is a sub-second operation that has little or no impact on end user queries.

### Partitioning the Oracle Retail Data Model for Easier Data Access

Range partitioning also helps ensure that only the necessary data to answer a query is scanned (Oracle Retail Data Model uses Interval partitioning which is an extension to range partitioning). Consider the case where business users predominately accesses the sales data on a weekly basis (for example, total sales per week) then range partitioning this table by day ensures that the data is accessed in the most efficient manner, as only seven partitions must be scanned to answer the business users query instead of the entire table. The ability to avoid scanning irrelevant partitions is known as partition pruning. For more information, see *Oracle Database VLDB and Partitioning Guide*.

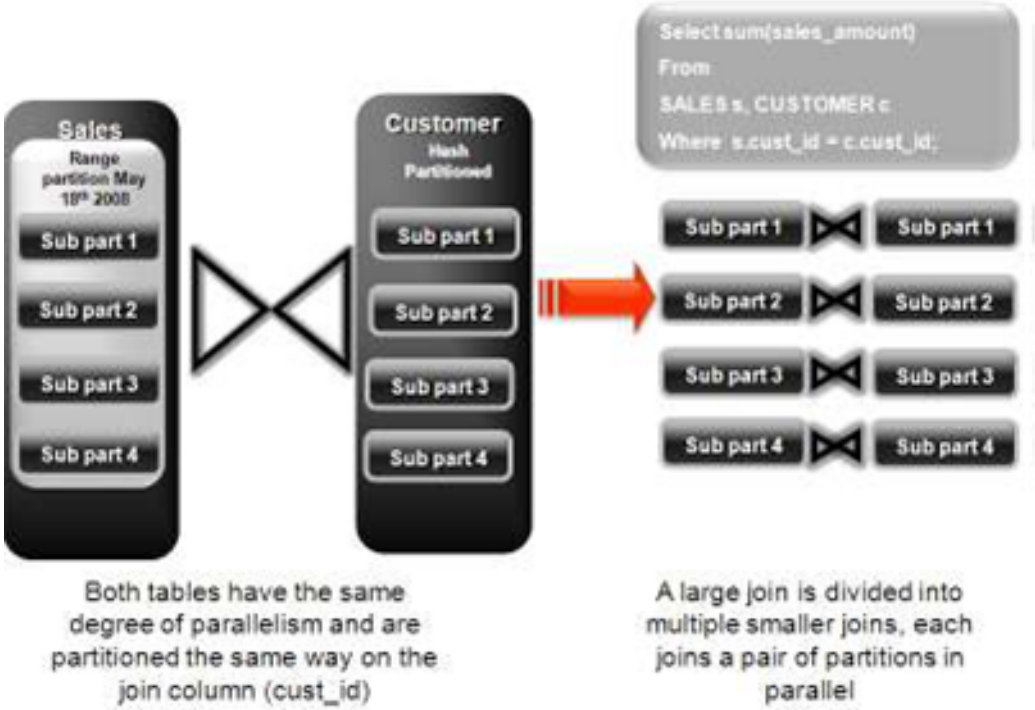
### Partitioning the Oracle Retail Data Model for Join Performance

Sub-partitioning by hash is used predominately for performance reasons. Oracle Database uses a linear hashing algorithm to create sub-partitions.

A major performance benefit of hash partitioning is partition-wise joins. Partition-wise joins reduce query response time by minimizing the amount of data exchanged among parallel execution servers when joins execute in parallel. This significantly reduces response time and improves both CPU and memory resource usage. In a clustered data warehouse, this significantly reduces response times by limiting the data traffic over the interconnect (IPC), which is the key to achieving good scalability for massive join operations. Partition-wise joins can be full or partial, depending on the partitioning scheme of the tables to be joined.

As illustrated in Figure 2–2, a full partition-wise join divides a join between two large tables into multiple smaller joins. Each smaller join, performs a joins on a pair of partitions, one for each of the tables being joined. For the optimizer to choose the full partition-wise join method, both tables must be equi-partitioned on their join keys. That is, they have to be partitioned on the same column with the same partitioning method. Parallel execution of a full partition-wise join is similar to its serial execution, except that instead of joining one partition pair at a time, multiple partition pairs are joined in parallel by multiple parallel query servers. The number of partitions joined in parallel is determined by the Degree of Parallelism (DOP).

*Figure 2–2 Partitioning for Join Performance*




#### Recommendations: Number of Hash Partitions

To ensure that the data gets evenly distributed among the hash partitions it is highly recommended that the number of hash partitions is a power of 2 (for example, 2, 4, 8, and so on). A good rule of thumb to follow when deciding the number of hash partitions a table should have is `2 X # of CPUs` rounded to up to the nearest power of 2.

If your system has 12 CPUs, then 32 would be a good number of hash partitions. On a clustered system the same rules apply. If you have 3 nodes each with 4 CPUs, then 32 would still be a good number of hash partitions. However, ensure that each hash partition is at least 16MB. Many small partitions do not have efficient scan rates with parallel query. Consequently, if using the number of CPUs makes the size of the hash partitions too small, use the number of Oracle RAC nodes in the environment (rounded to the nearest power of 2) instead.

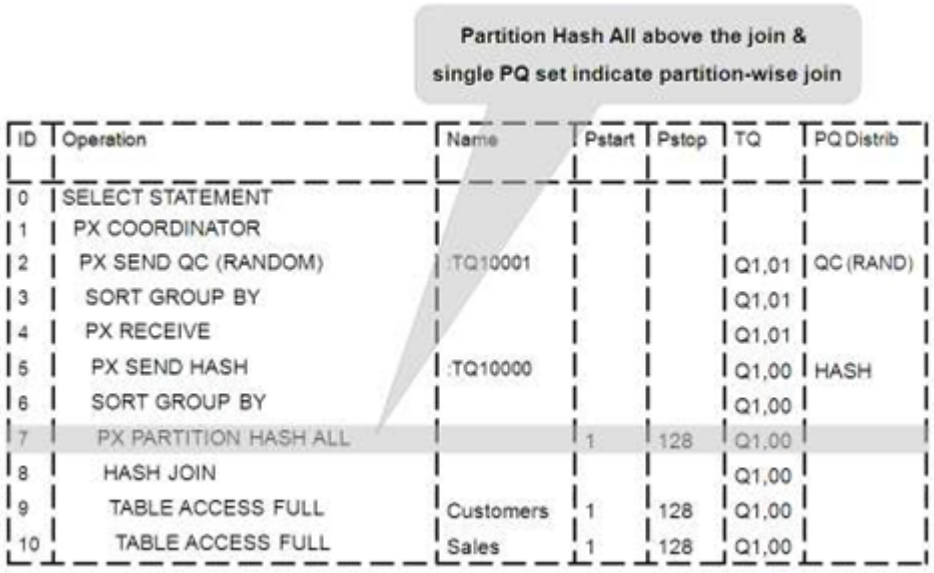
## Parallel Execution in Oracle Retail Data Model

Parallel Execution enables a database task to be parallelized or divided into smaller units of work, thus allowing multiple processes to work concurrently. By using parallelism, a terabyte of data can be scanned and processed in minutes or less, not hours or days.

Figure 2–3 illustrates the parallel execution of a full partition-wise join between two tables, Sales and Customers. Both tables have the same degree of parallelism and the same number of partitions. They are range partitioned on a date field and sub partitioned by hash on the cust_id field. As illustrated in the picture, each partition pair is read from the database and joined directly.

There is no data redistribution necessary, thus minimizing IPC communication, especially across nodes. Figure 2–3 shows the execution plan you would see for this join.

***Figure 2–3 Parallel Execution of a Full Partition-Wise Join Between Two Tables***



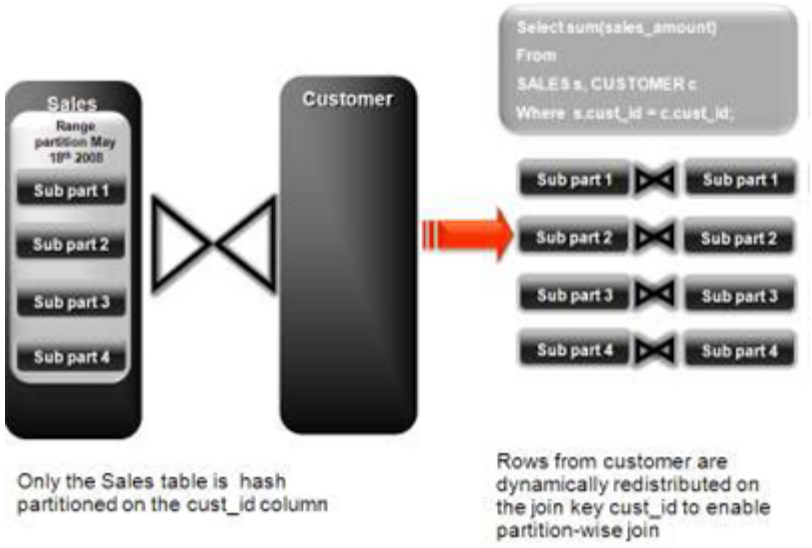

To ensure that you get optimal performance when executing a partition-wise join in parallel, use the degree of parallelism as a multiple of the number of partitions. The number of partitions should be multiple of the number of cores. To get best performance use degree of parallelism that is the same as the number of partitions to be processed in a query (this should be equal to number of CPU cores).

What happens if only one table that you are joining is partitioned? In this case the optimizer could pick a partial partition-wise join. Unlike full partition-wise joins, partial partition-wise joins can be applied if only one table is partitioned on the join key. Hence, partial partition-wise joins are more common than full partition-wise joins. To execute a partial partition-wise join, Oracle Database dynamically repartitions the other table based on the partitioning strategy of the partitioned table.

After the other table is repartitioned, the execution is similar to a full partition-wise join. The redistribution operation involves exchanging rows between parallel execution servers. This operation leads to interconnect traffic in Oracle RAC environments, since data must be repartitioned across node boundaries.

Figure 2–4 illustrates a partial partition-wise join. It uses the same example as in Figure 2–3, except that the customer table is not partitioned. Before the join operation is executed, the rows from the customers table are dynamically redistributed on the join key.

***Figure 2–4 Partial Partition-Wise Join***




## Enabling Parallel Execution for a Session

Parallel query is the most commonly used parallel execution feature in Oracle Database. Parallel execution can significantly reduce the elapsed time for large queries. To enable parallelization for an entire session, execute the following statement.

```
alter session enable parallel query;
```

## Enabling Parallel Execution of DML Operations

Data Manipulation Language (DML) operations such as `INSERT`, `UPDATE`, and `DELETE` can be parallelized by Oracle Database. Parallel execution can speed up large DML operations and is particularly advantageous in data warehousing environments. To enable parallelization of DML statements, execute the following statement.

```
alter session enable parallel dml;
```

When you issue a DML statement such as an `INSERT`, `UPDATE`, or `DELETE`, Oracle Database applies a set of rules to determine whether that statement can be parallelized. The rules vary depending on whether the statement is a DML `INSERT` statement, or a DML `UPDATE` or `DELETE` statement.

- The following rules apply when determining how to parallelize DML `UPDATE` and `DELETE` statements:
  - Oracle Database can parallelize `UPDATE` and `DELETE` statements on partitioned tables, but only when multiple partitions are involved.
  - You cannot parallelize `UPDATE` or `DELETE` operations on a nonpartitioned table or when such operations affect only a single partition.
- The following rules apply when determining how to parallelize DML `INSERT` statements:
  - Standard `INSERT` statements using a `VALUES` clause cannot be parallelized.
  - Oracle Database can parallelize only `INSERT . . . SELECT . . . FROM` statements.

### Enabling Parallel Execution at the Table Level

The setting of parallelism for a table influences the optimizer. Consequently, when using parallel query, also enable parallelism at the table level by issuing the following statement.

```
alter table <table_name> parallel 32;
```

# 3

# Access Layer Customization

This chapter provides information about customizing the access layer of Oracle Retail Data Model. It includes the following topics:

- Introduction to Customizing the Access Layer of Oracle Retail Data Model
- Derived Tables in the Oracle Retail Data Model
- Dimensional Components in the Oracle Retail Data Model
- Materialized Views in the Oracle Retail Data Model

**See also:** Chapter 2, "Physical Model Customization"

## Introduction to Customizing the Access Layer of Oracle Retail Data Model

The access layer of Oracle Retail Data Model provides the calculated and summarized ("flattened") perspectives of the data needed by business intelligence tools. Access layer objects are populated using the data from the foundation layer 3NF objects.

The access layer objects in the `ordm_sys` schema include: derived tables, aggregate objects, OLAP cubes, and materialized views. This layer also contains data mining models. The results of the these models are stored in derived tables.

When designing and customizing access layer objects:

- Follow the general guidelines for customizing physical objects given in "General Recommendations When Designing Physical Structures" on page 2-8.
- Design the access layer objects to support the business intelligence reports and queries that your site makes. See Chapter 5, "Report and Query Customization."

The following topics provide specialized information about designing and customizing access layer objects:

- Derived Tables in the Oracle Retail Data Model
- Dimensional Components in the Oracle Retail Data Model
- Materialized Views in the Oracle Retail Data Model

## Derived Tables in the Oracle Retail Data Model

Derived tables contain data which is generated from foundation tables using transformation or aggregation operations (or sometimes, both). There are some derived tables such as `DWD_RTL_SL_RETRN_ITEM_DAY` which are populated after performing minimal aggregations as well as transformations on Foundation layer

data. Derived tables have a DWD_ prefix and typically contain data at a specific granularity of time (typically, at DAY level of the time dimension).

There are two main types of derived tables in the default Oracle Retail Data Model and the way you customize these tables varies by type:

- Tables that hold the results of a calculation such as the DWD_EMP_LBR table that contains employee labor details at the day level. For information on customizing these tables, see "Creating New Derived Tables for Calculated Data" on page 3-2.
- Result tables for the data mining models (for example, DWD_CUST_MNNG). For information on customizing data mining models, see "Customizing Oracle Retail Data Model Data Mining Models" on page 3-2.

> **See:** The Derived Tables topic in *Oracle Retail Data Model Reference* for a list of all of the derived tables in the default Oracle Retail Data Model. For a list of only those derived tables that are results tables for the data mining models, see the chapter on Data Mining Models in *Oracle Retail Data Model Reference*.

## Creating New Derived Tables for Calculated Data

If, during fit-gap analysis, you identified a need for calculated data that is not provided by the default derived tables, you can meet this need by defining new tables. When designing these tables, name the tables following the convention of using the DWD_ prefix for derived tables.

## Customizing Oracle Retail Data Model Data Mining Models

Some derived (DWD_) tables in the default ordm_sys schema are the results of data mining models defined in the default Oracle Retail Data Model. Those models are also defined in the default ordm_sys schema.

All Oracle Retail Data Model mining models use materialized views as source input. Those materialized views are defined in ordm_mining_etl.sql file in $ORACLE_HOME/ordm/pdm/mining/src. Each mining model uses a different materialized view as its source.

When creating a customized Oracle Retail Data Model warehouse, data mining models may be customized as follows:

- Create a model as discussed in "Creating a New Data Mining Model for Oracle Retail Data Model" on page 3-2.
- Modify an existing model as discussed in "Modifying Oracle Retail Data Model Data Mining Models" on page 3-3.

> **See also:** "Tutorial: Customizing the Customer Life Time Value Prediction Data Mining Model" on page 3-3.

### Creating a New Data Mining Model for Oracle Retail Data Model

To write a new data mining model:

1. Ensure that the ordm_sys schema includes a definition for a materialized view that you can use as input to the model. Define a new materialized view, if necessary.
2. Create the model as you would any data mining model. Follow the instructions given in *Oracle Data Mining Concepts*. Add the model to the ordm_sys schema.

3. Add any physical tables needed by the model into the `ordm_sys` schema. Follow the naming conventions outlined in "Conventions When Customizing the Physical Model" on page 2-4 and use a `DWD_` prefix for results tables.
4. In the `ordm_sys` schema, grant `SELECT` privileges to the results tables created in Step 3.
5. Modify the intra-ETL to support the use of the data mining model.

### Modifying Oracle Retail Data Model Data Mining Models

To customize Oracle Retail Data Model mining models, take the following steps:

1. Change the definition for source materialized views used as input to the mining model.
2. Train the model again by calling Oracle Retail Data Model mining package.
3. Ensure that the model reflects the new definition (for example, that a new column has been added).

***Example 3–1 Adding a New Column to a Mining Model in Oracle Retail Data Model***

To add a new column to `create_cust_ltv_glmr`, take the following steps:

1. Add the new column to the following materialized view that is used as input to `create_cust_ltv_glmr`.

   ```
   DMV_CUST_ACCT_SRC
   ```

2. Train the model by issuing the following statement:

   ```
   exec pkg_ordm_mining.create_cust_ltv_glmr;
   ```

3. Execute the following statement to query the result table and ensure the new column name is included in the query result:

```
SELECT DISTINCT attribute_name FROM TABLE(DBMS_DATA_MINING.GET_MODEL_DETAILS_GLM('CUST_LTV_GLMR'));
```

> **See also:** "Refreshing Oracle Retail Data Model Data Mining Models" on page 4-19, and "Troubleshooting Data Mining Model Creation" on page 4-22.

### Tutorial: Customizing the Customer Life Time Value Prediction Data Mining Model

After you have populated Oracle Retail Data Model foundation layer and executed the intra-ETL to populate derived tables, you can leverage the prebuilt Oracle Retail Data Model data mining models for more advanced analysis and predictions.

This tutorial shows how to predict the Life Time Value of customers who are registered members at a retail store, for the next three years based on populated Oracle Retail Data Model warehouse. Using prebuilt Oracle Retail Data Model data mining models you can easily and very quickly see the prediction results of your customers, without having to go through all of the data preparation, training, testing, and applying process that you must perform in a traditional from-scratch mining project. See *Oracle Data Mining Concepts* for more information about the Oracle Database mining training and scoring (applying) process.

After initially generating a data mining model, as time goes by, the customer information, behavior, and purchase history change. Consequently, you must refresh the previously trained data mining models based on the latest customer and usage data. You can follow the process in this tutorial to refresh the data mining models to acquire predictions based the on latest customer information.

This tutorial shows you how to investigate the Customer Life Time Value Prediction model through Oracle Retail Data Model mining APIs. To use different parameters in the training process, or customize the model in more advanced fashion, you can either modify mining settings tables, the tables with DM_ as prefix, or use the Oracle Data Miner GUI tool (an extension to Oracle SQL Developer).

This tutorial consists of the following:

- Tutorial Prerequisites
- Preparing Your Environment
- Generating the Model
- Checking the Result

**Tutorial Prerequisites** Before starting this tutorial:

1. Review the Oracle by Example (OBE) tutorial "Using Oracle Data Miner 11g Release 2." To access this tutorial, open the Oracle Learning Library in your browser by following the instructions in "Oracle Technology Network" on page xii; and, then, search for the tutorial by name.
2. Install Oracle Retail Data Model.
3. Populate the base, reference, and lookup tables.
4. Execute the intra-ETL.

Ensure that the following tables contain valid data:

```
DWB_RTL_TRX

DWR_CUST

DWR_BSNS_MO

DWR_CUST_ACCT

DWR_CUST_RSTRCT_INFO
```

---

**Note:** If you have not populated Oracle Retail Data Model with data from operational systems using sample data will not result in meaningful mining model outcomes. To understand the data mining models, you can use the sample data provided with Oracle Retail Data Model by taking the following steps:

- Ensure that during the install, you generated the calendar data covering range of 2005-2012. For example, the parameters of starting from 20050101 for 8 years satisfy this condition.
- Download the sample data (ordm_mining_sample.zip) and import the data into your new `ordm_sys` schema.

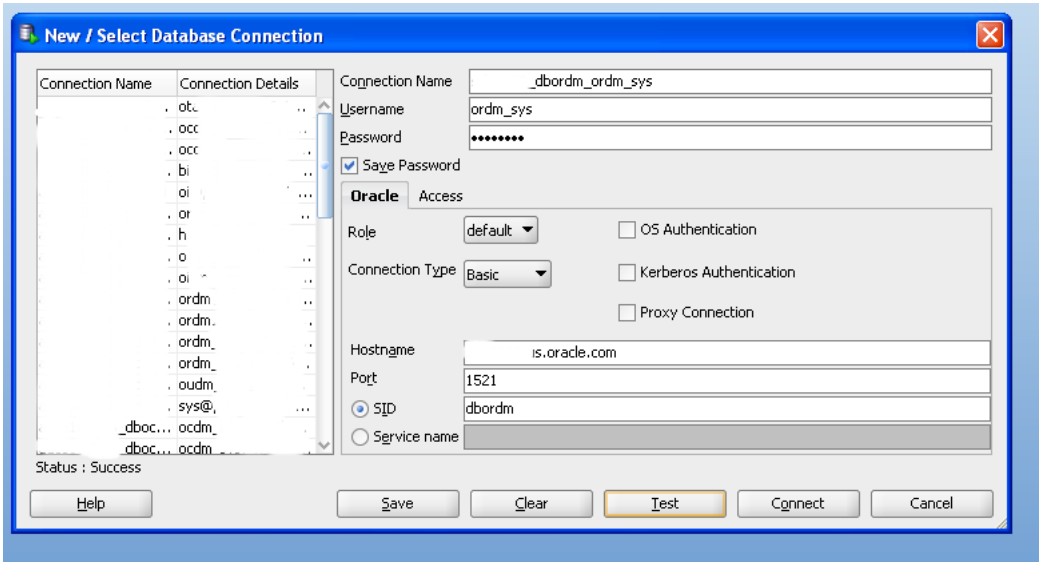
---

**Preparing Your Environment** This tutorial requires a valid, populated Oracle Retail Data Model warehouse.

To prepare the environment, do the following:

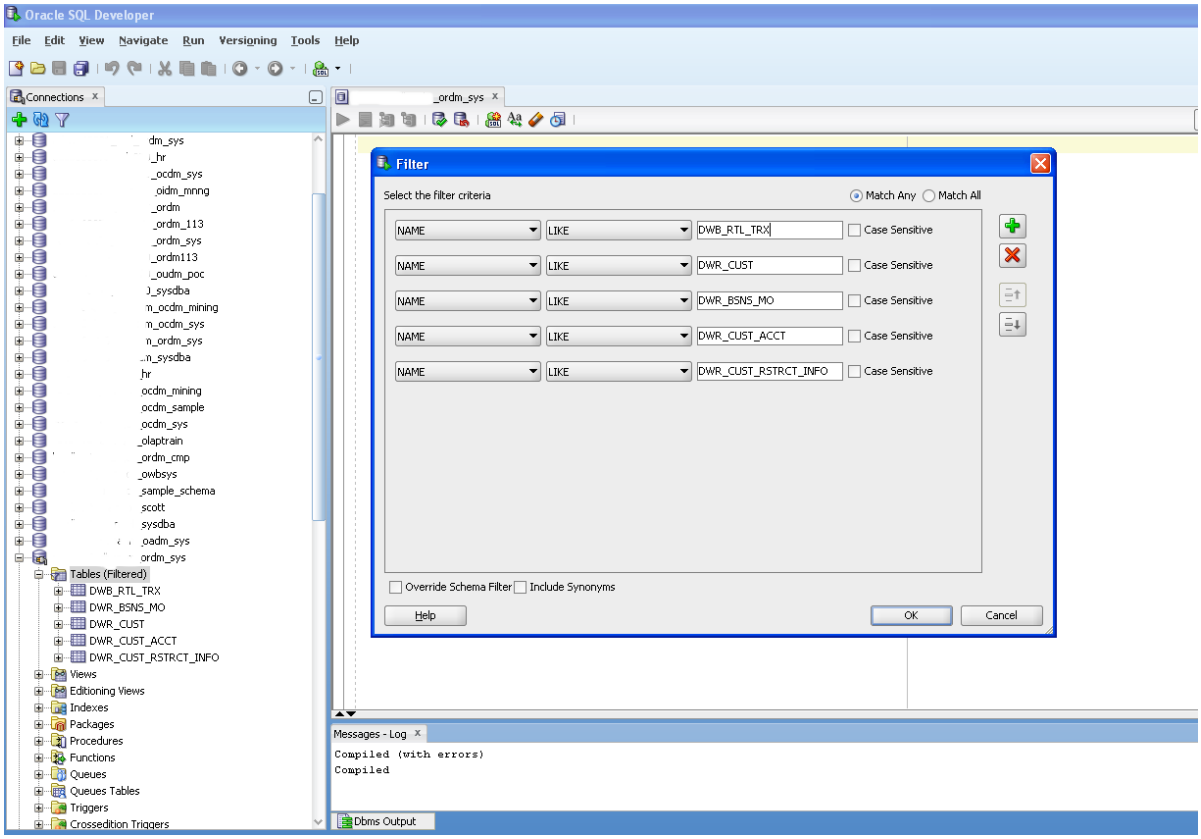
1. In Oracle SQL Developer, connect to the `ordm_sys` schema, as shown in Figure 3–1.

***Figure 3–1 Oracle SQL Developer with ORDM_SYS Schema***




2. After you connect to the `ordm_sys` schema, you can see all of the tables. You can narrow down the list by right clicking "Tables" and then applying filters, as shown in Figure 3–2.

***Figure 3–2 Applying Filters***



3. (Optional) As mentioned in "Tutorial Prerequisites", if you have not populated the tables with your own data, you can use the sample data. After you download the sample data, take the following steps to import the data:

   a. Grant dba to `ordm_sys` using the following statement:

      ```
      grant dba to ordm_sys
      ```

   b. Disable all foreign keys on those tables required by the tutorial. First, issue the following statement that generates SQL statements:

      ```
      SELECT 'ALTER TABLE ' || table_name || ' DISABLE CONSTRAINT ' ||
      CONSTRAINT_NAME || ' CASCADE;' FROM all_constraints
      WHERE status='ENABLED' AND owner='ORDM_SYS'
      AND constraint_type=
      'R' and table_name IN
      ('DWB_RTL_TRX','DWR_CUST','DWR_BSNS_MO',
      'DWR_CUST_ACCT','DWR_CUST_RSTRCT_INFO') ;
      ```

      Then, to actually disable the foreign keys for the following tables, execute the SQL statements generated by the previous SELECT statement:

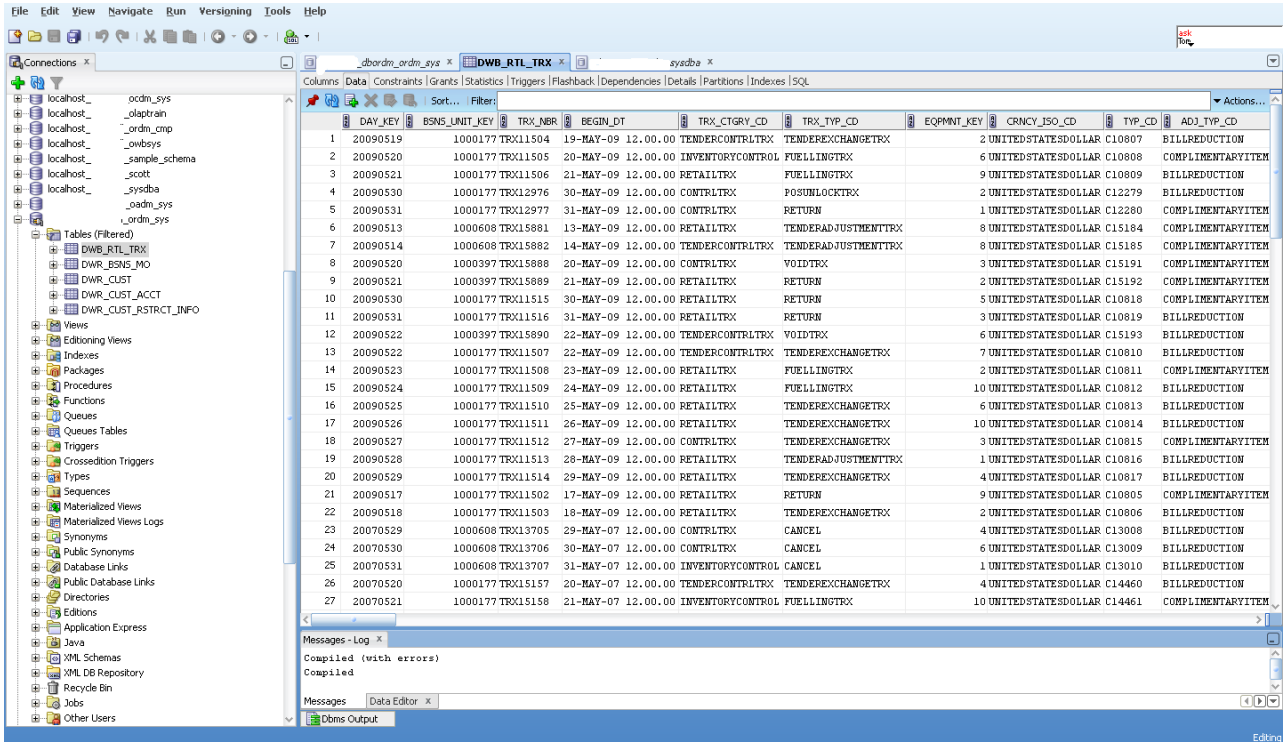
      ```
      DWB_RTL_TRX
      DWR_CUST
      DWR_BSNS_MO
      DWR_CUST_ACCT
      DWR_CUST_RSTRCT_INFO
      ```

   c. Ensure that the sample dump, `ordm_mining_sample.dmp`, is in default data dump directory, DATA_PUMP_DIR. Then, import the sample mining dump into `ordm_sys` schema by issuing the following statement (replace *password* with your password for `ordm_sys`):

      ```
      impdp ordm_sys/password directory=DATA_DUMP_DIR dumpfile=ordm_mining_
      sample.dmp content=DATA_ONLY table_exist_action=truncate TABLES=ordm_
      sys.DWB_RTL_TRX, ordm_sys.DWR_CUST, ordm_sys.DWR_BSNS_MO, ordm_sys.DWR_
      CUST_ACCT , ordm_sys.DWR_CUST_RSTRCT_INFO
      ```

4. Review the tables to ensure that they contain valid data (using either your customer data or the sample mining data), as shown in Figure 3–3.

***Figure 3–3 Review Tables to Ensure Valid Data***




5. Check the mining result table DWD_CUST_MNNG is empty before executing the model procedure in Oracle Retail Data Model Mining APIs.

**Generating the Model** This tutorial uses two procedures from Oracle Retail Data Model APIs:

- `pkg_ordm_mining.refresh_mining_source`: refreshes all mining source materialized views.
- `pkg_ordm_mining.create_cust_ltv_glmr`: generates the Customer Life Time Value Prediction Model.

Take the following steps to use these procedures:

1. Refresh the Oracle Retail Data Model mining source materialized views by executing the following SQL statements:

```
SELECT count(*) FROM dmv_cust_acct_src;
EXEC pkg_ordm_mining.refresh_mining_source;
SELECT count(*)FROM dmv_cust_acct_src;
```

   These statements do the following:

   - Display the number of records in `DMV_CUST_ACCT_SRC` materialized view before materialized view refresh.
   - Refresh the mining source materialized views.
   - Display the number of records in `DMV_CUST_ACCT_SRC` materialized view after materialized view refresh.

2. Generate the Customer Life Time Value Prediction Model by executing the following statements:

```
SELECT count(*) FROM dwd_cust_mnng;
```

```
EXEC pkg_ordm_mining. create_cust_ltv_glmr;
SELECT count(*) FROM dwd_cust_mnng;
```

These statements do the following:

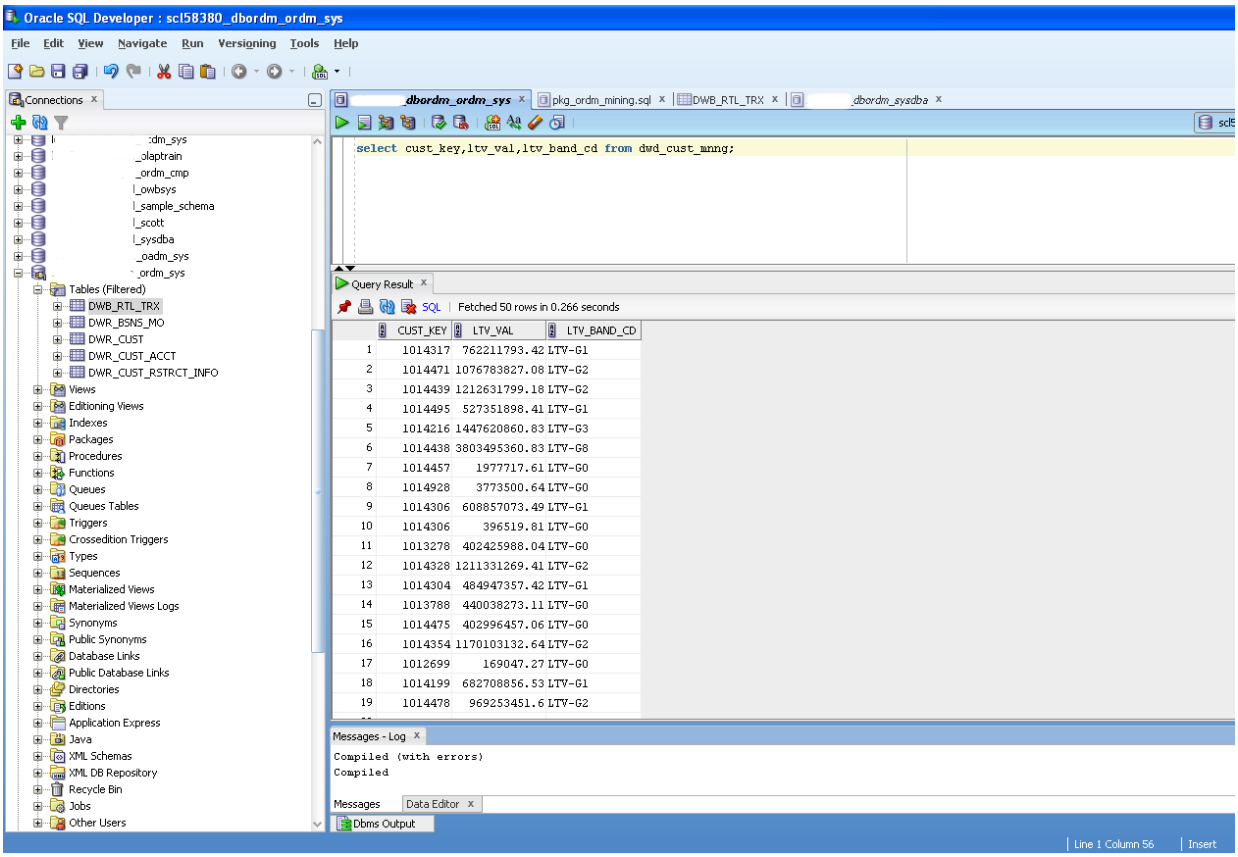
- Show the records count in `dwd_cust_mnng` table before data mining model build.
- Train the data mining model.
- Show the records count in the `dwd_cust_mnng` table after data mining model build.

**Checking the Result** After refreshing the mining source materialized views and building the data mining model, check the mining prediction results in the `dwd_cust_mnng` table as shown in the following steps:

1. Issue the following query:

```
SELECT cust_key,ltv_val,ltv_band_cd FROM dwd_cust_mnng;
```

***Figure 3–4 Checking the Result***




2. For each customer identified by cust_key, the ltv_val column gives the prediction of customer life time value, a continuous value. The ltv_band_cd column is populated by binning the prediction value, ltv_val.

# Dimensional Components in the Oracle Retail Data Model

There is often much discussion regarding the 'best' modeling approach to take for any given data warehouse with each style, classic 3NF and dimensional having their own strengths and weaknesses. It is likely that data warehouses must do more to embrace the benefits of each model type rather than rely on just one - this is the approach that was adopted in designing the Oracle Retail Data Model. The foundation layer of the Oracle Retail Data Model is a 3NF model. The default Oracle Retail Data Model also provides a dimensional model of the data. This dimensional model of the data is a perspective that summarizes and aggregates data, rather than preserving detailed transaction information.

Familiarize yourself with dimensional modeling by reading the following topics before you begin to customize the dimensional model of the default Oracle Retail Data Model:

- Characteristics of a Dimensional Model
- Characteristics of Relational Star and Snowflake Tables
- Characteristics of the OLAP Dimensional Model
- Characteristics of the OLAP Cubes in Oracle Retail Data Model
- Defining New Oracle OLAP Cubes for Oracle Retail Data Model
- Changing an Oracle OLAP Cube in Oracle Retail Data Model
- Creating a Forecast Cube for Oracle Retail Data Model
- Choosing a Cube Partitioning Strategy for Oracle Retail Data Model
- Choosing a Cube Data Maintenance Method for Oracle Retail Data Model

## Characteristics of a Dimensional Model

The simplicity of a dimensional model is inherent because it defines objects that represent real-world business entities. Analysts know which business measures they are interested in examining, which dimensions and attributes make the data meaningful, and how the dimensions of their business are organized into levels and hierarchies.

In the simplest terms, a dimensional model identifies the following objects:

- **Measures.** Measures store quantifiable business data (such as sales, expenses, and inventory). Measures are sometimes called "facts". Measures are organized by one or more dimensions and may be stored or calculated at query time:
  - **Stored Measures**. Stored measures are loaded and stored at the leaf level. Commonly, there is also a percentage of summary data that is stored. Summary data that is not stored is dynamically aggregated when queried.
  - **Calculated Measures.** Calculated measures are measures whose values are calculated dynamically at query time. Only the calculation rules are stored in the database. Common calculations include measures such as ratios, differences, moving totals, and averages. Calculations do not require disk storage space, and they do not extend the processing time required for data maintenance.
- **Dimensions.** A dimension is a structure that categorizes data to enable users to answer business questions. Commonly used dimensions are Customers, Products, and Time. A dimension's structure is organized hierarchically based on parent-child relationships. These relationships enable:

- Navigation between levels.

  Hierarchies on dimensions enable drilling down to lower levels or navigation (rolling up) to higher levels. Drilling down on the Time dimension member 2011 typically navigates you to the quarters Q1 2011 through Q4 2011. In a calendar year hierarchy, drilling down on Q1 2011 would navigate you to the months, January 2011 through March 2011. These kinds of relationships make it easy for users to navigate large volumes of multidimensional data.

- Aggregation from child values to parent values.

  The parent represents the aggregation of its children. Data values at lower levels aggregate into data values at higher levels. Dimensions are structured hierarchically so that data at different levels of aggregation are manipulated efficiently for analysis and display.

- Allocation from parent values to child values.

  The reverse of aggregation is allocation and is heavily used by planning budgeting, and similar applications. Here, the role of the hierarchy is to identify the children and descendants of particular dimension members of "top-down" allocation of budgets (among other uses).

- Grouping of members for calculations.

  Share and index calculations take advantage of hierarchical relationships (for example, the percentage of total profit contributed by each product, or the percentage share of product revenue for a certain category, or costs as a percentage of the geographical region for a retail location).

A dimension object helps to organize and group dimensional information into hierarchies. This represents natural 1:n relationships between columns or column groups (the levels of a hierarchy) that cannot be represented with constraint conditions. Going up a level in the hierarchy is called rolling up the data and going down a level in the hierarchy is called drilling down the data.

There are two ways that you can implement a dimensional model:

- **Relational tables in a star schema configuration.** This traditional method of implementing a dimensional model is discussed in "Characteristics of Relational Star and Snowflake Tables" on page 3-10.
- **Oracle OLAP Cubes**. The physical model provided with Oracle Retail Data Model provides a dimensional perspective of the data using Oracle OLAP cubes. This dimensional model is discussed in "Characteristics of the OLAP Dimensional Model" on page 3-12.

## Characteristics of Relational Star and Snowflake Tables

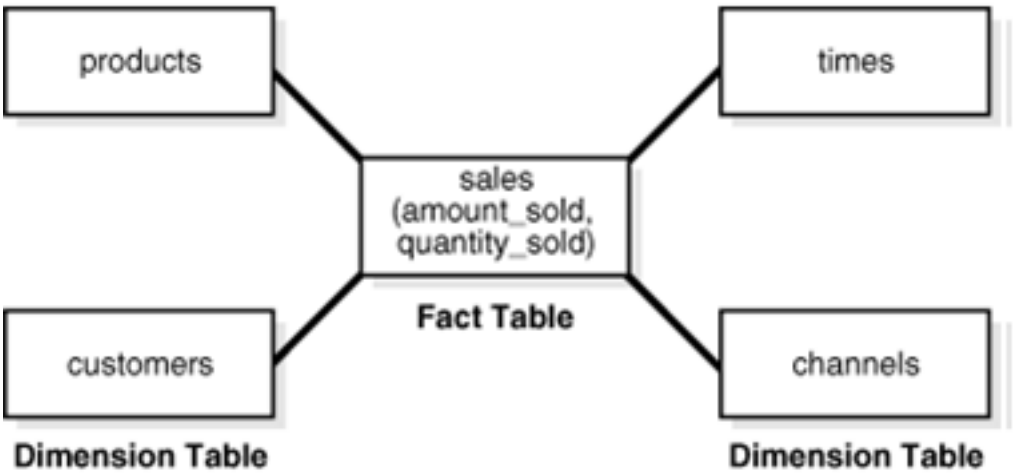
In the case of relational tables, the dimensional model has historically been implemented as a star or snowflake schema. Dimension tables (which contain information about hierarchies, levels, and attributes) join to one or more fact tables. Fact tables are the large tables that store quantifiable business measurements (such as sales, expenses, and inventory) and typically have foreign keys to the dimension tables. Dimension tables, also known as lookup or reference tables. contain the relatively static or descriptive data in the data warehouse.

A star schema borders on a physical model, as drill paths, hierarchy and query profile are embedded in the data model itself rather than the data. This in part at least, is what makes navigation of the model so straightforward for end users. Star schemas usually have a large fact table surrounded by smaller dimension tables. Dimension tables do

not change very much. Most of the information that the users need are in the fact tables. Therefore, star schemas have fewer table joins than do 3NF models.

A star schema is so called because the diagram resembles a star, with points radiating from a center. The center of the star consists of one or more fact tables and the points of the star are the dimension tables, as shown in Figure 3–5.

***Figure 3–5 Star Schema Diagram***




Snowflake schemas are slight variants of a simple star schema where the dimension tables are further normalized and broken down into multiple tables. The snowflake aspect only affects the dimensions and not the fact table and is therefore considered conceptually equivalent to star schemas. Snowflake dimensions are useful and indeed necessary when there are fact tables of differing granularity. A month-level derived or aggregate table (or materialized view) must be associated with a month level snowflake dimension table rather than the default (lower) Day level star dimension table.

> **See also:** "Declaring Relational Dimension Tables" on page 3-11 and "Validating Relational Dimension Tables" on page 3-11.

### Declaring Relational Dimension Tables

When a relational table acts as a dimension to a fact table, it is recommended that you declare that table as a dimension (even though it is not necessary). Defined dimensions can yield significant performance benefits, and support the use of more complex types of rewrite.

To define and declare the structure of the dimension use the `CREATE DIMENSION` command. Use the `LEVEL` clause to identify the names of the dimension levels.

### Validating Relational Dimension Tables

To improve the data quality of the dimension data in the data warehouse, it is recommended that you validate the declarative information about the relationships between the dimension members after any modification to the dimension data.

To perform this validation, use the `VALIDATE_DIMENSION` procedure of the `DBMS_DIMENSION` package. When the `VALIDATE_DIMENSION` procedure encounters any errors, the procedure places the errors into the `DIMENSION_EXCEPTIONS` table. To find the exceptions identified by the `VALIDATE_DIMENSION` procedure, query the `DIMENSION_EXCEPTIONS` table.

You can schedule a call to the VALIDATE_DIMENSION procedure as a post-process step to the regular Incremental Dimension load script. This can be done before the call to refresh the derived or aggregate tables of the data model through materialized view refresh, intra-ETL package calls.

## Characteristics of the OLAP Dimensional Model

Oracle OLAP Cubes logically represent data similar to relational star tables, although the data is actually stored in multidimensional arrays. Like dimension tables, cube dimensions organize members into hierarchies, levels, and attributes. The cube stores the measure (fact) data. The dimensions form the edges of the cube.

Oracle OLAP is an OLAP server embedded in the Oracle Database. Oracle OLAP provides native multidimensional storage and speed-of-thought response times when analyzing data across multiple dimensions. The database provides rich support for analytics such as time series calculations, forecasting, advanced aggregation with additive and nonadditive operators, and allocation operations.

By integrating multidimensional objects and analytics into the database, Oracle Database provides the best of both worlds: the power of multidimensional analysis along with the reliability, availability, security, and scalability of the Oracle Database.

Oracle OLAP is fully integrated into Oracle Database. At a technical level, this means:

- The OLAP engine runs within the kernel of Oracle Database.
- Dimensional objects are stored in Oracle Database in their native multidimensional format.
- Cubes and other dimensional objects are first class data objects represented in the Oracle data dictionary.
- Data security is administered in the standard way, by granting and revoking privileges to Oracle Database users and roles.
- OLAP cubes, dimensions, and hierarchies are exposed to applications as relational views. Consequently, applications can query OLAP objects using SQL as described in "Oracle OLAP Cube Views" on page 3-13 and Chapter 5, "Report and Query Customization."
- Oracle OLAP cubes can be enhanced so that they are materialized views as described in "Cube Materialized Views" on page 3-14.

> **See also:** *Oracle OLAP User's Guide* and"Characteristics of the OLAP Cubes in Oracle Retail Data Model" on page 3-15.

### Benefits of Using Oracle OLAP

Using Oracle OLAP provides significant benefits; Oracle OLAP offers the power of simplicity with One database, standard administration and security, and standard interfaces and development tools.

The Oracle OLAP dimensional data model is highly structured. Structure implies rules that govern the relationships among the data and control how the data can be queried. Cubes are the physical implementation of the dimensional model, and thus are highly optimized for dimensional queries. The OLAP engine leverages this innate dimensionality in performing highly efficient cross-cube joins for inter-row calculations, outer joins for time series analysis, and indexing. Dimensions are pre-joined to the measures. The technology that underlies cubes is based on an indexed multidimensional array model, which provides direct cell access.

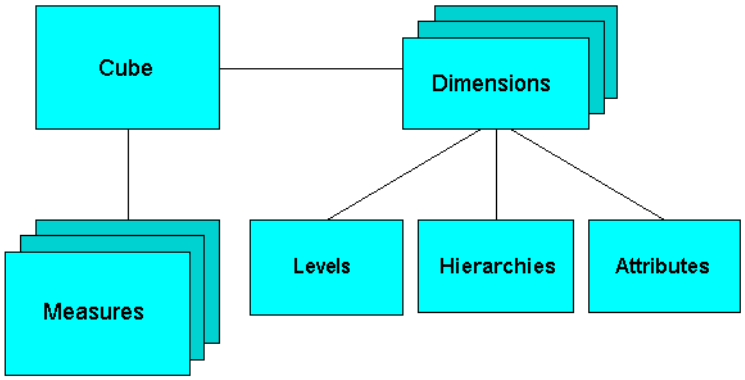
The OLAP engine manipulates dimensional objects in the same way that the SQL engine manipulates relational objects. However, because the OLAP engine is optimized to calculate analytic functions, and dimensional objects are optimized for analysis, analytic and row functions can be calculated much faster in OLAP than in SQL.

The dimensional model enables Oracle OLAP to support high-end business intelligence tools and applications such as OracleBI Discoverer Plus OLAP, OracleBI Spreadsheet Add-In, Oracle Business Intelligence Suite Enterprise Edition, BusinessObjects Enterprise, and Cognos ReportNet.

### Oracle OLAP Dimensional Objects

Oracle OLAP dimensional objects include cubes, measures, dimensions, hierarchies, levels and attributes. The OLAP dimensional objects are described in detail in *Oracle OLAP User's Guide*. Figure 3–6 shows the general relationships among the objects.

***Figure 3–6 Diagram of the OLAP Dimensional Model***




### Oracle OLAP Cube Views

When you define an OLAP cube, Oracle OLAP automatically generates a set of relational views on the cube and its dimensions and hierarchies

- Cube view. Each cube has a cube view that presents the data for all the measures and calculated measures in the cube. You can use a cube view like a fact table in a star or snowflake schema. However, the cube view contains all the summary data in addition to the detail level data. The default name of a cube view is *cube*_VIEW.
- Dimension and hierarchy views. Each dimension has one dimension view plus a hierarchy view for each hierarchy associated with the dimension. The default name for a dimension view is *dimension*_VIEW. For a hierarchy view, the default name is *dimension_hierarchy*_VIEW.

These views are related in the same way as fact and dimension tables in a star schema. Cube views serve the same function as fact tables, and hierarchy views and dimension views serve the same function as dimension tables. Typical queries join a cube view with either a hierarchy view or a dimension view.

SQL applications query these views to display the information-rich contents of these objects to analysts and decision makers. You can also create custom views that follow the structure expected by your applications, using the system-generated views like base tables.

**See also:** The discussion on querying dimensional objects in *Oracle OLAP User's Guide* and Chapter 5, "Report and Query Customization."

### Cube Materialized Views

Oracle OLAP cubes can be enhanced so that they also contain materialized views as part of the underlying implementation. A cube that has been enhanced in this way is said to contain a cube materialized view and the cube materialized view is identified with a CB$ prefix (for example: the SLSQR cube contains a cube materialized view `CB$SLSQR`). Cube materialized views can be incrementally refreshed through the Oracle Database materialized view subsystem, and if the option "Enable Query Rewrite" has been enabled for the cube, they can also serve as targets for transparent rewrite of queries against the source tables.

The OLAP dimensions associated with a cube materialized view are also defined with materialized view capabilities.

#### Necessary Cube Characteristics for Cube Materialized Views

A cube must conform to the following requirements before it can be designated as a cube materialized view:

- All dimensions of the cube have at least one level and one level-based hierarchy. Ragged and skip-level hierarchies are not supported. The dimensions must be mapped.
- All dimensions of the cube use the same aggregation operator, which is either `SUM`, `MIN`, or `MAX`.
- The cube has one or more dimensions and one or more measures.
- The cube is fully defined and mapped. For example, if the cube has five measures, then all five are mapped to the source tables.
- The data type of the cube is `NUMBER`, `VARCHAR2`, `NVARCHAR2`, or `DATE`.
- The source detail tables support dimension and rely constraints. If they have not been defined, then use the Relational Schema Advisor to generate a script that defines them on the detail tables.
- The cube is compressed.
- The cube can be enriched with calculated measures, but it cannot support more advanced analytics in a cube script.

#### Adding Materialized View Capabilities

To add materialized view capabilities to an OLAP cube, take the following steps:

1. In the Analytic Workspace Manager, connect to the `ordm_sys` schema.
2. From the cube list, select the cube which to enable.
3. In the right pane, select the **Materialized Views** tab.
4. Select the option **Enable Materialized View Refresh of the Cube**, and select the **Refresh Method**: **Fast** and **Constraints**: **Trusted** and also select the option Include a count of measure values in the materialized view, then click **Apply**. Leave the option **Enable for Query Rewrite** unchecked by default.
5. For certain cubes, for example the SLSQR cube, enable the option **Enable for Query Rewrite**. This is the only cube in Oracle Retail Data Model which is setup for cube based Materialized View Query Rewrite capability.

**Note:** You cannot enable the cube materialized view for a forecast cube. The cubes SLS, INV and SLSQR are the only cubes in Oracle Retail Data Model which use this Cube based Materialized View functionality.

**Oracle by Example:** For more information on working with OLAP cubes, see the following OBE tutorials:

- "Querying OLAP 11*g* Cubes"
- "Using Oracle OLAP 11*g* With Oracle BI Enterprise Edition"

To access the tutorials, open the Oracle Learning Library in your browser by following the instructions in "Oracle Technology Network" on page xii; and, then, search for the tutorials by name.

**See also:** *Oracle OLAP User's Guide*

## Characteristics of the OLAP Cubes in Oracle Retail Data Model

The default access layer of Oracle Retail Data Model provides a dimensional perspective of the data using Oracle OLAP cubes.

There are OLAP cubes defined in the default `ordm_sys` schema. These cubes have the general characteristics described in "Characteristics of the OLAP Dimensional Model" on page 3-12. Specifically, OLAP cubes in the Oracle Retail Data Model have the following characteristics:

- The cubes were defined and built using the Analytical Workspace Manager (AWM) client tool.
- OLAP cubes are loaded with data from DWB, DWD, and DWV tables (objects).
- Some OLAP Cubes are related to the Sales and Inventory Forecast process. They are not mapped to any relational source and data for these cubes is generated from within the Oracle Retail Data Model OLAP Analytical Workspace.
- A relational view (with a `_VIEW` suffix) is defined over each of the OLAP cubes.
- Several of the OLAP cubes in the Oracle Retail Data Model have cube materialized views enabled (that is, contain CB$ objects). These are the cubes: SLS, INV, and SLSQR.

For information on the using OLAP cubes in your customized version of Oracle Retail Data Model, see *Oracle OLAP User's Guide* and the following topics:

- Defining New Oracle OLAP Cubes for Oracle Retail Data Model
- Changing an Oracle OLAP Cube in Oracle Retail Data Model
- Creating a Forecast Cube for Oracle Retail Data Model
- Choosing a Cube Partitioning Strategy for Oracle Retail Data Model
- Choosing a Cube Data Maintenance Method for Oracle Retail Data Model

## Defining New Oracle OLAP Cubes for Oracle Retail Data Model

You can add new OLAP cubes to the `ordm_sys` schema. For consistency's sake, design and define these new cubes as described in "Characteristics of the OLAP Cubes in Oracle Retail Data Model" on page 3-15.

Take the following steps to define new cubes:

1. Ensure that there is an appropriate relational object (table or view: either `DWB_` , or `DWD_`, or `DWA_`, or even a `DWV_`) to use as the "lowest leaf" level data for the cube. Essentially this step is to ensure that the task of adding new cubes has been designed correctly and that the implementation team is aware of the various candidate source objects in Oracle Retail Data Model.

2. Use the AWM to define new Cubes for a customized version of Oracle Retail Data Model. Follow the instructions given for creating cubes and dimensions in *Oracle OLAP User's Guide*.

   Use the information provided in "Characteristics of the OLAP Dimensional Model" on page 3-12. and the Oracle OLAP User's Guide to guide you when you design and define new OLAP cubes. Also, if you are familiar with a relational star schema design as outlined in "Characteristics of Relational Star and Snowflake Tables" on page 3-10, then you can use this understanding to help you design an OLAP Cube:

   - Fact tables correspond to cubes.
   - Data columns in the fact tables correspond to measures.
   - Foreign key constraints in the fact tables identify the dimension tables.
   - Dimension tables identify the dimensions.
   - Primary keys in the dimension tables identify the base-level dimension members.
   - Parent columns in the dimension tables identify the higher level dimension members.
   - Columns in the dimension tables containing descriptions and characteristics of the dimension members identify the attributes.

   You can also get insights into the dimensional model by looking at the reports included with Oracle Retail Data Model.

   **See:** *Oracle Retail Data Model Installation Guide* for more information on installing the sample reports and deploying the Oracle Retail Data Model RPD and webcat on the Business Intelligence Suite Enterprise Edition instance.

   **Tip:** While investigating your source data, you may decide to create relational views that more closely match the dimensional model that you plan to create.

3. Add materialized view capabilities to the OLAP cubes as described in "Adding Materialized View Capabilities" on page 3-14.

   **See also:** *Oracle OLAP User's Guide*, "Defining New Oracle OLAP Cubes for Oracle Retail Data Model" on page 3-16, and the sample reports in *Oracle Retail Data Model Reference*.

**Oracle by Example:** For more information on creating OLAP cubes, see the "Building OLAP 11*g* Cubes" OBE tutorial.

To access the tutorial, open the Oracle Learning Library in your browser by following the instructions in "Oracle Technology Network" on page xii; and, then, search for the tutorial by name.

## Changing an Oracle OLAP Cube in Oracle Retail Data Model

Common customizations to Oracle Retail Data Model cubes are changing the dimensions or the measures of the cube.

To change the measures or dimensions of one cube, you must take the following steps:

1. Use the information in *Oracle Retail Data Model Reference* to identify the relational object, that is the table or view, from which the OLAP cube is populated.
2. Change the structure of the object identified in Step 1.
3. Change the OLAP cube and cube materialized views to reflect the new structure.

## Creating a Forecast Cube for Oracle Retail Data Model

To create a forecast cube for Oracle Retail Data Model:

1. Create a cube to contain the results of the forecast as described in "Defining New Oracle OLAP Cubes for Oracle Retail Data Model" on page 3-16.

   **Note:** You cannot enable materialized views for an Oracle Retail Data Model forecast cube.

2. Write an OLAP DML forecasting context program as described in *Oracle OLAP DML Reference*.

## Choosing a Cube Partitioning Strategy for Oracle Retail Data Model

Partitioning is a method of physically storing the contents of a cube. It improves the performance of large cubes in the following ways:

- Improves scalability by keeping data structures small. Each partition functions like a smaller measure.
- Keeps the working set of data smaller both for queries and maintenance, since the relevant data is stored together.
- Enables parallel aggregation during data maintenance. Each partition can be aggregated by a separate process.
- Simplifies removal of old data from storage. Old partitions can be dropped, and new partitions can be added.

The number of partitions affects the database resources that can be allocated to loading and aggregating the data in a cube. Partitions can be aggregated simultaneously when sufficient resources have been allocated.

The Cube Partitioning Advisor analyzes the source tables and develops a partitioning strategy. You can accept the recommendations of the Cube Partitioning Advisor, or you can make your own decisions about partitioning.

If your partitioning strategy is driven primarily by life-cycle management considerations, then you should partition the cube on the Time dimension. Old time periods can then be dropped as a unit, and new time periods added as a new partition. The Cube Partitioning Advisor has a Time option, which recommends a hierarchy and a level in the Time dimension for partitioning.

The level on which to partition a cube is determined based on a trade off between load performance and query performance.

Typically, you do not want to partition on too low a level (for example, on the DAY level of a TIME dimension) because if you do then too many partitions must be defined at load time which slows down an initial or historical load. Also, a large number of partitions can result in unusually long Analytic Workspace attach times and slows down the Time Series-based calculations. Also, a Quarterly Cumulative measure (Quarter to Date Measure) needs to access 90 or 91 partitions to calculate a value for one Customer and Organization. All dimension members above the partition level of partition dimension (including those belonging to nondefault hierarchies) would be present in a single default template. Day level partitioning makes this very heavy since all higher level members are stored in default template. However, the advantage of partitioning `DAY` if the OLAP Cube load frequency is daily then there you must only load from a new partition in fact table into a single partition in the OLAP cube every day. This greatly improves the load performance since percentage-based refresh can be enabled if the cube is materialized-view enabled and has materialized-view logs.

#### Recommendations: Cube Partitioning Strategy

Usually a good compromise between the differing load and query performance requirements is to use an intermediate level like `MONTH` as the partition level. Time series calculations within a month (week to date, month to date, and so on) are fast and higher level calculations such as year to date need to refer to 12 partitions at most. Also this way the monthly partition is defined and created only one time (that is during the initial load on first of each month) and is then reused for each subsequent load that month. The aggregation process may be triggered off at the month level (instead of specific day level) and some redundant aggregations (of previously loaded dates of current month) may occur each time but it should result in satisfactory load and query performance.

> **See also:** "The discussion on choosing a partition strategy in *Oracle OLAP User's Guide*, "Indexes and Partitioned Indexes in Oracle Retail Data Model" on page 2-11, and "Partitioning and Materialized Views" on page 3-22.

## Choosing a Cube Data Maintenance Method for Oracle Retail Data Model

While developing a dimensional model of your data, it is a good idea to map and load each object immediately after you create it so that you can immediately detect and correct any errors that you made to the object definition or the mapping.

However, in a production environment, you should perform routine maintenance as quickly and easily as possible. For this stage, you can choose among data maintenance methods. You can refresh all cubes using the Maintenance Wizard. This wizard enables you to refresh a cube immediately, or submit the refresh as a job to the Oracle job queue, or generate a PL/SQL script. You can run the script manually or using a scheduling utility, such as Oracle Enterprise Manager Scheduler or the `DBMS_SCHEDULER` PL/SQL package. The generated script calls the `BUILD` procedure of the `DBMS_CUBE` PL/SQL package. You can modify this script or develop one from the start using this package.

The data for a partitioned cube is loaded and aggregated in parallel when multiple processes have been allocated to the build. You are able to see this in the build log.

In addition, each cube can support these data maintenance methods:

- Custom cube scripts
- Cube materialized views

If you are defining cubes to replace existing materialized views, then you use the materialized views as an integral part of data maintenance. Note, however, that materialized view capabilities restrict the types of analytics that can be performed by a custom cube script.

> **See also:** *Oracle OLAP User's Guide* and "Types of Materialized Views and Refresh options" on page 3-20

---

**Oracle by Example:** See the following OBE tutorial for an example of how Oracle uses cube materialized views for transparent access to a relational star schema:

- "Querying OLAP 11*g* Cubes"

To access the tutorial, open the Oracle Learning Library in your browser by following the instructions in "Oracle Technology Network" on page xii; and, then, search for the tutorial by name.

---

## Materialized Views in the Oracle Retail Data Model

Materialized views are query results that have been stored or "materialized" in advance as schema objects. From a physical design point of view, materialized views resemble tables or partitioned tables and behave like indexes in that they are used transparently and improve performance.

In the past, organizations using summaries spent a significant amount of time and effort creating summaries manually, identifying which summaries to create, indexing the summaries, updating them, and advising their users on which ones to use. With the advent of materialized views, a database administrator creates one or more materialized views, which are the equivalent of a summary. Thus, the workload of the database administrator is eased and the user no longer needed to be aware of the summaries that had been defined. Instead, the end user queries the tables and views at the detail data level. The query rewrite mechanism in the Oracle server automatically rewrites the SQL query to use the summary tables and reduces response time for returning results from the query.

Materialized views improve query performance by precalculating expensive join and aggregation operations on the database before executing and storing the results in the database. The query optimizer automatically recognizes when it can use an existing materialized view to satisfy a request.

The default Oracle Retail Data Model defines many materialized views. In the default `ordm_sys` schema, you can identify these materialized views by looking at objects with the prefixes listed in the following table.

| Prefix | Description |
|---|---|
| DWA_ | Aggregate table or relational materialized view. This prefix is generalized for aggregate tables. Aggregate objects are implemented using either Materialized Views or tables.<br><br>See: Aggregate tables in *Oracle Retail Data Model Reference* for a list of these objects in the default data model. |
| CB$ | Materialized view used to support/deliver required functionality for an Oracle OLAP cube. This is an internal object built and maintained automatically by the Oracle OLAP server in the database.<br><br>See: OLAP cube materialized views in *Oracle Retail Data Model Reference* for a list of these objects in the default data model.<br><br>"Characteristics of the OLAP Cubes in Oracle Retail Data Model" on page 3-15 for information on OLAP cubes.<br><br>Note: Do not report or query against this object. Instead access the relational view of an OLAP cube (that is, the object with the _VIEW suffix). |
| DMV_ | Data mining views that are *not* an aggregate table or a cube materialized view.<br><br>See: *Oracle Retail Data Model Reference* to identify these objects in the default data model. |

The following topics provide more information on using and creating materialized views in your customized Oracle Retail Data Model:

- Types of Materialized Views and Refresh options
- Choosing Indexes for Materialized Views
- Partitioning and Materialized Views
- Compressing Materialized Views

## Types of Materialized Views and Refresh options

Refresh option vary by the type of materialized view:

- Refresh Options for Materialized Views with Aggregates
- Refresh Options for Materialized Views Containing Only Joins
- Refresh Options for Nested Materialized Views

> **See:** *Oracle OLAP User's Guide* for a discussion of creating materialized views of Oracle OLAP cubes.

### Refresh Options for Materialized Views with Aggregates

In data warehouses, materialized views normally contain aggregates. The DWA_ tables in the default Oracle Retail Data Model are this type of materialized view.

For a materialized view with aggregates, for fast refresh to be possible:

- The SELECT list must contain all of the GROUP BY columns (if present)
- There must be a COUNT(*) and a COUNT(*column*) on any aggregated columns.
- Materialized view logs must be present on all tables referenced in the query that defines the materialized view. The valid aggregate functions are: SUM, COUNT(*x*), COUNT(*), AVG, VARIANCE, STDDEV, MIN, and MAX, and the expression to be aggregated can be any SQL value expression.

Fast refresh for a materialized view containing joins and aggregates is possible after any type of DML to the base tables (direct load or conventional INSERT, UPDATE, or DELETE).

You can define that the materialized view be refreshed ON COMMIT or ON DEMAND. A REFRESH ON COMMIT materialized view is automatically refreshed when a transaction that does DML to a materialized view's detail tables commits.

When you specify REFRESH ON COMMIT, the table commit can take more time than if you have not. This is because the refresh operation is performed as part of the commit process. Therefore, this method may not be suitable if many users are concurrently changing the tables upon which the materialized view is based.

### Refresh Options for Materialized Views Containing Only Joins

Some materialized views contain only joins and no aggregates (for example, when a materialized view is created that joins the sales table to the times and customers tables). The advantage of creating this type of materialized view is that expensive joins are precalculated.

Fast refresh for a materialized view containing only joins is possible after any type of DML to the base tables (direct-path or conventional INSERT, UPDATE, or DELETE).

A materialized view containing only joins can be defined to be refreshed ON COMMIT or ON DEMAND. If it is ON COMMIT, the refresh is performed at commit time of the transaction that does DML on the materialized view's detail table.

If you specify REFRESH FAST, Oracle Database performs further verification of the query definition to ensure that fast refresh can be performed if any of the detail tables change. These additional checks are:

- A materialized view log must be present for each detail table unless the table supports partition change tracking. Also, when a materialized view log is required, the ROWID column must be present in each materialized view log.
- The rowids of all the detail tables must appear in the SELECT list of the materialized view query definition.

If some of these restrictions are not met, you can create the materialized view as REFRESH FORCE to take advantage of fast refresh when it is possible. If one table does not meet all of the criteria, but the other tables do the materialized view is still fast refreshable with respect to the other tables for which all the criteria are met.

To achieve an optimally efficient refresh:

- Ensure that the defining query does not use an outer join that behaves like an inner join. If the defining query contains such a join, consider rewriting the defining query to contain an inner join.
- If the materialized view contains *only* joins, the ROWID columns for each table (and each instance of a table that occurs multiple times in the FROM list) must be present in the SELECT list of the materialized view.
- If the materialized view has remote tables in the FROM clause, all tables in the FROM clause must be located on that same site. Further, ON COMMIT refresh is not supported for materialized view with remote tables. Except for SCN-based materialized view logs, materialized view logs must be present on the remote site for each detail table of the materialized view and ROWID columns must be present in the SELECT list of the materialized view.

### Refresh Options for Nested Materialized Views

A nested materialized view is a materialized view whose definition is based on another materialized view. A nested materialized view can reference other relations in the database in addition to referencing materialized views.

In a data warehouse, you typically create many aggregate views on a single join (for example, rollups along different dimensions). Incrementally maintaining these distinct materialized aggregate views can take a long time, because the underlying join has to be performed many times.

Using nested materialized views, you can create multiple single-table materialized views based on a joins-only materialized view and the join is performed just one time. In addition, optimizations can be performed for this class of single-table aggregate materialized view and thus refresh is very efficient.

Some types of nested materialized views cannot be fast refreshed. Use `EXPLAIN_MVIEW` to identify those types of materialized views.

You can refresh a tree of nested materialized views in the appropriate dependency order by specifying the `nested =TRUE` parameter with the `DBMS_MVIEW.REFRESH` parameter.

***Example 3–2 Refreshing Oracle Retail Data Model Nested Materialized Views***

For example, if you call `DBMS_MVIEW.REFRESH ('DWA_CUST_TYP_ORDR_DEPT_MO', nested => TRUE)`, the `REFRESH` procedure first refreshes the `DWA_CUST_TYP_ORDR_SBC_WK` materialized view, and then refreshes the `DWA_CUST_TYP_ORDR_DEPT_MO` materialized view.

## Choosing Indexes for Materialized Views

The two most common operations on a materialized view are query execution and fast refresh, and each operation has different performance requirements:

- Query execution might need to access any subset of the materialized view key columns, and might need to join and aggregate over a subset of those columns. Consequently, for best performance, create a single-column bitmap index on each materialized view key column.
- In the case of materialized views containing only joins using fast refresh, create indexes on the columns that contain the rowids to improve the performance of the refresh operation.
- If a materialized view using aggregates is fast refreshable, then an index appropriate for the fast refresh procedure is created unless `USING NO INDEX` is specified in the `CREATE MATERIALIZED VIEW` statement.

**See also:** "Indexes and Partitioned Indexes in Oracle Retail Data Model" on page 2-11

## Partitioning and Materialized Views

Because of the large volume of data held in a data warehouse, partitioning is an extremely useful option when designing a database. Partitioning the fact tables improves scalability, simplifies system administration, and makes it possible to define local indexes that can be efficiently rebuilt. Partitioning the fact tables also improves the opportunity of fast refreshing the materialized view because this may enable partition change tracking refresh on the materialized view.

Partitioning a materialized view has the same benefits as partitioning fact tables. When a materialized view is partitioned a refresh procedure can use parallel DML in more scenarios and partition change tracking-based refresh can use truncate partition to efficiently maintain the materialized view.

> **See also:** *Oracle Database VLDB and Partitioning Guide*, "Partitioned Tables in Oracle Retail Data Model" on page 2-12, "Indexes and Partitioned Indexes in Oracle Retail Data Model" on page 2-11, and "Choosing a Cube Partitioning Strategy for Oracle Retail Data Model" on page 3-17

### Using Partition Change Tracking

It is possible and advantageous to track freshness to a finer grain than the entire materialized view. The ability to identify which rows in a materialized view are affected by a certain detail table partition, is known as partition change tracking. When one or more of the detail tables are partitioned, it may be possible to identify the specific rows in the materialized view that correspond to a modified detail partition(s). those rows become stale when a partition is modified while all other rows remain fresh.

You can use partition change tracking to identify which materialized view rows correspond to a particular partition. Partition change tracking is also used to support fast refresh after partition maintenance operations on detail tables. For instance, if a detail table partition is truncated or dropped, the affected rows in the materialized view are identified and deleted. Identifying which materialized view rows are fresh or stale, rather than considering the entire materialized view as stale, allows query rewrite to use those rows that refresh while in `QUERY_REWRITE_INTEGRITY = ENFORCED` or `TRUSTED` modes.

Several views, such as `DBA_MVIEW_DETAIL_PARTITION`, detail which partitions are stale or fresh. Oracle does not rewrite against partial stale materialized views if partition change tracking on the changed table is enabled by the presence of join dependent expression in the materialized view.

To support partition change tracking, a materialized view must satisfy the following requirements:

- At least one detail table referenced by the materialized view must be partitioned.
- Partitioned tables must use either range, list or composite partitioning.
- The top level partition key must consist of only a single column.
- The materialized view must contain either the partition key column or a partition marker or `ROWID` or join dependent expression of the detail table.
- If you use a `GROUP BY` clause, the partition key column or the partition marker or `ROWID` or join dependent expression must be present in the `GROUP BY` clause.
- If you use an analytic window function or the `MODEL` clause, the partition key column or the partition marker or `ROWID` or join dependent expression must be present in their respective `PARTITION BY` subclauses.
- Data modifications can only occur on the partitioned table. If partition change tracking refresh is being done for a table which has join dependent expression in the materialized view, then data modifications should not have occurred in any of the join dependent tables.
- The `COMPATIBILITY` initialization parameter must be a minimum of `9.0.0.0.0`.

- Partition change tracking is not supported for a materialized view that refers to views, remote tables, or outer joins.

## Compressing Materialized Views

Using data compression for a materialized view brings you a additional dramatic performance improvement.

Consider data compression when using highly redundant data, such as tables with many foreign keys. In particular, likely candidates are materialized views created with the `ROLLUP` clause.

**See also:** "Data Compression in Oracle Retail Data Model" on page 2-9

# 4

# ETL Implementation and Customization

This chapter discusses the ETL (Extraction, Transformation and Loading) procedures you use to populate an Oracle Retail Data Model warehouse. It includes the following topics:


- The Role of ETL in the Oracle Retail Data Model
- Creating Source-ETL for Oracle Retail Data Model
- Customizing Intra-ETL for the Oracle Retail Data Model
- Performing an Initial Load of an Oracle Retail Data Model Warehouse
- Refreshing the Data in Oracle Retail Data Model Warehouse
- Managing Errors During Oracle Retail Data Model Intra-ETL Execution


## The Role of ETL in the Oracle Retail Data Model

Figure 2–1 shows the three layers in Oracle Retail Data Model warehouse environment: the optional staging layer, the foundation layer, and the access layer. As shown in Figure 4–1, you use two types of ETL (extraction, transformation and loading) to populate these layers:

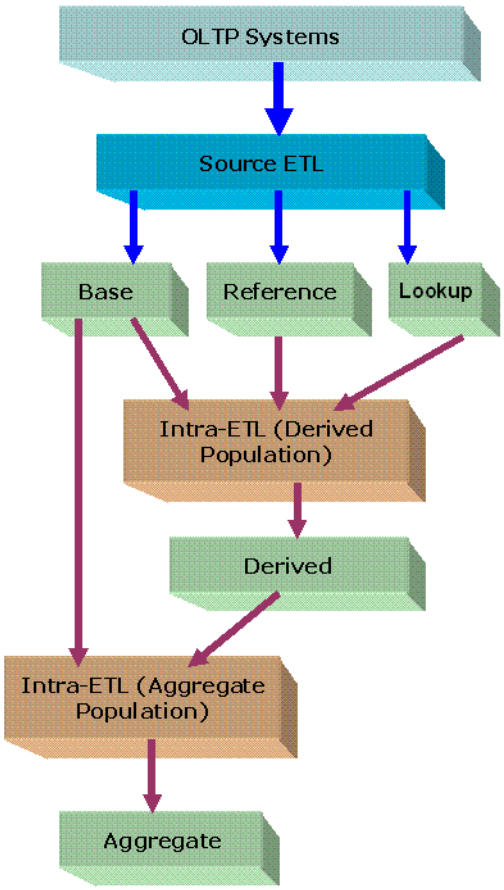
- **Source-ETL**. ETL that populates the staging layer (if any) or the foundation layer with data from the transactional system is known as source ETL.

  Oracle Retail Data Model does *not* include source-ETL. You must create source-ETL yourself using your understanding of your transactional system and your customized Oracle Retail Data Model. See "Creating Source-ETL for Oracle Retail Data Model" on page 4-2 for more information on creating source-ETL.

- **Intra-ETL**. ETL that populates the access layer using the data in the foundation layer is known as intra-ETL.

  Oracle Retail Data Model *does* include intra-ETL. You can modify the default intra-ETL to populate a customized access layer from a customized foundation layer. See "Customizing Intra-ETL for the Oracle Retail Data Model" on page 4-9 for more information on the intra-ETL.

*Figure 4–1 ETL Flow Diagram*




# Creating Source-ETL for Oracle Retail Data Model

ETL that populates the staging layer or the foundation layer of an Oracle Retail Data Model warehouse with data from a transactional system is known as source-ETL.

Due to the large number of available transactional applications and multiple versions that may be in use, source-ETL is not provided with Oracle Retail Data Model. You must write your own source-ETL scripts using Oracle Warehouse Builder or another ETL tool or mapping tool.

> **See also:** *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide*.

**Oracle By Example:** See the following OBE tutorials for more information on Oracle Warehouse Builder:

- "Setting Up the Oracle Warehouse Builder 11*g* Release 2 Environment"
- "Improved User Interface, Usability, and Productivity With OWB 11*g*"
- "Using Data Transformation Operators with Source and Target Operators"
- "Working with Pluggable Mappings"
- "Examining Source Data Using Data Profiling with Database 11*g* Release 2"

To access the tutorials, open the Oracle Learning Library in your browser by following the instructions in "Oracle Technology Network" on page xii; and, then, search for the tutorials by name.

The following topics provide general information about writing source-ETL:

- Source-ETL Design Considerations
- ETL Architecture for Oracle Retail Data Model Source-ETL
- Creating a Source to Target Mapping Document for the Source-ETL
- Designing a Plan for Rectifying Source-ETL Data Quality Problems
- Designing Source-ETL Workflow and Jobs Control
- Designing Source-ETL Exception Handling
- Writing Source-ETL that Loads Efficiently

## Source-ETL Design Considerations

Keep the following points in mind when designing and writing source-ETL for Oracle Retail Data Model:

- You can populate the calendar data using the calendar population scripts provided with Oracle Retail Data Model (the calendar population scripts populate data for the business and gregorian calendars, other calendars need to be populated using source ETL). See *Oracle Retail Data Model Reference* for more information on the Oracle Retail Data Model calendar population scripts.
- Populate the tables in the following order:
  1. Lookup tables
  2. Reference tables
  3. Base tables
- Analyze the tables in one category before loading the tables in the next category (for example, analyze the reference tables before loading the lookup tables). Additionally, you must analyze all of the tables loaded by the source-ETL process before executing the intra-ETL processes).

  **See:** The topic about analyzing tables, indexes and clusters in *Oracle Database Administrator's Guide*.

## ETL Architecture for Oracle Retail Data Model Source-ETL

ETL typically extracts data from the transactional system, checks for data quality, cleanses the data and ensures consistency of terms, currency, units of measures, and so on, as it consolidates and populates the physical objects in the data warehouse with 'clean' data.

The fundamental services upon which data acquisition is constructed are as follows:

- Data sourcing
- Data movement
- Data transformation
- Data loading

From a logical architecture perspective, there are many different ways to configure these building blocks for delivering data acquisition services. The major architectural styles available that cover a range of options to be targeted within a data warehousing architecture include:

- **Batch Extract, Transform, and Load** and **Batch Extract, Load, Transform, Load**

  Batch Extract, Transform and Load (ETL) and Batch Extract, Load, Transform, Load (ELTL) are the traditional architectures in a data warehouse implementation. The difference between these is where the transformation proceeds, in or out of the database.

- **Batch Hybrid Extract, Transform, Load, Transform, Load**

  Batch Hybrid Extract, Transform, Load, Transform, Load (ETLTL) is a hybrid strategy. This strategy provides the most flexibility to remove hand coding approaches to transformation design, apply a metadata-driven approach, and still be able to leverage the data processing capabilities of the enterprise warehouse. In this targeted design, the transformation processing is first performed outside the warehouse as a pre-processing step before loading the staging tables, and then further transformation processing is performed within the data warehouse before the final load into the target tables.

- **Real-time Extract, Transform, Load**

  Real-time Extract, Transform, Load (rETL) is appropriate when service levels for data freshness demand more up-to-date information in the data warehousing environment. In this approach, the transactional system must actively publish events of interest so that the rETL processes can extract them from a message bus (queue) on a timely basis. A message-based paradigm is used with publish and subscribe message bus structures or point-to-point messaging with reliable queues.

When designing source-ETL for Oracle Retail Data Model, use the architecture that best meets your business needs.

## Creating a Source to Target Mapping Document for the Source-ETL

Before you begin building your extract systems, create a logical data interface document that maps the relationship between original source columns and target destination columns in the tables. This document ties the very beginning of the ETL system to the very end.

Columns in the data mapping document are sometimes combined. For example, the source database, table name, and column name could be combined into a single target column. The information within the concatenated column would be delimited with a

period. Regardless of the format, the content of the logical data mapping document has been proven to be the critical element required to sufficiently plan ETL processes.

## Designing a Plan for Rectifying Source-ETL Data Quality Problems

Data cleaning consists of all the steps required to clean and validate the data feeding a table and to apply known business rules to make the data consistent. The perspectives of the cleaning and conforming steps are less about the upside potential of the data and more about containment and control.

If there are data quality problems, then build a plan, in agreement with IT and business users, for how to rectify these problems.

Answer the following questions:

- Is data missing?
- Is the data wrong or inconsistent?
- Should the problem be fixed in the source systems?

Set up the following processes and programs:

- Data quality measurement process.
- Data quality reporting and action program and people responsibility.

## Designing Source-ETL Workflow and Jobs Control

All data movement among ETL processes are composed of jobs. An ETL workflow executes these jobs in the proper sequence and with regard to the necessary dependencies. General ETL tools, such as Oracle Warehouse Builder, support this kind of workflow, job design, and execution control.

The following list includes tips for when you design ETL jobs and workflow:

- Use common structure across all jobs (source system to transformer to target data warehouse).
- Have a one-to-one mapping from source to target.
- Define one job per Source table.
- Apply generic job structure and template jobs to allow for rapid development and consistency.
- Use an optimized job design to leverage Oracle load performance based on data volumes.
- Design parameterized job to allow for greater control over job performance and behavior.
- Maximize Jobs parallelism execution.

## Designing Source-ETL Exception Handling

Your ETL tool or your developed mapping scripts generate status and error handling tables. All ETL procedures log status and errors into a log table. Execution status may be monitored using an ETL tool or by examining the log table.

## Writing Source-ETL that Loads Efficiently

Whether you are developing mapping scripts and loading into a staging layer or directly into the foundation layer the goal is to get the data into the warehouse in the most expedient manner. To achieve good performance during the load you must begin by focusing on where the data to be loaded resides and how you load it into the database. For example, you should not use a serial database link or a single JDBC connection to move large volumes of data. The most common and preferred mechanism for loading large volumes of data is loading from flat files.

The following topics discuss best practices for ensuring your source-ETL loads efficiently:

- Using a Staging Area for Flat Files
- Preparing Raw Data Files for Source-ETL
- Source-ETL Data Loading Options
- Parallel Direct Path Load Source-ETL
- Source-ETL Data Loading Options

### Using a Staging Area for Flat Files

The area where flat files are stored before being loaded into the staging layer of a data warehouse system is commonly known as staging area. The overall speed of your load is determined by:

- How quickly the raw data can be read from staging area.
- How quickly the raw data can be processed and inserted into the database.

#### Recommendations: Using a Staging Area

Stage the raw data across as many physical disks as possible to ensure that reading it is not a bottleneck during the load.

Also, if you are using the Oracle Exadata Database Machine, the best place to stage the data is in an Oracle Database File System (DBFS) stored on the Exadata storage cells. DBFS creates a mountable cluster file system which can you can use to access files stored in the database. Create the DBFS in a separate database on the Database Machine. This allows the DBFS to be managed and maintained separately from the data warehouse.

Mount the file system using the `DIRECT_IO` option to avoid thrashing the system page cache while moving the raw data files in and out of the file system.

> **See:** *Oracle Database SecureFiles and Large Objects Developer's Guide* for more information on setting up DBFS.

### Preparing Raw Data Files for Source-ETL

To parallelize the data load Oracle Database must be able to logically break up the raw data files into chunks, known as granules. To ensure balanced parallel processing, the number of granules is typically much higher than the number of parallel server processes. At any given point in time, a parallel server process is allocated one granule to work on. After a parallel server process completes working on its granule, another granule is allocated until all of the granules are processed and the data is loaded.

#### Recommendations: Preparing Raw Data Files for Source-ETL

Follow these recommendations:

- Deliminate each row using a known character such as a new line or a semicolon. This ensures that Oracle can look inside the raw data file and determine where each row of data begins and ends to create multiple granules within a single file.
- If a file is not position-able and seek-able (for example the file is compressed or zip file), then the files cannot be broken up into granules and the entire file is treated as a single granule. In this case, only one parallel server process can work on the entire file. To parallelize the loading of compressed data files, use multiple compressed data files. The number of compressed data files used determines the maximum parallel degree used by the load.
- When loading multiple data files (compressed or uncompressed):
  - Use a single external table, if at all possible
  - Make the files similar in size
  - Make the size of the files a multiple of 10 MB
- If you must have files of different sizes, list the files from largest to smallest. By default, Oracle assumes that the flat file has the same character set as the database. If this is not the case, specify the character set of the flat file in the external table definition to ensure the proper character set conversions can take place.

## Source-ETL Data Loading Options

Oracle offers several data loading options

- External table or SQL*Loader
- Oracle Data Pump (import and export)
- Change Data Capture and Trickle feed mechanisms (such as Oracle GoldenGate)
- Oracle Database Gateways to open systems and mainframes
- Generic Connectivity (ODBC and JDBC)

The approach that you take depends on the source and format of the data you receive.

### Recommendations: Loading Flat Files

If you are loading from files into Oracle Database you have two options: SQL*Loader or external tables.

Using external tables offers the following advantages:

- Allows transparent parallelization inside the database.
- You can avoid staging data and apply transformations directly on the file data using arbitrary SQL or PL/SQL constructs when accessing external tables. SQL Loader requires you to load the data as-is into the database first.
- Parallelizing loads with external tables enables a more efficient space management compared to SQL*Loader, where each individual parallel loader is an independent database sessions with its own transaction. For highly partitioned tables this could potentially lead to a lot of wasted space.

Create an external table using the standard `CREATE TABLE` statement. However, to load from flat files the statement must include information about where the flat files reside outside the database. The most common approach when loading data from an external table is to issue a `CREATE TABLE AS SELECT (CTAS)` statement or an `INSERT AS SELECT (IAS)` statement into an existing table.

### Parallel Direct Path Load Source-ETL

A direct path load parses the input data according to the description given in the external table definition, converts the data for each input field to its corresponding Oracle Database data type, then builds a column array structure for the data. These column array structures are used to format Oracle data blocks and build index keys. The newly formatted database blocks are then written directly to the database, bypassing the standard SQL processing engine and the database buffer cache.

The key to good load performance is to use direct path loads wherever possible:

- A `CREATE TABLE AS SELECT (CTAS)` statement always uses direct path load.
- A simple `INSERT AS SELECT (IAS)` statement does *not* use direct path load. In order to achieve direct path load with an IAS statement you must add the `APPEND` hint to the command.

Direct path loads can also run in parallel. To set the parallel degree for a direct path load, either:

- Add the `PARALLEL` hint to the CTAS statement or an IAS statement.
- Set the `PARALLEL` clause on both the external table and the table into which the data is loaded.

  After the parallel degree is set:

  - A `CTAS` statement automatically performs a direct path load in parallel.
  - An `IAS` statement does not automatically perform a direct path load in parallel. To enable an `IAS` statement to perform direct path load in parallel, you must alter the session to enable parallel DML by executing the following statement.

    ```
    alter session enable parallel DML;
    ```

### Partition Exchange Load for Oracle Retail Data Model Source-ETL

A benefit of partitioning is the ability to load data quickly and easily with minimal impact on the business users by using the `EXCHANGE PARTITION` command. The `EXCHANGE PARTITION` command enables swapping the data in a nonpartitioned table into a particular partition in your partitioned table. The `EXCHANGE PARTITION` command does not physically move data, instead it updates the data dictionary to exchange a pointer from the partition to the table and vice versa.

Because there is no physical movement of data, an exchange does not generate redo and undo. In other words, an exchange is a sub-second operation and far less likely to impact performance than any traditional data-movement approaches such as `INSERT`.

#### Recommendations: Partitioning Tables

Partition the larger tables and fact tables in the Oracle Retail Data Model warehouse.

***Example 4–1 Using Exchange Partition Statement with a Partitioned Table***

Assume that there is a large table called `Sales`, which is range partitioned by day. At the end of each business day, data from the online sales system is loaded into the `Sales` table in the warehouse.

The following steps ensure the daily data gets loaded into the correct partition with minimal impact to the business users of the data warehouse and optimal speed:

1. Create external table for the flat file data coming from the online system

2. Using a CTAS statement, create a nonpartitioned table called tmp_sales that has the same column structure as Sales table
3. Build any indexes that are on the Sales table on the tmp_sales table
4. Issue the EXCHANGE PARTITION command.

```
Alter table Sales exchange partition p2 with
    table top_sales including indexes without validation;
```

5. Gather optimizer statistics on the newly exchanged partition using incremental statistics.

The EXCHANGE PARTITION command in this example, swaps the definitions of the named partition and the tmp_sales table, so the data instantaneously exists in the right place in the partitioned table. Moreover, with the inclusion of the INCLUDING INDEXES and WITHOUT VALIDATION clauses, Oracle swaps index definitions and does not check whether the data actually belongs in the partition - therefore, the exchange is very quick.

---

**Note:** The assumption being made in this example is that the data integrity was verified at date extraction time. If you are unsure about the data integrity, omit the WITHOUT VALIDATION clause so that the Database checks the validity of the data.

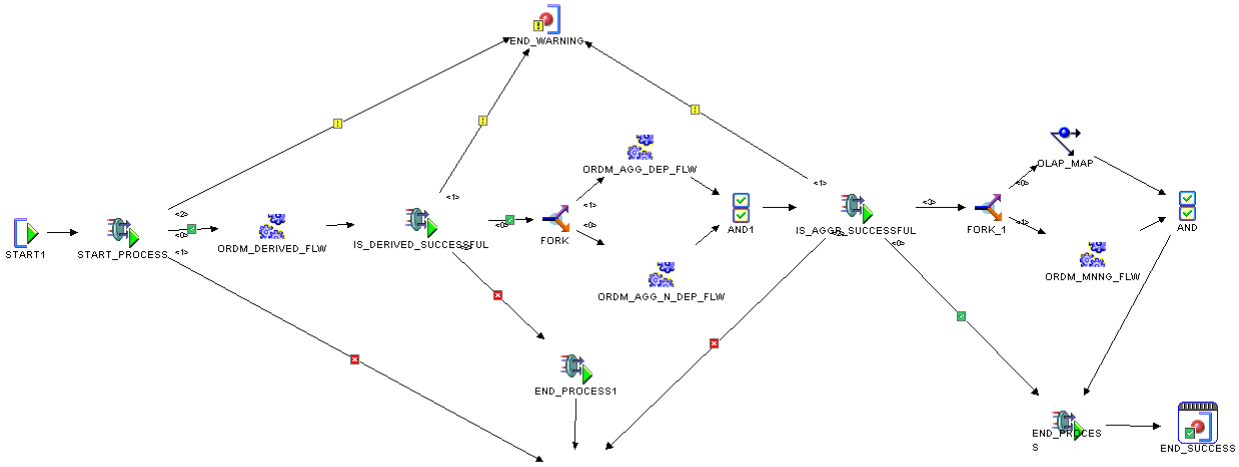
---

## Customizing Intra-ETL for the Oracle Retail Data Model

The Oracle Retail Data Model supports the use of ETL tools such as Oracle Warehouse Builder to define the workflow to execute the intra-ETL process. You can, of course, write your own Intra-ETL. However, an intra-ETL component is delivered with Oracle Retail Data Model that is a process flow designed using the Oracle Warehouse Builder Workflow component. This process flow is named ORDM_INTRA_ETL_FLW.

As shown in Figure 4–2, the ORDM_INTRA_ETL_FLW process flow uses the data in the Oracle Retail Data Model base, reference, and lookup tables to populate all of the other Oracle Retail Data Model structures. Within this package the dependency of each individual program is implemented and enforced so that each program executes in the proper order.

***Figure 4–2 ORDM Main Intra-ETL Process Flow***




You can change the original intra-ETL script for your specific requirements. However, perform a complete impact analysis before you make the change. Package the changes as a patch to the original Oracle Retail Data Model intra-ETL mapping.

The `ORDM_INTRA_ETL_FLW` process flow consists of the following sub-processes and includes the dependency of individual sub-process flows and executes them in the proper order:

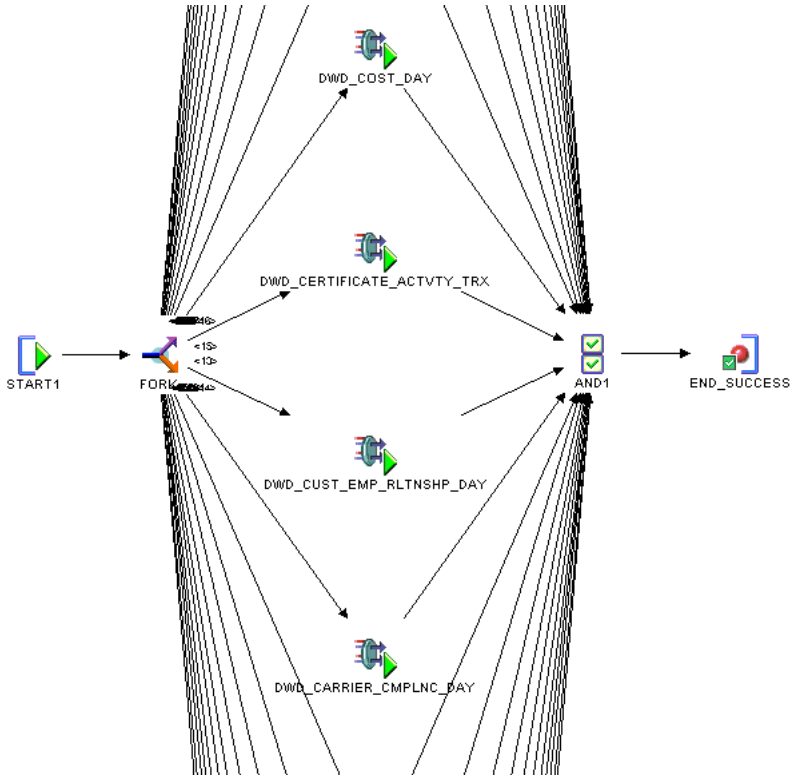
- ORDM_DERIVED_FLW
- ORDM_AGG_N_DEP_FLW
- ORDM_AGG_DEP_FLW
- OLAP_MAP Mapping Flow
- ORDM_MNNG_FLW

## ORDM_DERIVED_FLW

The `ORDM_DERIVED_FLW` sub-process flow contains all the PL/SQL package code for populating derived tables, based on the content of the base, reference, and lookup tables.

Figure 4–3 shows the `ORDM_DERIVED_FLW` sub-process flow for populating derived tables.

***Figure 4–3 Intra-ETL Derived Process Flow***




After the ORDM_DERIVED_FLW starts successfully, it moves to the fork. The sub-process FORK performs the derived ETL execution (these run in parallel). For each activity, a record is inserted into a control table and the state is set to RUNNING and the respective ETL is executed. Once ETL execution completes successfully, the control table record that was inserted before ETL execution is updated with COMPLETED-SUCCESS status; otherwise it is updated with COMPLETED-ERROR status.

The AND activity specifies whether all the parallel activities run to completion. Then the flow switches to the next activity, for example END_SUCCESS.

This sub-process uses the following technologies:

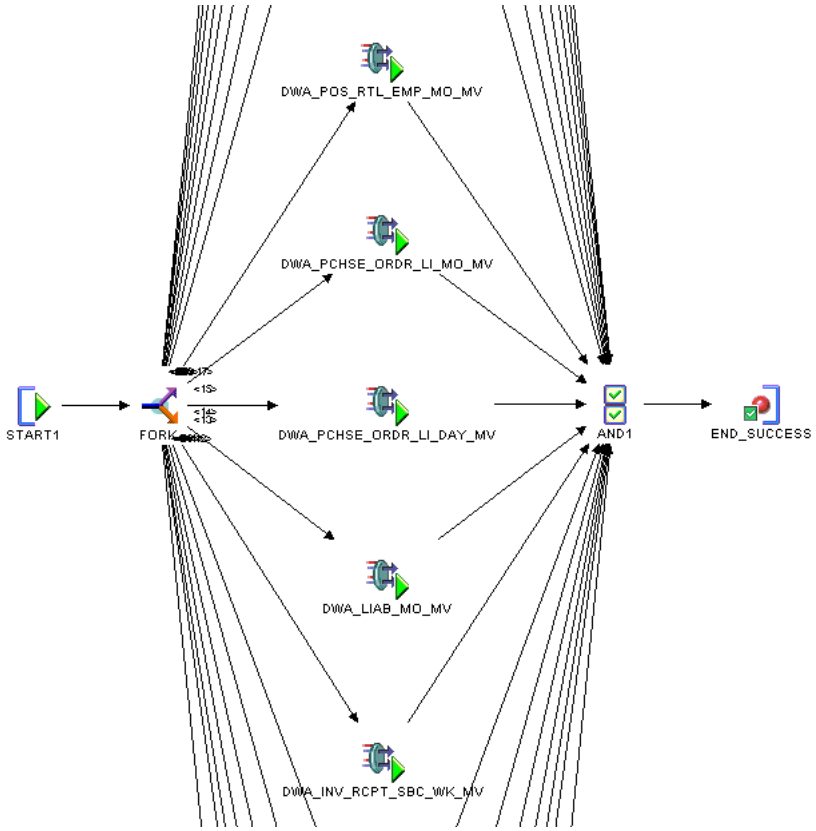
- Table: Whenever ETL package is executed, data is inserted into a derived table based on the values of ETL parameters in the DWC_ETL_PARAMETER control table.

## ORDM_AGG_N_DEP_FLW

For each activity, the ORDM_AGG_N_DEP_FLW sub-process flow invokes a PL/SQL procedure for refreshing materialized views. The activities in the sub-process flow are all independent, and hence can run in parallel. This sub-process flow has dependency on ORDM_DERIVED_FLW sub-process, that is, ORDM_AGG_N_DEP_FLW is executed only ORDM_DERIVED_FLW is executed successfully.

Figure 4–4 shows the ORDM_AGG_N_DEP_FLW sub-process flow for refreshing all independent materialized views.

***Figure 4–4 Intra-ETL Independent MV Process Flow***




After the ORDM_AGG_N_DEP_FLW is initiated and starts successfully the flow moves to the Fork. The FORK process makes the aggregates run in parallel. The AND activity specifies all the parallel aggregates must complete; the flow then switches over to the next activity, (for example, END_SUCCESS).

This sub-process uses the following technologies:

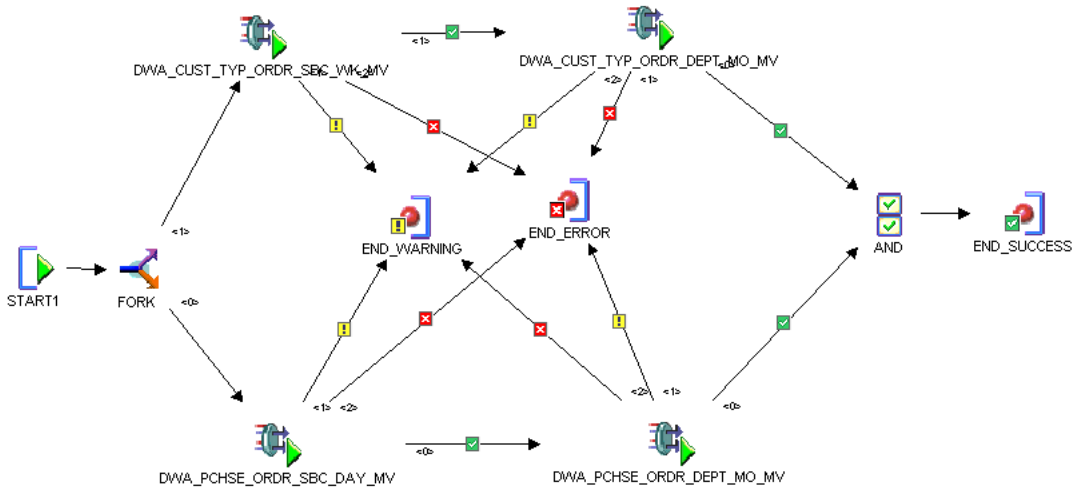
- Materialized View: A materialized view is used to hold the aggregation data. Whenever this is refreshed then the modified data are reflected in the corresponding aggregate table and this leads to a significant increase in the query execution performance. Moreover usage of materialized view allows Oracle Retail Data Model to make use of the Oracle Query Rewrite feature for better SQL optimization and hence improved performance.
- FAST Refresh: This refresh type is used to refresh the aggregates with only the incremental data (inserted and modified) in base and derived tables after the immediately previous refresh and this incremental refresh leads to much better performance and hence shorter intra-ETL window.

## ORDM_AGG_DEP_FLW

For each activity, the ORDM_AGG_DEP_FLW sub-process flow invokes a PL/SQL procedure for refreshing materialized views. The activities in the sub-process flow have dependencies. This sub-process flow has dependency on ORDM_DERIVED_FLW sub-process, that is, ORDM_AGG_DEP_FLW is executed only after ORDM_DERIVED_FLW runs successfully.

Figure 4–5 shows the ORDM_AGG_DEP_FLW sub-process flow for refreshing all independent materialized views.

*Figure 4–5 Intra-ETL Aggregate Process Flow*




After the ORDM_AGG_DEP_FLW is initiated and starts successfully the process flow moves to the Fork. The FORK process makes the aggregates to run in parallel. The AND activity specifies all the parallel aggregates must complete, then the flow switches over to the next activity, (for example, END_SUCCESS).

This sub-process uses the following technologies:

- Materialized View: A materialized view holds the aggregation data. Whenever this is refreshed then the modified data is reflected in the corresponding aggregate table and this leads to a significant increase in the query execution performance. Moreover usage of materialized view allows Oracle Retail Data Model to make use of the Oracle Query Rewrite feature for better SQL optimization and hence improved performance.
- FAST Refresh: This refresh type is used to refresh the aggregates with only the incremental data (inserted and modified) in base and derived tables after the immediately previous refresh and this incremental refresh leads to much better performance and hence shorter intra-ETL window.

## OLAP_MAP Mapping Flow

The OLAP_MAP mapping invokes PKG_ORDM_OLAP_ETL_AW_LOAD.OLAP_ETL_AW_BUILD function of OLAP ETL package that can load from Oracle Retail Data Model reference and derived tables to Oracle Retail Data Model Analytical Workspace dimensions and cubes respectively and calculate the forecast data. It reads OLAP ETL parameters from DWC_OLAP_ETL_PARAMETER table.

Figure 4–6 shows the OLAP_MAP mapping that invokes the OLAP ETL package.

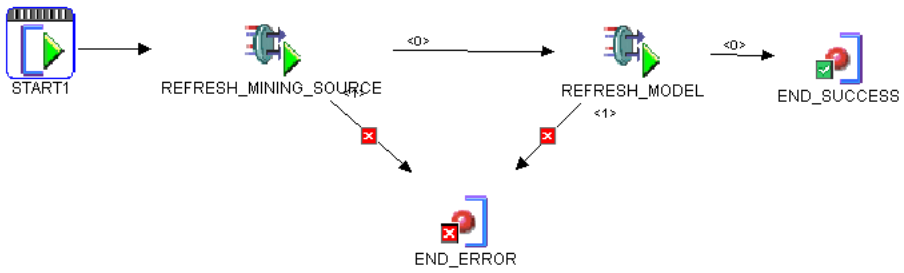
*Figure 4–6 OLAP Map Process Flow*


## ORDM_MNNG_FLW

The mining process flow, ORDM_MNNG_FLW, first refreshes mining source materialized views then refreshes the mining models.

Figure 4–7 shows the mining process flow, ORDM_MNNG_FLW.

*Figure 4–7 Mining Flow Process*




# Performing an Initial Load of an Oracle Retail Data Model Warehouse

Performing an initial load of an Oracle Retail Data Model is a multistep process:

1. Load the reference, lookup, and base tables Oracle Retail Data Model warehouse by executing the source-ETL that you have written using the guidelines given in "Creating Source-ETL for Oracle Retail Data Model" on page 4-2.
2. Load the remaining structures in the Oracle Retail Data Model, by taking the following steps:
   a. Update the parameters in `DWC_ETL_PARAMETER` control table in the `ordm_sys` schema so that the ETL can use this information (that is, the beginning and end date of the ETL period) when loading the derived and aggregate tables and views.

      For an initial load of an Oracle Retail Data Model warehouse, specify the values shown in the following table.

| Columns | Value |
|---|---|
| PROCESS_NAME | 'ORDM-INTRA-ETL' |
| FROM_DATE_ETL | The beginning date of the ETL period. |
| TO_DATE_ETL | The ending date of the ETL period. |

**See:** *Oracle Retail Data Model Reference* for more information on the DWC_ETL_PARAMETER control table.

**b.** Update the Oracle Retail Data Model OLAP ETL parameters in DWC_OLAP_ETL_PARAMETER control table in the ordm_sys schema to specify the build method and other build characteristics so that the ETL can use this information when loading the OLAP cube data.

For an initial load of the analytic workspace, specify values following the guidelines in Table 4–1.

***Table 4–1 Explanation of Load Parameters in DWC_OLAP_ETL_PARAMETER Along with (typical) Initial Load Values***

| Column Name | Value |
|---|---|
| BUILD_METHOD | Use the build method parameter to indicate a full or a fast (partial) refresh. The following are the possible values for BUILD_METHOD:<br>■ C: Complete refresh clears all dimension values before loading. (Default value).<br>■ F: Fast refresh of a cube materialized view, which performs an incremental refresh and re-aggregation of only changed rows in the source table.<br>■ ?: Fast refresh if possible, and otherwise a complete refresh.<br>■ P: Recomputes rows in a cube materialized view that are affected by changed partitions in the detail tables.<br>■ S: Fast solve of a compressed cube. A fast solve reloads all the detail data and re-aggregates only the changed values.<br>Note:<br>In a fast refresh, only changed rows are inserted in the cube and the affected areas of the cube are re-aggregated.<br>The C, S, and ? methods always succeed and can be used on any cube.<br>The F and P methods require that the cube have a materialized view that was created as a fast or a rewrite materialized view.<br>For initial load, specify C which specifies a complete refresh which clears all dimension values before loading. |
| BUILD_METHOD_TYPE | HISTORICAL or INCREMENTAL indicating whether this is an initial load of OLAP AW or an incremental load of the OLAP AW.<br>For initial load, specify HISTORICAL |
| CALC_FCST | One of the following values depending on whether you calculate forecast cubes:<br>■ Y specifies calculate forecast cubes.<br>■ N specifies do not calculate forecast cubes.<br>For initial load, specify Y. |

***Table 4–1 (Cont.) Explanation of Load Parameters in DWC_OLAP_ETL_PARAMETER Along with (typical) Initial Load Values***

| Column Name | Value |
|---|---|
| CUBENAME | One of the following values that specifies the cubes you build:<br>■ ALL specifies a build of the cubes in the Oracle Retail Data Model analytic workspace.<br>■ *cubename*[[\|*cubename*]...] specifies one or more cubes to build.<br>For initial load, specify ALL. |
| FCST_MTHD | If the value for the CALC_FCST column is Y, then specify AUTO; otherwise, specify NULL. Another valid value is MANUAL which sets the forecasting approach to APPMANUAL instead of APPAUTO (APPAUTO and APPMANUAL are internal terms used by Oracle OLAP Forecasting command). This parameter is ignored if CALC_FCST column is N.<br>For initial load, specify AUTO. |
| FCST_ST_MO | If the value for the CALC_FCST column is Y, then specify value specified as BY *YYYY* M*X* which is the "end business month" of a historical period; otherwise, specify NULL. This parameter is ignored if CALC_FCST column is N. *X* is month number in a year.<br>For example:<br>BY 2011 M7, or BY 2011 M11<br>For the sample data present in the sample schema installed with Oracle Retail Data Model Sample Reports, for initial load, specify:<br>BY 2012 M1 |
| HIST_ST_MO | If the value for the CALC_FCST column is Y, then specify value specified as BY *YYYY* M*X* which is the "start business month" of historical data; otherwise, specify NULL. This parameter is ignored if CALC_FCST column is N. *X* is the month number in a year.<br>For example: BY 2011 M7, or BY 2011 M11<br>For the sample data present in the sample schema installed with Oracle Retail Data Model Sample Reports, for initial load, specify:<br>BY 2010 M1 |
| MAXJOBQUEUES | A decimal value that specifies the number of parallel processes to allocate to this job. (Default value is 4.) The value that you specify varies depending on the setting of the JOB_QUEUE_PROCESSES database initialization parameter.<br>For initial load, specify 4 |
| NO_FCST_YRS | If the value for the CALC_FCST column is Y, specify a decimal value that specifies how many years forecast data to calculate; otherwise, specify NULL. This parameter is ignored if CALC_FCST column is N.<br>For initial load, specify 2 |
| OTHER1 | Not used. Specify NULL. |
| OTHER2 | Not used. Specify NULL. |
| PROCESS_NAME | 'ORDM-OLAP-ETL' |

**c.** Execute the intra-ETL as described in "Executing the Default Oracle Retail Data Model Intra-ETL" on page 4-16.

**See also:** "Refreshing the Data in Oracle Retail Data Model Warehouse" on page 4-18

## Executing the Default Oracle Retail Data Model Intra-ETL

Oracle Retail Data Model provides you with a database package named PKG_INTRA_ETL_PROCESS that is a complete Intra-ETL process. This intra-ETL process is

composed of individual population programs (database packages and MV refresh scripts). This package includes the dependency for each individual program and executes the programs in the proper order.

You can execute the intra-ETL packages provided with Oracle Retail Data Model in the following ways.

- As a Workflow within Oracle Warehouse Builder as described in "Executing the ORDM_INTRA_ETL_FLW Workflow from Oracle Warehouse Builder" on page 4-17.
- Without using Oracle Warehouse Builder Workflow as described in "Executing the Intra-ETL Without Using Oracle Warehouse Builder" on page 4-17.

> **See:** "Monitoring the Execution of the Intra-ETL Process" on page 4-20, "Recovering an Intra ETL Process" on page 4-21, and "Troubleshooting Intra-ETL Performance" on page 4-21.

### Executing the ORDM_INTRA_ETL_FLW Workflow from Oracle Warehouse Builder

You can execute the `ORDM_INTRA_ETL_FLW` process from within Oracle Warehouse Builder.

**To deploy the ORDM_INTRA_ETL_FLW process flow, take the following steps:**

1. Confirm that Oracle Warehouse Builder Workflow has been installed as described in *Oracle Retail Data Model Installation Guide*.
2. Within Oracle Warehouse Builder, go to the **Control Center Manager**.
3. Select `OLAP_PFLW`, then select `AGR_PFLW`, then select the main process flow `ORDM_INTRA_ETL_FLW`.
4. Right-click `ORDM_INTRA_ETL_FLW` and select **set action**.
   - If this is the first deployment, set action to **Create**.
   - If this is a later deployment, set action to **Replace**.

   Deploy the process flow.

After the deployment finishes successfully, `ORDM_INTRA_ETL_FLW` is ready to execute.

> **See:** *Oracle Warehouse Builder Sources and Targets Guide* for information about Oracle Warehouse Builder.

### Executing the Intra-ETL Without Using Oracle Warehouse Builder

You do not have to execute the Intra-ETL as a workflow in Oracle Warehouse Builder. You can, instead, execute it as follows:

- Executing the Intra-ETL by Using the PKG_INTRA_ETL_PROCESS.RUN Procedure

#### Executing the Intra-ETL by Using the PKG_INTRA_ETL_PROCESS.RUN Procedure

You can use the `PKG_INTRA_ETL_PROCESS.RUN` procedure to start the Intra-ETL process. This procedure can be invoked manually, or by another process such as Source-ETL, or according to a predefined schedule such as Oracle Job Scheduling.

The database package `PKG_INTRA_ETL_PROCESS` is a complete Intra-ETL process composed of individual population programs (database packages and MV refresh scripts). This package includes dependency for each individual program and executes the programs in the proper order.

PKG_INTRA_ETL_PROCESS.RUN does not accept parameters. This procedure calls other programs in the correct order to load the data for current day (according to the Oracle system date). The PKG_INTRA_ETL_PROCESS.RUN procedure uses the DWC_ control tables to track the loading progress and results.

# Refreshing the Data in Oracle Retail Data Model Warehouse

The section, "Performing an Initial Load of an Oracle Retail Data Model Warehouse" on page 4-14 describes how to perform an initial load of an Oracle Retail Data Model data warehouse. After this initial load, you must load new data into your Oracle Retail Data Model data warehouse regularly so that it can serve its purpose of facilitating business analysis.

To load new data into your Oracle Retail Data Model warehouse, you extract the data from one or more operational systems and copy that data into the warehouse. The challenge in data warehouse environments is to integrate, rearrange and consolidate large volumes of data over many systems, thereby providing a new unified information base for business intelligence.

The successive loads and transformations must be scheduled and processed in a specific order that is determined by your business needs. Depending on the success or failure of the operation or parts of it, the result must be tracked and subsequent, alternative processes might be started.

You can do a full incremental load of the relational tables and views, OLAP cubes, and data mining models simultaneously, or you can refresh the data sequentially, as follows:

1. Refreshing Oracle Retail Data Model Relational Tables and Views
2. Refreshing Oracle Retail Data Model OLAP Cubes
3. Refreshing Oracle Retail Data Model Data Mining Models

In either case, you can manage errors during the execution of the intra-ETL as described in "Managing Errors During Oracle Retail Data Model Intra-ETL Execution" on page 4-20.

## Refreshing Oracle Retail Data Model Relational Tables and Views

Refreshing the relational tables and views in an Oracle Retail Data Model is a multi-step process:

1. Refresh the reference, lookup, and base tables in the Oracle Retail Data Model warehouse with transactional data by executing the source-ETL that you have written.
2. Update the parameters of the DWC_ETL_PARAMETER control table in the ordm_sys schema. For an incremental load of an Oracle Retail Data Model warehouse, specify the values shown in the following table (that is, the beginning and end date of the ETL period).

| Columns | Value |
|---|---|
| PROCESS_NAME | 'ORDM-INTRA-ETL' |
| FROM_DATE_ETL | The beginning date of the ETL period. |
| TO_DATE_ETL | The ending date of the ETL period. |

**See:** *Oracle Retail Data Model Reference* for more information on the DWC_ETL_PARAMETER control table.

3. Refresh the derived tables and aggregate tables which are materialized views in Oracle Retail Data Model by executing the DRVD_FLOW and AGGR_FLOW subprocess of the ORDM_INTRA_ETL_FLW process flow. See "Executing the Default Oracle Retail Data Model Intra-ETL" on page 4-16 for more information.

**See also:** *Oracle Warehouse Builder Sources and Targets Guide*

## Refreshing Oracle Retail Data Model OLAP Cubes

On a scheduled basis you must update the OLAP cube data with the relational data that has been added to the Oracle Retail Data Model data warehouse since the initial load of the OLAP cubes.

You can execute the Oracle Retail Data Model ETL to update the OLAP cubes in the following ways

- Refresh *all* of the data in the warehouse by executing the Oracle Warehouse Builder Workflow ORDM_INTRA_ETL_FLW in one of the ways that are described in "Executing the Default Oracle Retail Data Model Intra-ETL" on page 4-16.

  The OLAP Cubes are populated through OLAP_MAP which is a part of Oracle Retail Data Model intra-ETL main workflow ORDM_INTRA_ETL_FLW.

- Refresh *only* the OLAP cube data by executing the OLAP_MAP Oracle Warehouse Builder mapping in the Oracle Warehouse Builder control center.

**Note:** You must refresh the corresponding materialized view of the OLAP cubes you are refreshing before you execute OLAP_MAP. (For the mapping between OLAP cube and materialized views, refer to *Oracle Retail Data Model Reference*.

Take these steps to perform an incremental load of the analytic workspace that is part of the Oracle Retail Data Model warehouse:

1. Update the aggregate tables which are materialized views in Oracle Retail Data Model. See "Refreshing Oracle Retail Data Model Relational Tables and Views" on page 4-18 for more information.
2. Execute the intra-ETL to load the cube data in one of the ways described in "Executing the Default Oracle Retail Data Model Intra-ETL" on page 4-16.
3. If necessary, to recover from errors during the execution of OLAP_MAP take the following steps.
   a. Change the value of the BUILD_METHOD column of the ordm_sys.DWC_OLAP_ETL_PARAMETER table to "C".
   b. In Oracle Warehouse Builder, rerun the OLAP_MAP map.

## Refreshing Oracle Retail Data Model Data Mining Models

The MINING_FLW sub-process flow of the ORDM_INTRA_ETL_FLW process flow triggers the data mining model refreshment. After the initial load of the warehouse, it is recommended to refresh the data mining models monthly. Refreshing the data models is integrated into the MINING_FLW sub-process flow. You can also manually refresh the data models.

The way you refresh a data mining model varies depending on whether you want to refresh all of the models or only one model:

- To manually refresh *all* mining models, call the following procedure.

  ```
  PKG_ordm_mining.REFRESH_MODEL( MONTH_CODE,P_PROCESS_NO)
  ```

  This procedure performs the following tasks for each model:

  1. Refreshes the mining source materialized views based on the latest data from ordm_sys schema.
  2. Trains each model on the new training data.
  3. Applies each model onto the new apply data set.

- To manually re-create only one mining model, you can call the corresponding procedure. For example, to re-create the employee combination model, you can call the following procedure.

  ```
  create_emp_cmbntn_glmr_model;
  ```

  "Tutorial: Customizing the Customer Life Time Value Prediction Data Mining Model" on page 3-3 provides detailed instructions for refreshing a single data mining model.

  **See also:** "Troubleshooting Data Mining Model Creation" on page 4-22

# Managing Errors During Oracle Retail Data Model Intra-ETL Execution

This topic discusses how you can identify and manage errors during intra-ETL execution. It contains the following topics:

- Monitoring the Execution of the Intra-ETL Process
- Recovering an Intra ETL Process
- Troubleshooting Intra-ETL Performance

## Monitoring the Execution of the Intra-ETL Process

Two `ordm_sys` schema control tables, `DWC_INTRA_ETL_PROCESS` and `DWC_INTRA_ETL_ACTIVITY`, monitor the execution of the intra-ETL process. These tables are documented in *Oracle Retail Data Model Reference*.

Each normal run (as opposed to an error-recovery run) of a separate intra-ETL execution performs the following steps:

1. Inserts a record into the `DWC_INTRA_ETL_PROCESS` table with a monotonically increasing system generated unique process key, `SYSDATE` as process start time, `RUNNING` as the process status, and an input date range in the `FROM_DATE_ETL` and `TO_DATE_ETL` columns.
2. Invokes each of the individual intra-ETL programs in the appropriate order of dependency. Before the invocation of each program, the procedure inserts a record into the intra-ETL Activity detail table, `DWC_INTRA_ETL_ACTIVITY`, with values for:
   - `ACTIVITY_KEY`, a system generated unique activity key.
   - `PROCESS_KEY`, the process key value corresponding to the intra-ETL process.
   - `ACTIVITY_NAME`, an individual program name.

- ACTIVITY_DESC, a suitable activity description.
- ACTIVITY_START_TIME, the value of SYSDATE.
- ACTIVITY_STATUS, the value of RUNNING.

3. Updates the corresponding record in the DWC_INTRA_ETL_ACTIVITY table for the activity end time and activity status after the completion of each individual ETL program (either successfully or with errors). For successful completion of the activity, the procedure updates the status as 'COMPLETED-SUCCESS'. When an error occurs, the procedure updates the activity status as 'COMPLETED-ERROR', and also updates the corresponding error detail in the ERROR_DTL column.
4. Updates the record corresponding to the process in the DWC_INTRA_ETL_ PROCESS table for the process end time and status, after the completion of all individual intra-ETL programs. When all the individual programs succeed, the procedure updates the status to 'COMPLETED-SUCCESS', otherwise it updates the status to 'COMPLETED-ERROR'.

You can monitor the execution state of the intra-ETL, including current process progress, time taken by individual programs, or the complete process, by viewing the contents of the DWC_INTRA_ETL_PROCESS and DWC_INTRA_ETL_ACTIVITY tables corresponding to the maximum process key. Monitoring can be done both during and after the execution of the intra-ETL procedure.

## Recovering an Intra ETL Process

To recover an intra-ETL process

1. Identify the errors by looking at the corresponding error details that are tracked against the individual programs in the DWC_INTRA_ETL_ACTIVITY table.
2. Correct the causes of the errors.
3. Re-invoke the intra-ETL process.

The ORDM_INTRA_ETL_FLW process identifies whether it is a normal run or recovery run by referring the DWC_INTRA_ETL_ACTIVITY table. During a recovery run, ORDM_INTRA_ETL_FLW executes only the necessary programs. For example, in the case of a derived population error as a part of the previous run, this recovery run executes the individual derived population programs which produced errors in the previous run. After their successful completion, the run executes the aggregate population programs and materialized view refresh in the appropriate order.

In this way, the intra-ETL error recovery is almost transparent, without involving the data warehouse or ETL administrator. The administrator must only correct the causes of the errors and re-invoke the intra-ETL process. The intra-ETL process identifies and executes the programs that generated errors.

## Troubleshooting Intra-ETL Performance

To troubleshoot the performance of the intra-ETL:

- Check the execution plan as described in "Checking the Execution Plan" on page 4-22.
- Monitor parallel DML executions as described in "Monitoring PARALLEL DML Executions" on page 4-22.
- Check that data mining models were created correctly as described in "Troubleshooting Data Mining Model Creation" on page 4-22.

### Checking the Execution Plan

Use SQLDeveloper or other tools to view the package body of the code generated by Oracle Warehouse Builder.

For example, take the following steps to examine a map:

1. Copy out the main query statement from code viewer.

   Copy from `"CURSOR "AGGREGATOR_c" IS …."` to end of the query, which is right above another `"CURSOR "AGGREGATOR_c$1" IS"`.

2. In SQLDeveloper worksheet, issue the following statement to turn on the parallel DML:

   ```
   Alter session enable parallel dml;
   ```

3. Paste the main query statement into another SQL Developer worksheet and view the execution plan by clicking `F6`.

   Carefully examine the execution plan to make the mapping runs according to a valid plan.

### Monitoring PARALLEL DML Executions

Check that you run the mapping in parallel mode by executing the following SQL statement to count the executed "Parallel DML/Query" operations:

```
column name format a50
column value format 999,999
SELECT NAME, VALUE
FROM GV$SYSSTAT
WHERE UPPER (NAME) LIKE '%PARALLEL OPERATIONS%'
  OR UPPER (NAME) LIKE '%PARALLELIZED%'
  OR UPPER (NAME) LIKE '%PX%'
;
```

If you run mapping in parallel mode, you should see "DML statements parallelized" increased by 1 (one) every time the mapping was invoked. If not, you do not see this increase, then the mapping was not invoked as "parallel DML".

If you see "queries parallelized" increased by 1 (one) instead, then typically it means that the `SELECT` statement inside of the `INSERT` was parallelized, but that `INSERT` itself was not parallelized.

> **See also:** "Parallel Execution in Oracle Retail Data Model" on page 2-14

### Troubleshooting Data Mining Model Creation

After the data mining models are created, check the error log in `ordm_sys.dwc_intra_etl_activity` table. For example, execute the following code.

```
set line 160
col ACTIVITY_NAME format a30
col ACTIVITY_STATUS format a20
col error_dtl format a80
select activity_name, activity_status,  error_dtl from dwc_intra_etl_activity;
```

If all models are created successfully, the activity_status is all `"COMPLETED-SUCCESS"`. If the activity_status is `"COMPLETED-ERROR"` for a certain step, check the ERROR_DTL column, and fix the problem accordingly.

The following examples illustrate how to troubleshoot some common error messages returned in ERROR_DTL and ACTIVITY_NAME when working with Oracle Retail Data Model:

- Example 4–2, "Troubleshooting an ORA-20991 Error for Oracle Retail Data Model"
- Example 4–3, "Troubleshooting the "Message not available ... [Language=ZHS]" Error"
- Example 4–4, "Troubleshooting an ORA-40113 Error for Oracle Retail Data Model"
- Example 4–5, "Troubleshooting an ORA-40112 Error for Oracle Retail Data Model"
- Example 4–6, "Troubleshooting an ORG-11130 Error for Oracle Retail Data Model"

#### *Example 4–2 Troubleshooting an ORA-20991 Error for Oracle Retail Data Model*

Assume that the returned error is `ORA-20991: Message not available ... [Language=ZHS]CURRENT_MONTH_KEY`.

This error may happen when there is not enough data in the DWR_BSNS_MO table. For example, if the calendar data is populated with 2004~2009 data, the mining model refresh for Year 2010 may result in this error.

To fix this error, execute the Oracle Retail Data Model calendar utility script again to populate the calendar with sufficient data. For example:

```
Execute Calendar_Population.run('2005-01-01',10);
```

See *Oracle Retail Data Model Reference* for information on the calendar population utility script.

#### *Example 4–3 Troubleshooting the "Message not available ... [Language=ZHS]" Error*

Assume that the returned error is `Message not available ... [Language=ZHS]`.

'ZHS' is a code for a language. The language name it relates to can appear as different name depending on the database environment. This error happens when ordm_sys.DWC_MESSAGE.LANGUAGE does not contain messages for the current language.

Check the values in the DWC_MESSAGE table and, if required, update to the language code specified by the Oracle session variable `USERENV('lang')`.

#### *Example 4–4 Troubleshooting an ORA-40113 Error for Oracle Retail Data Model*

Assume that the returned error is `ORA-40113: insufficient number of distinct target values, for 'create_chrn_dt_model`.

This error happens when the target column for the training model contains only one value or no value when it is expecting more than one value.

For example, for the customer churn prediction model, the target column is: dmv_cust_acct_src.chrn_ind

To troubleshoot this error:

1. Execute a SQL query to check if there are enough values in this column.

   Using the customer churn prediction model as an example, issue the following statement.

   ```
   select chrn_ind, count(*) from DMV_CUST_ACCT_SRC  group by chrn_ind;
   ```

   The following is a result of the query.

```
CHRN_IND   COUNT(*)
-- -----------
0         10000
```

2. Check whether `dwr_cust.prmry_eff_to_dt` column has few non-null values.
3. Execute the following statement to refresh the mining source materialized views:

```
exec pkg_ordm_mining.refresh_mining_source;
```

***Example 4–5 Troubleshooting an ORA-40112 Error for Oracle Retail Data Model***

Assume that the returned error is:

```
ORA-40112:insufficient number of valid data rows, for " create_chrn_dt_model "
```

For this model, the target column is `dmv_cust_acct_src.chrn_ind`.

To troubleshoot this error:

1. Execute the following SQL statement:

```
select count(chrn_ind) from dmv_cust_acct_src;
```

2. Check to see that the value returned by this query is greater than 0 (zero) and similar to number of customers. If the number is 0 or too small, check the data in source tables of mining source materialized view, `dmv_cust_acct_src` :

```
DWB_RTL_TRX
DWR_CUST
```

***Example 4–6 Troubleshooting an ORG-11130 Error for Oracle Retail Data Model***

Assume that the returned error is `ORG-11130:no data found in the collection, for "create_sentiment_svm_model"`.

This error occurs when there is not enough data in the source table for customer sentiment model training: `dm_cust_cmnt` .

To ensure that some text is loaded for customer sentiment analysis:

1. Issue the following SQL statement:

```
SELECT cust_key, count(cust_cmmnt)
from dm_cust_cmnt
group by cust_key;
```

2. Check the number of text comments from the `dm_cust_cmnt`.
3. If there is not enough data in the dm_cust_cmnt table, check the ETL logic of `dm_cust_intrqacn_cmnt` table first, then check data in the base interaction event tables.

# 5

# Report and Query Customization

This chapter provides information about creating reports, queries, and dashboards against the data in Oracle Retail Data Model warehouse. It contains the following topics:


- Reporting Approaches in Oracle Retail Data Model
- Customizing Oracle Retail Data Model Reports
- Writing Your Own Queries and Reports
- Optimizing Star Queries
- Troubleshooting Oracle Retail Data Model Report Performance
- Writing As Is and As Was Queries
- Tutorial: Creating a New Oracle Retail Data Model Dashboard
- Tutorial: Creating a New Oracle Retail Data Model Report


## Reporting Approaches in Oracle Retail Data Model

There are two main approaches to creating reports from data in Oracle Retail Data Model warehouse:

### Relational Reporting

With relational reporting, reports are created against the analytical layer entities using the fact entities as the center of the star with the reference entities (that is, DWR_ and DWL_ tables) acting as the dimensions of the star. Typically the fact entities include the derived and aggregate entities (that is, DWD_ tables and DWA_ objects). When you need more detailed reports, generate the reports with base tables (DWB_), Reference tables (DWR_) and lookup tables (DWL_).

The reference tables (that is, DWR_ tables) typically represent dimensions which contain a business hierarchy and are present in the form of snowflake entities containing a table for each level of the hierarchy. This allows us to attach the appropriate set of reference entities for the multiple subject area and fact entities composed of differing granularity.

For example, you can use the set of tables comprising DWR_SKU_ITEM and DWR_ITEM, DWR_ITEM_SBC, DWR_ITEM_CLASS, and DWR_ITEM_DEPT tables to query against the SKU ITEM level Space Utilization entity such as DWD_SPACE_UTLZTN_ITEM_DAY. On the other hand, you need to use the higher level snowflakes at ITEM level and above such as DWR_ITEM, DWR_ITEM_SBC, DWR_ITEM_CLASS, and DWR_ITEM_DEPT to query against the DEPARTMENT level Space Utilization entity such as DWA_SPACE_UTLZTN_DEPT_DAY.

The lookup tables (that is tables, with the DWL_ prefix) represent the simpler dimensions comprising a single level containing a flat list of values.

### OLAP Reporting

With OLAP reporting, Oracle OLAP cubes are accessed using SQL against the dimension and cube (fact) views. Cubes and dimensions are represented using a star schema design. Dimension views form a constellation around the cube (or fact) view. The dimension and cube views are relational views with names ending with _VIEW. Typically, the dimension view used in the reports is named *dimension_hierarchy*_VIEW and the cube view is named *cube*_VIEW.

Unlike the corresponding relational dimension objects stored in DWR_ tables, the OLAP dimension views contains information relating to the entire dimension including all the levels of the hierarchy logically partitioned by a level column (identified as level_name). On a similar note, the cube views also contain the facts pertaining to the cross-combination of the levels of individual dimensions which are part of the cube definition. Also the join from the cube view and the dimension views are based on the dimension keys along with required dimension level filters.

Although the OLAP views are also modeled as a star schema, there are certain unique features to the OLAP reporting methodology which requires special modeling techniques in Oracle Business Intelligence Suite Enterprise Edition.

> **See also:** The Oracle By Example tutorial, entitled "Using Oracle OLAP 11g With Oracle BI Enterprise Edition". To access the tutorial, open the Oracle Learning Library in your browser by following the instructions in "Oracle Technology Network" on page xii; and, then, search for the tutorials by name.

The rest of this chapter explains how to create Oracle Retail Data Model reports. For examples of Oracle Retail Data Model reports, see:

- "Writing As Is and As Was Queries" on page 5-6
- "Tutorial: Creating a New Oracle Retail Data Model Dashboard" on page 5-12
- "Tutorial: Creating a New Oracle Retail Data Model Report" on page 5-20
- The reports provided with Oracle Retail Data Model that are documented in *Oracle Retail Data Model Reference*.

## Customizing Oracle Retail Data Model Reports

Sample reports and dashboards are delivered with Oracle Retail Data Model. These reports illustrate the analytic capabilities provided with Oracle Retail Data Model -- including the OLAP and data mining capabilities.

> **See:** *Oracle Retail Data Model Installation Guide* for more information on installing the reports and deploying the Oracle Retail Data Model RPD and webcat on the Business Intelligence Suite Enterprise Edition instance.

The Oracle Retail Data Model reports were developed using Oracle Business Intelligence Suite Enterprise Edition which is a comprehensive suite of enterprise business intelligence products that delivers a full range of analysis and reporting capabilities. Thus, the reports also illustrate the ease with which you can use Oracle Business Intelligence Suite Enterprise Edition Answers and Dashboard presentation tools to create useful reports.

Oracle Business Intelligence Suite Enterprise Edition products are used to display and customize the predefined sample reports, or new reports can be created:

- **Oracle BI Answers**. Provides end user ad hoc capabilities in a pure Web architecture. You can interact with a logical view of the information, completely hidden from data structure complexity while simultaneously preventing runaway queries. You can easily create charts, pivot tables, reports, and visually appealing dashboards.
- **Oracle BI Interactive Dashboards**. Provides any knowledge worker with intuitive, interactive access to information. You can work with live reports, prompts, charts, tables, pivot tables, graphics, and tickers, and have full capability for drilling, navigating, modifying, and interacting with these results.

> **See:** *Oracle Retail Data Model Reference* for detailed information on the sample reports.

## Writing Your Own Queries and Reports

The `ordm_sys` schema defines the relational tables and views in Oracle Retail Data Model. You can use any SQL reporting tool to query and report on these tables and views.

Oracle Retail Data Model also supports On Line Analytic Processing (OLAP) reporting using OLAP cubes defined in the `ordm_sys` schema. You can query and write reports on OLAP cubes using SQL tools to query the views that are defined for the cubes or using OLAP tools to directly query the OLAP components.

> **See also:** "Reporting Approaches in Oracle Retail Data Model" on page 5-1, "Oracle OLAP Cube Views" on page 3-13, and the discussion on querying dimensional objects in *Oracle OLAP User's Guide*.

***Example 5–1 Creating a Relational Query for Oracle Retail Data Model***

For example, assume that you want to know the total number of hours worked by each employee for September 2011. To answer this question, you might have to query the tables described in the following table.

| Entity Name | Table Name | Description |
|---|---|---|
| Employee Labor Derived | DWD_EMP_LBR | Summary of employee labor in a day. |
| Employee | DWR_EMP | An individual who works for the retail organization, accepts direction from the retail store management and satisfies the statutory criteria requiring that payroll taxes and benefit contributions be paid by the retailer. |
| Business Month | DWR_BSNS_MO | Months as defined in the Business calendar. |

To perform this query, execute the following SQL statement:

```
SELECT DE.emp_key,DBM.mo_cd,SUM(DEL.hrs_wrkd) AS tot_hrs_wrkd
FROM dwd_emp_lbr DEL, dwr_emp DE, dwr_bsns_mo DBM
WHERE DEL.emp_key = DE.emp_key
AND TO_DATE(DEL.day_key,'YYYYMMDD') BETWEEN DBM.mo_strt_dt AND DBM.mo_end_dt
AND DBM.mo_desc = 'BY 2011 M9'
GROUP BY DE.emp_key, DBM.mo_cd;
```

The result of this query is:

```
EMP_KEY    MO_CD       TOT_HRS_WRKD
1005875    20110905    82
1005479    20110905    46
1005552    20110905    32
1005539    20110905    46
1005813    20110905    44
1005460    20110905    50
1006208    20110905    54
1005829    20110905    56
```

# Optimizing Star Queries

A typical query in the access layer is a join between the fact table and some number of dimension tables and is often referred to as a star query. In a star query each dimension table is joined to the fact table using a primary key to foreign key join. Normally the dimension tables do not join to each other.

Typically, in this kind of query all of the `WHERE` clause predicates are on the dimension tables and the fact table. Optimizing this type of query is very straight forward.

To optimize this type of query, do the following:

- Create a bitmap index on each of the foreign key columns in the fact table or tables
- Set the initialization parameter `STAR_TRANSFORMATION_ENABLED` to `TRUE`.

Using this option enables the optimizer feature for star queries which is off by default for backward compatibility.

If your environment meets these two criteria, your star queries should use a powerful optimization technique that rewrites or transforms your SQL called star transformation. Star transformation executes the query in two phases:

1. Retrieves the necessary rows from the fact table (row set).
2. Joins this row set to the dimension tables.

The rows from the fact table are retrieved by using bitmap joins between the bitmap indexes on all of the foreign key columns. The end user never needs to know any of the details of `STAR_TRANSFORMATION`, as the optimizer automatically chooses `STAR_TRANSFORMATION` when it is appropriate.

Example 5–2, "Star Transformation" gives the step by step process to use `STAR_TRANSFORMATION` to optimize a star query.

#### *Example 5–2 Star Transformation*

A business question that could be asked against the star schema in Figure 5-1, "Star Schema Diagram" would be "What was the total number of Widgets sold in Nashua during the month of May 2008?"

1. The original query.

```
SELECT SUM(DRSRL.qty) tot_Widgets_in_Nashua
FROM dwb_rtl_sl_rtrn_li DRSRL, dwr_cust DC, dwr_sku_item DSI, dwr_day DD
WHERE DRSRL.cust_key = DC.cust_key
AND   DRSRL.sku_item_key = DSI.sku_item_key
AND   DRSRL.day_key = DD.bsns_day_key
AND   DC.city = 'Nashua'
AND   DSI.sku_item_name = 'Widgets'
AND   DD.bsns_mo_desc = 'BY 2008 M5'
```

```
;
```

As you can see all of the where clause predicates are on the dimension tables and the fact table (dwb_rtl_sl_rtrn_li) is joined to each of the dimensions using their foreign key, primary key relationship.

2. Take the following actions:
   a. Create a bitmap index on each of the foreign key columns in the fact table or tables.
   b. Set the initialization parameter STAR_TRANSFORMATION_ENABLED to TRUE.
3. The rewritten query. Oracle rewrites and transfers the query to retrieve only the necessary rows from the fact table using bitmap indexes on the foreign key columns:

```
SELECT SUM(DRSRL.qty) tot_Widgets_sld_in_Nashua
FROM dwb_rtl_sl_rtrn_li DRSRL
WHERE DRSRL.cust_key IN (SELECT DC.cust_key FROM dwr_cust DC WHERE DC.city =
'Nashua')
AND   DRSRL.sku_item_key IN (SELECT DSI.sku_item_key FROM dwr_sku_item DSI
WHERE DSI.sku_item_name = 'Widgets')
AND   DRSRL.day_key IN (SELECT DD.bsns_day_key FROM dwr_day DD WHERE DD.bsns_
mo_desc = 'BY 2008 M5')
;
```

   By rewriting the query in this fashion you can now leverage the strengths of bitmap indexes. Bitmap indexes provide set based processing within the database, allowing you to use various fact methods for set operations such as AND, OR, MINUS, and COUNT. So, you use the bitmap index on day_key to identify the set of rows in the fact table corresponding to sales in May 2008. In the bitmap the set of rows are actually represented as a string of 1's and 0's. A similar bitmap is retrieved for the fact table rows corresponding to the sale of the Widgets and another is accessed for sales made in Nashua. At this point there are three bitmaps, each representing a set of rows in the fact table that satisfy an individual dimension constraint. The three bitmaps are then combined using a bitmap AND operation and this newly created final bitmap is used to extract the rows from the fact table needed to evaluate the query.
4. Using the rewritten query, Oracle joins the rows from fact tables to the dimension tables.

   The join back to the dimension tables is normally done using a hash join, but the Oracle Optimizer selects the most efficient join method depending on the size of the dimension tables.

The typical execution plan for a star query when STAR_TRANSFORMATION has kicked in. The execution plan may not look exactly as you expect. There is no join back to the dwr_cust table after the rows have been successfully retrieved from the dwb_rtl_sl_rtrn_li table. In the select list, there is not anything actually selected from the dwr_cust table so the optimizer knows not to bother joining back to that dimension table. You may also notice that for some queries even if STAR_TRANSFORMATION does kick in it may not use all of the bitmap indexes on the fact table. The optimizer decides how many of the bitmap indexes are required to retrieve the necessary rows from the fact table. If an additional bitmap index would not improve the selectivity, the optimizer does not use it. The only time you see the dimension table that corresponds to the excluded bitmap in the execution plan is during the second phase, or the join back phase.

# Troubleshooting Oracle Retail Data Model Report Performance

Take the following actions to identify problems generating a report created using Oracle Business Intelligence Suite Enterprise Edition:

1. In the (Online) Oracle BI Administrator Tool, select **Manage,** then **Security**, then **Users**, and then **ordm**.

   Ensure that the value for **Logging level** is `7`.

2. Open the Oracle Retail Data Model Repository, select **Manage**, and then **Cache**.
3. In the right-hand pane of the Cache Manager window, select all of the records, then right-click and select **Purge**.
4. Run the report or query that you want to track using the SQL log.
5. Open the query log file (`NQQuery.log`) under `OracleBI\server\Log`.

   The last query SQL is the log of the report you have just run. If an error was returned in your last accessed report, there is an error at the end of this log.

The following examples illustrate how to use these error messages:

- Example 5–3, "Troubleshooting a Report: A Table Does Not Exist"
- Example 5–4, "Troubleshooting a Report: When the Database is Not Connected"

***Example 5–3 Troubleshooting a Report: A Table Does Not Exist***

Assume the log file shows the following error:

```
Query Status: Query Failed: [encloser: 17001] Oracle Error code:
942, message: ORA-00942: table or view does not exist.
```

This error occurs when the physical layer in your Oracle Business Intelligence Suite Enterprise Edition repository uses a table which actually does not exist in the Database.

To find out which table has problem:

1. Copy the SQL query to the database environment.
2. Execute the query.

The table which does not exist is marked out by the database client.

***Example 5–4 Troubleshooting a Report: When the Database is Not Connected***

Assume the log file contains the following error.

**Error:** `Query Status: Query Failed: [nQSError: 17001] Oracle Error code: 12545, message: ORA-12545: connect failed because target host or object does not exist.`

**Meaning:** This error occurs when the Database is not connected.

**Action:** Check connecting information in physical layer and ODBC connection to ensure that the repository is connecting to the correct database.

# Writing As Is and As Was Queries

Two common query techniques are "as is" and "as was" queries:

- Characteristics of an As Is Query

- Characteristics of an As Was Query
- Examples: As Is and As Was Queries


## Characteristics of an As Is Query

An As Is query has the following characteristics:

- The resulting report shows the data as it happened.
- The snowflake dimension tables are also joined using the surrogate key columns (that is the primary key and foreign key columns).
- The fact table is joined with the dimension tables (at leaf level) using the surrogate key column.
- Slowly-changing data in the dimensions are joined with their corresponding fact records and are presented individually.
- It is possible to add up the components if the different versions share similar characteristics.

## Characteristics of an As Was Query

An As Was query (also known as point-in-time analysis) has the following characteristics:

- The resulting report shows the data that would result from freezing the dimensions and dimension hierarchy at a specific point in time.
- Each snowflake table is initially filtered by applying a point-in-time date filter which selects the records or versions which are valid as of the analysis date. This structure is called the point-in-time version of the snowflake.
- The filtered snowflake is joined with an unfiltered version of itself by using the natural key. All of the snowflake attributes are taken from the point-in-time version alias. The resulting structure is called the composite snowflake.
- A composite dimension is formed by joining the individual snowflakes on the surrogate key.
- The fact table is joined with the composite dimension table at the leaf level using the surrogate key column.
- The point-in-time version is super-imposed on all other possible SCD versions of the same business entity -- both backward as well as forward in time. Joining in this fashion gives the impression that the dimension is composed of only the specific point-in-time records.
- All of the fact components for various versions add up correctly due to the super-imposition of point-in-time attributes within the dimensions.

## Examples: As Is and As Was Queries

Based on the data shown in Data used for the examples illustrates the characteristics of As Is and As Was queries:

- Example 5–5, "As Is Query for Sales Split by Item Subclass"
- Example 5–6, "As Was Query for Sales Split by Item Subclass"

### Data used for the examples

Assume that the data warehouse has DWR_SKU_ITEM , DWR_ITEM , DWR_ITEM_SBC and DWB_RTL_SL_RTRN_LI fact tables. As of January 1, 2012, these tables include the following values.

**SKU Item Table**

| SKU_ITEM_KEY | SKU_ITEM_NBR | ITEM_KEY | SKU_ITEM_NAME | EFF_FROM_DT | EFF_TO_DT |
|---|---|---|---|---|---|
| 1 | 389338009222279 | 1 | Antacid | 1/1/2011 | 12/31/2099 |
| 2 | 41204000420118700 | 2 | Apple Tea | 1/1/2011 | 12/31/2099 |
| 3 | 3632320042915860 | 3 | Vitamin Water | 1/1/2011 | 12/31/2099 |
| 4 | 38465200581121200 | 4 | Mineral Water | 1/1/2011 | 12/31/2099 |

**Item Table**

| ITEM_KEY | ITEM_NBR | SBC_KEY | ITEM_NAME | EFF_FROM_DT | EFF_TO_DT |
|---|---|---|---|---|---|
| 1 | 100309620 | 10 | Antacid | 1/1/2011 | 12/31/2099 |
| 2 | 100309700 | 1 | Apple Tea | 1/1/2011 | 12/31/2099 |
| 3 | 100309970 | 1 | Vitamin Water | 1/1/2011 | 12/31/2099 |
| 4 | 100309110 | 1 | Mineral Water | 1/1/2011 | 12/31/2099 |

**Item Subclass Table**

| SBC_KEY | SBC_CD | ITEM_CLAS_KEY | SBC_NAME | EFF_FROM_DT | EFF_TO_DT |
|---|---|---|---|---|---|
| 1 | BVGS | 1 | BEVERAGES | 1/1/2011 | 12/31/2099 |
| 10 | HLTH | 10 | HEALTH | 1/1/2011 | 12/31/2099 |

**Retail Sales Table**

| DAY_KEY | SKU_ITEM_KEY | SLS_AMT |
|---|---|---|
| 20111230 | 1 | 100 |
| 20111230 | 2 | 100 |
| 20111230 | 3 | 100 |
| 20111230 | 4 | 100 |
| 20111231 | 1 | 100 |
| 20111231 | 2 | 100 |
| 20111231 | 3 | 100 |
| 20111231 | 4 | 100 |

Assume that the following event occurred in January 2012:

On January 2, 2012 Apple Tea and Vitamin Water, these two SKUs moved from subclass Beverages to Health. Consequently, as shown, January 3, 2012, DWR_SKU_

ITEM, DWR_ITEM, DWR_ITEM_SBC and DWB_RTL_SL_RTRN_LI tables have new data.

**SKU Item Table**

| SKU_ITEM_KEY | SKU_ITEM_NBR | ITEM_KEY | SKU_ITEM_NAME | EFF_FROM_DT | EFF_TO_DT |
|---|---|---|---|---|---|
| 1 | 389338009222279 | 1 | Antacid | 1/1/2011 | 12/31/2099 |
| 2 | 41204000420118700 | 2 | Apple Tea | 1/1/2011 | 1/1/2012 |
| 3 | 3632320042915860 | 3 | Vitamin Water | 1/1/2011 | 1/1/2012 |
| 4 | 38465200581121200 | 4 | Mineral Water | 1/1/2011 | 12/31/2099 |
| 1008 | 41204000420118700 | 1008 | Apple Tea | 1/2/2012 | 12/31/2099 |
| 1009 | 3632320042915860 | 1009 | Vitamin Water | 1/2/2012 | 12/31/2099 |

**Item Table**

| ITEM_KEY | ITEM_NBR | SBC_KEY | ITEM_NAME | EFF_FROM_DT | EFF_TO_DT |
|---|---|---|---|---|---|
| 1 | 100309620 | 10 | Antacid | 1/1/2011 | 12/31/2099 |
| 2 | 100309700 | 1 | Apple Tea | 1/1/2011 | 1/1/2012 |
| 3 | 100309970 | 1 | Vitamin Water | 1/1/2011 | 1/1/2012 |
| 4 | 100309110 | 1 | Mineral Water | 1/1/2011 | 12/31/2099 |
| 1008 | 100309700 | 10 | Apple Tea | 1/2/2012 | 12/31/2099 |
| 1009 | 100309970 | 10 | Vitamin Water | 1/2/2012 | 12/31/2099 |

**Item Subclass Table**

| SBC_KEY | SBC_CD | ITEM_CLAS_KEY | SBC_NAME | EFF_FROM_DT | EFF_TO_DT |
|---|---|---|---|---|---|
| 1 | BVGS | 1 | BEVERAGES | 1/1/2011 | 12/31/2099 |
| 10 | HLTH | 10 | HEALTH | 1/1/2011 | 12/31/2099 |

**Retail Sales Table**

| DAY_KEY | SKU_ITEM_KEY | SLS_AMT |
|---|---|---|
| 20111230 | 1 | 100 |
| 20111230 | 2 | 100 |
| 20111230 | 3 | 100 |
| 20111230 | 4 | 100 |
| 20111231 | 1 | 100 |
| 20111231 | 2 | 100 |
| 20111231 | 3 | 100 |
| 20111231 | 4 | 100 |

| | | |
|---|---|---|
| 20120102 | 1 | 100 |
| 20120102 | 1008 | 100 |
| 20120102 | 1009 | 100 |
| 20120102 | 4 | 100 |

Assuming the data used for the examples, to show the sales data split by Item Subclass, the SQL statement in Example 5–5 that joins the sales fact table and the item dimensions on the SKU_ITEM_KEY surrogate key. Also snowflake tables for Item dimension DWR_ITEM is joined with DWR_SKU_ITEM by ITEM_KEY and DWR_ITEM_SBC is joined with DWR_ITEM by SBC_KEY.

***Example 5–5 As Is Query for Sales Split by Item Subclass***

```
SELECT SKU.sku_item_name
,SBC.sbc_name
,SUM(SLS_AMT) sls_amt
FROM dwb_rtl_sl_rtrn_li SLS, dwr_sku_item SKU, dwr_item ITEM, dwr_item_sbc SBC
WHERE SLS.sku_item_key = SKU.sku_item_key
AND SKU.item_key = ITEM.item_key
AND ITEM.sbc_key = SBC.sbc_key
GROUP BY SBC.sbc_name, SKU.sku_item_name
ORDER BY SBC.sbc_name, SKU.sku_item_name
;
```

The result of the query is shown in the table Table 5–1.

***Table 5–1 As Is Query Result for Sales Split by Item Subclass***

| SKU_ITEM_NAME | SBC_NAME | SLS_AMT |
|---|---|---|
| Apple Tea | BEVERAGES | 200 |
| Mineral Water | BEVERAGES | 300 |
| Vitamin Water | BEVERAGES | 200 |
| Antacid | HEALTH | 300 |
| Apple Tea | HEALTH | 100 |
| Vitamin Water | HEALTH | 100 |

Assuming the data used for the examples, issue the SQL statement shown in Example 5–6 to show the sales data split by subclass using an analysis date of January 1, 2012.

***Example 5–6 As Was Query for Sales Split by Item Subclass***

```
SELECT SKU.sku_item_name
,SBC.sbc_name
,SUM(SLS_AMT) sls_amt
FROM dwb_rtl_sl_rtrn_li SLS, dwr_item_sbc SBC,
( SELECT ACT.item_key, PIT.item_nbr, PIT.sbc_key, PIT.item_name
FROM dwr_item ACT INNER JOIN
(SELECT item_key, item_nbr, sbc_key, item_name
FROM dwr_item
WHERE TO_DATE('20120101','YYYYMMDD') BETWEEN eff_from_dt AND eff_to_dt
)PIT
```

```
ON ACT.item_nbr = PIT.item_nbr
) ITEM,
( SELECT ACT.sku_item_key, PIT.sku_item_nbr, PIT.item_key, PIT.sku_item_name
FROM dwr_sku_item ACT INNER JOIN
(SELECT sku_item_key, sku_item_nbr, item_key, sku_item_name
FROM dwr_sku_item
WHERE TO_DATE('20120101','YYYYMMDD') BETWEEN eff_from_dt AND eff_to_dt
)PIT
ON ACT.sku_item_nbr = PIT.sku_item_nbr
) SKU
WHERE SLS.sku_item_key = SKU.sku_item_key
AND SKU.item_key = ITEM.item_key
AND ITEM.sbc_key = SBC.sbc_key
GROUP BY SBC.sbc_name, SKU.sku_item_name
ORDER BY SBC.sbc_name, SKU.sku_item_name
```

The results of this query are shown in Table 5–2. Since Apple Tea and Vitamin Water were under Beverages subclass, the sales amount for these two are accounted under the Beverages subclass.

***Table 5–2 As Was Query Result for Sales Split by Item Subclass (Beverages Subclass)***

| SKU_ITEM_NAME | SBC_NAME | SLS_AMT |
|---|---|---|
| Apple Tea | BEVERAGES | 300 |
| Mineral Water | BEVERAGES | 300 |
| Vitamin Water | BEVERAGES | 300 |
| Antacid | HEALTH | 300 |

Assume instead that you issued the exact same query except that the to_date phrase you specify was January 3, 2012 rather than January 1, 2012. All sales for these two products, Apple Tea and Vitamin Water, would be accounted under the Health subclass.

```
SELECT SKU.sku_item_name
,SBC.sbc_name
,SUM(SLS_AMT) sls_amt
FROM dwb_rtl_sl_rtrn_li SLS, dwr_item_sbc SBC,
( SELECT ACT.item_key, PIT.item_nbr, PIT.sbc_key, PIT.item_name
FROM dwr_item ACT INNER JOIN
(SELECT item_key, item_nbr, sbc_key, item_name
FROM dwr_item
WHERE TO_DATE('20120103','YYYYMMDD') BETWEEN eff_from_dt AND eff_to_dt
)PIT
ON ACT.item_nbr = PIT.item_nbr
) ITEM,
( SELECT ACT.sku_item_key, PIT.sku_item_nbr, PIT.item_key, PIT.sku_item_name
FROM dwr_sku_item ACT INNER JOIN
(SELECT sku_item_key, sku_item_nbr, item_key, sku_item_name
FROM dwr_sku_item
WHERE TO_DATE('20120103','YYYYMMDD') BETWEEN eff_from_dt AND eff_to_dt
)PIT
ON ACT.sku_item_nbr = PIT.sku_item_nbr
) SKU
WHERE SLS.sku_item_key = SKU.sku_item_key
AND SKU.item_key = ITEM.item_key
AND ITEM.sbc_key = SBC.sbc_key
GROUP BY SBC.sbc_name, SKU.sku_item_name
ORDER BY SBC.sbc_name, SKU.sku_item_name
```

;

The result of this query is shown in Table 5–3.

***Table 5–3 As Was Query Result for Sales Split by Item Subclas (Health Subclass)***

| SKU_ITEM_NAME | SBC_NAME | SLS_AMT |
|---|---|---|
| Mineral Water | BEVERAGES | 300 |
| Antacid | HEALTH | 300 |
| Apple Tea | HEALTH | 300 |
| Vitamin Water | HEALTH | 300 |

# Tutorial: Creating a New Oracle Retail Data Model Dashboard

This tutorial explains how to create a dashboard based on dashboards in the Oracle Retail Data Model webcat included with the sample Oracle Business Intelligence Suite Enterprise Edition reports delivered with Oracle Retail Data Model.

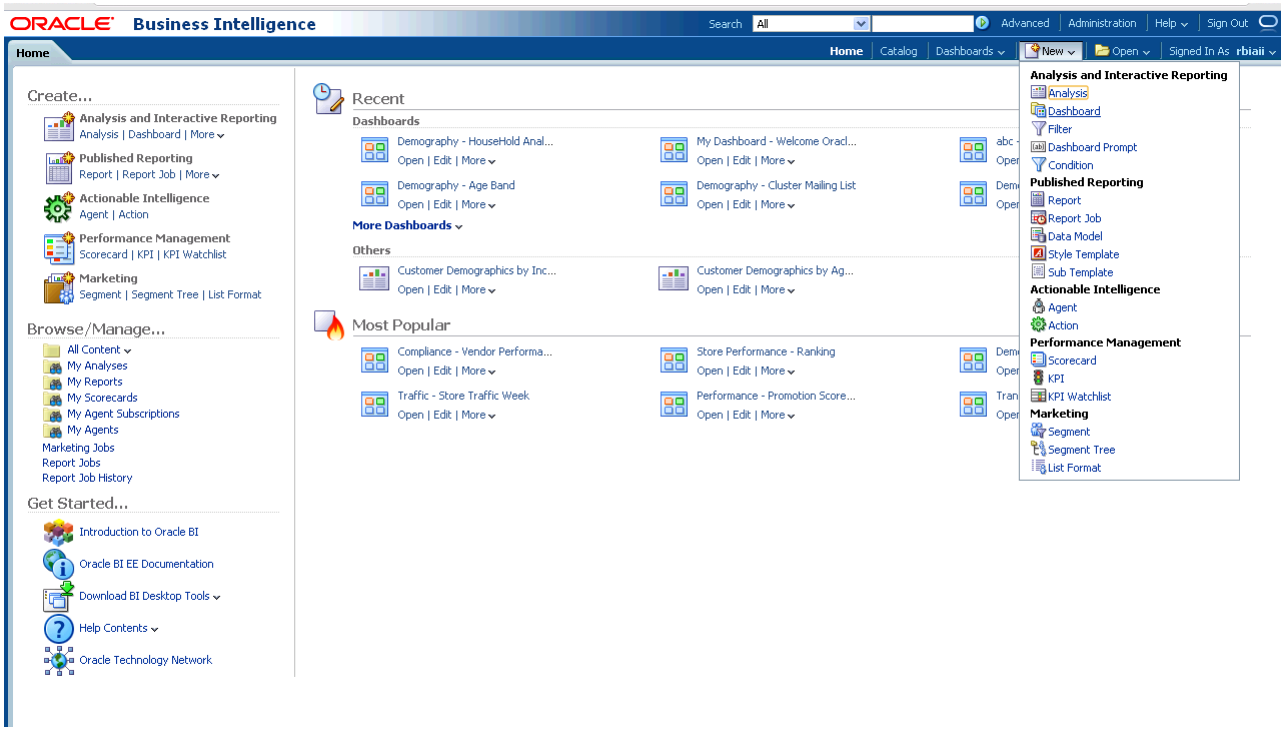
> **See:** *Oracle Retail Data Model Installation Guide* for more information on installing the sample reports and deploying the Oracle Retail Data Model RPD and webcat on the Business Intelligence Suite Enterprise Edition instance.

In this example, assume that you are creating a dashboard named "Category Manager", and you put both "Customer Sales Value by Year " and "Bottom N Customer" into this new dashboard.

To create a dashboard, take the following steps:

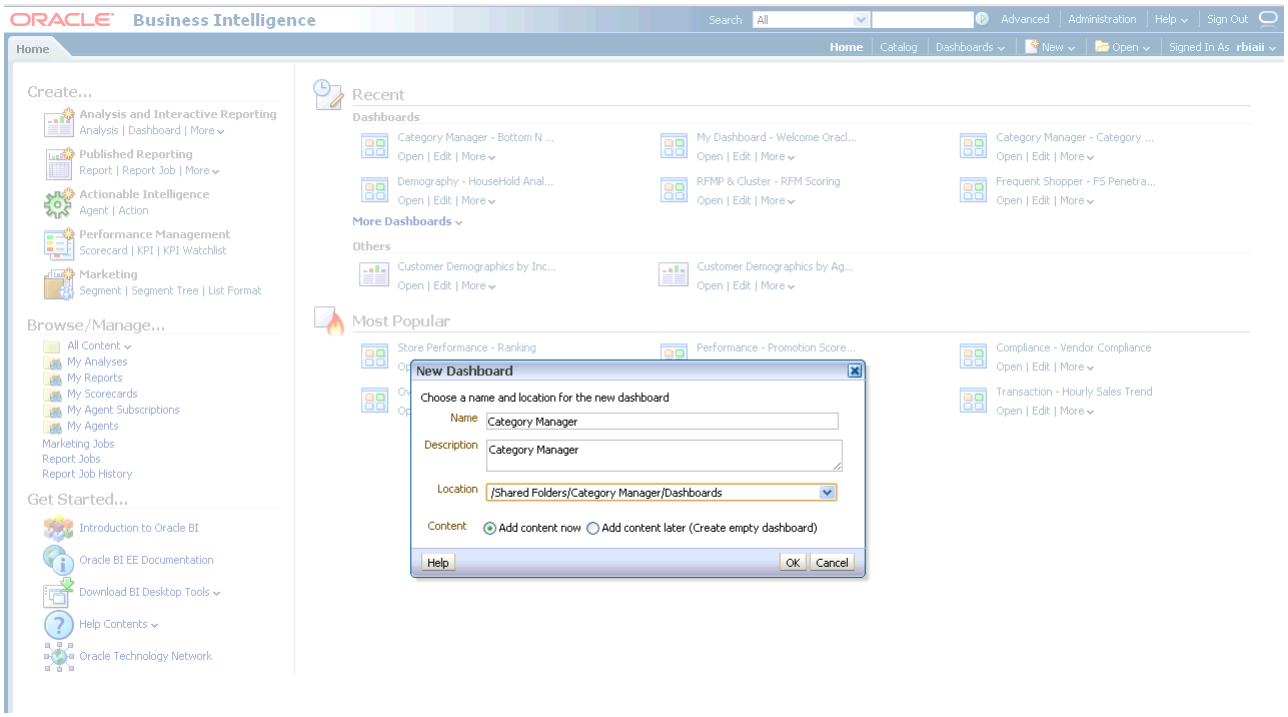
1. In a browser, open the login page at `http://`*`servername`*`:9704/analytics` where *`servername`* is the server on which the webcat is installed.
2. Login with username of `ordm`, and provide the password.
3. Select **New**, and then select **Dashboard** to create an Oracle Business Intelligence Suite Enterprise Edition dashboard. For example, see Figure 5–1.

***Figure 5–1 New Dashboard Start: BIEE Home***



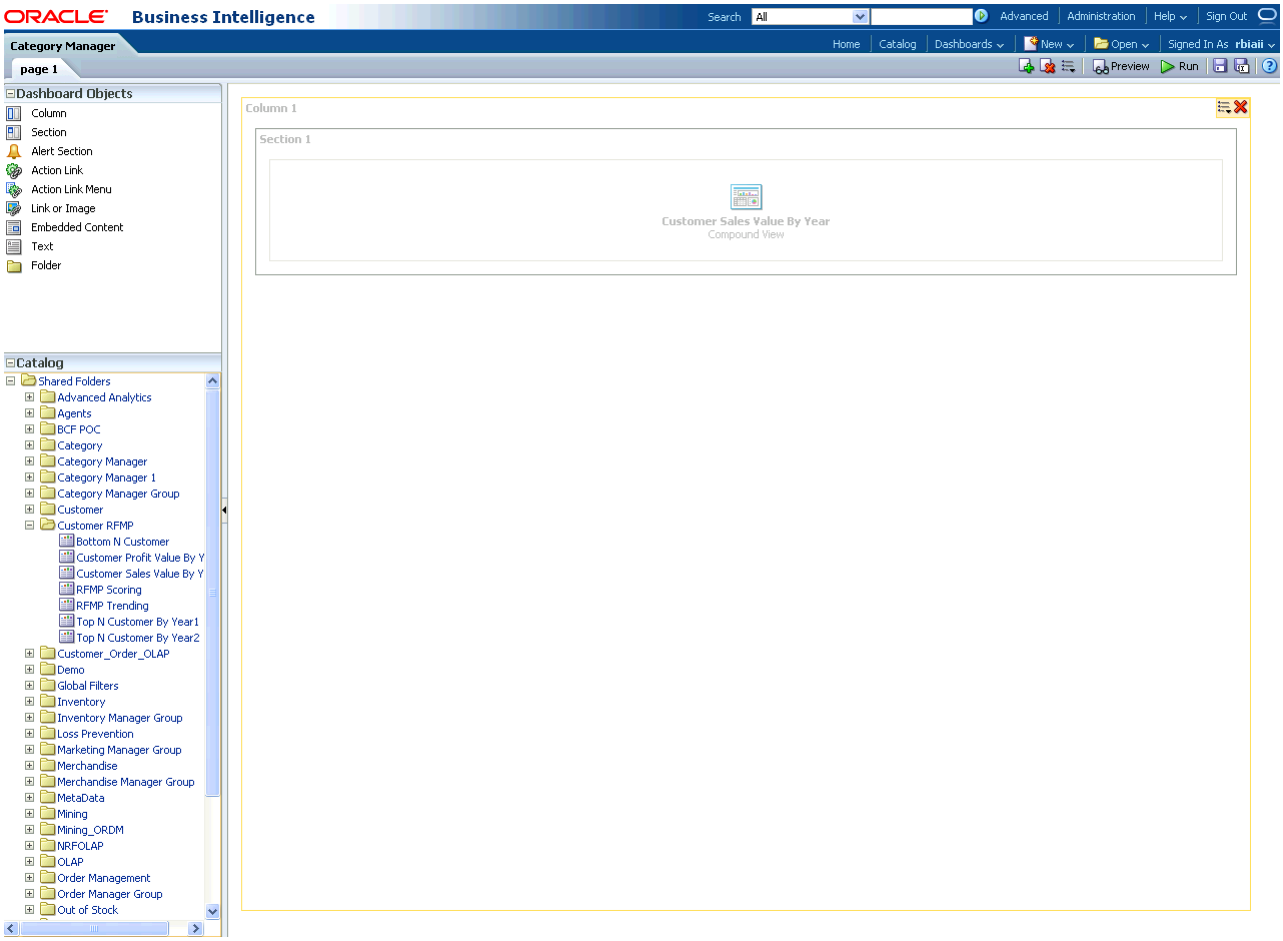

4. Enter a name and description, as shown in Figure 5–2. Select the save location to save the dashboard to the Dashboards folder, then click **OK**.

***Figure 5–2 New Dashboard: Enter Name***



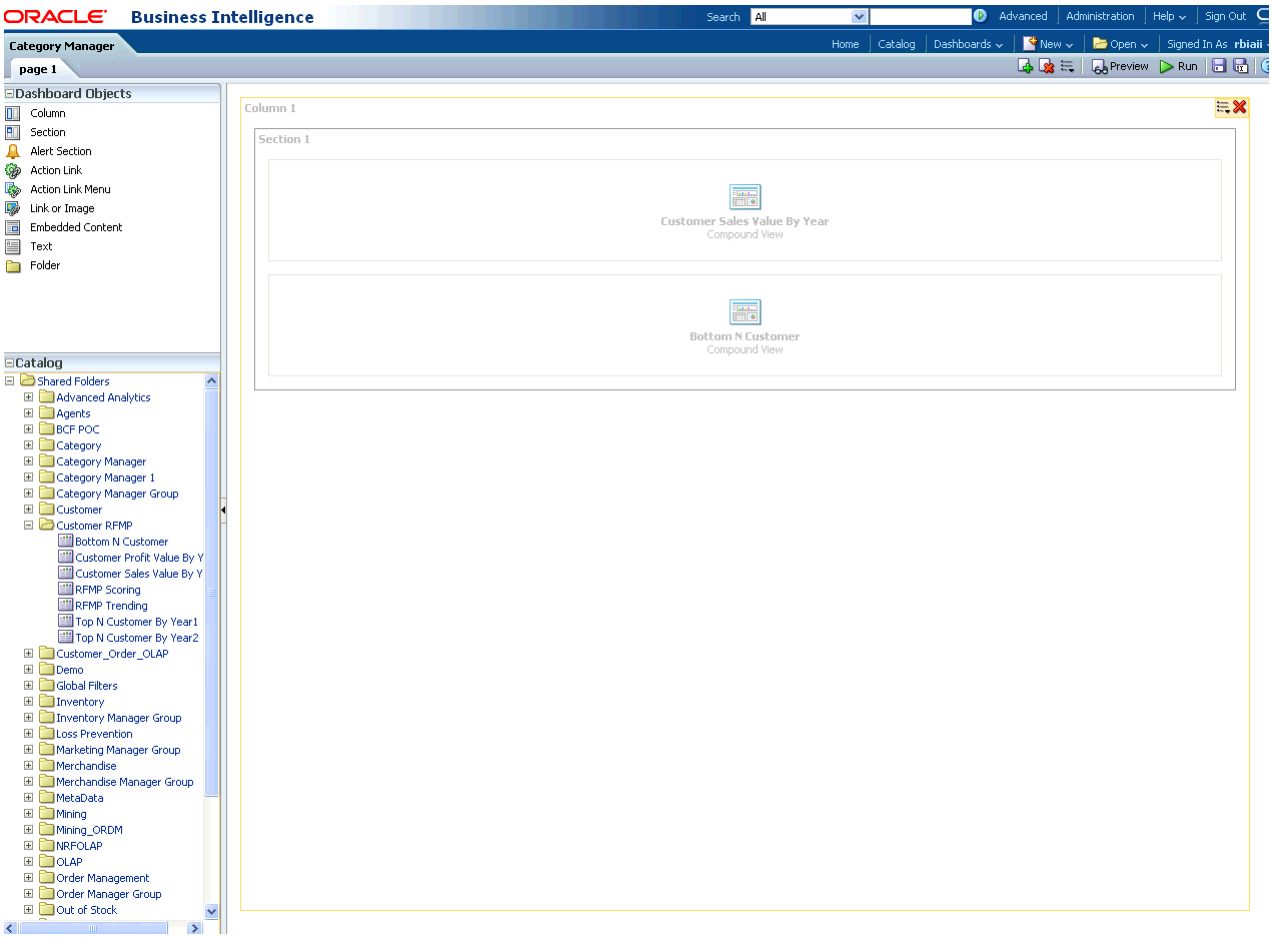
5. In the **Catalog** view, expand the Customer **RFMP** folder, as shown in Figure 5–3. In the navigator, you can see the **Customer Sales Value by Year** Report; drag it from catalog view into the right panel.

***Figure 5–3 New Dashboard: Catalog View***



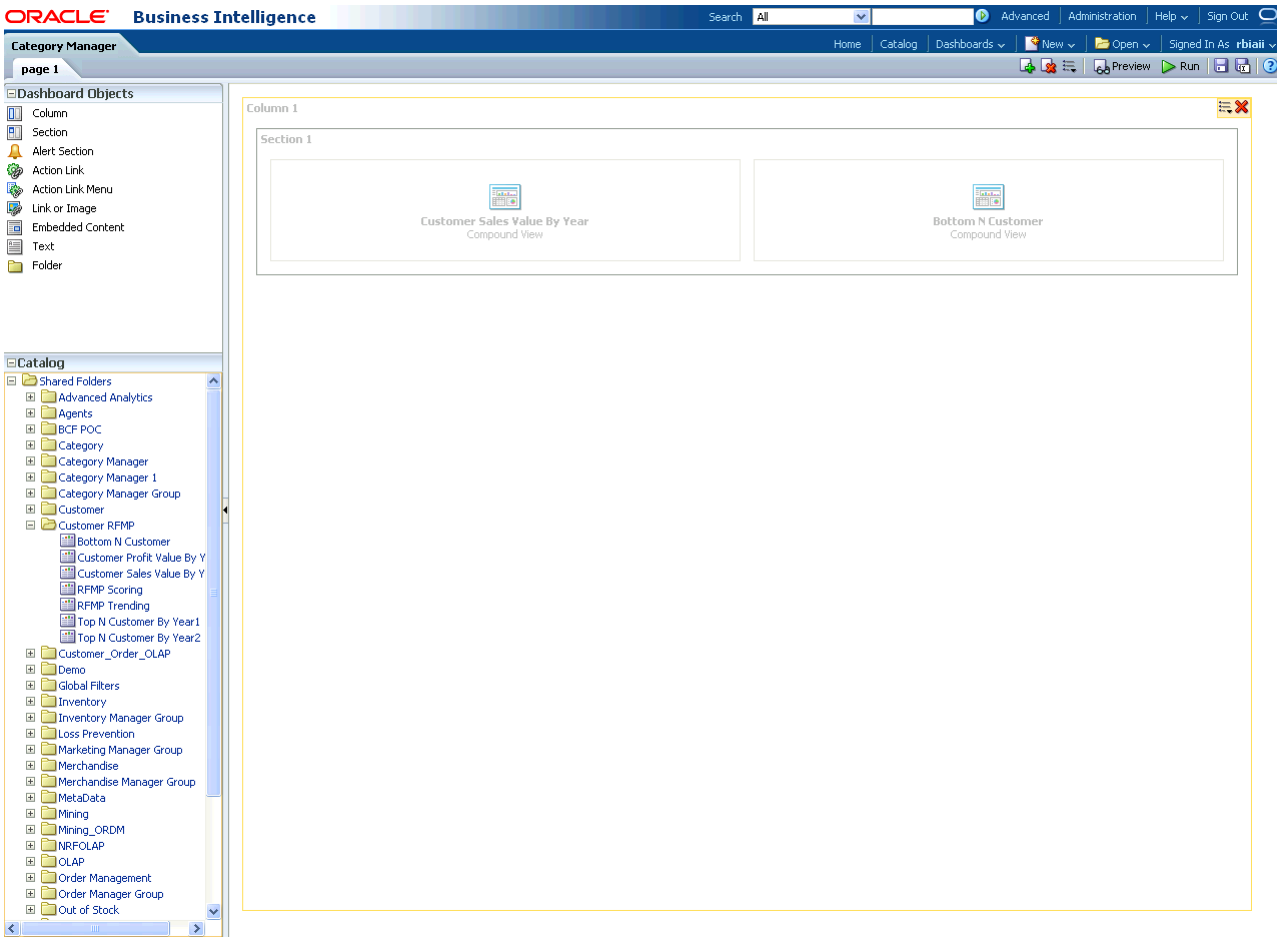

6. From the navigator, in the same folder, drag the **Bottom N Customer** report into the right pane, as shown in Figure 5–4.

***Figure 5–4 New Dashboard: Catalog View with New Reports Vertical***



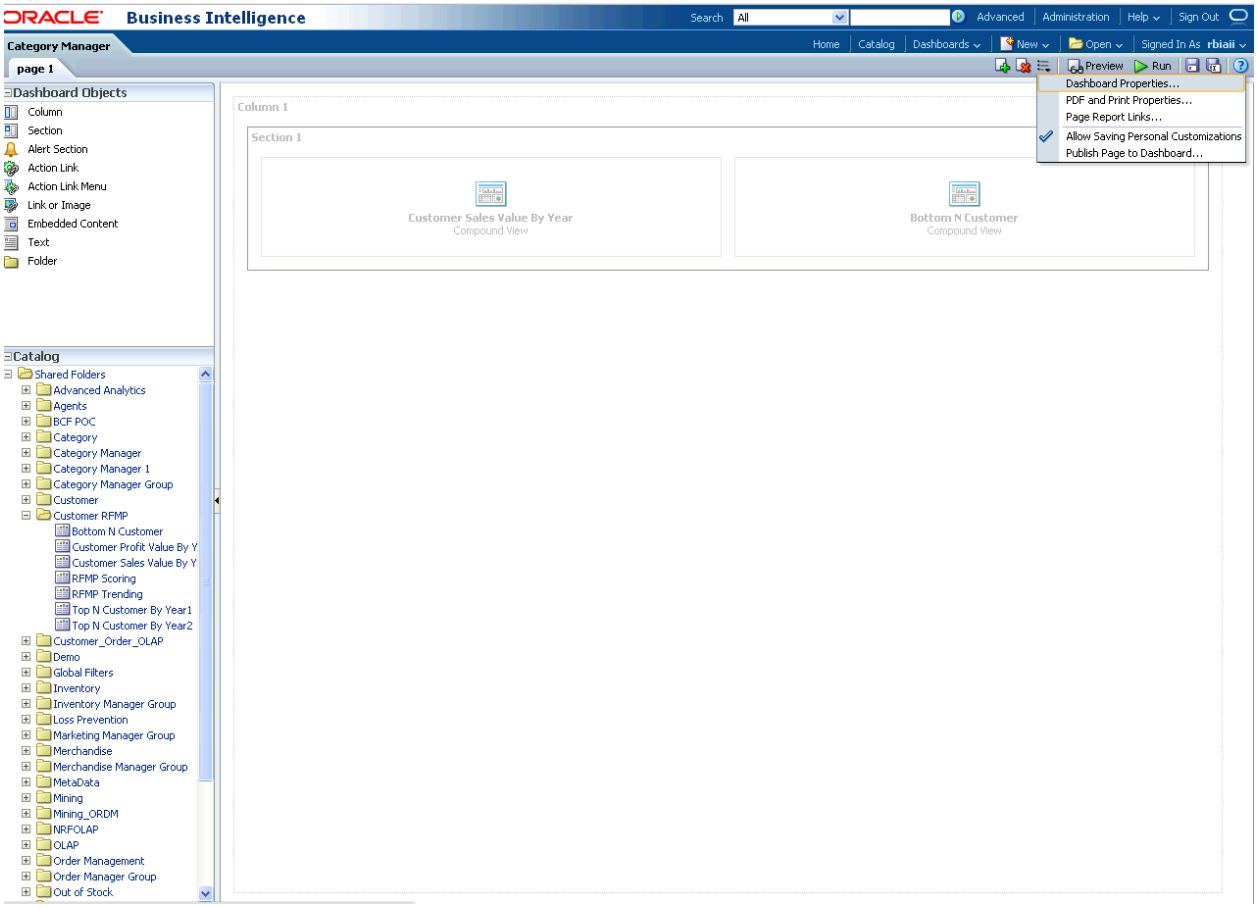

7. You can change the layout of this section to organize the two reports, either **horizontal** or **vertical**, as shown in Figure 5–4, and Figure 5–5.

***Figure 5–5 New Dashboard: Catalog View with New Reports Horizontal***



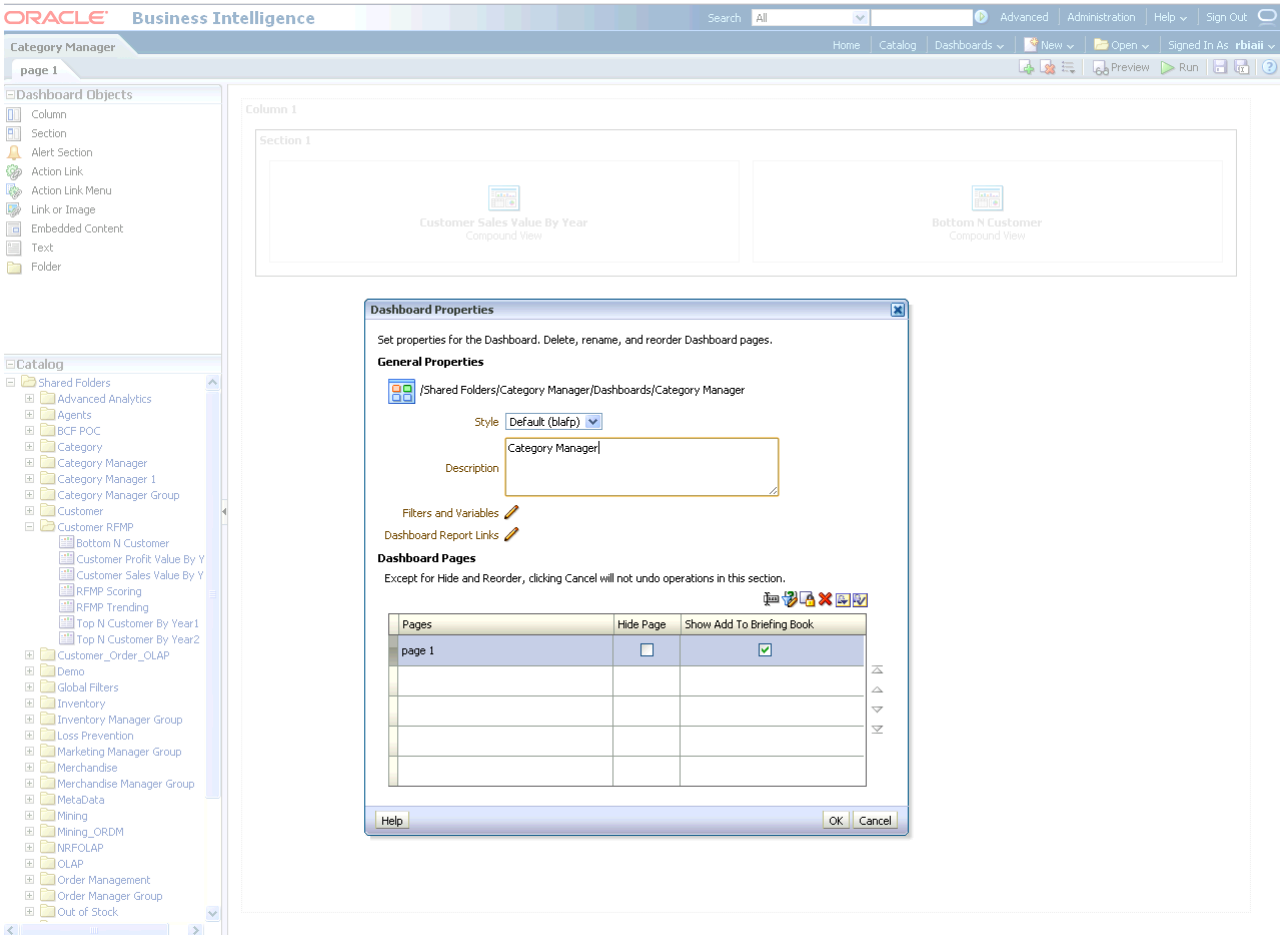

8. The page has the default name, "Page1". To change the name, do the following:
   a. Select the **Dashboard**, as shown in Figure 5–6.

***Figure 5–6 New Dashboard: Select Name for Page***



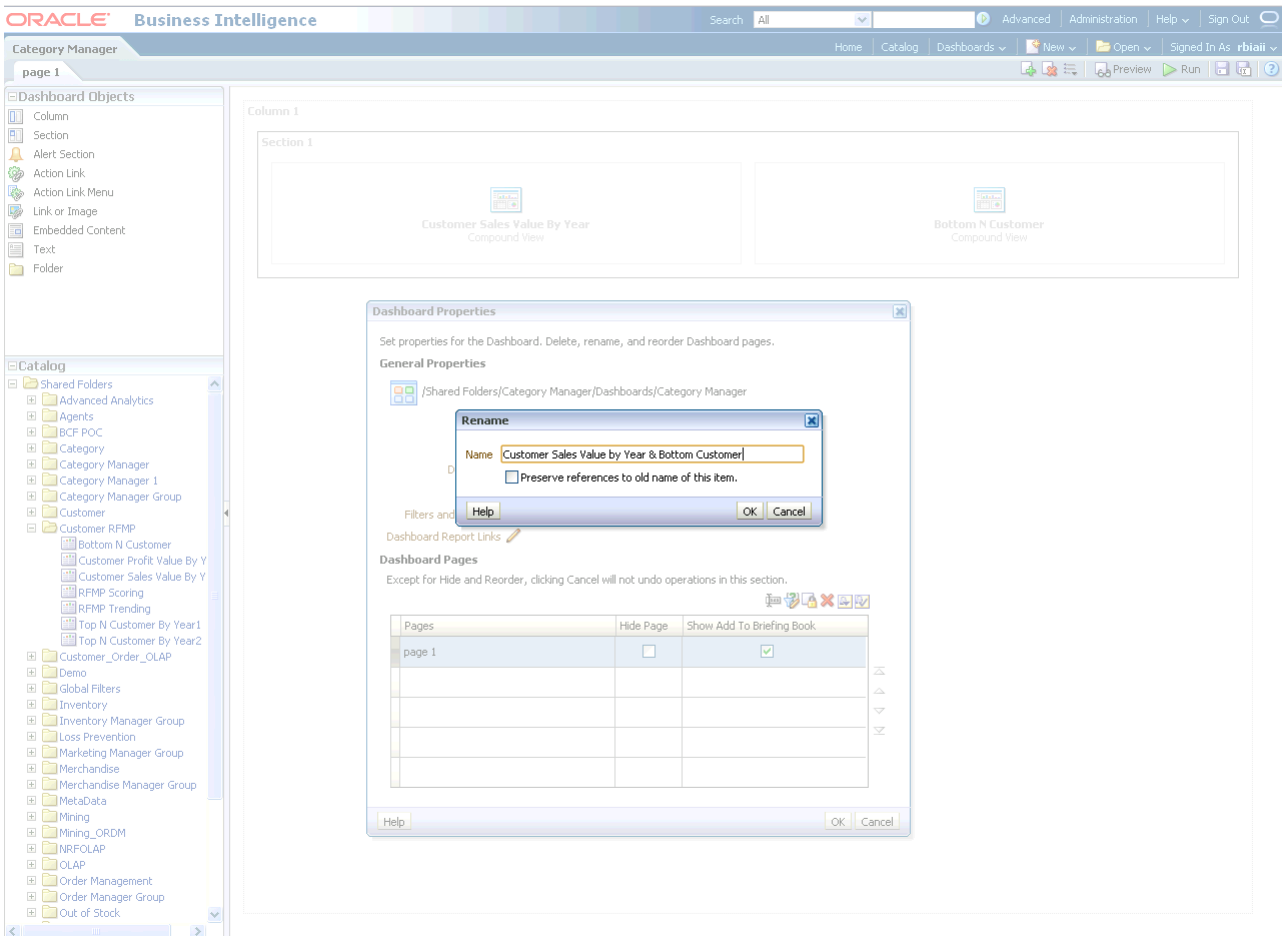

**b.** In Dashboard Properties window, click **Change Name**, as shown in Figure 5–7.

***Figure 5–7 New Dashboard: Enter New Page Name***



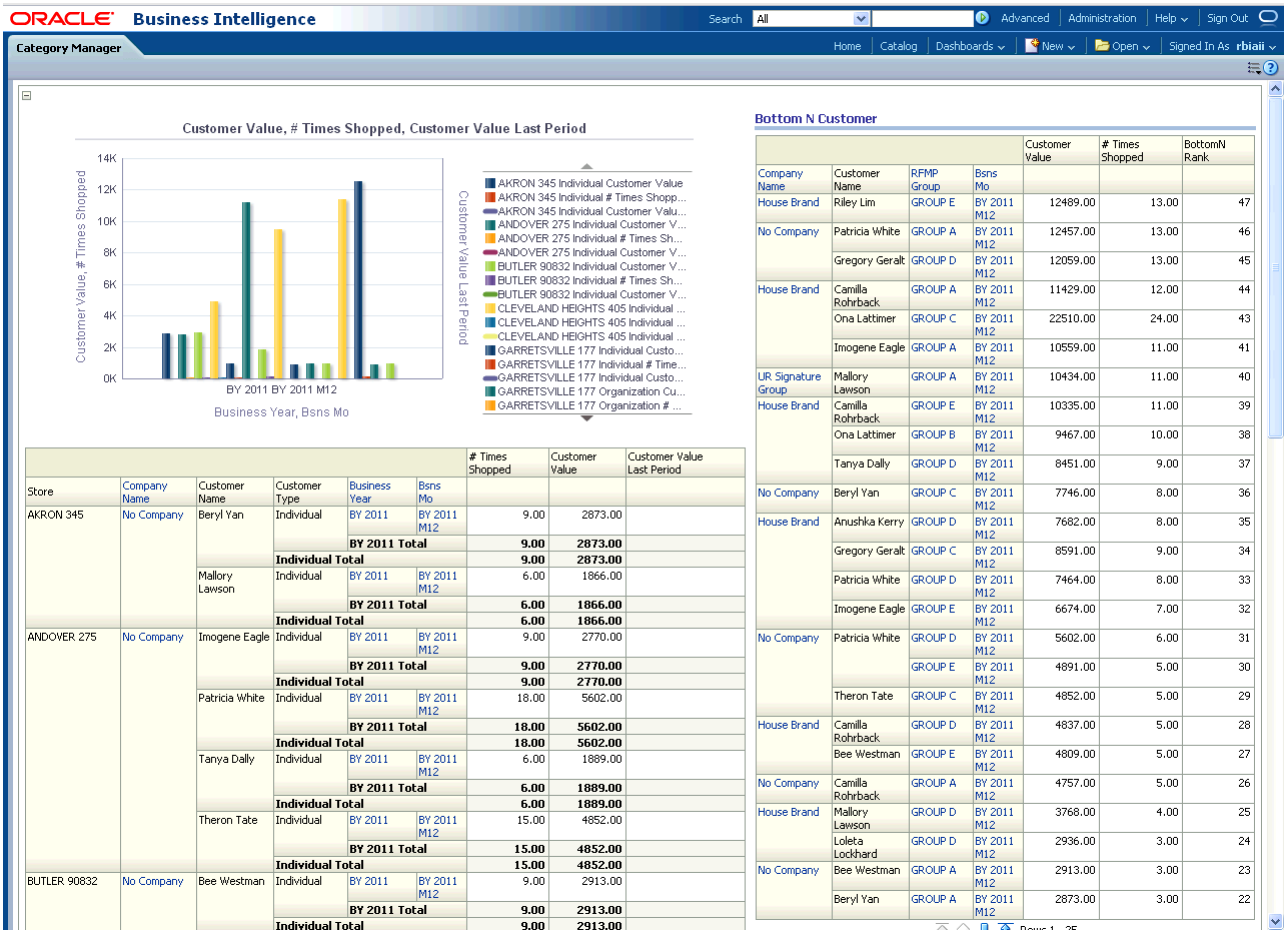

**c.** Change the name to "**Customer Sales Value by Year & Bottom Customer**", as shown in Figure 5–8. Then click **OK.**

***Figure 5–8 New Dashboard: Rename Page Dialog***




9. Click **Save** on the top of the dashboard. The new dashboard is shown in Figure 5–9.

***Figure 5–9 New Dashboard: Display with Two New Reports***




**Oracle by Example:** For more information on creating dashboards see the "Creating Analyses and Dashboards 11*g*" OBE tutorial.

To access the tutorial, open the Oracle Learning Library in your browser by following the instructions in "Oracle Technology Network" on page xii; and, then, search for the tutorial by name.

# Tutorial: Creating a New Oracle Retail Data Model Report

This tutorial explains how to create a report based on the Oracle Retail Data Model webcat included with the sample Oracle Business Intelligence Suite Enterprise Edition reports delivered with Oracle Retail Data Model.

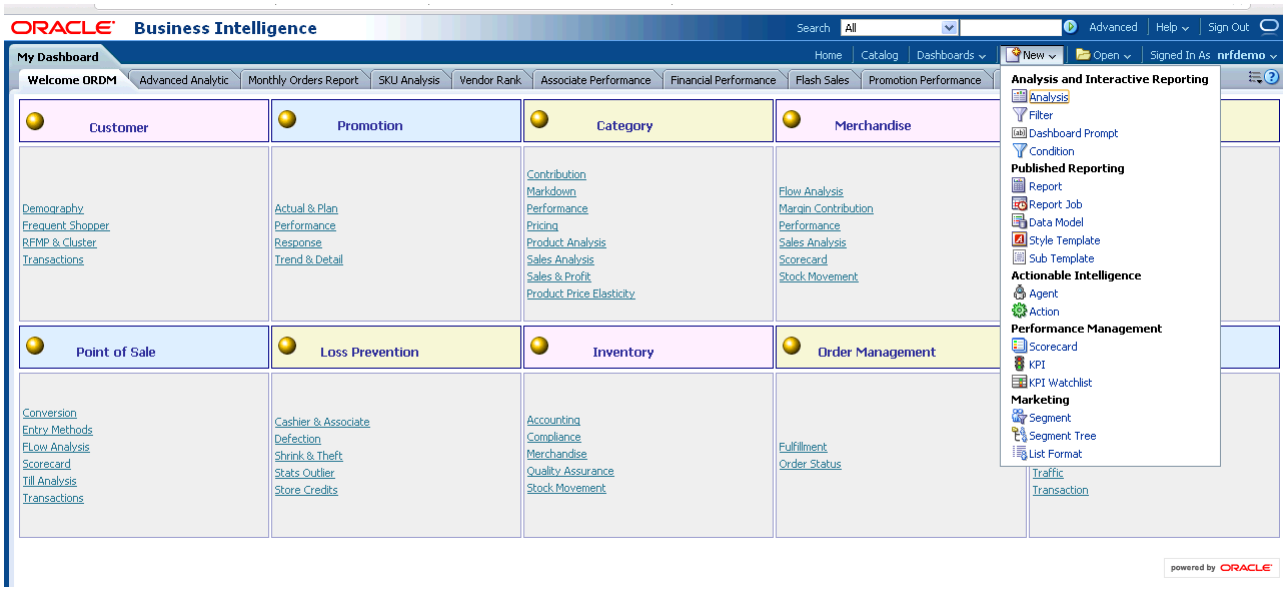
**See:** *Oracle Retail Data Model Installation Guide* for more information on installing the sample reports and deploying the Oracle Retail Data Model RPD and webcat on the Business Intelligence Suite Enterprise Edition instance.

In this example, assume that you are creating a report named "Vendor Sales by Channel" to show sales values by time, channel, and vendor.

To create a report, take the following steps:

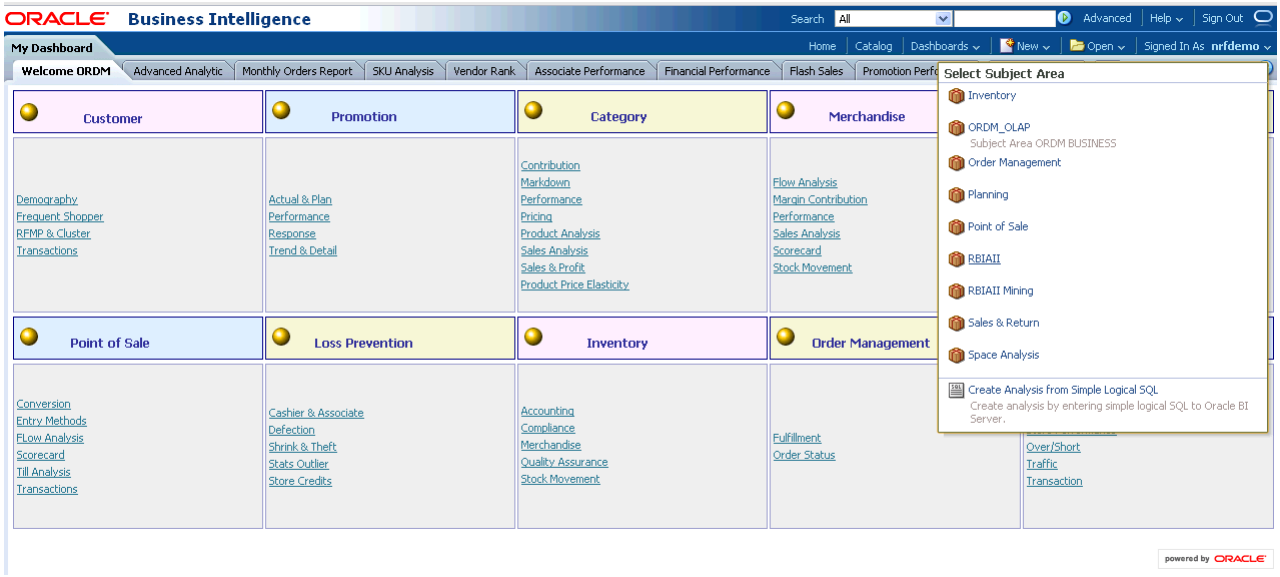
1. In the browser, open the login page at `http://servername:9704/analytics` where *servername* is the server on which the webcat is installed.
2. Login with username of `ordm`, and provide the password.
3. Select **New**, and then select **Analysis** to create an Oracle Business Intelligence Suite Enterprise Edition report, as shown in Figure 5–10.

***Figure 5–10 Analysis Report: Welcome Page with New Menu***



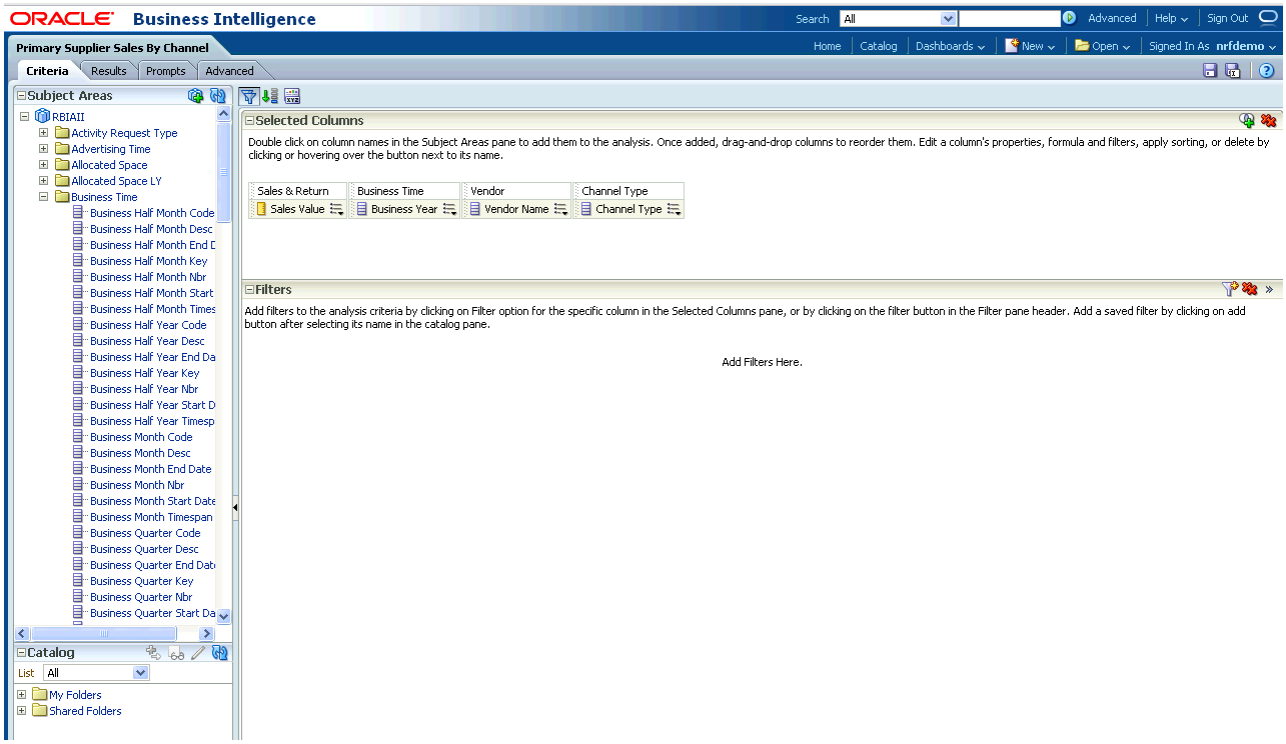

4. Select **Subject Area**, and then select **RBIAII** to create a relational report, as shown in Figure 5–11.

***Figure 5–11 Analysis Report: Welcome Page with Select Subject Area Menu***



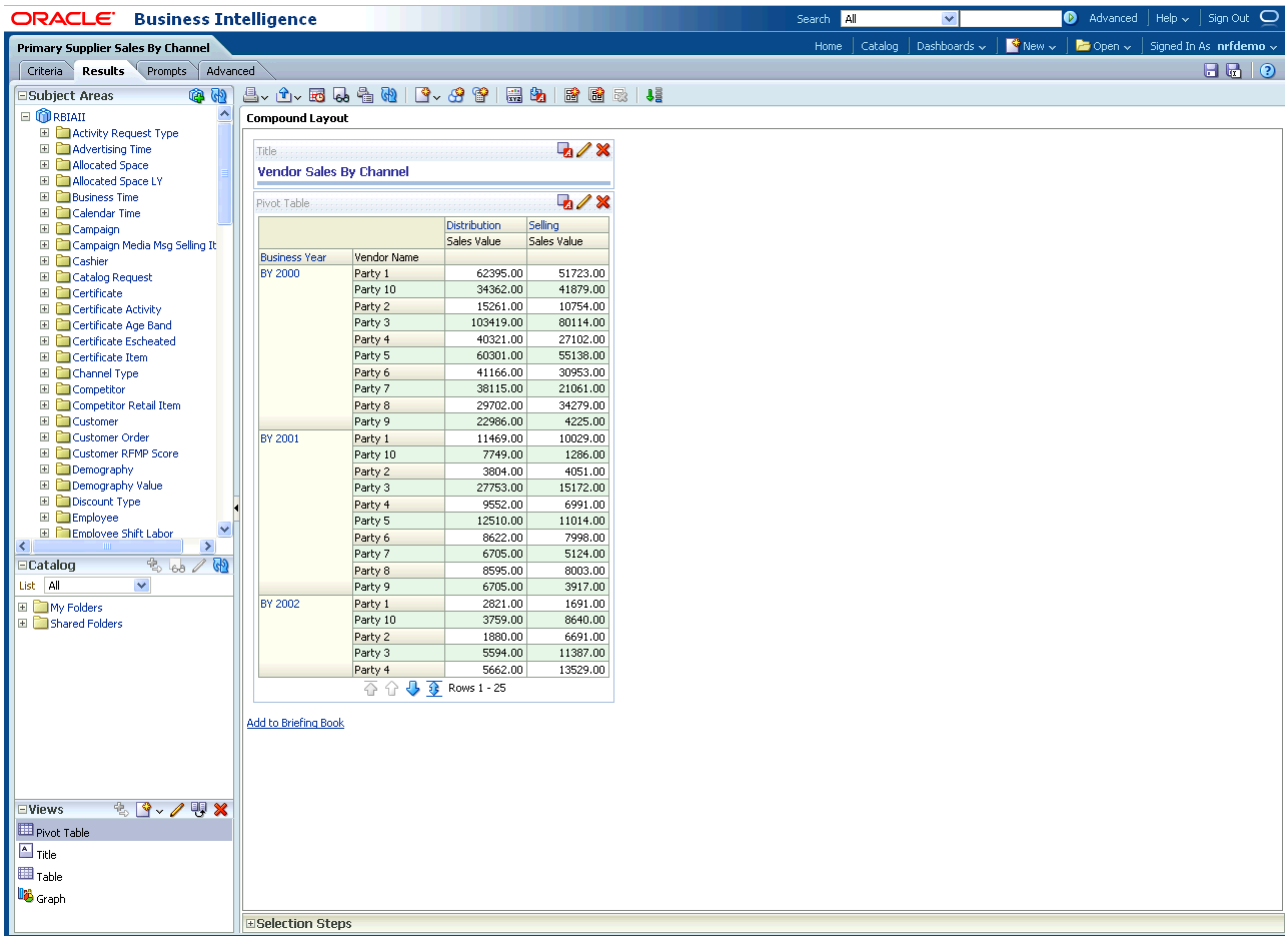

5. Drag and drop the dimension and fact columns into the **Selected Columns** panel, as shown in Figure 5–12.

***Figure 5–12 Analysis Report: Welcome Page with Selected Columns***



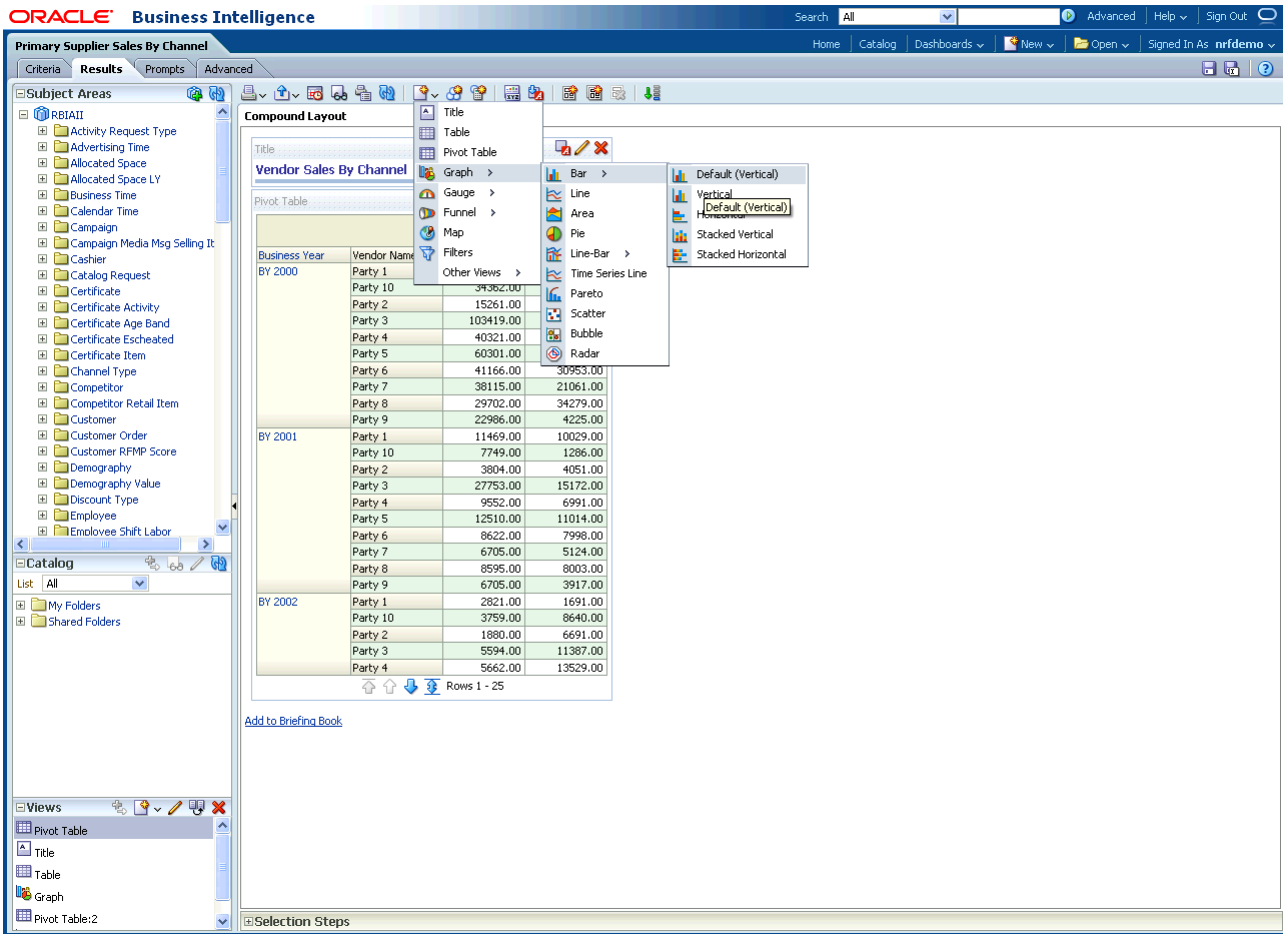

6. Select the **Results** tab to view the report, as shown in Figure 5–13.

***Figure 5–13 Analysis Report: Results View of Report***



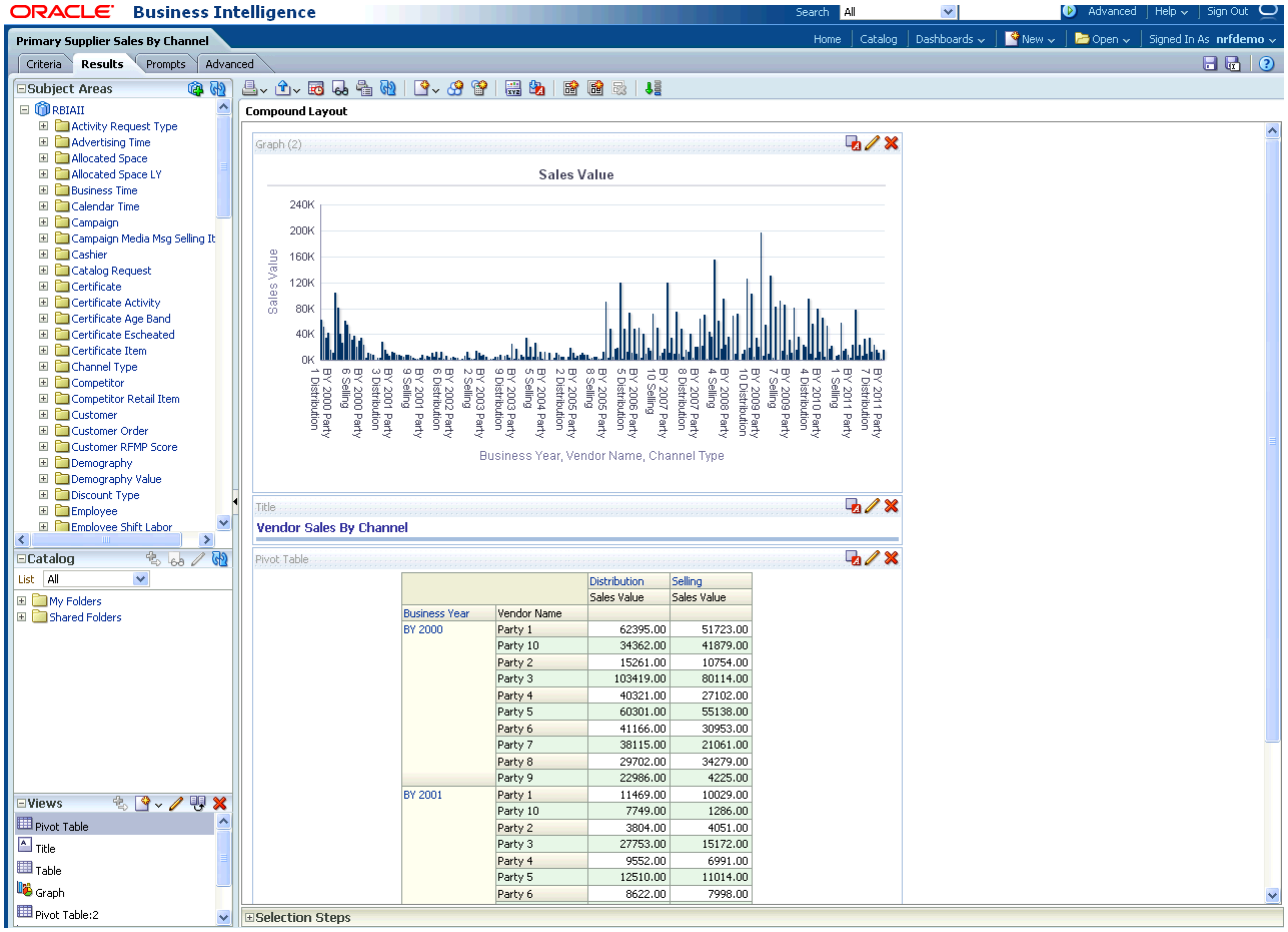

7. Select **New View** to add a chart to the report, as shown in Figure 5–14.

***Figure 5–14 New Report: Create New View***




8. Select **Save** to save this report. The report including the new values is shown in Figure 5–15.

***Figure 5–15 New Report: Final View of New Reports***




| Business Year | Vendor Name | Distribution Sales Value | Selling Sales Value |
|---|---|---|---|
| BY 2000 | Party 1 | 62395.00 | 51723.00 |
| | Party 10 | 34362.00 | 41879.00 |
| | Party 2 | 15261.00 | 10754.00 |
| | Party 3 | 103419.00 | 80114.00 |
| | Party 4 | 40321.00 | 27102.00 |
| | Party 5 | 60301.00 | 55138.00 |
| | Party 6 | 41166.00 | 30953.00 |
| | Party 7 | 38115.00 | 21061.00 |
| | Party 8 | 29702.00 | 34279.00 |
| | Party 9 | 22986.00 | 4225.00 |
| BY 2001 | Party 1 | 11469.00 | 10029.00 |
| | Party 10 | 7749.00 | 1286.00 |
| | Party 2 | 3804.00 | 4051.00 |
| | Party 3 | 27753.00 | 15172.00 |
| | Party 4 | 9552.00 | 6991.00 |
| | Party 5 | 12510.00 | 11014.00 |
| | Party 6 | 8622.00 | 7998.00 |

**Oracle by Example:** For more information on creating a report, see the "Creating Analyses and Dashboards 11*g*" OBE tutorial.

To access the tutorial, open the Oracle Learning Library in your browser by following the instructions in "Oracle Technology Network" on page xii; and, then, search for the tutorial by name.

# 6

# Metadata Collection and Reports

You use the Oracle Retail Data Model metadata browser generation packages to generate and update the Oracle Retail Data Model metadata browser.

There are four main tables and other staging tables and views in the metadata generation package. The tables are: MD_ENTY, MD_PRG, MD_KPI, and MD_REF_ENTY_KPI; these are the tables that support metadata browser reports.

This chapter includes the following sections:


- Metadata Collection and Population
- Metadata Reports and Dashboard


## Metadata Collection and Population

Use the following steps to generate Oracle Retail Data Model Logical Data Model metadata, and use the following sequence to ensure proper metadata collection and loading.

1. **Collect LDM Metadata**

   In step you extract the Logical Data Model repository metadata from Oracle SQL Developer Data Modeler (OSDM) into a database schema. Use manual steps to generate Logical Data Model repository tables in the database with Oracle SQL Developer Data Modeler.

   - Start Oracle SQL Developer Modeler
   - Open LDM ->  File -> Export -> to Report schema.

2. **Collect Sample Dashboard Metadata**

   In this step you extract BIEE dashboard metadata from webcat to csv file.

   Use OBIEE catalog manager to open SQL Developer sample report webcat.

   Tools -> create Report -> Select type to report on -> select dashboard

   Select columns one by one as per the md_dashboard.ldr specified in the meta_data folder, then save as a csv format file, md_dashboard.csv.

   Put this file in the meta_data folder.

   Column Sequence:

   a. Name
   b. Description
   c. Path

d. Folder

e. Analysis Path

f. Analysis Name

g. Analysis Description

h. Dashboard Page Description

i. Dashboard Page Name

j. Dashboard Page Path

k. Owner

3. **Collect Sample Report Metadata**: Extract BIEE report metadata from webcat to csv file.

   Use OBIEE catalog manager to open Oracle Retail Data Model sample report webcat.

   - Tools -> create Report -> Select type to report on -> select Analysis -> select columns one by one as per the md_dashboard.ldr specified in the meta_data folder.
   - Save the file as csv format, md_dashboard.csv. Put the file under meta_data folder

   Column Sequence:

   a. NAME

   b. DESCRIPTION

   c. TABLE_NAME

   d. COLUMN_NAME

   e. FOLDER

   f. PATH

   g. SUBJECT_AREA

   h. FORMULA

4. **Collect Sample RPD Metadata**

   Extract BIEE RPD metadata from RPD to csv file.

   Use Administrator Tool to open Oracle Retail Data Model sample report RPD:

   - Tools -> Utilities -> Repository Documentation -> Execute -> select location -> set xls file name as md_rpd
   - Save as csv format md_rpd.csv and put under meta_data folder.

5. **Load Naming Convention Information**: Load Oracle Retail Data Model PDM naming convention information from csv into a staging table.Use sqlloader to load data from name_conversion.csv into MD_NAME_CONVERSION table. Sqlloader format file: Name_conversion.ldr

```
Name_conversion.ldr:
OPTIONS (SKIP=1)
LOAD DATA
INFILE      'name_conversion.csv'
BADFILE     'name_conversion.csv.bad'
DISCARDFILE 'name_conversion.csv.dsc'
```

```
truncate
INTO TABLE MD_NAME_CONVERSION
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
TRAILING NULLCOLS
(
ABBREVIATION    ,
FULL_NAME
)
```

**6. Load Sample Dashboard Metadata**

Load sample dashboard metadata from csv into a staging table.

Use sqlloader to load data from md_dashboard.csv into MD_DASHBOARD table. Sqlloader format file: md_dashboard.ldr.

Md_dashboard.ldr:

```
OPTIONS (SKIP=1)
LOAD DATA
INFILE      'md_dashboard.csv'
BADFILE     'md_dashboard.csv.bad'
DISCARDFILE 'md_dashboard.csv.dsc'
truncate
INTO TABLE MD_DASHBOARD
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
TRAILING NULLCOLS
(
NAME char(2000),
DESCRIPTION char(2000),
PATH char(2000),
FOLDER char(2000),
ANALYSIS_PATH char(2000),
ANALYSIS_NAME char(2000),
ANALYSIS_DESCRIPTION char(2000),
DASHBOARD_PAGE_DESCRIPTION char(2000),
DASHBOARD_PAGE_NAME char(2000),
DASHBOARD_PAGE_PATH char(2000),
OWNER char(2000)
)
```

**7. Load Sample Report Metadata**

Load sample report metadata from csv into a staging table.

Use sqlloader to load data from md_report.csv into MD_REPORT table. Sqlloader format file: md_report.ldr.

Md_dashboard.ldr:

```
OPTIONS (SKIP=1)
LOAD DATA
INFILE      'md_dashboard.csv'
BADFILE     'md_dashboard.csv.bad'
DISCARDFILE 'md_dashboard.csv.dsc'
truncate
INTO TABLE MD_DASHBOARD
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
TRAILING NULLCOLS
(
NAME char(2000),
DESCRIPTION char(2000),
```

```
PATH char(2000),
FOLDER char(2000),
ANALYSIS_PATH char(2000),
ANALYSIS_NAME char(2000),
ANALYSIS_DESCRIPTION char(2000),
DASHBOARD_PAGE_DESCRIPTION char(2000),
DASHBOARD_PAGE_NAME char(2000),
DASHBOARD_PAGE_PATH char(2000),
OWNER char(2000)
)
```

8. **Load Sample RPD Metadata**

   Load sample RPD metadata from csv into a staging table.

   Use sqlloader to load data from md_rpd.csv into MD_RPD table. Sqlloader format file: md_rpd.ldr.

   Md_rpd.ldr:

```
OPTIONS (SKIP=0)
LOAD DATA
INFILE      'md_rpd.csv'
BADFILE     'md_rpd.csv.bad'
DISCARDFILE 'md_rpd.csv.dsc'
truncate
INTO TABLE MD_RPD
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
TRAILING NULLCOLS
(
 SUBJECT_AREA
,PRESENTATION_TABLE
,PRESENTATION_COLUMN char(500)
,DESC_PRESENTATION_COLUMN
,BUSINESS_MODEL
,DERIVED_LOGICAL_TABLE
,DERIVED_LOGICAL_COLUMN
,DESC_DERIVED_LOGICAL_COLUMN
,EXPRESSION char(1000)
,LOGICAL_TABLE
,LOGICAL_COLUMN
,DESC_LOGICAL_COLUMN
,LOGICAL_TABLE_SOURCE
,EXPRESSION_1 char(1000)
,INITIALIZATION_BLOCK
,VARIABLE
,DATABASE
,PHYSICAL_CATALOG
,PHYSICAL_SCHEMA
,PHYSICAL_TABLE
,ALIAS
,PHYSICAL_COLUMN
,DESC_PHYSICAL_COLUMN
)
```

9. **Load LDM/PDM Metadata (Table MD_ENTY)**

   Load LDM/PDM mapping and related information into table MD_ENTY.

   For information on this step, see "Load LDM/PDM Metadata (Table MD_ENTY)" on page 6-5.

10. **Load Program (Intra-ETL) Metadata (Table MD_PRG)**

    Load Intra-ETL program input/output and related information into table MD_PRG.

    For information on this step, see "Load Program (Intra-ETL) Metadata (Table MD_PRG)" on page 6-6

11. **Load Reports and KPI Metadata (Table - MD_KPI and MD_REF_ENTY_KPI)**

    Load sample report metadata into MD_KPI and load report/PDM/LDM mapping related information into table MD_REF_ENTY_KPI.

    For information on this step see "Load Reports and KPI Metadata (Table MD_KPI and MD_REF_ENTY_KPI):" on page 6-7.

## Load LDM/PDM Metadata (Table MD_ENTY)

### Source Tables Required

| Source Table Name | Description |
|---|---|
| DMRS_ATTRIBUTES | Containing attributes of the particular entity |
| DMRS_ENTITIES | Containing entity name with unique id |
| MD_NAME_CONVERSION | Containing full name and abbreviation of the distinct word used in the LDM |

### Staging Tables/Views

| Staging Table/View Name | Description |
|---|---|
| MD_OIDM_ATTR_COL_NAME_MAP | Used to store abbreviate the column names based on the standard abbreviation used in the project. |
| MD_DM_ALL_ENT_ATTR | Used to generate and keep the entity description. |

### Loading MD_ENTY (MD_ENTY_POP.SQL)

#### GIVE_ABBRV

Type: Function

This database function GIVE_ABBRV provides the abbreviation for a named token from the table MD_NAME_CONVERSION.

Source Table

MD_NAME_CONVERSION

**Columns**: ABBREVIATION

Target

**Table**: MD_OIDM_ATTR_COL_NAME_MAP

**Columns**: column_name_abbr

#### MD_DM_ALL_ENT_ATTR

Type: View

This database view provides the description of each entity.

| Source Table | Target View |
|---|---|
| DMRS_ENTITIES | MD_DM_ALL_ENT_ATTR |

### PL/SQL Program to Update Column Name

Type: PL/SQL Program

This program updates the column name based on the result of function GIVE_ABBRV.

| Source Tables | Target Table |
|---|---|
| MD_OIDM_ATTR_COL_NAME_MAP | MD_OIDM_ATTR_COL_NAME_MAP |
| DMRS_ATTRIBUTES | Column: column_name_abbr |

### PL/SQL program to insert initial data into MD_OIDM_ATTR_COL_NAM

Type: PL/SQL Program

Provides initial loading for table MD_OIDM_ATTR_COL_NAME_MAP

| Source Tables | Target Table |
|---|---|
| MD_DM_ALL_ENT_ATTR<br>DMRS_ENTITIES | MD_OIDM_ATTR_COL_NAME_MAP |

### PL/SQL program to load data into MD_ENTY

Type: PL/SQL Program

Loads data into MD_ENTY from all the staging tables.

| Source Table | Target Table |
|---|---|
| MD_OIDM_ATTR_COL_NAME_MAP | MD_ENTY |

## Load Program (Intra-ETL) Metadata (Table MD_PRG)

### Source Tables Required

| Source Table Name | Description |
|---|---|
| USER_DEPENDENCIES | This database view describes dependencies between procedures, packages, functions, package bodies, and triggers owned by the current user, including dependencies on views created without any database links. |
| MD_RPD_RPT | This table contains the sample report related information. |

### Staging Tables/Views

| Staging Table/View Name | Description |
|---|---|
| MD_INTRA_ETL | Used to generate and keep the relational/OLAP ETL program metadata information. |
| MD_MINING | Used to generate and keep the data mining ETL program metadata information. |

### Loading MD_PRG (MD_PRG_POP.SQL, MD_MIN_PRG_POP.SQL)

Program: MD_INTRA_ETL

Type: View

This view extracts information for relational and OLAP Intra-ETL packages. The structure is the same as MD_PRG.

| Source View | Target View |
|---|---|
| USER_DEPENDENCIES | MD_INTRA_ETL |

Program: MD_MINING

Type: View

This view extracts information for the data mining Intra-ETL packages. The structure of the view same as MD_PRG.

| Source View | Target View |
|---|---|
| USER_DEPENDENCIES | MD_MINING |

Program: PL/SQL program to load ETL mapping data into MD_PRG.

Type: PL/SQL Program

Load ETL program data into MD_PRG from all the staging views

| Source Views | Target Table |
|---|---|
| MD_INTRA_ETL<br>MD_MINING | MD_PRG |

Program: PL/SQL program insert report data into MD_PRG

Type: PL/SQL Program

Load report data into MD_PRG from report staging table.

| Source Table | Target Table |
|---|---|
| MD_RPD_RPT | MD_PRG |

## Load Reports and KPI Metadata (Table MD_KPI and MD_REF_ENTY_KPI):

### Source Tables Required

| Source Table Name | Description |
|---|---|
| MD_RPD | This tables stores all the RPD metadata information, it is directly loaded from md_rpd.csv |
| MD_REPORT | This tables stores all the report (analysis) metadata information, it is directly loaded from md_report.csv |
| MD_DASHBOARD | This tables stores all the sample report dashboard metadata information, it's directly loaded from md_dashboard.csv |

### Staging Tables/Views

| Staging Table/View Name | Description |
|---|---|
| MD_RPD_CALC_PHY | Stores the missing physical tables and columns for derived measures. Wrote a query to find out missing Physical tables and columns for derived measures. |
| MD_REPORT1 | MD_REPORT1 has the same structure of MD_RPT, it is used to store comma separated tables and columns to the new row, by that it can directly join with physical tables and columns from MD_RPD_CALC_PHY. |
| MD_RPT_DASH | Contains all mappings information between RPD and reports. |
| MD_RPD_RPT_DASH | Stores all the mappings information of Report, RPD and Dashboard. |

### Loading MD_KPI and MD_REF_ENTY_KPI (SAMPLE_REP_POP.SQL)

Program: PL/SQL program Insert non calculated columns Data Into MD_RPD_CALC_PHY

Type: PL/SQL Program

This program extracts those base KPIs or non calculated column information and inserts into MD_RPD_CALC_PHY.

| Source Table | Target Table |
|---|---|
| MD_RPD | MD_RPD_CALC_PHY |

Program: PROCEDURE Proc_DelmValuePopulate2

Type: Procedure

This procedure loads comma separated data to new row of the MD_REPORT1 table.

| Source Table | Target Table |
|---|---|
| MD_REPORT | MD_REPORT1 |

Program: PL/SQL program to create and perform initial load of data into MD_RPD_RPT

Type: PL/SQL Program

This program creates and performs initial load of data for the table MD_RPD_RPT.

| Source Tables | Target Table |
|---|---|
| MD_RPD_CALC_PHY<br>MD_REPORT1 | MD_RPD_RPT |

Program: PL/SQL program to create and initial load data into MD_RPD_RPT_DASH.

Type: PL/SQL Program

This program creates and performs initial load of data for table MD_RPD_RPT_DASH.

| Source Tables | Target Table |
|---|---|
| MD_RPD_CALC_PHY<br>MD_RPT_DASH<br>MD_RPD_RPT_DASH | MD_RPD_RPT_DASH |

Program: PL/SQL program to create and initial load data into MD_RPD_RPT.

Type: PL/SQL Program

This program creates performs initial load of data for table MD_RPD_RPT.

| Source Tables | Target Table |
|---|---|
| MD_RPD_CALC_PHY<br>MD_REPORT1 | MD_RPD_RPT |

Program: MD_DRVD_KP

Type: View

This view extracts and keeps the information for all the calculated KPIs.

| Source Table | Target Table |
|---|---|
| MD_RPD_RPT_DASH | MD_DRVD_KPI |

Program: PL/SQL program to create and performs initial load of data into MD_KPI.

Type: PL/SQL Program

This program creates and performs initial load of data for table MD_KPI.

| Source Table | Target Table |
|---|---|
| MD_RPD_RPT_DASH | MD_KPI |

Program: PL/SQL program to create and initial load data into MD_REF_ENTY_KPI.

Type: PL/SQL Program

This program creates and performs the initial load of data for table MD_REF_ENTY_KPI.

| Source Table | Target Table |
|---|---|
| MD_RPD_RPT_DASHI | MD_REF_ENTY_KPI |

## Metadata Reports and Dashboard

To customize the Oracle Retail Data Model reports, you must understand the dependencies among Oracle Retail Data Model objects, especially how the report KPIs are mapped to the physical tables and columns.

The "Oracle Retail Data Model Metadata" browser that helps you discover these dependencies. When you install Oracle Retail Data Model with its sample reports, the metadata browser is delivered as a sample Dashboard.

> **See:** *Oracle Retail Data Model Installation Guide* for more information on installing the sample reports and deploying the Oracle Retail Data Model RPD and webcat on the Business Intelligence Suite Enterprise Edition instance.

There are four tabs (reports) in the Oracle Retail Data Model Metadata browser:

- Details: Measure-Entity Tab Business Areas and Measures Attributes and Entities
- Details: Entity-Measure Tab Entity to Attribute Measures
- Details Program-Table Tab
- Details: Table-Program Tab

### Details: Measure-Entity Tab Business Areas and Measures Attributes and Entities

On the Measure-Entity tab the measure descriptions, computational formulas with physical columns, physical tables, and corresponding entities can be viewed by Business Area.

To browse the data, select the business area and measure description that you are interested in.

Report table uses and column used details:

| Table | Associated Columns |
|---|---|
| MD_ENTY_VIEW | ENTY_NAME1, ATTRBT_NAME |
| MD_KPI | KPI_NAME, KPI_DESC, CMPUT_LOGIC, BSNS_AREA |
| MD_PRG_VIEW | PHY_TAB_NAME |

### Details: Entity-Measure Tab Entity to Attribute Measures

The Entity-Measure tab displays the measures supported by the entities and how they are calculated. You can discover information about particular entities and attributes.

For example, take the following steps to learn more about an entity:

1. Select the entity.
2. Click **GO**.

Report table uses and column used details:

| Table | Associated Columns |
|---|---|
| MD_ENTY | ENTY_NAME, ATTRBT_NAME |
| MD_KPI | KPI_NAME, CMPUT_LOGIC |

### Details Program-Table Tab

The Program-Table tab displays the input and output tables used in the selected programs.

For example, take the following steps to learn more about intra-ETL mappings:

1. Select the program type (that is, intra-ETL or report) and program name for showing particular report or intra-ETL information.
2. Select **GO**.

Report table uses and column used details:

| Table | Associated Columns |
|---|---|
| MD_PRG | PRG_TYP, PRG_NAME, PHY_TAB_NAME, SB_PRG_TYP, PRG_MODEL, SB_PRG_DESC |

### Details: Table-Program Tab

The Table-Program tab lists the Programs used by a given table and whether that table is an input or output, or both, of that program. To discover what reports use a particular table, you must move a particular table from the right pane to the left (Selected) pane.

For example, to see the reports that use a particular table, take the following steps:

1. In the right pane of the Table-Program tab, select the table.
2. Move the table to the Selected list on the left by clicking on < (left arrow), and click **OK**.
3. Select **GO**.

   The reports for the selected table are displayed.

Report table uses and column used details:

| Table | Associated Columns |
|---|---|
| MD_PRG | PRG_NAME, PHY_TAB_NAME, SB_PRG_TYP, PRG_MODEL, SB_PRG_DESC |

# 7

# Multi-Currency Support and Configuration

This chapter includes the following topics:


- Multi-Currency Overview
- Multi-Currency Data Field Naming Conventions
- Multi-Currency Data Movement
- Currency Data Flow
- Currency DWC_CRNCY_CONF Table


## Multi-Currency Overview

Oracle Retail Data Model supports four currency types:

- Base
- Reporting1
- Reporting2
- Reporting3

You configure and setup these currencies in the DWL_CRNCY_CONF table.

Base Currency is the Standard or default currency for the Oracle Retail Data Model installation. The currencies: Reporting1/2/3 are the three available Reporting Currencies available through the Oracle Retail Data Model Analytical Layer.

In addition, Oracle Retail Data Model supports Local (Lcl) and Transactional (Txn) currencies to store other aspects of the business. These currencies need to be stored in the table DWL_CRNCY. It is expected that a full set of currencies would be defined in this table. This table contains the Oracle Retail Data Model Base and Reporting1/2/3 currencies as well as any other currencies which are used by the Retailer. For example, if the Retailer has operations in ten countries with ten different currencies and if the Base and Reporting1/2/3 currencies are a subset of these ten currencies, then you need to configure the four mandatory currencies in DWL_CRNCY_CONF and also setup the ten currencies in the DWL_CRNCY table (including the four mandatory and the six additional currencies).

The exchange rates, calculated with respect to the Base currency, should be stored in table DWB_EXCHANG_RATE_CRNCY_DAY. You populate this table for enabling currency conversions during Intra-ETL process, from any of the ten defined transaction currencies to the four mandatory currencies. This table can also be used for dynamically converting Base currency figures to any other currency within the Reporting Layer.

The Intra-ETL performs currency conversions at the Base Layer and populates the Base, Reporting 1/2/3 Currencies in respective AMT (amount) columns in the Analytical Layer (Derived/Aggregate).

From a Reporting perspective, certain reports can be built which are enabled for multi-currency analysis. Reports enabled for multi-currency analysis can show a drop down containing Base or one of the three Reporting Currencies and any other currency available in the `DWL_CRNCY` table. Reports delivered in Base and Reporting 1/2/3 currencies are available in a pre-calculated mode. The Reports (requests) do not need to perform any conversions while requesting a currency from within this group. For any other currency, the reports will need to perform currency conversion dynamically and the converted results would be displayed in the report (performance will be worse when compared to a report request involving a mandatory currency).

**Note:** Most of the currency conversion reports (especially the non-mandatory currencies) are relational in nature. They would be built to run against the relational source tables in Oracle Retail Data Model. The OLAP component as well as OLAP Reports can support only the mandatory Base and Reporting 1/2/3 currencies.

## Multi-Currency Data Field Naming Conventions

The naming conventions for the various currency related fields in Oracle Retail Data Model model are as follows:

- **Base Currency**: *_AMT: Represents the Base/Standard currency amount for Oracle Retail Data Model.
- **Local Currency**: *_AMT_LCL: Represents the local currency amount for an organization. This column can be summed / added up to organization level of Org Hierarchy. The local currency is defined in the Organization in field DWR_ORG_BSNS_UNIT. PRMRY_CRNCY_ISO_CD.
- **Transaction Currency**: *_AMT_TXN: Represents the transactional amount coming from source into this column. As a transaction can occur in many currencies, you need to store the currency code along with the Transaction record details. The column TXN_CRNCY_CD in the base fact table stores the transactional currency code.
- **Reporting Currency1**: *_AMT_RPT: This column contains the amount in Reporting Currency 1. This currency code is as configured/set up in table DWC_CRNCY_CONF. This is the value in column CRNCY_CD_VAL corresponding to CRNCY_CD_TYP value GLBL_CRNCY_RPT_CD.
- **Reporting Currency2**: *_AMT_RPT2: This column contains the amount in Reporting Currency 2. This currency code is as configured/set up in table DWC_CRNCY_CONF. This is the value in column CRNCY_CD_VAL corresponding to CRNCY_CD_TYP value GLBL_CRNCY_RPT2_CD.
- **Reporting Currency3**: *_AMT_RPT3: This column contains the amount in Reporting Currency 3. This currency code is as configured/set up in table DWC_CRNCY_CONF. This is the value in column CRNCY_CD_VAL corresponding to CRNCY_CD_TYP value GLBL_CRNCY_RPT3_CD.

# Multi-Currency Data Movement

Oracle Retail Data Model obtains the transaction amount and the currency information from the source system and loads this information into interface tables. The transaction and currency data movement occurs as follows:

- Movement from Interface to Base Tables
- Movement from Base to Derived Tables
- Movement from Derived to Aggregate Tables

## Movement from Interface to Base Tables

The movement from Interface to Base tables occurs as follows:

- The value of the *_AMT column is calculated based on the currency rate and the rate is picked up from the table DWB_EXCHNG_RATE_CRNCY_DAY.
- The value of the *_AMT_LCL column is calculated based on the currency rate and the rate is picked up from the table DWB_EXCHNG_RATE_CRNCY_DAY.
- The value of the *_AMT_TXN column is the same as the transaction amount and the corresponding currency code is placed in the new column.
- The value of the *_AMT_RPT column is calculated based on currency rate and the rate is picked up from the table DWB_EXCHNG_RATE_CRNCY_DAY.
- The value of the *_AMT_RPT2 column is calculated based on the currency rate and the rate is picked up from the table DWB_EXCHNG_RATE_CRNCY_DAY.
- The value of the *_AMT_RPT3 column is calculated based on the currency rate and the rate is picked up from the table DWB_EXCHNG_RATE_CRNCY_DAY.

### Handling Currency at the Base Level

The Interface input file should have the Txn_Cd of the record being loaded. The transactions are converted into Base crncy and loaded in _AMT columns in the DWB_* table. The original (txn) value is loaded into the *_AMT_TXN column in DWB_ tables. Also the _AMT_RPT/2/3 columns contain the default Reporting currency values (converted). The column TXN_CRNCY_CD indicates the TXN currency.

## Movement from Base to Derived Tables

The movement from Base to derived tables occurs as follows:

- It is not possible to sum up the values in the *_AMT_TXN columns from the DWB table to the DWD table as the DWD record can encompass many transactional currencies. Thus, the *_AMT_TXN column is not present in the Derived Table.
- The *_AMT_LCL in DWD table could be summed up at Business Unit level from corresponding column in DWB. If Derived table is at higher level, then this column should not be present in the Derived table.
- The *_AMT column in DWD table would be summed from the corresponding column in DWB table.
- The *_AMT_RPT column in DWD table would be summed from the corresponding column in DWB table.
- The *_AMT_RPT2 column in DWD table would be summed from the corresponding column in DWB table.

- The *_AMT_RPT3 column in DWD table would be summed from the corresponding column in DWB table.

## Movement from Derived to Aggregate Tables

The movement from derived to aggregate tables occurs as follows:

- The *_AMT_LCL in DWA table could be summed up at Business Unit level from corresponding column in DWB/DWD. If Aggregate table is at higher level, then this column should not be present in the Aggregate table.
- The *_AMT column in DWA table would be summed up from corresponding column in DWB/DWD table.
- The *_AMT_RPT column in DWA table would be summed up from corresponding column in DWB/DWD table.
- The *_AMT_RPT2 column in DWA table would be summed up from corresponding column in DWB/DWD table.
- The *_AMT_RPT3 column in DWA table would be summed from corresponding column in DWB/DWD table.

### Handling Data Movement from Base to Derived and Aggregate Layers

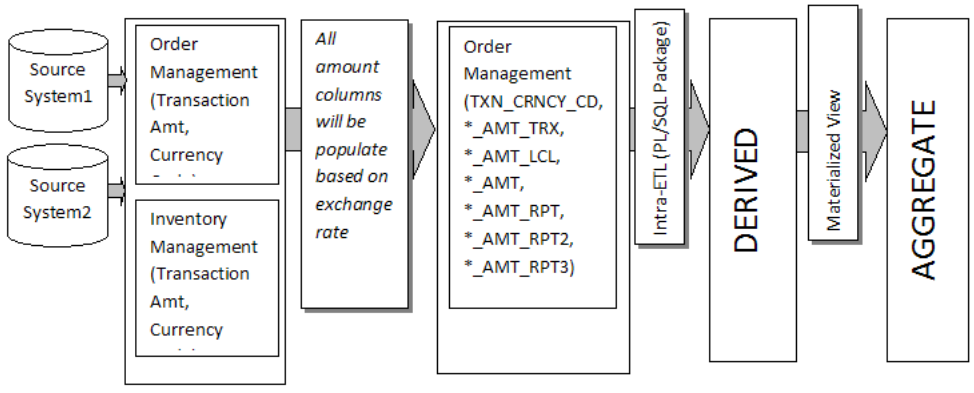
Base information can be summed up and loaded into respective columns in Derived table. Derived table will contain the _AMT (Base) and _AMT_RPT, _AMT_RPT2/3 for RPT1/2/3 currencies. The _AMT_TXN will not be present in DWD table as it does not apply (multiple Txn currencies cannot be rolled up into single value).

Similarly Derived information is summed up and loaded into DWA entities.

# Currency Data Flow

Figure 7–1 shows the data flow showing the movement of currency information through the layers of Oracle Retail Data Model.

***Figure 7–1 Currency and Transaction Amount Data Flow in Oracle Retail Data Model***




# Currency DWC_CRNCY_CONF Table

Table 7–1 shows the currency configuration table DWC_CRNCY_CONF details. This table stores the Base Currency and the three reporting currencies.

*Table 7–1 DWC_CRNCY_CONF Table Details*

| Column Name | Data Type | Nullable | Remarks |
|---|---|---|---|
| CRNCY_CD_TYP | VARCHAR2 (30) | No | Types of Currency Codes. Possible values:<br>GLBL_CRNCY_RPT_CD<br>GLBL_CRNCY_RPT2_CD<br>GLBL_CRNCY_RPT3_CD<br>BASE_CRNCY_CD<br>This is the PK column for this table |
| CRNCY_CD_VAL | VARCHAR2 (30) | No | Value of corresponding Currency code like USD, GBP, INR, and so on |
| WID | NUMBER(30) | No | System field |

The information in the columns shown in Table 7–1 and should be set up during Oracle Retail Data Model installation and should not be modified or updated subsequently.

For example, the currency configuration table DWC_CRNCY_CONF entries suitable for a customer with a base currency GBP and requires reporting in three currencies: FRC, USD and EUR is shown in Table 7–2.

*Table 7–2 DWC_CRNCY_CONF Sample Values*

| CRNCY_CD_TYP | CRNCY_CD_VAL | WID |
|---|---|---|
| BASE_CRNCY_CD | GBP | 1 |
| GLBL_CRNCY_RPT_CD | FRC | 1 |
| GLBL_CRNCY_RPT2_CD | USD | 1 |
| GLBL_CRNCY_RPT3_CD | EUR | 1 |

Use the table DWB_EXCHNG_RATE_CRNCY_DAY to store the exchange rates for currency conversion.

The table DWL_CRNCY stores all the currencies required by Oracle Retail Data Model. This table should contain the four mandatory configuration currencies as defined in DWC_CRNCY_CONF table. This table must also contain any additional currencies which may be used as Transactional or Local currencies (Transactions) or as additional Reporting currencies. Certain Reports can be built which perform currency conversion dynamically and display results in terms of these additional Reporting Currencies.

# A

# Sizing and Configuring an Oracle Retail Data Model Warehouse

This appendix provides information about sizing and configuring an Oracle Retail Data Model warehouse. It contains the following topics:

- Sizing an Oracle Retail Data Model Warehouse
- Configuring a Balanced System for Oracle Retail Data Model

## Sizing an Oracle Retail Data Model Warehouse

Businesses now demand more information sooner and are delivering analytics from their Enterprise Data Warehouse (EDW) to an ever-widening set of users and applications. To keep up with this increase in demand the EDW must now be near real-time and be highly available. Regardless of the design or implementation of a data warehouse the initial key to good performance lies in the hardware configuration used. This has never been more evident than with the recent increase in the number of data warehouse appliances in the market.

But how do you go about sizing such a system? You must first understand how much throughput capacity is required for your system and how much throughput each individual CPU or core in your configuration can drive, thus the number one task is to calculate the database space requirement in your data warehouse.

There are two data volume estimate resources in a data warehouse environment:

- The estimated raw data extract from source systems. This estimate affects the ETL system configuration and the staging layer database space in the data warehouse system. Because this value is determined by your specific transactional systems, you must calculate this information yourself.
- The space needed for data stored to support the objects defined in the default Oracle Retail Data Model schema. This appendix provides information you can use to make this calculation.

### Calculation Factors When Making a Data Volume Calculation for an Oracle Retail Data Model Warehouse

Consider the following calculation factors when making a data volume calculation:

- Calculates data unit volume within different type:
- Reference and lookup tables data. Assume this data is permanently stored.
- Base tables data (transaction data). Assume that this data is stored within its life cycle.

- Star schema (derived and summary). Assume that this data is stored within its life cycle.
- Calculate each type of data retention.
- Define how many months or years of each type of tables to retain.
- Calculate data growth.
- Assume that annual growth rate: applies to both transaction and reference data and data in the star schema.
- Assume that annual change rate applies only to reference data.
- Calculate Staging Area data requirements, if proposed.

  **Tip:** Multiply ETL volume by day by number of days held for problem resolution and re-run of transform with new extract from source systems.

- Calculate data volume for indexes, temporary tables, and transaction logs.
- Calculate the space requirement for business intelligence tools, such as cubes, and data mining.
- Consider the redo log and Oracle ASM space requirement.
- Consider the RAID architecture [RAID 1, 0+1, 5]
- Consider the backup strategy.
- Consider the compress factor if applied.
- Consider the OS and file system disk space requirements.

### Formula to Determine Minimum Disk Space Requirements for an Oracle Retail Data Model Warehouse

Use the following formula, based on the factors outlined in "Calculation Factors When Making a Data Volume Calculation for an Oracle Retail Data Model Warehouse" on page A-1, to determine the minimum disk space requirements for an Oracle Retail Data Model warehouse.

```
Disk Space Minimum Requirements = Raw data size * Database space
factor * (1+GrthperY)nY*OS and File system factor * Compress
Factor * Storage Redundant factor
```

where:

- `Raw data size = (reference and lookup data per year + base/transaction data per year + derived and summary data per year +staging data +other data(OLAP/Data Mining))`
- `Database space factor = Indexes + Temporary Tables + Logs]`
- `GrthperY` = growth rate per year
- `OS and File system factor` is the install and configuration and maintain space for OS and DB
- `Redundant factor= ASM disk space and RAID factor. [RAID 1=2, RAID 5=1.25 or 1.33]`
- `Compress factor` depends how you apply the compress function. If you are executing on an Oracle Exadata Database machine, it has a huge savings in disk space by using compression.

# Configuring a Balanced System for Oracle Retail Data Model

Many data warehouse operations are based upon large table scans and other I/O-intensive operations, which perform vast quantities of random I/Os. To achieve optimal performance the hardware configuration must be sized end to end to sustain this level of throughput. This type of hardware configuration is called a balanced system. In a balanced system all components - from the CPU to the disks - are orchestrated to work together to guarantee the maximum possible I/O throughput. I/O performance is always a key consideration for data warehouse designers and administrators. The typical workload in a data warehouse is especially I/O intensive, with operations such as large data loads and index builds, creation of materialized views, and queries over large volumes of data. Design the underlying I/O system for a data warehouse to meet these heavy requirements.

To create a balanced system, answer the following questions:

- How many CPUs are required? What speed is required?
- What amount of memory is required? Data warehouses do not have the same memory requirements as mission-critical transactional applications?
- How many I/O bandwidth components are required? What is the desired I/O speed?

Each component must be able to provide sufficient I/O bandwidth to ensure a well-balanced I/O system.

The following topics provide more information about configuring a balanced system for Oracle Retail Data Model:


- Maintaining High Throughput in an Oracle Retail Data Model Warehouse
- Configuring I/O in an Oracle Retail Data Model for Bandwidth not Capacity
- Planning for Growth of Your Oracle Retail Data Model
- Testing the I/O System Before Building the Oracle Retail Data Model Warehouse
- Balanced Hardware Configuration Guidelines for Oracle Retail Data Model


## Maintaining High Throughput in an Oracle Retail Data Model Warehouse

The hardware configuration and data throughput requirements for a data warehouse are unique mainly because of the sheer size and volume of data. Before you begin sizing the hardware configuration for your data warehouse, estimate the highest throughput requirement to determine whether current or proposed hardware configuration can deliver the necessary performance. When estimating throughput, use the following criteria:

- The amount of data accessed by queries during peak time, and the acceptable response time
- The amount of data that is loaded within a window of time

## Configuring I/O in an Oracle Retail Data Model for Bandwidth not Capacity

Based on the data volume calculated and the highest throughput requirement, you can estimate the I/O throughput along with back-end ETL process and front end business intelligence applications by time unit. Typically, a value of approximately 200MB per second I/O throughput per core is a good planning number for designing a balanced system. All subsequent critical components on the I/O path - the Host Bus Adapters,

fiber channel connections, the switch, the controller, and the disks - have to be sized appropriately.

When running a data warehouse on an Oracle Real Application Cluster (Oracle RAC) it is just as important to size the cluster interconnect with the same care and caution you would use for the I/O subsystem throughput.

When configuring the storage subsystem for a data warehouse, it should be simple, efficient, highly available and very scalable. An easy way to achieve this is to apply the S.A.M.E. methodology (Stripe and Mirror Everything). S.A.M.E. can be implemented at the hardware level or by using Oracle ASM (Automatic Storage Management) or by using a combination of both. There are many variables in sizing the I/O systems, but one basic rule of thumb is that the data warehouse system has multiple disks for each CPU (at least two disks for each CPU at a bare minimum) to achieve optimal performance.

## Planning for Growth of Your Oracle Retail Data Model

A data warehouse designer plans for future growth of a data warehouse. There are several approaches to handling the growth in a system, and the key consideration is to be able to grow the I/O system without compromising on the I/O bandwidth. You cannot, for example, add four disks to an existing system of 20 disks, and grow the database by adding a new tablespace striped across only the four new disks. A better solution would be to add new tablespaces striped across all 24 disks, and over time also convert the existing tablespaces striped across 20 disks to be striped across all 24 disks.

## Testing the I/O System Before Building the Oracle Retail Data Model Warehouse

When creating a data warehouse on a new system, test the I/O bandwidth before creating all of the database data files to validate that the expected I/O levels are being achieved. On most operating systems, you can perform the test using simple scripts to measure the performance of reading and writing large test files.

## Balanced Hardware Configuration Guidelines for Oracle Retail Data Model

You can reference the follow tips for a balanced hardware configuration:

- Total throughput = `#cores X 100-200MB` (depends on the chip set)
- Total host bus adaptor (HBA) throughput = Total core throughput

> **Note:** If total core throughput is 1.6 GB, you need four 4 Gbit HBAs.

- Use one disk controller per HBA port (throughput capacity must be equal).
- Switches must have the capacity as HBAs and disk controllers.
- Use a maximum of ten physical disk per controller (that is, use smaller drives: 146 or 300 GB).
- Use a minimum of 4 GB of memory per core (8 GB if using compress).
- Interconnect bandwidth equals I/O bandwidth (InfiniBand).

Oracle now provides the Oracle Database Machine, Exadata which combines industry-standard hardware from Oracle, Oracle Database 11*g* Release 2, and Oracle Exadata Storage Server Software to create a faster, more versatile database machine.

It's a completely scalable and fault tolerant package for all data management, especially for data warehousing.

Oracle also has a series of Optimized Warehouse Reference configurations that help customers take the risk out of designing and deploying Oracle data warehouses. Using extensive field experience and technical knowledge, Oracle and its hardware partners have developed a choice of data warehouse reference configurations that can support various sizes, user populations and workloads. These configurations are fast, reliable and can easily scale from 500 GB to over 100 TB on single and clustered servers to support tens to thousands of users.

# Index


## A

access layer, 2-2
  customizing, 3-1
  Oracle Retail Data Model, 2-3, 3-1
As Is reports, 5-7
As Was reports, 5-7

## C

compression
  in Oracle Retail Data Model, 2-9
  materialized views, 3-24
configuring Oracle Retail Data Model
    warehouse, A-3
conventions
  when customizing physical model, 2-4
cubes
  adding materialized view capabilities to, 3-14
  changing the dimensions of, 3-17
  changing the measures of, 3-17
  customizing, 3-15
  data maintenance methods, 3-18
  forecast, 3-17
  in Oracle Retail Data Model, 3-15, 3-16
  partitioning, 3-17
customizing
  access layer, 3-1
  cubes, 3-15
  Oracle Retail Data Model, 1-3
  physical data model, 2-1

## D

dashboards, Oracle Retail Data Model, 5-2, 5-12
data governance committee, responsibilities of, 1-5
data mining models
  customizing, 3-2
derived tables
  in Oracle Retail Data Model, 3-1
dimensional components, Oracle Retail Data
    Model, 3-9

## E

error handling
  during intra-ETL execution, 4-20
ETL for Oracle Retail Data Model, 4-1

## F

fit-gap analysis for Oracle Retail Data Model, 1-10
forecast cube in Oracle Retail Data Model, 3-17
foundation layer
  defined, 2-2
  Oracle Retail Data Model, 2-3
foundation layer of Oracle Retail Data Model
  common change scenarios, 2-6

## H

HCC, 2-10
hybrid columnar compression
  and Oracle Retail Data Model, 2-10

## I

implementers of Oracle Retail Data Model
  prerequisite knowledge, 1-4
implementing
  Oracle Retail Data Model, 1-3
indexes
  in Oracle Retail Data Model, 2-11
  materialized views, 3-22
  partitioning, 2-12
integrity constraints
  in Oracle Retail Data Model, 2-11
intra-ETL
  managing errors, 4-20
  monitoring execution of, 4-20
  Oracle Retail Data Model, 4-1
  recovery, 4-21
  troubleshooting, 4-21
intra-ETL, Oracle Retail Data Model
  executing, 4-17

## J

join performance, improving, 2-13

## K

keys, surrogate

in Oracle Retail Data Model, 2-10

## L

loading Oracle Retail Data Model data, 4-14

## M

materialized views
compressing, 3-24
in Oracle Retail Data Model, 3-19
indexing, 3-22
partition change tracking, 3-23
partitioning, 3-22
refresh options
refreshing
materialized views, 3-20
Metadata Dependency Manager
with Oracle Retail Data Model, 1-9
metadata management
repository, 1-7, 6-9
with Oracle Retail Data Model, 1-6
metadata repository, 1-7
browsing, 1-7, 6-9
with Oracle Retail Data Model, 1-7, 6-9

## N

naming conventions
for physical model of Oracle Retail Data Model, 2-4

## O

Oracle data mining models
Oracle Retail Data Model, 3-2
Oracle Retail Data Model
access layer, 2-3, 3-1
components of, 1-2
customizing, 1-3
customizing physical model, 2-1, 2-3, 2-4, 2-8
dashboards, 5-2
data governance, 1-5
described, 1-1
dimensional components, 3-9
ETL, 4-1
fit-gap analysis, 1-10
foundation layer, 2-3
implementing, 1-3
intra-ETL, 4-1
loading, 4-14
Metadata Dependency Manager, 1-9
metadata management, 1-6
metadata repository, 1-7, 6-9
Oracle products used by, 1-2
Oracle Warehouse Builder, using with, 1-9
physical layers of, 2-2
pre-implementation tasks, 1-4
querying, 5-3
refreshing data, 4-18
reporting, 5-1, 5-3
sample reports, 5-2
source-ETL, 4-1, 4-2, 4-4, 4-5
staging layer, 2-2
tablespaces, design recommendations, 2-8
Oracle Retail Data Model implementers
prerequisite knowledge for, 1-4
Oracle Retail Data Model warehouse
configuring, A-3
sizing, A-1
Oracle Warehouse Builder
with Oracle Retail Data Model, 1-9

## P

parallel execution
enabling for a session, 2-16
enabling for DML operations, 2-16
in Oracle Retail Data Model, 2-14
partition change tracking, 3-23
partition exchange load, 4-8
partitioned indexes in Oracle Retail Data Model, 2-11
partitioning
cubes, 3-17
for join performance, 2-13
for manageability, 2-13
for source-ETL, 4-8
indexes, 2-12
materialized views, 3-22
tables, 2-12
partitions, changing, 2-8
physical model of Oracle Retail Data Model
characteristics of, 2-1, 2-3, 2-4
customizing, 2-4
general recommendations for, 2-8

## Q

querying Oracle Retail Data Model, 5-3

## R

refreshing
Oracle Retail Data Warehouse, 4-18
reporting
Oracle Retail Data Model, 5-1, 5-3
reports
approaches to, 5-1
As Is, 5-7
As Was, 5-7
troubleshooting performance, 5-6
reports, Oracle Retail Data Model
creating new, 5-20

## S

sample reports
customizing, 5-2
sizing
Oracle Retail Data Model warehouse, A-1

source-ETL
exception handling, 4-5
jobs control, 4-5
loading considerations, 4-6
Oracle Retail Data Model, 4-1, 4-2, 4-4, 4-5, 4-6
parallel direct path load, 4-8
partitioning for, 4-8
workflow, 4-5
staging layer, 2-1
Oracle Retail Data Model, 2-2
star queries, optimizing, 5-4
surrogate keys
in Oracle Retail Data Model, 2-10

## T

tables
compressing, 2-9
derived, 3-1
partitioning, 2-12
tablespace in Oracle Retail Data Model, 2-8