Oracle® Communications Data Model

Implementation and Operations Guide

Release 11.3.2

E28442-05

October 2013

ORACLE®

Oracle Communications Data Model Implementation and Operations Guide, Release 11.3.2

E28442-05

Contents

1 Introduction to Oracle Communications Data Model Customization

2 Physical Model Customization

3 Access Layer Customization

4 ETL Implementation and Customization

5 Report and Query Customization

6 Metadata Collection and Reports

List of Examples

List of Figures

Preface

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

This preface contains the following topics:

Audience

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

Oracle Communications Data Model Documentation Set

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

- *Oracle Communications Data Model Installation Guide*
- *Oracle Communications Data Model Reference*
- *Oracle Communications Data Model Adapters and Analytics Installation Guide*
- *Oracle Communications Data Model Adapters and Analytics User's Guide*
- *Oracle Communications 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 the tutorial or demo (within "All") by name.

For example, search within "All" for "OCDMTutorial" to go to the tutorial for Oracle Communications Data Model.

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 Communications Data Model Customization

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

What is the Oracle Communications Data Model?

Oracle Communications Data Model is a standards-based, pre-built approach to data warehousing for the communications industry enabling customers to realize the power of insight more quickly. Oracle Communications 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 communications specific data model. You can use Oracle Communications Data Model in any application environment. Also, you can easily extend the model.

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

Oracle Communications Data Model provides much of the data modeling work that you must do for a communications business intelligence solution. The Oracle Communications Data Model logical and physical data models were designed following best practices for communications service providers. Oracle Communications Data Model is aligned with TM Forum's Information Framework (SID) Release 12 as discussed in *Oracle Communications Data Model Reference*.

Components of Oracle Communications Data Model

Oracle Communications Data Model includes the following components:

- Logical model which is a third normal form (3NF) entity-object standards-based model. The logical model is described in *Oracle Communications Data Model Reference*.
- Physical model defined as one Oracle Database schema. This schema defines all the relational, OLAP, and data mining components.
- Intra-ETL database packages and SQL scripts to extract, transform, and load (ETL) data from the Oracle Communications Data Model 3NF physical tables to the derived and aggregate tables in Oracle Communications Data Model.
- Sample reports and dashboards developed using Oracle Business Intelligence Suite Enterprise Edition.
- DDL and installation scripts

Note: When you use the Oracle Installer to install Oracle Communications Data Model, you have the choice of performing two different types of installations:

- Installation of the Oracle Communications Data Model component, itself
- Installation of sample reports (and schemas)

See *Oracle Communications Data Model Installation Guide* for detailed information on the different types of installation.

The installer for Oracle Communications Data Model Add-ons installs the separately licensed options to Oracle Communications Data Model described in "Oracle Products That Make Up Oracle Communications Data Model" on page 1-2.

See: *Oracle Communications Data Model Reference* for detailed descriptions of the components.

Oracle Products That Make Up Oracle Communications Data Model

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

- Oracle Database with OLAP, Data Mining and Partitioning Option
- Oracle Communications Data Model Application Adapters and Analytics Add-ons
- Oracle Development Tools
- Oracle Business Intelligence Suite Enterprise Edition Presentation Tools

Oracle Database with OLAP, Data Mining and Partitioning Option

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

Oracle Communications Data Model Application Adapters and Analytics Add-ons

The Oracle Communications Data Model Add-ons provides several separately licensed options to Oracle Communications Data Model:

- **Oracle Communications BRM Adapter**: this separately licensed option loads data from an Oracle Communications Billing and Revenue Management source system into Oracle Communications Data Model. You can load data in both an initial and an incremental manner. The data from Oracle Communications Billing and Revenue Management populates the Oracle Communications Data Model derived and aggregate tables, reports, and mining models.
- **Oracle Communications NCC Adapter**: this separately licensed option populates the foundation layer of an Oracle Communications Data Model warehouse with data from an Oracle Communications Network Charging and Control (NCC) system.
- **Oracle Communications Billing Analytics**: this separately licensed option provides an end-to-end analytic application developed on Oracle Communications Data Model that leverages data fed through the BRM Adapter and the NCC Adapter, provides comprehensive and visually appealing dashboards and enables users to:
  - Understand existing business customers across all their products and services
  - Predict and analyze customer behavior to increase revenue and retention
  - Reduces time, cost, risk, and complexity of custom solution
- **Oracle Communications Social Network Analytics**: this separately licensed option provides reports and tools to assist you with Acquiring Customers, Retaining Customers, and Growing Your Customer Base, and with tools and reports for:
  - Identification of social network communities.
  - Network metrics characterizing the social graph.
  - Network distribution information.
  - Predictive scores for churn and influence at a node level, and potential revenue/value at risk.
  - User interface targeted at business users and flexible ad-hoc reporting

Oracle Development Tools

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

Table 1–1 Oracle Development Tools Used with Oracle Communications Data Model

Name	Use
SQL Developer or SQL*Plus	To modify, customize, and extend database objects
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 sample dashboard reports that are provided with Oracle Communications Data Model.

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

Steps for Implementing an Oracle Communications Data Model Warehouse

Although Oracle Communications Data Model was designed following best practices for communications service providers, usually the model requires some customization to meet your business needs.

The reasons that you might customize Oracle Communications Data Model include:

- Your business does not have a business area that is included in the Oracle Communications Data Model.
- You must apply a table or column, or change a calculation or business rule in the Intra-ETL due to the unique way your company does business.

Typical physical model modifications include: adding, deleting, modifying, or renaming tables and columns; or altering foreign keys, constraints, or indexes.

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

1. Perform the organizational tasks outlined in "Before You Begin Customizing the Oracle Communications Data Model" on page 1-5.
2. Create a fit-gap analysis report by following the process outlined "Performing Fit-Gap Analysis for Oracle Communications Data Model" on page 1-7.
3. In a development environment, install a copy of the Oracle Communications Data Model.
4. Customize Oracle Communications 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 2-6. When writing the ETL, follow the guidelines in "ETL for the Foundation Layer of an Oracle Communications Data Model Warehouse" 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 Oracle Communications Data Model" on page 4-9.
5. In a test environment, make a copy of your customized version of Oracle Communications Data Model. Then, following the documentation you created in Step 2, test the customized version of Oracle Communications Data Model
6. Following your typical procedures, roll the tested customized version of Oracle Communications Data Model out into pre-production and, then, production.

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

Before You Begin Customizing the Oracle Communications Data Model

Before you begin customizing Oracle Communications 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 1-6.
- Implementation team. This team consists of IT engineers who have the expertise outlined in "Prerequisite Knowledge for Implementers" on page 1-5. This team has the responsibilities outlined in "Steps for Implementing an Oracle Communications Data Model Warehouse" on page 1-4.
- Fit-gap analysis team. This team consists of business analysts who can identify the business requirements and scope of the Oracle Communications 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 Communications Data Model" on page 1-7.

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). The order of implementation either top-down (per business or subject area) or bottom-up (source-leading) should be based on the "quick wins" (easy implementation, clean and known source, no or very little changes, out-of-the-box reports), themselves ordered by business relevance (from a Return On Investment perspective and from a strategic perspective).
- Agree on the timing and the working arrangements.

Prerequisite Knowledge for Implementers

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

- Experience performing information and data analysis and data modeling. (Experience using Oracle SQL Data Modeler, is a plus).
- Hands on experience developing ETL or ELT, prefereable in the chosen ETL tool (ODI, Golden Gate, and so on).
- Knowledge of the source applications, their data and their table structures from which you want to load data into Oracle Communications Data Model.
- An understanding of the Oracle technology stack, especially data warehouse (Database, Data Warehouse, OLAP, Data Mining, 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
- Oracle Business Intelligence Suite Enterprise Edition Administrator, Answers, and Dashboards

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 Communications 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 direction and response for the governance framework and should at least cover the following areas:

- The entire data warehouse life cycle.
- Agree on the data to process and make available to end-users.
- Determine what is the minimum quality criteria for the data that is available to end users and determine 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 data warehouse.
- The policies, procedures and standards for data resource and data access, and the implications it may have on the existing or future business processes. For the later, the committee must make sure to communicate early enough to the right persons the process change request to ease the integration and to save time.
- 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 "Overview of Managing Metadata for Oracle Communications Data Model" on page 6-1' 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.

Performing Fit-Gap Analysis for Oracle Communications Data Model

Fit-gap analysis is where you compare your information needs and communications business requirements with the structure that is available with Oracle Communications Data Model. You identify any required functionality that is not included in the 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 Communications Data Model to fit your communications 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 the data structures of your source to the Oracle Communications Data Model schema:
   - Starting from business requirements, questions, and rules, identify any objects that are *not* in the Oracle Communications Data Model.
   - Compare the Oracle Communications Data Model to your existing application model if have one.
   - Compare the Oracle Communications Data Model to the application data that you are using as a data source to the Oracle Communications Data Model warehouse.
3. Determine the differences between your needs and Oracle Communications 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 Communications 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 Communications 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 create and to identify and make any changes to the intra-ETL provided with Oracle Communications Data Model.
   c. Reports: A clear distinction should be made between reports end-users could create themselves from the default data and data structure available in the OBIEE repository, and those that would require additions (from earlier

changes in Physical model, or simply because the considered entity was not accessible by default in the repository). Note that this step could be done a bit later, in a second phase, not for the initial implementation, with the risk to possibly miss an important source of information if the business interviews were not run properly.

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 Communications 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 Communications Data Model Warehouse" on page 1-4.

Data Encryption and Security for Oracle Communications Data Model

To comply with privacy and data protection requirements, Oracle Communications Data Model is certified with Transparent Data Encryption and Oracle Database Vault.

For more information on using Transparent Data Encryption, see *Oracle Database Administrator's Guide* and for information on using Oracle Database Vault, see *Oracle Database Vault Administrator's Guide*.

For more information on these topics, see:

`http://www.oracle.com/technetwork/database/options/advanced-security/index-099011.html`

`http://www.oracle.com/technetwork/database/options/database-vault/index-085211.html`

2

Physical Model Customization

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

See also: Chapter 3, "Access Layer Customization"

Characteristics of the Default Physical Model

The default physical model of Oracle Communications Data Model defines:

1800+ tables and 40,000+ columns
5000+ industry-specific measures and KPIs
7 pre-built data mining models
22 pre-built OLAP cubes

The default physical model of the Oracle Communications 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 Communications 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 operational 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.

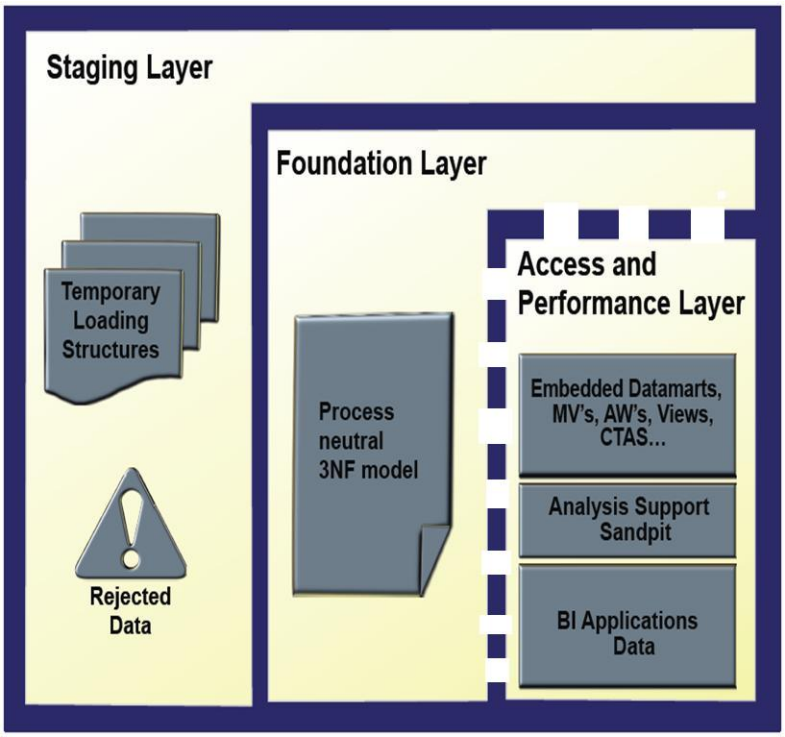
Note: In some implementations this layer is not necessary, because all data transformation processing is done as needed as data is extracted from the source system before it is inserted directly into the foundation layer.

- **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 many 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 Communications Data Model Warehouse

Oracle Communications Data Model warehouse environment also consists of three layers However, as indicated by the dotted line in Figure 2–1, "Layers of an Oracle Communications Data Model Warehouse" on page 2-2, in the Oracle Communications Data Model the definitions of the foundation and access layers are combined in a single schema.

Figure 2–1 Layers of an Oracle Communications Data Model Warehouse



The layers in the Oracle Communications Data Model warehouse are:

- **Staging layer**. As in a "traditional" data warehouse environment, an Oracle Communications Data Model warehouse environment can have a staging layer. Because the definition of this layer varies by customer, a definition of this area is not provided as part of Oracle Communications Data Model.

  Note: If you are using an application adapter for Oracle Communications Data Model to populate the foundation layer of Oracle Communications Data Model, then that adapter defines and populates an Oracle Communications Data Model staging layer.

- **Foundation and Access layers.** The physical objects for these layers are defined in a single schema, the `ocdm_sys` schema:

  - **Foundation layer.** The foundation layer of the Oracle Communications Data Model is defined by base tables that present the data in 3NF (that is, tables that have the `DWB_` prefix). This layer also includes reference, lookup, and control tables defined in the `ocdm_sys` schema (that is, the tables that have the `DWR_` , `DWL_`, `DWC_` prefixes).

  - **Access layer.** The access layer of Oracle Communications 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 `DWV_` prefix), and cube views (defined with `_VIEW` suffix). These structures provide a summarized or "flattened" perspectives 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. The access layer also includes the tables with prefixes as shown in Table 2–1.

    See: *Oracle Communications Data Model Reference* for detailed information on the `ocdm_sys` schema.

Customizing the Oracle Communications Data Model Physical Model

The starting point for the Oracle Communications 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 the Oracle Communications 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 Communications Data Model as outlined in"Characteristics of the Default Physical Model" on page 2-1 and presented in detail in *Oracle Communications Data Model Reference*.
3. Modify the foundation level of your physical model of Oracle Communications Data Model, as needed. See "Common Change Scenarios" 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 2-7 to guide you when designing the physical objects.
- Follow the conventions used when creating the default physical model of Oracle Communications 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 `ocdm_sys` schema.

4. Modify the access layer of your physical model of Oracle Communications Data Model as discussed in Chapter 3, "Access Layer Customization".
5. Modify existing or create a new Intra-ETL packages to feed the changes you make in the access layer, as discussed in Chapter 4, "ETL Implementation and 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 convert the logical data model into the physical layer. For example, answer the following questions before you design the physical data model:

- Do you need the physical data model to cover the full scope of the logical data model, or only part of the scope?

  "Common Change Scenarios" on page 2-6 provides 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 Communications 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 Communications Data Model, the conventions outlined below were followed. 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 `ocdm_sys` schema delivered with Oracle Communications Data Model uses the prefixes and suffixes shown Table 2–1 to identify object types.

Table 2–1 Default Physical Object Prefixes and Suffixes in Oracle Communications Data Model

Prefix or Suffix	Used for Name of These Objects
`_VIEW`	A relational view of an OLAP cube, dimension, or hierarchy.
`CCB_`	Customized OLAP cubes.
`CUBE`	Created when OLAP cubes are built. Used to store logs and results.
`DM$`	Created when the mining models are trained. Used to store trained model and logs.
`DMV_`	Materialized view created for performance reasons (that is, *not* an aggregate table or an OLAP cube).
`DR$`	Created when the mining models are trained. Used to store trained model and logs.
`DWA_`	Aggregate tables which are materialized views.
`DWB_`	Base transaction data (3NF) tables.
`DWC_`	Control tables.
`DWD_`	Derived tables -- including data mining result tables.
`DWL_`	Lookup tables.
`DWR_`	Reference data tables.
`DWV_`	Relational view of time dimension

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

Domain Definition Standards

A domain is a set of values allowed for a column. The domain can be enforced by a foreign key, check constraints, or the application on top of the database. Define the standards for each domain across the model such as:

- Date and time type, such as `'YYYY-MM-DD'`. For example, be aware that most date columns (abbreviation DT) in Oracle Communications Data Model may contain the time, such as `EVT_STRT_DT`. There is no separate `TIME` column.
- Numeric value in different situations. For example, all columns of type `COUNT` are `NUMBER(16,0)` while all monetary-like columns (`AMOUNT`) are `NUMBER(22,7)`.
- Character string length in different situations. For example, all Code columns are `VARCHAR2(120)`, Name (`NAME`) and Description columns (`DSCR`) are respectively 500 and 1000 characters long (with some exceptions). Indicator columns (`IND`) are `CHAR(1)`.

- Coded value definition such as key or description. For example, all "Key" columns are `NUMBER(30)`.

Foundation Layer Customization

The first step in customizing the physical model of Oracle Communications Data Model is customizing the foundation layer of the physical data model. Since, as mentioned in "Layers in the Default Oracle Communications Data Model Warehouse" on page 2-2, the foundation layer of the physical model mirrors the 3NF logical model of Oracle Communications 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 Communications 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" 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 2-7 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 Communications Data Model with caution. The foundation layer of the physical model of the Oracle Communications Data Model has (at its core) a set of generic structures that allow it to be flexible and extensible. You may disable temporarily Foreign Keys and constraints if required but do not forget to set them back when the entities are back in use. Before making extensive additions, deletions, or changes, ensure that you understand the full range of capabilities of Oracle Communications Data Model and that you cannot handle your requirements using the default objects in the foundation layer.

Before making changes, review Appendix C, "Product Assumptions" in *Oracle Communications Data Model Reference*. For more information, see *Oracle Communications Data Model Reference*.

Common Change Scenarios

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 Communications Data Model, add new tables and columns.

  Carefully study the default foundation layer of the physical data model of Oracle Communications 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 Communications Data Model Development Team for possible inclusion in future releases of Oracle Communications 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 the Oracle Communications 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 Communications Data Model. Also identify the impact on any applications based on the new design.

General Recommendations When Designing Physical Structures

The `ocdm_sys` schema delivered with Oracle Communications 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 Communications Data Model:

- Tablespaces in the Oracle Communications Data Model
- Data Compression in the Oracle Communications Data Model
- Tables for Supertype and Subtype Entities in Oracle Communications Data Model
- Surrogate Keys in the Physical Model
- Integrity Constraints in Oracle Communications Data Model
- Indexes and Partitioned Indexes in the Oracle Communications Data Model
- Partitioned Tables in the Oracle Communications Data Model
- Parallel Execution in the Oracle Communications Data Model

Tablespaces in the Oracle Communications 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 Communications Data Model warehouse.

Changing the Tablespace and Partitions Used by Tables

You can change the tablespace and partitions used by Oracle Communications Data Model tables. What you do depends on whether the Oracle Communications 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 Communications Data Model defines the partitioned tables as interval partitioning, which means the partitions are created only when new data arrives.

  Consequently, for Oracle Communications 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 by *tablespace 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 Communications Data Model tables that do not have partitions (that is, lookup tables and reference tables), to change the existing tablespace for a table then issue the following statement.

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

Data Compression in the Oracle Communications 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 the CTAS or IAS statement ("Create Table As Select" or "Insert As Select" types of statements). Specify an `ORDER BY` a `NOT NULL` column (ideally non numeric) that has many distinct values (1,000 to 10,000).

> **See also:** "Types of Data Compression Available" on page 2-8 and "Compressing Materialized Views" on page 3-29.

Types of Data Compression Available

Oracle Database offers the following types of compression:

- Basic or Standard Compression
- OLTP Compression

- Hybrid Columnar Compression

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 Communications Data Model leverages the compress feature for all base, derived, and aggregate tables which reduces the amount of data being stored, reduces memory usage (more data per memory block), and increases query performance.

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

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

Example 2–1 Creating a Compressed Table for Oracle Communications Data Model

To create a compressed table named `CWB_ACCS_MTHD_PORT_HIST`.

```
COMPRESS
Create table CWB_ACCS_MTHD_PORT_HIST
(NP_RQST_HDR_CD                 VARCHAR2(30)
,ACCS_MTHD_KEY                   NUMBER(30)  NOT NULL ENABLE
,EXTRNL_OPRTR_KEY                NUMBER(30)
…..
)
tablespace TBS_BASE
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 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 Hybrid Columnar Compression, 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 Hybrid Columnar Compression is not recommended; instead, the use of OLTP compression is recommended.

Hybrid Columnar Compression provides different levels of compression, focusing on query performance or compression ratio respectively. With Hybrid Columnar Compression 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 Hybrid Columnar Compression in *Oracle Database Concepts*.

Tables for Supertype and Subtype Entities in Oracle Communications Data Model

A supertype is a generic entity type that has a relationship with one or more subtypes.

A subtype is a sub-grouping of the entities in an entity type that is meaningful to the organization and that shares common attributes or relationships distinct from other subgroups.

- Subtypes inherit all supertype attributes
- Subtypes have attributes that are different from other subtypes

Recommendations: Tables for Supertype and Subtype Entities

Create separate tables for the super type and all sub type entities for the following reasons:

- Data integrity enforced at database level. (using `NOT NULL` column constraints)
- Relationships can be accurately modeled and enforced including those which apply to only one subtype
- Physical model closely resembles the logical data model.
- It is easier to correlate the logical data model with the physical data model and support the logical data model enhancements and changes.
- Physical data model reflects true business rules (for example, if there are some attributes or relationships mandatory for only one subtype.)

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:

- Have to allocate 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 to operational systems

Integrity Constraints in Oracle Communications 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, therefore, most foreign key columns in Oracle Communications Data Model are nullable.

Indexes and Partitioned Indexes in the Oracle Communications 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 Communications 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 non-partitioning key, or a performance requirement).

See also: "Partitioned Tables in the Oracle Communications Data Model" on page 2-12, "Choosing Indexes for Materialized Views" on page 3-28, "Choosing a Cube Partitioning Strategy for Oracle Communications Data Model" on page 3-23, and "Partitioning and Materialized Views" on page 3-28.

Partitioned Tables in the Oracle Communications 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 Communications Data Model for Manageability
- Partitioning the Oracle Communications Data Model for Easier Data Access
- Partitioning the Oracle Communications 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 the Oracle Communications Data Model" on page 2-12, "Choosing a Cube Partitioning Strategy for Oracle Communications Data Model" on page 3-23, and "Partitioning and Materialized Views" on page 3-28.

Partitioning the Oracle Communications Data Model for Manageability

Range partitioning helps improve the manageability and availability of large volumes of data.

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.

Oracle Communications Data Model uses Interval Partitioning as an extension of Range Partitioning, so that you provide just the first partition higher limit and interval to create the first partition and the following partitions are created automatically as and when data comes. The (hidden) assumption is that the data flow is more or less similar over the various intervals.

Partitioning the Oracle Communications Data Model for Easier Data Access

Range partitioning also helps ensure that only the necessary data to answer a query is scanned. Assuming that the 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.

Partitioning the Oracle Communications Data Model for Join Performance

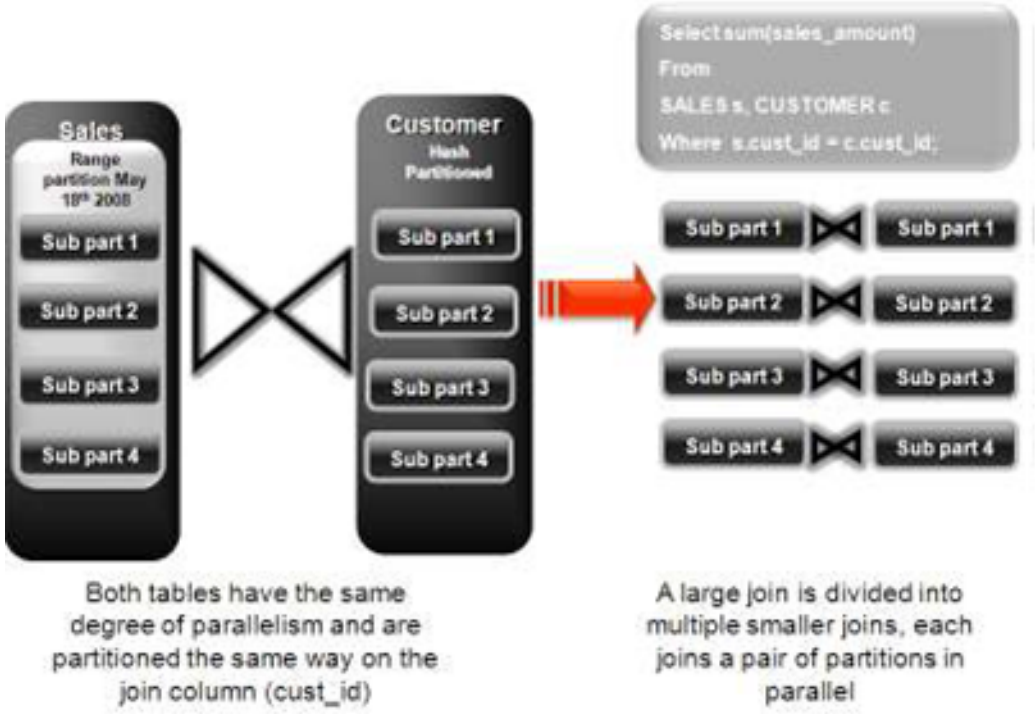
Sub-partitioning by hash is used predominately for performance reasons. Oracle Database uses a linear hashing algorithm to create sub-partitions.

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.

Figure 2–2 shows how 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

In order 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 16 MB. 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.

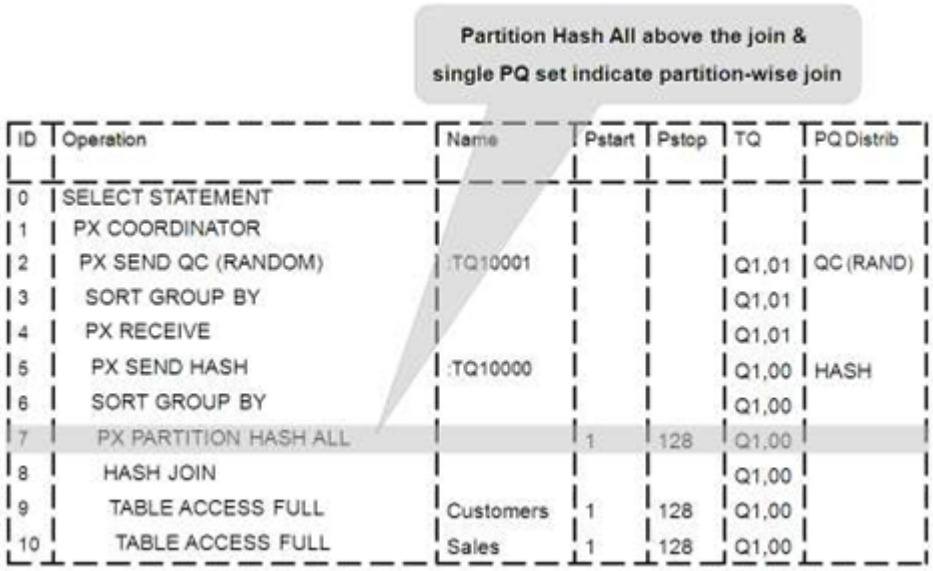
Parallel Execution in the Oracle Communications Data Model

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. Below figure shows the execution plan you would see for this join.

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



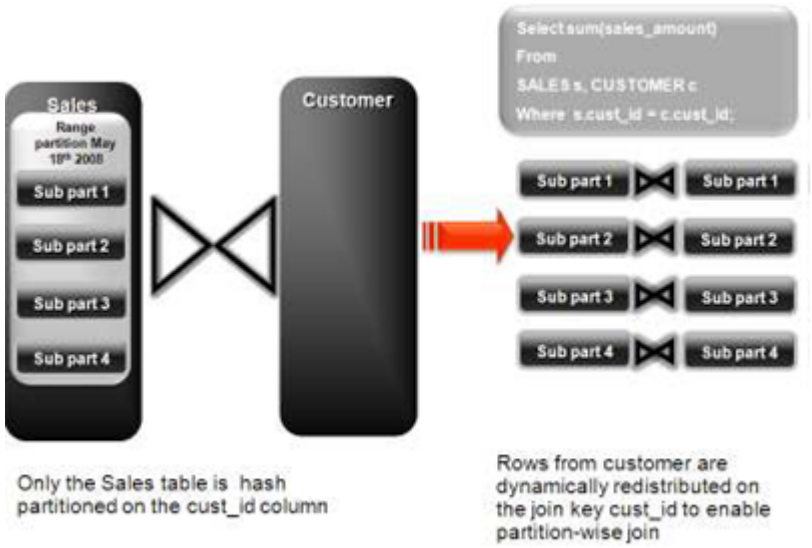
To ensure that you get optimal performance when executing a partition-wise join in parallel, specify a number for the partitions in each of the tables that is larger than the degree of parallelism used for the join. If there are more partitions than parallel servers, each parallel server is given one pair of partitions to join, when the parallel server completes that join, it requests another pair of partitions to join. This process repeats until all pairs have been processed. This method enables the load to be balanced dynamically (for example, 128 partitions with a degree of parallelism of 32).

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;
```

> **Note:** It is usually suggested to set at session level rather than at the system level.

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 Communications Data Model. It includes the following topics:

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

See also: Chapter 2, "Physical Model Customization"

Introduction to Customizing the Access Layer of Oracle Communications Data Model

The access layer of Oracle Communications 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 `ocdm_sys` schema include: derived and aggregate tables and OLAP cube views. This layer also contains data mining models. The results of these models are stored in derived tables. The models themselves are also defined in the `ocdm_sys` schema.

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 2-7.
- Design access layer objects to support business intelligence reports and queries that your business end-users require. 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 Communications Data Model
- Aggregate Tables in the Oracle Communications Data Model
- Dimensional Components in the Oracle Communications Data Model
- Materialized Views in the Oracle Communications Data Model

Derived Tables in the Oracle Communications Data Model

Derived tables have a DWD_ prefix. Derived tables are tables where one of the following apply:

- Have as values the result of a non-aggregate calculation against the data in the foundation layer tables.
- Have some minimal level of aggregation, typically at the day level (for example, DWD_CNT_DAY).
- Summarize, at day or month level, all activities over the period of specific processes, split in various attributes (for example, DWD_ACCT_BAL_MO).

Some derived tables leverage, and hence are dependent on, other derived tables (for example DWD_CUST_MNNG).

Depending on the type of derived table you customize derived tables as follows:

- Tables that hold the results of a calculation such as the DWD_BER_FER_ERR_RATIO_DAY table that contains values that are the daily BER (Bit Error Rate) and FER (Frame Error Rate) statistics about the network elements. 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 Communications Data Model Data Mining Models" on page 3-2.

> **See:** The Derived Tables topic in *Oracle Communications Data Model Reference* for a list of all of the derived tables in the default Oracle Communications 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 Communications 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 or, alternatively, by adding missing dimensions and measures to existing derived tables. When designing these tables, name the tables following the convention of using the CWD_ prefix (for Customized Warehouse Derived) or DWD_ (for Data Warehouse Derived). Make sure all the main dimensions are put first and have Foreign Keys to their corresponding reference or lookup tables. Attributes that add information only, avoiding costly joins, should not be part of the Primary Key of the derived table. Some dimensions that are part of a hierarchy do not necessarily need to be part of the Primary Key. All measures should be put afterward, grouped if possible by similar meaning. Make sure that all monetary measures should have the LOCAL and REPORTING amount associated, and if possible the currency (as required in the TM Forum SID).

> **See:** *Oracle Communications Data Model Reference* for details on dimensions that are part of a hierarchy.

Customizing Oracle Communications Data Model Data Mining Models

Some derived (DWD_) tables in the default ocdm_sys schema are the results of data mining models defined in the default Oracle Communications Data Model. Those models are defined in the default ocdm_sys schema that also comes with Oracle Communications Data Model.

Oracle Communications Data Model data mining models get source data from views defined on two derived tables (DWD_) and a base table (DWB_). These tables are:

- DWD_CUST_DNA
- DWD_VAS_SBRP_QCK_SUMM
- DWB_EVT_PRTY_INTRACN

Derived tables store data mining models prediction results and model rules, in (DWD_), and use reference tables (DWR_), and look up tables (DWL_). These tables are:

- DWD_CUST_PROD_AFFLTN
- DWD_CUST_DNA
- DWR_CUST_DT_NODE
- DWD_CHRN_SVM_FACTOR
- DWD_CHRN_SVM_ROC
- DWD_PROMO_SVM_FACTOR
- DWD_PROMO_SVM_ROC
- DWR_CUST_SGMNT
- DWR_CUST_SGMNT_DTL
- DWL_MNNG_LTV_BAND
- DWL_MNNG_LT_SRVVL_BAND

When Oracle Communications Data Model is installed the installer copies the mining source scripts to $ORACLE_HOME/ocdm/pdm/mining/src.

Table 3–1 lists the mining scripts.

Table 3–1 Oracle Communications Data Model Mining Scripts

Script Name	Description
cust_sntmnt_manual_score.sql	Predefined dictionary for customer comment scoring.
ocdm_mining_init.sql	Initializes mining environment and executes the other three mining scripts.
pkg_mining_etl.sql	Defines views, which have training/apply data, on source tables.
pkg_ocdm_mining.sql	Core mining package that has a procedure for each model. Each procedure drops, creates mining model and scores mining model.

When you create a customized Oracle Communications Data Model warehouse, you can customize the data mining models in the following ways:

- Create a model as discussed in "Creating a New Data Mining Model for Oracle Communications Data Model".
- Modify an existing model as discussed in "Modifying Oracle Communications Data Model Data Mining Models".

See also: "Tutorial: Customizing the Churn Prediction Model" on page 3-5.

Creating a New Data Mining Model for Oracle Communications Data Model

To create a data mining model:

1. Define the problem and identify input attributes. Also identify target attribute if the mining problem is supervised.
2. Check if the existing mining source views defined in `pkg_mining_etl.sql` script support the requirement of your problem. Modify the definition of views to support your requirement. Do not remove any columns from view definition unless you are sure that those columns do not make any sense.
3. If the existing mining source views do not support required fields, create a new table or view to support your requirements. Add the new table to `pkg_mining_etl.sql` PL/SQL package. Follow the naming conventions outlined in "Conventions When Customizing the Physical Model" on page 2-4 and use a `DWD_` prefix for results tables. Modify the intra-ETL programs to support your mining problem requirements.
4. For each mining problem that Oracle Data Mining supports, there is more than one algorithm. Create a setting table for your mining problem and follow the naming convention. The prefix for a setting table is "`DM_`". Add the definition of new setting table to `ocdm_mining_init.sql` script.
5. Add a procedure for your mining problem to `pkg_ocdm_mining` PL/SQL package. This procedure should create mining model and score the trained mining model on apply data. Compile the package. To create the mining model for your problem, invoke the newly added procedure. Make sure your new procedure works according to your expectations. Check `user_mining_models` data dictionary view for trained model. There are few more data dictionary views that give more information on the trained models. For more details, refer to *Oracle Data Mining Concepts*.

Modifying Oracle Communications Data Model Data Mining Models

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

1. Change the definition of source views used as input to the mining model.
2. If required, change the definition of source derived table, `DWD_CUST_DNA`. Do not remove any existing columns. Only add new columns with `NULL` enable.
3. Modify the intra-ETL package of `DWD_CUST_DNA` table. Execute the intra-ETL package to load data into `DWD_CUST_DNA` table.
4. Refresh mining views by executing following statement. You need to pass training day and apply day key:

```
SQL> exec PKG_MINING_ETL.refresh_mining_views(l_trnng_day_key,l_apply_day_key);
```

5. Train the model again by calling Oracle Communications Data Model mining package.
6. 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 Communications Data Model

To add a new column to `create_prpd_churn_svm_model`, perform the following steps:

1. Add the new column to views that are used in `create_prpd_churn_svm_model`.

- DWV_PRPD_CUST_CHRN_SRC
- DWV_PRPD_CUST_CHRN_TST
- DWV_PRPD_CUST_CHRN_APPLY

2. Train the model by issuing the following statement:

```
pkg_ocdm_mining.create_prpd_churn_svm_model(training_day_key);
```

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

```
SQL> SELECT attribute_name
        FROM TABLE(SELECT ATTRIBUTE_SET FROM
        TABLE(DBMS_DATA_MINING.GET_MODEL_DETAILS_SVM('OCDM_PRPD_CHURN_SVM'))
     WHERE CLASS=1);
```

See also: "Refreshing the Data in an Oracle Communications Data Model Warehouse" on page 4-17, and Chapter 7, "Working with User Privileges in Oracle Communications Data Model"

Tutorial: Customizing the Churn Prediction Model

After you have populated Oracle Communications Data Model foundation layer and the derived tables, you can leverage the prebuilt Oracle Communications Data Model mining model for some more advanced analysis and predictions.

This tutorial shows you how to predict the prepaid customers who will terminate the service in next three months (churners) based on the populated Oracle Communications Data Model warehouse. Using prebuilt Oracle Communications Data Model Mining models, you can easily and very quickly see the prediction result 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 the initially generated mining model, as time goes by, the customer information and their behavior change. Consequently, you must refresh the previous trained mining models based on the latest customer and usage data. You can follow the process in this tutorial to refresh the mining models to acquire predictions on latest customer information.

This tutorial shows you how to generate the Churn Prediction model through Oracle Communications Data Model Mining APIs. To use different parameters in the training process, or customize the model in more advanced fashion, you can also use Oracle Data Miner to do the same work.

This tutorial consists of the following:

- Tutorial Prerequisites
- Preparing Your Environment
- Generating the Prepaid Churn Prediction Model
- Checking the Result

Tutorial Prerequisites Before starting this tutorial:

1. Review the Oracle by Example (OBE) tutorial "Using Oracle Data Miner 11*g* 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 Communications Data Model.
3. Populate the base, reference, and lookup tables.
4. Execute the intra-ETL.

Ensure that the following tables contain valid data:

DWB_ACCT_STAT_HIST
DWB_BRDBND_USG_EVT
DWB_FIXED_LN_CALL_EVT
DWB_ISP_USG_EVT
DWB_MMS_EVT
DWB_NP_RQST_HDR
DWB_SMS_EVT
DWB_WRLS_CALL_EVT
DWB_WRLS_CNTNT_DNLDG_EVT
DWD_ACCT_DEBT_MO
DWD_ACCT_PYMT_DAY
DWD_CNTCT_CNTR_DAY
DWD_CUST_DNA
DWD_PRPD_ACCT_STTSTC_DAY
DWL_AGE_BND
DWL_AGE_ON_NET_BND
DWL_DEBT_AGNG_BND
DWR_ACCT
DWR_ADDR_LOC
DWR_AGRMNT
DWR_CUST
DWR_BSNS_MO
DWR_CUST_SCR
DWR_DEMOG_ATRIB
DWR_EXTRNL_OPRTR
DWR_HH
DWR_INDVL_DEMOG_VAL
DWR_JB
DWR_PROD_SBRP
DWR_SOC_JB

Note: If you have not populated the real customer data and you only want to learn the Oracle Communications Data Model mining model, you can use the sample data by taking the following steps:

1. Ensure that during the install you generated the calendar data covering the range of 2011 through 2012. For example, by setting the parameters of starting from `"20070101"` for 10 years satisfy this condition.
2. Download the sample data (`ocdm_for_mining.dmp.zip`):

   http://download.oracle.com/technetwork/database/options/comm-data-model/ocdm_for_mining.dmp.zip

   Then unzip the sample data and import that data into your new `ocdm_sys` schema using the following command:

   ```
   imp system/<password> file=ocdm_for_mining.dmp fromuser=ocdm_
   sample touser=ocdm_sys ignore=y log=mining_dump_import.log
   ```

Preparing Your Environment

This tutorial requires a valid, populated Oracle Communications Data Model warehouse.

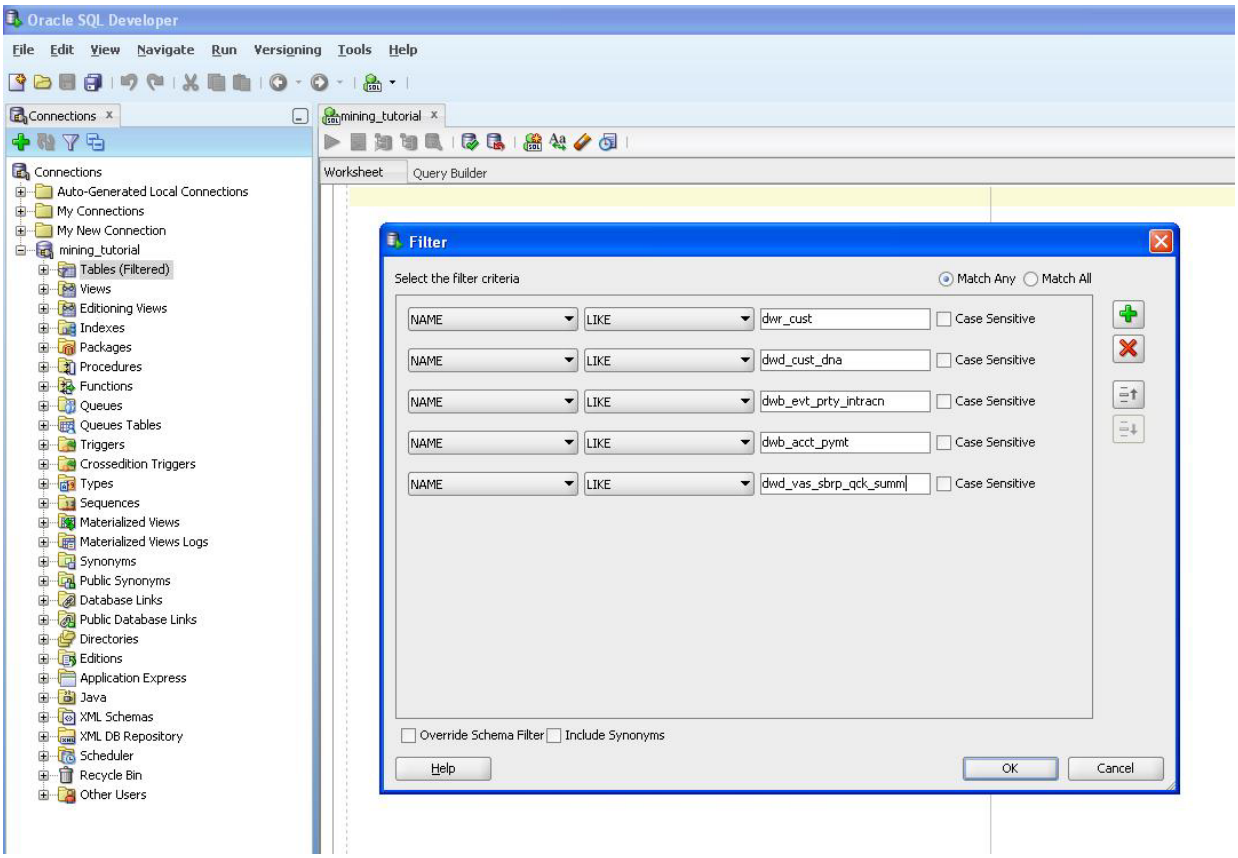
Oracle by Example: For more information about using SQL Developer, refer to tutorial "Getting Started with Oracle SQL Developer 3.0". To access this tutorial, open the Oracle Learning Library in your browser by following the instructions in "Oracle Technology Network" on page xii; and, then, search for the tutorial by name.

To prepare the environment, take the following steps:

1. In SQL Developer, connect to the `ocdm_sys` schema.

   Tip: SQL Developer can be found on any Oracle Database Installation under `$ORACLE_HOME/sqldeveloper`.

2. After you connect to the `ocdm_sys` schema, you can see all the tables in that schema. You can narrow down the list by right clicking the "table" and then applying filters:



3. (Optional) As mentioned in the "Preparing Your Environment" on page 3-7", if you have not populated tables with your data you can try using sample data. After you download the sample data, follow these steps to import the sample data:

   a. Connect to sqlplus as sysdba:

   ```
   sqlplus / as sysdba
   ```

   b. Execute following commands to generate disable constraint script:

   ```
   spool disable_constraints.sql
   SET PAGESIZE 15000
   SET LINESIZE 1000
   conn ocdm_sys/ocdm_sys
   SELECT 'ALTER TABLE ' || table_name || ' disable constraint ' ||
   constraint_name || ';'
   FROM user_constraints
   WHERE STATUS='ENABLED'
   AND constraint_type = 'R';


   SELECT 'ALTER TABLE ' || table_name || ' disable constraint ' ||
   constraint_name || ';'
   FROM user_constraints
   WHERE STATUS='ENABLED'
   AND constraint_type = 'U';

   SELECT 'ALTER TABLE ' || table_name || ' disable constraint ' ||
   constraint_name || ';'
   FROM user_constraints
   WHERE STATUS='ENABLED'
   ```

```
AND constraint_type = 'C';

SELECT 'ALTER TABLE ' || table_name || ' disable constraint ' ||
constraint_name || ';'
FROM user_constraints
WHERE STATUS='ENABLED'
AND constraint_type = 'P';

spool off
```

c. Execute `disable_constraints.sql` script as `ocdm_sys` user:

```
sqlplus ocdm_sys/ocdm_sys @disable_constraints.sql
```

d. Import the sample dump into `ocdm_sys` schema by executing the following commands:

```
imp system/<password> file=ocdm_for_mining.dmp fromuser=ocdm_sample
touser=ocdm_sys ignore=y log=mining_dump_import.log
```

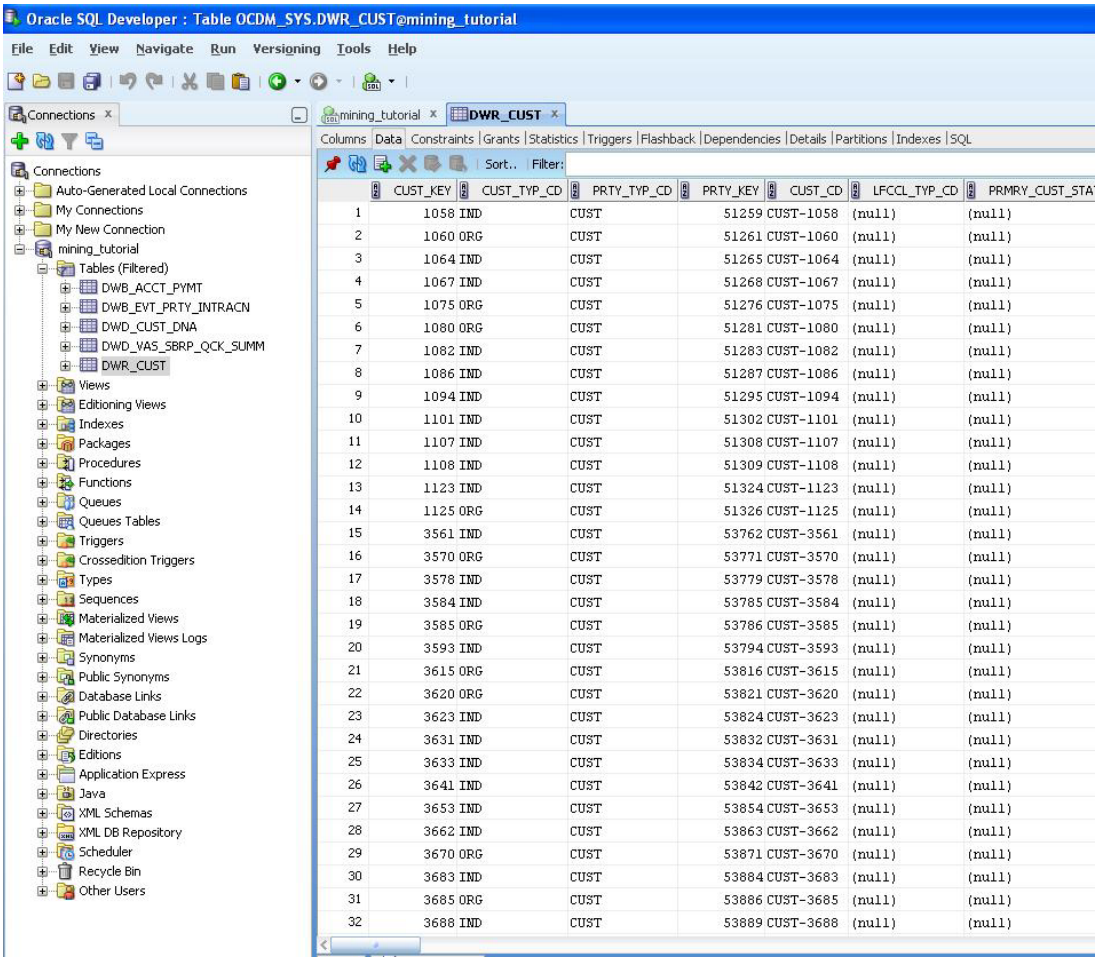
e. Execute the following commands to generate enable constraint script:

```
spool enable_constraints.sql
SET PAGESIZE 15000
SET LINESIZE 1000
conn ocdm_sys/ocdm_sys
SELECT 'ALTER TABLE ' || table_name || ' enable constraint ' || constraint_
name || ';'
FROM user_constraints
WHERE STATUS='DISABLED'
AND constraint_type = 'R';


SELECT 'ALTER TABLE ' || table_name || ' enable constraint ' || constraint_
name || ';'
FROM user_constraints
WHERE STATUS='DISABLED'
AND constraint_type = 'U';

SELECT 'ALTER TABLE ' || table_name || ' enable constraint ' || constraint_
name || ';'
FROM user_constraints
WHERE STATUS='DISABLED'
AND constraint_type = 'C';

SELECT 'ALTER TABLE ' || table_name || ' enable constraint ' || constraint_
name || ';'
FROM user_constraints
WHERE STATUS='DISABLED'
AND constraint_type = 'P';

spool off
```

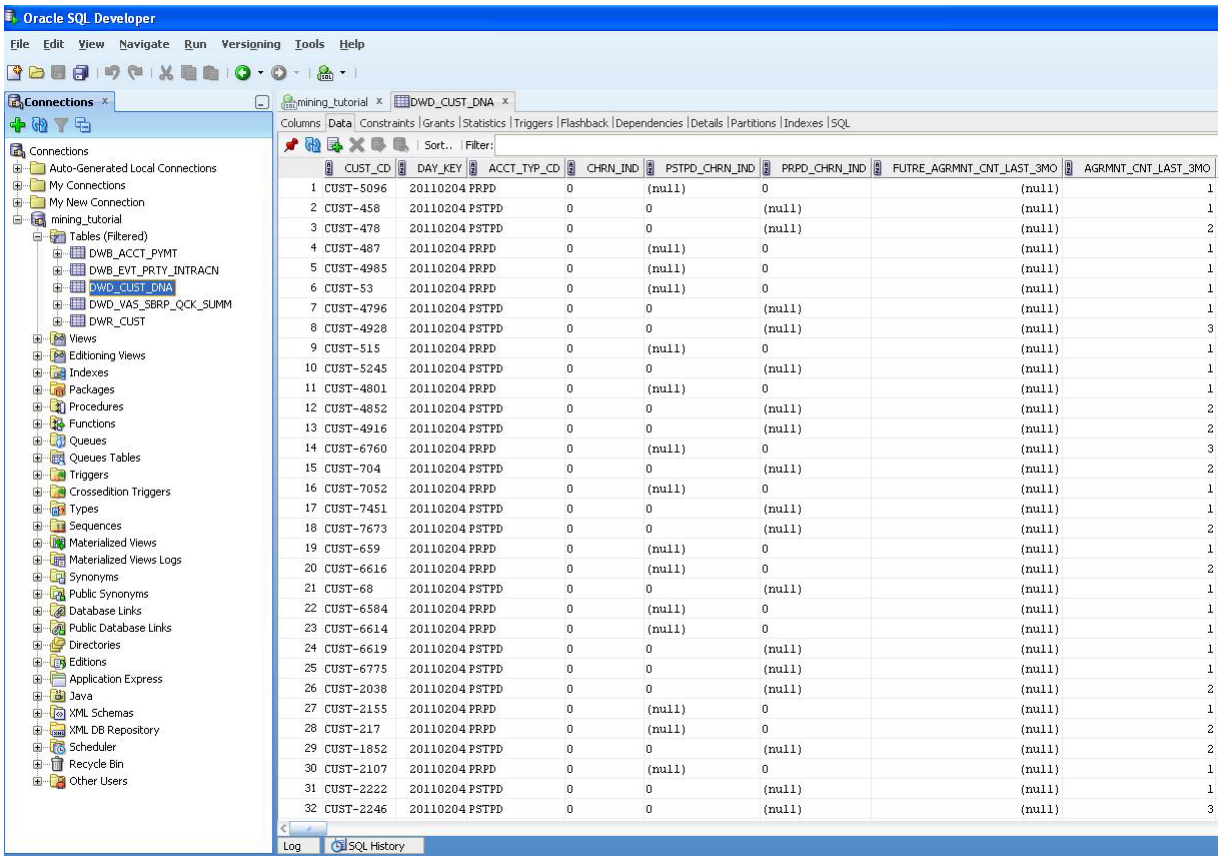
f. Execute `enable_constraints.sql` script as `ocdm_sys` user:

```
sqlplus ocdm_sys/ocdm_sys @enable_constraints.sql
```

4. Review tables to ensure they contain valid data, either from your own customer data, or from the sample data:



5. Review the `DWD_CUST_DNA` table:



6. Select and check each of the following tables to ensure that the table is properly populated:

   DWR_CUST

   DWB_EVT_PRTY_INTRACN

   DWB_ACCT_PYMT

   DWD_CUST_DNA

   DWD_VAS_SBRP_QCK_SUMM

Generating the Prepaid Churn Prediction Model

For generating prepaid churn prediction model, refresh the mining views defined on mining source tables DWD_CUST_DNA and DWD_VAS_SBRP_QCK_SUMM.

1. Refresh mining views by passing training day key and apply day key. If you are using mining sample dump provided, then pass 20110201 as training day key and 20110301 as apply day key:

   ```
   sqlplus  ocdm_sys/ocdm_sys
   SQL> exec PKG_MINING_ETL.refresh_mining_views(l_trnng_day_key,l_apply_day_key);
   ```

2. Check user_mining_models data dictionary view for mining models. It should return no data if you have not previously trained any models:

   ```
   SQL> SELECT * FROM user_mining_models;
   ```

3. Train the prepaid churn prediction model by executing following:

```
sqlplus  ocdm_sys/ocdm_sys
SQL> exec PKG_OCDM_MINING.create_prpd_churn_svm_model (l_trnng_day_key);
```

4. Check `user_mining_models` data dictionary view for trained prepaid churn prediction mining model:

```
SQL> SELECT * FROM user_mining_models;
```

5. Check `DWD_CHRN_SVM_FACTOR` table for prepaid churn SVM factors:

```
SQL> SELECT * FROM DWD_CHRN_SVM_FACTOR WHERE ACCT_TYP_CD = 'PRPD';
```

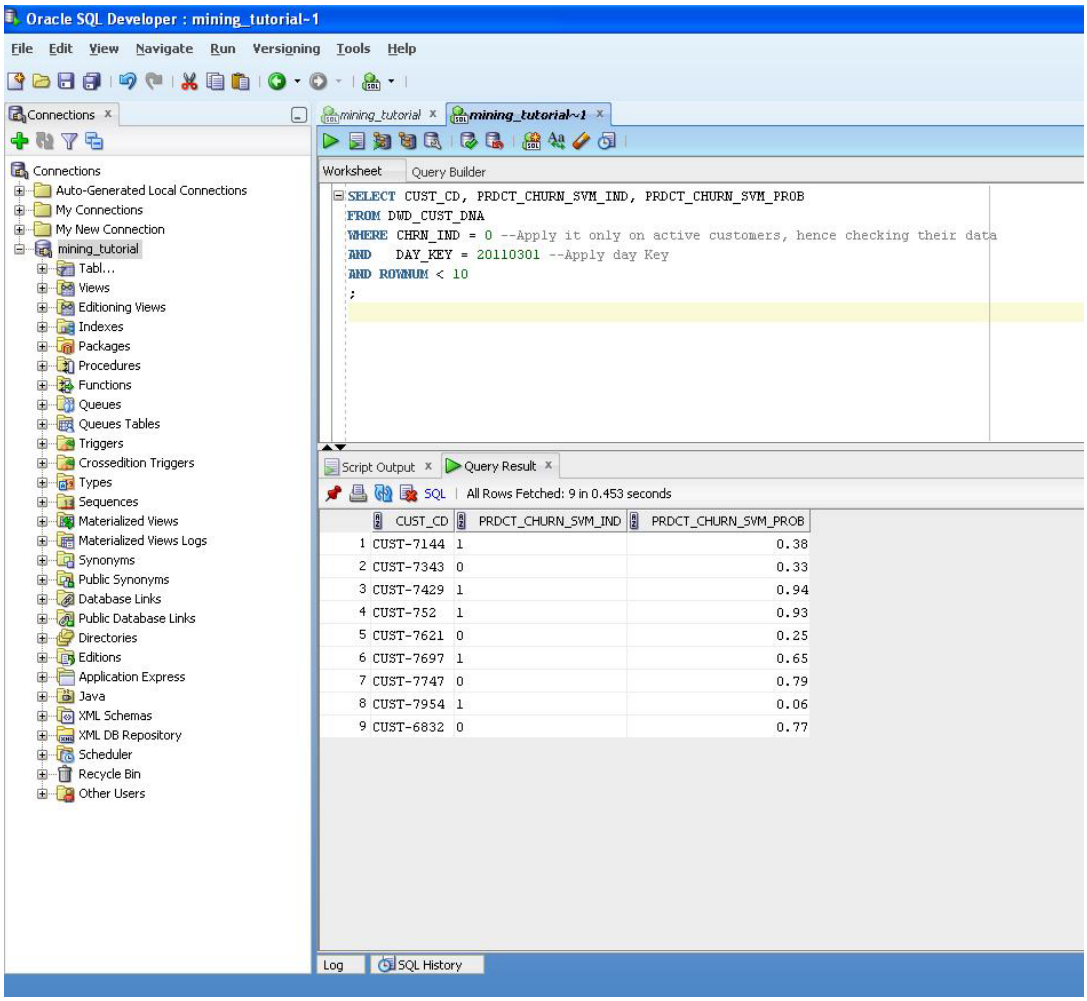
Note: This tutorial does not refresh all models. It only refreshes one prepaid churn prediction model. To refresh all of the default mining models based on latest customer data, follow the instructions in "Refreshing the Data in an Oracle Communications Data Model Warehouse" on page 4-17.

Checking the Result After you refresh, train, and generate the data mining model, check the `DWD_CUST_DNA` results table in `ocdm_sys` schema as follows:

1. Issue the following query.

```
SQL> SELECT CUST_CD, PRDCT_CHURN_SVM_IND, PRDCT_CHURN_SVM_PROB
FROM DWD_CUST_DNA
WHERE CHRN_IND = 0 --Apply it only on active customers, hence checking their
data
AND   DAY_KEY = 20110301 --Apply day Key
AND ROWNUM < 10
;
```

   This provides results:



For each customer identified by `CUST_CD`, the `PRDCT_CHRN_SVM_IND` column gives a Boolean prediction of whether a customer will churn in next three months. Zero (0) stands for non-churner, while one (1) stands for churner. The `PRDCT_CHURN_SVM_PROB` column provides a more detailed probability (0~1) that specifies how likely a customer is going to churn.

2. (Optional) If you have also installed the Oracle Communications Data Model sample reports in Oracle Business Intelligence Suite Enterprise Edition 11g, you can also view the results as an Oracle Business Intelligence Suite Enterprise Edition report.

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

Aggregate Tables in the Oracle Communications Data Model

Aggregate tables are tables that aggregate or "roll up" the data to one level higher than a base or derived table (and other functions can also be in the aggregate tables such as average, count, min, max, and others). The aggregate tables in the default Oracle Communications Data Model are actually materialized views and have a `DWA_` prefix.

These aggregate tables provide a view of the data similar to the view provided by a fact table in a snowflake schema.

The default Oracle Communications Data Model defines several aggregate tables. For example, the DWA_BER_FER_ERR_RATIO_MONTH table aggregates the values of the DWD_BER_FER_ERR_RATIO_DAY table to the month level.

> **See:** The "Aggregate Tables" topic in *Oracle Communications Data Model Reference* for a list of the aggregate tables in the default Oracle Communications Data Model.

If, during fit-gap analysis, you identified a need for simple aggregated data that is not provided by the default aggregate tables, you can define new materialized views. When designing these tables, keep the following points in mind:

- Create a query for the materialized view that aggregates up only a single level. For example, if aggregating over time, then aggregate only from day to month.

  > **Note:** When you must aggregate up many levels (for example in time, month, quarter, and year) or different hierarchies (for example, the fiscal and calendar hierarchies for a time dimension), do not define a DWA_ table; instead, define the aggregations by creating OLAP cubes.

  > **See also:** "Materialized Views in the Oracle Communications Data Model" on page 3-25 and "Defining New Oracle OLAP Cubes for Oracle Communications Data Model" on page 3-21.

- Name the tables following the conventions outlined in "General Naming Conventions for Physical Objects" on page 2-4 and use a DWA_ prefix.

Dimensional Components in the Oracle Communications 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 Communications Data Model. The foundation layer of the Oracle Communications Data Model is a 3NF model. The default Oracle Communications 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 Communications 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 Communications Data Model
- Defining New Oracle OLAP Cubes for Oracle Communications Data Model
- Changing an Oracle OLAP Cube in Oracle Communications Data Model

- Creating a Forecast Cube for Oracle Communications Data Model
- Choosing a Cube Partitioning Strategy for Oracle Communications Data Model
- Choosing a Cube Data Maintenance Method for Oracle Communications 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 also 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, totals and moving 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 to higher levels (rolling up). Drilling down on the Time dimension member 2012 typically navigates you to the quarters Q1 2012 through Q4 2012. In a calendar year hierarchy for 2012, drilling down on Q1 2012 would navigate you to the months, January 12 through March 12. These kinds of relationships make it easy for users to navigate through 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 3-16.
- **Oracle OLAP Cubes**. The physical model provided with Oracle Communications 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 3-18.

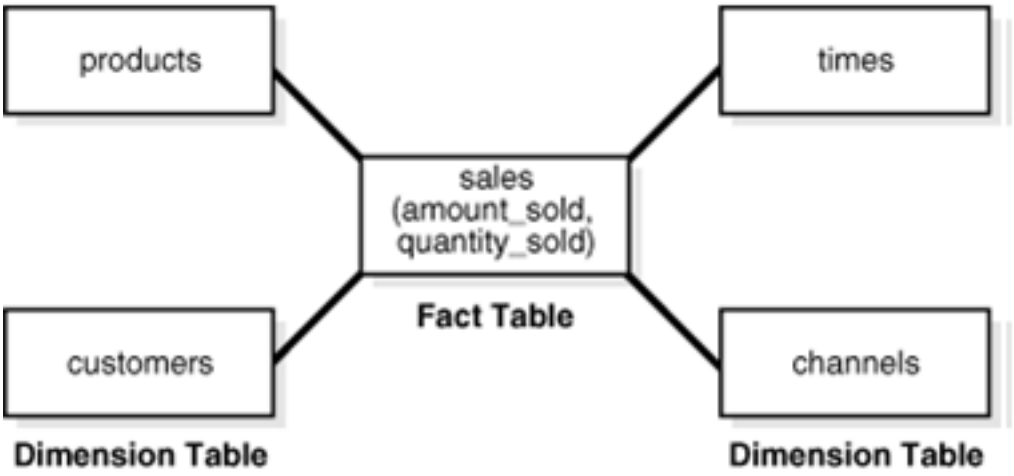
Characteristics of Relational Star and Snowflake Tables

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 is a relational schema whose design represents a multidimensional data model. The star schema consists of one or more fact tables and one or more dimension tables that are related through foreign keys. This allows drill paths, hierarchy and query profile to be 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.

Figure 3–1 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 3-17 and "Validating Relational Dimension Tables" on page 3-17.

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 3-19 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 3-20.

> **See also:** *Oracle OLAP User's Guide* and"Characteristics of the OLAP Cubes in Oracle Communications Data Model" on page 3-21.

Benefits of Using Oracle OLAP

The benefits of using Oracle OLAP are significant; Oracle OLAP offers the power of simplicity and provides: One database, standard administration and security, 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.

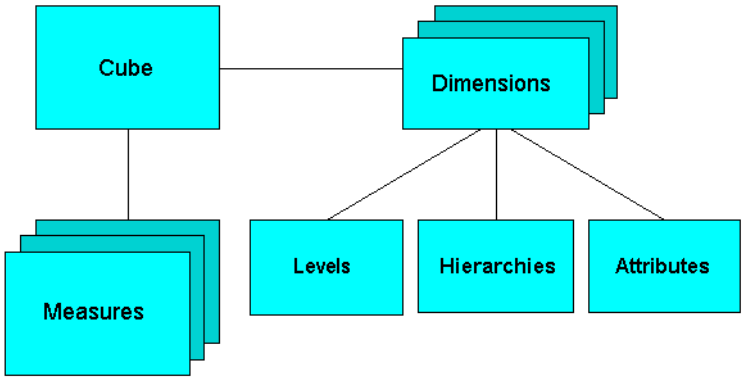
The OLAP engine manipulates dimensional objects in the same way that the SQL engine manipulates relational objects. However, because the OLAP engine is optimized to calculate analytic functions, and dimensional objects are optimized for analysis, analytic and row functions can be calculated much faster 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 3–2 shows the general relationships among the objects.

Figure 3–2 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 are materialized views. A cube that has been enhanced in this way is called a cube materialized view and has a `CB$` prefix. Cube materialized views can be incrementally refreshed through the Oracle Database materialized view subsystem, and they can 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 `ocdm_sys` schema.
2. From the cube list, select the cube which you want to enable.
3. In the right pane, select the **Materialized Views** tab.
4. Select **Enable Materialized View Refresh of the Cube**. then click **Apply**.

> **Note:** You cannot enable the cube materialized view for a forecast cube.

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 Communications Data Model

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

There are OLAP cubes defined in the default `ocdm_sys` schema. These cubes have the general characteristics described in "Characteristics of the OLAP Dimensional Model" on page 3-18. Specifically, OLAP cubes in the Oracle Communications Data Model have the following characteristics:

- All of the default OLAP cubes are loaded with data from `DWA_` tables that are materialized views.
- The cubes were defined and built using the Analytical Workspace Manager (AWM) client tool.
- A relational view (with a `_VIEW` suffix) is defined over each of the OLAP cubes.
- All of the OLAP cubes in the Oracle Communications Data Model are cube materialized views (that is, `CB$` objects).

Note: Immediately after installation, all materialized views underlying the OLAP cubes are disabled by default. To enable the cube materialized views, you must follow the steps outlined in "Adding Materialized View Capabilities" on page 3-20.

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

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

Defining New Oracle OLAP Cubes for Oracle Communications Data Model

You can add new OLAP cubes to the `ocdm_sys` schema. For consistency's sake, design and define these new cubes as described in "Characteristics of the OLAP Cubes in Oracle Communications Data Model" on page 3-21.

Take the following steps to define new cubes:

1. Ensure that there is an aggregate table (DWA_) to use as the "lowest leaf" data for the cube. See "Aggregate Tables in the Oracle Communications Data Model" on page 3-13 for information on creating new tables.
2. Use the AWM to define new Cubes for a customized version of Oracle Communications 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 3-18. 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 3-16, 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 sample reports included with Oracle Communications Data Model.

   See: *Oracle Communications Data Model Installation Guide* for more information on installing the sample reports and deploying the Oracle Communications 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 3-20.

   See also: *Oracle OLAP User's Guide*, "Defining New Oracle OLAP Cubes for Oracle Communications Data Model" on page 3-21, and the sample reports in *Oracle Communications 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 Communications Data Model

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

Since all Oracle Communications Data Model cubes load data from tables with the DWA_ prefix, to change the measures or dimensions of one cube, you must take the following steps:

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

Creating a Forecast Cube for Oracle Communications Data Model

To create a forecast cube for Oracle Communications Data Model:

1. Create a cube to contain the results of the forecast as described in "Defining New Oracle OLAP Cubes for Oracle Communications Data Model" on page 3-21.

   Note: You cannot enable materialized views for an Oracle Communications 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 Communications 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 like year to date needs 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 the Oracle Communications Data Model" on page 2-12, and "Partitioning and Materialized Views" on page 3-28.

Choosing a Cube Data Maintenance Method for Oracle Communications 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 want to 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 3-26

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 Communications 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 can 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 an existing materialized view can and should be used to satisfy a request.

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

Prefix	Description
`DWA_`	Aggregate tables which are materialized views. **See:** Aggregate tables in *Oracle Communications Data Model Reference* for a list of these objects in the default data model. "Aggregate Tables in the Oracle Communications Data Model" on page 3-13 for more information on customizing these objects,.

Prefix	Description
CB$	An OLAP cube enhanced with materialized view capabilities. **See:** OLAP cube materialized views in *Oracle Communications Data Model Reference* for a list of these objects in the default data model. "Characteristics of the OLAP Cubes in Oracle Communications Data Model" on page 3-21 for information on OLAP cubes. **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_	Materialized views created for performance reasons (that is, *not* an aggregate table or a cube materialized view). **See:** *Oracle Communications 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 Communications 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 Communications 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.

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 the Oracle Communications Data Model" on page 2-12

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 the Oracle Communications Data Model" on page 2-12, "Indexes and Partitioned Indexes in the Oracle Communications Data Model" on page 2-12, and "Choosing a Cube Partitioning Strategy for Oracle Communications Data Model" on page 3-23

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 the Oracle Communications Data Model" on page 2-8, and "Aggregate Tables in the Oracle Communications Data Model" on page 3-13.

4

ETL Implementation and Customization

This chapter discusses the ETL (extraction, transformation and loading) programs you use to populate an Oracle Communications Data Model warehouse. It includes the following topics:

- The Role of ETL in the Oracle Communications Data Model
- ETL for the Foundation Layer of an Oracle Communications Data Model Warehouse
- Customizing Intra-ETL for Oracle Communications Data Model
- Performing an Initial Load of an Oracle Communications Data Model Warehouse
- Refreshing the Data in an Oracle Communications Data Model Warehouse
- Managing Errors During Oracle Communications Data Model Intra-ETL Execution

The Role of ETL in the Oracle Communications Data Model

Figure 2–1, "Layers of an Oracle Communications Data Model Warehouse" on page 2-2 illustrated the three layers in Oracle Communications Data Model warehouse environment: the optional staging layer, the foundation layer, and the access layer. As illustrated by Figure 4–1, you use two types of ETL (extraction, transformation and loading) to populate these layers:

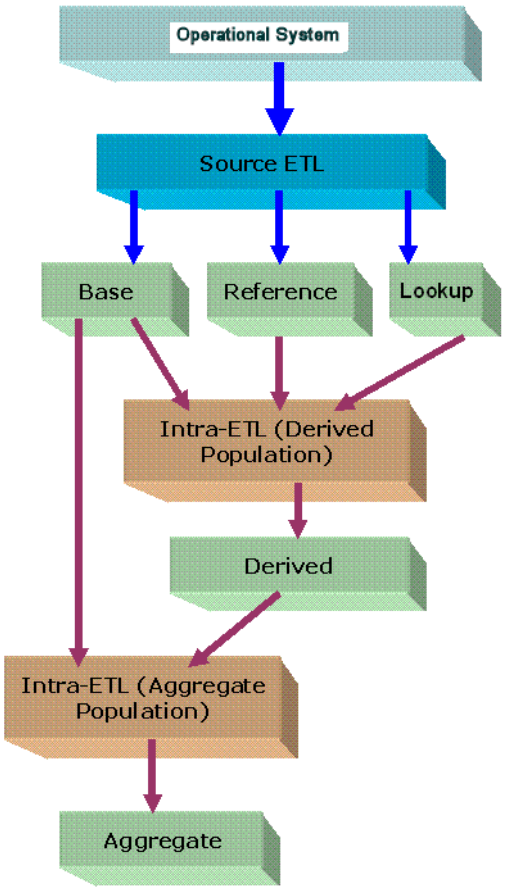
- **Source-ETL**. ETL that populates the staging layer (if any) and the foundation layer (that is, the base, reference, and lookup tables) with data from the operational system is known as source ETL.

  Oracle Communications Data Model does *not* include source-ETL scripts. Unless you are using an application adapter for Oracle Communications Data Model, you must create source-ETL yourself using your understanding of your operational and other source systems and your customized Oracle Communications Data Model. See "ETL for the Foundation Layer of an Oracle Communications Data Model Warehouse" on page 4-2 for more information on creating source-ETL.

- **Intra-ETL**. ETL that populates the access layer (that is, the derived tables, aggregate tables, materialized views, OLAP cubes, and data mining models) using the data in the foundation layer is known as intra-ETL.

  Oracle Communications 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 Oracle Communications Data Model" for more information on the intra-ETL.

Figure 4–1 ETL Flow Diagram



ETL for the Foundation Layer of an Oracle Communications Data Model Warehouse

ETL that populates the foundation layer of an Oracle Communications Data Model warehouse (that is, the base, reference, and lookup tables) with data from an operational system is known as source-ETL.

You can populate the foundation layer of an Oracle Communications Data Model warehouse in the following ways:

- If an application adapter for Oracle Communications Data Model is available for the system from which you want to populate the foundation layer of an Oracle Communications Data Model warehouse, you can use that adapter to populate the foundation layer. For more information, see "Using an Application Adapter to Populate the Foundation Layer" on page 4-2.
- Write your own source-ETL scripts using Oracle Data Integrator or another ETL tool and then use those scripts to populate the foundation layer. For more information, see "Writing Your Own Source-ETL" on page 4-3.

Using an Application Adapter to Populate the Foundation Layer

If an Application Adapter for Oracle Communications Data Model is available for the application that populates your Operational system, you use that adapter to populate the foundation layer of your Oracle Communications Data Model warehouse.

Writing Your Own Source-ETL

If you are not using an application adapter for Oracle Communications Data Model, you must write your own source-ETL scripts using Oracle Data Integrator or another ETL tool or mapping tool.

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

- Source-ETL Design Considerations
- ETL Architecture for Oracle Communications 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

> **See Also:** *Oracle® Fusion Middleware Developer's Guide for Oracle Data Integrator*

Source-ETL Design Considerations

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

- You can populate the calendar data by using the calendar population scripts provided with Oracle Communications Data Model and described in *Oracle Communications Data Model Reference*.
- 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 Communications Data Model Source-ETL

ETL (or EL-T, that is, Extract, Load and Transform) first extracts data from the original sources, assures the quality of the data, cleans the data, and makes the data consistent across the original sources. ETL then populates the physical objects with the "clean" data so that query tools, report writers, dashboards and so on can access the 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 architecture's in a data warehouse implementation. The difference between them is where the transformation proceed 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 OLTP 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. In such cases, the staging area can be used as a real-time Operational Data Store, at least for the source concerned, and aggregation could run directly from the Operational Data Store (operational system) to the Access layer, or to the presentation layer in specific cases.

When designing source-ETL for Oracle Communications 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 fields and target destination fields 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.

There are several potential data quality issues, related to each other, that the staging area needs to handle:

- Data Validity: Is the data content and type sufficient to be usable, and as expected (and "profile" in case one uses this advanced option)?
- Data Accuracy: correct addresses, correct with respect some "true" standard (or as such defined).
- Data Completeness: is all the required data there? What to do when data is missing? What represents the minimum set of required data?
- Data Consistency: that is, consistency of the data between the various sources and what rules one applies for inconsistencies.
- Data Latency: A sub-part of data consistency, but treated separately because of its importance: when does data arrive, over which period and in which one can we combine, which one not?
- Data Reasoning: This is more at reporting level but can be applied at the staging level: Does the data I see make sense from a business perspective? Can I really combine the data as an end-user would expect?

As a consequence, a multi-layer staging is generally required or expected.

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 data quality reporting and action program and people responsibility.

Set up the following processes and programs:

- Set up a data quality measurement process.
- Set up the 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 the necessary dependencies. General ETL tools, such as Oracle Warehouse Builder, support this kind of workflow, job design, and execution control.

Below are some tips 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.

As a general principle, all ETL logs status and errors into a table. You monitor execution status using an ETL tool or by querying this log table directly.

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. In order 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
- Partition Exchange Load for Oracle Communications Data Model Source-ETL

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 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 In order 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 in order 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 whole file is treated as a single granule. In this case, only one parallel server process can work on the entire file. In order 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 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.

You can 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 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. In order 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 Communications 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 Communications 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 - so 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.

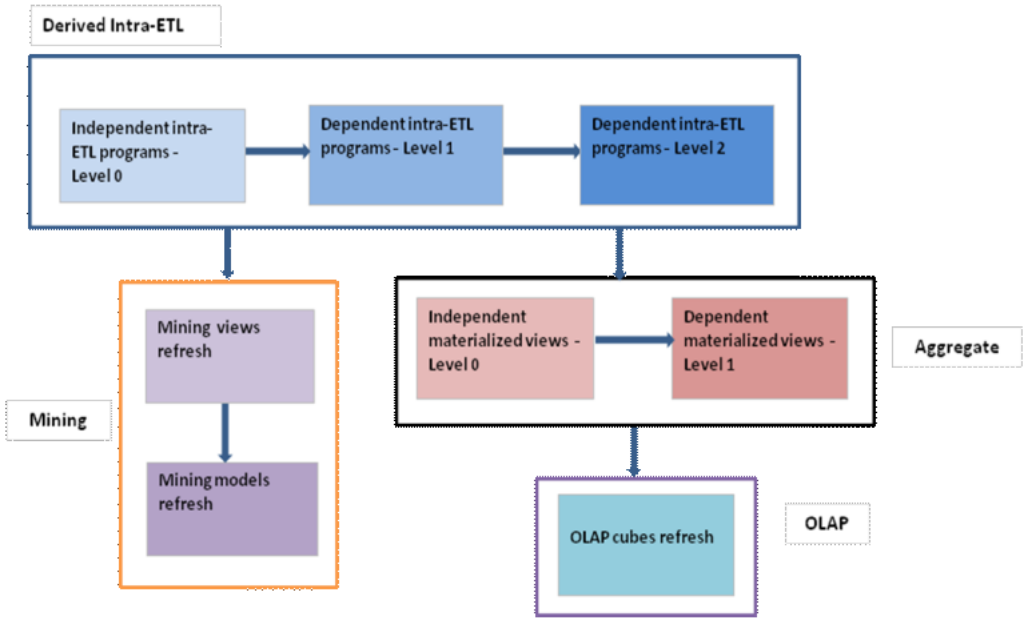
Customizing Intra-ETL for Oracle Communications Data Model

The Oracle Communications Data Model uses workflow implemented using PL/SQL packages to execute the intra-ETL process. The workflow consists of four major components:

1. Lookup Values
2. Executing Derived Intra-ETL Programs:
   a. Independent Derived intra-ETL programs - Level 0
   b. First level dependent Derived intra-ETL programs - Level 1
   c. Second level dependent Derived intra-ETL programs - Level 2
3. Refreshing Aggregate Materialized Views:
   a. Independent Aggregate materialized views - Level 0
   b. First level dependent Aggregate materialized views - Level 1
4. Refreshing Data mining models
5. Refreshing OLAP Cubes

Figure 4–2 illustrates the Oracle Communications Data Model intra-ETL workflow.

Figure 4–2 Oracle Communications Data Model Intra-ETL Workflow



Handling Lookup Values in Staging

Some Intra-ETLs expect some default values in order to work properly. They are usually associated with codes and stored as numbers but saved as text. For example, a typical status code (`STAT_CD`) is expected to start with 1 for pre-activated status, 2 with active status, 4 with deactivated status, and 5 with canceled status.

The advantage of defining numbers saved as text is that this allows the addition of custom codes that can be associated with an active state (for example 21, 2199, 21000001, and so on) that will be taken into account without having to change anything in the codes.

But of course, you will need to map the original codes to the text values. When you use an ETL lookup Matrix, as table, this allows the identifier of the source system, source table, the source column and the source code, and the Oracle Communications Data Model target table, column, and code.

You can define a simple function to search and leverage the correct code and map it to a number (stored as text to allow the use of SQL TEXT functions). You might want to add Oracle Communications Data Model Lookup tables with a `SHORT_NAME` column that would represent the original code of the source system, or the one that the end-users use, for reporting purposes only.

An alternative approach is to change the default value in the Intra-ETLs. Note that as soon as multiple sources map to the same Oracle Communications Data Model table, you will need some similar lookup code unification process.

Executing Derived Intra-ETL Programs

The first workflow component is the **Derived intra-ETL** programs. This component has three subcomponents that handle the dependency among Derived intra-ETL programs:

1. Independent Derived intra-ETL programs, the first subcomponent, has Derived intra-ETL programs that get data from foundation layer tables, that is base, lookup, and reference tables.
2. Derived intra-ETL programs, the second subcomponent, handles parts that depend on the first subcomponent, Independent Derived intra-ETL programs. The second subcomponent intra-ETL programs get data from foundation layer tables, that is base, lookup, and reference tables and also from derived tables that have intra-ETL programs in first subcomponent.
3. The third subcomponent has Derived intra-ETL programs that depend on the both the first (Independent Derived intra-ETL programs) and the second subcomponents (First level dependent Derived intra-ETL programs). The third subcomponent intra-ETL programs get data from foundation layer tables, that is base, lookup, and reference tables and also from derived tables that have intra-ETL programs in both first and second subcomponents.

Intra-ETL programs in all three subcomponents are implemented using PL/SQL packages. All Intra-ETL packages except two (`DWD_CNT_DAY_PKG` and `DWD_CUST_DNA_PKG`) insert data for the ETL period mentioned in `DWC_ETL_PARAMETER` table for *"OCDM-INTRA-ETL"* process. The process name for `DWD_CNT_DAY` table is "`DWD_CNT_DAY`" and for `DWD_CUST_DNA` table it is "`DWD_CUST_DNA`". Modify the ETL period of all three processes according to your data load requirements. If you are trying to load data for ETL period, for which data is already loaded, intra-ETL program first truncates the partitions existing for the ETL period, and then loads data into the target derived table.

Modifying existing or adding new intra-ETLs is a common customization of Oracle Communications Data Model. If new data marts are required or if some existing data warehouse requires modifications, it is usual to either create an Intra-ETL from scratch or to copy an existing Intra-ETL and modify it. In both cases, the new or modified intra-ETL program needs to be added in the Package `PKG_INTRA_ETL_PROCESS`, and the old one needs to be switched off or commented, at the correct level of dependency.

Be sure to create or modify the target entity as required.

Refreshing Aggregate Materialized Views

This is the second component of the workflow. This component depends on the first component, "Executing Derived Intra-ETL Programs" on page 4-10. The execution of this component happens only when the execution of the first component completes successfully. This component has two subcomponents to deal with the dependency among the Aggregate materialized views:

1. *Independent Aggregate materialized views*, the first subcomponent, has aggregate materialized views that do not depend on any other aggregate materialized views and most of them get data from derived tables and reference tables. Whereas few materialized views get data from foundation layer tables and derived tables.
2. *First level dependent Aggregate materialized views*, the second component, has aggregate materialized views that depend on the first subcomponent, *Independent Aggregate materialized views*. The aggregate materialized views in this subcomponent get data from aggregate materialized views in first the subcomponent

Modifications or additions in this layer follow the same principle as the ones in the derived layer.

Refreshing Data mining models

This is the third component of the workflow. This component depends on the first component, "Executing Derived Intra-ETL Programs" on page 4-10. The execution of this component happens only when the execution of the first component completes successfully. This component refreshes data mining models based on the *training day* and *apply day* specified in ETL parameter table, `DWC_ETL_PARAMETER` table for *BUILD-MINING-MODELS* process.

The creation of new mining models or the adaptation of existing mining models for a specific business need could be seen as typical "customization" (or configuration in case of existing models). One should follow the standard mining process as described in the specific documentation for the Advanced Analytics option of the database (because data mining is a process as such, before being automated as part of the customized Intra-ETL processes).

Additional models, once finalized, should be seen as normal personalization of Oracle Communications Data Model to one's business.

As your model changes over time any customized models need to be reviewed and fine tuned to conform and provide useful information with the new data available, as part of the standard reprocessing of the mining models on a regular basis. It is usual to re-run and fine tune any given mining model at least every quarter, to make sure the current mining model takes into account any new trends from available data.

The mining intra-ETLs should be customized to correspondingly consider any model data additions or changes.

Refreshing OLAP Cubes

This is the fourth component of the workflow. This component depends on the second component, "Refreshing Aggregate Materialized Views" on page 4-11, which in turn depends on the first component, "Executing Derived Intra-ETL Programs" on page 4-10. The execution of this component happens only when the execution of the second component completes successfully. This component refreshes data in OLAP cubes and dimensions based on the parameters given in `DWC_OLAP_ETL_PARAMETER` table.

Similarly to data mining and aggregate customization, when you add a cube or change an existing cube is part of typical customization of Oracle Communications Data Model, your refresh must follows the same process as a cube creation or modification as described in the *Oracle OLAP User's Guide*. The Intra-ETL that fills the cubes should be correspondingly modified.

Executing Intra-ETL Workflow

Oracle Communications Data Model intra-ETL workflow is implemented using a PL/SQL package, `PKG_INTRA_ETL_PROCESS`. Each component and their subcomponents of intra-ETL workflow have one procedure each. All these procedures are private to the package. The package has only one public procedure, which invokes all the private procedures as depicted in Figure 4–2. Before executing the workflow, ensure that you set all ETL parameters in `DWC_OLAP_PARAMETER` and `DWC_OLAP_ETL_PARAMETER` tables. Invoking `PKG_INTRA_ETL_PROCESS.RUN` procedure starts the workflow execution.

Performing an Initial Load of an Oracle Communications Data Model Warehouse

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

1. Load the foundation layer of the Oracle Communications Data Model warehouse (that is, the reference, lookup, and base tables) as described in "Performing an Initial Load of the Foundation Layer".
2. Load the access layer of the Oracle Communications Data Model warehouse (that is, the derived and aggregate tables, materialized views, OLAP cubes, and data mining models) as described in "Performing an Initial Load of the Access Layer".

Performing an Initial Load of the Foundation Layer

The manner in which you perform an initial load of the foundation layer of an Oracle Communications Data Model warehouse (that is, the reference, lookup, and base tables) varies depending on whether you are using an application adapter for Oracle Communications Data Model:

- If you are using an application adapter for Oracle Communications Data Model, then you use that adapter to load the foundation layer. For example, you can use the NCC Adapter for Oracle Communications Data Model to populate the foundation layer of an Oracle Communications Data Model warehouse with data from an Oracle Communications Network Charging and Control system.
- If you are not using an application adapter, then you perform the initial load of the foundation layer using source-ETL that you create. See "Writing Your Own Source-ETL" for more information on creating this ETL.

Performing an Initial Load of the Access Layer

To perform an initial load of access layer of the Oracle Communications Data Model warehouse (that is, the derived and aggregate tables, materialized views, OLAP cubes, and data mining models) take the following steps:

1. Update the parameters in `DWC_ETL_PARAMETER` control table in the `ocdm_sys` schema for different processes 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 Communications Data Model warehouse, specify the values shown in the following tables:

   For *OCDM-INTRA-ETL* process:

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

   For *DWD_CUST_DNA* process:

Columns	Value
`PROCESS_NAME`	`'DWD_CUST_DNA'`

Columns	Value
FROM_DATE_ETL	The beginning date of the ETL period.
TO_DATE_ETL	The ending date of the ETL period.

For *DWD_CNT_DAY* process:

Columns	Value
PROCESS_NAME	'DWD_CNT_DAY'
FROM_DATE_ETL	The beginning date of the ETL period.
TO_DATE_ETL	The ending date of the ETL period.

For *OCDM-DWA-MV-DATE* process:

Columns	Value
PROCESS_NAME	' OCDM-DWA-MV-DATE'
FROM_DATE_ETL	The beginning date of the ETL period.
TO_DATE_ETL	The ending date of the ETL period.

For *BUILD-MINING-MODELS* process:

Columns	Value
PROCESS_NAME	'BUILD-MINING-MODELS'
FROM_DATE_ETL	The beginning date of the ETL period.
TO_DATE_ETL	The ending date of the ETL period.

For more information on DWC_ETL_PARAMETER control table, see *Oracle Communications Data Model Reference*.

2. Update the Oracle Communications Data Model OLAP ETL parameters in DWC_OLAP_ETL_PARAMETER control table in the ocdm_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 Values of OLAP ETL Parameters in the DWC_OLAP_ETL_PARAMETER table for Initial Load

Column Name	Value
PROCESS_NAME	' OCDM-OLAP-ETL'
BUILD_METHOD	C which specifies a complete refresh which clears all dimension values before loading.
CUBENAME	One of the following values that specifies the cubes you want to build: ▪ ALL specifies a build of the cubes in the Oracle Communications Data Model analytic workspace. ▪ cubename[[| cubename]...] specifies one or more cubes to build.
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.

Table 4–1 (Cont.) Values of OLAP ETL Parameters in the DWC_OLAP_ETL_PARAMETER table for Initial Load

Column Name	Value
CALC_FCST	One of the following values depending on whether you want to calculate forecast cubes: ▪ Y specifies calculate forecast cubes. ▪ N specifies do not calculate forecast cubes.
NO_FCST_YRS	If the value for the CALC_FCST column is Y, specify a decimal value that specifies how many years forecast data you want to calculate; otherwise, specify NULL.
FCST_MTHD	If the value for the CALC_FCST column is Y, then specify AUTO; otherwise, specify NULL.
FCST_ST_YR	If the value for the CALC_FCST column is Y, then specify value specified as 'BY YYYY' which is the "start business year" of a historical period; otherwise, specify NULL.
FCST_END_YR	If the value for the CALC_FCST column is Y, then specify value specified as 'BY YYYY' which is the "end business year" of a historical period; otherwise, specify NULL.
OTHER1	Specify NULL.
OTHER2	Specify NULL.

3. Execute the intra-ETL as described in "Executing the Default Oracle Communications Data Model Intra-ETL" on page 4-15.

Executing the Default Oracle Communications Data Model Intra-ETL

The intra-ETL workflow is implemented using PL/SQL package, PKG_INTRA_ETL_PROCESS. This package has a public procedure, Run, and also has private procedures for executing derived intra-ETL programs, refreshing aggregate materialized views, refreshing data mining models, and refreshing OLAP cubes. The public procedure, Run, invokes all the private procedures.

Before executing intra-ETL workflow, update ETL parameters in DWC_ETL_PARAMETER and DWC_OLAP_ETL_PARAMETER tables. It is suggested to not use ocdm_sys user to update ETL parameter tables and executing intra-ETL workflow. Ask your DBA to create a user for performing these tasks using following commands:

```
CREATE USER ocdm_user IDENTIFIED BY ocdm_user;
GRANT CREATE SESSION TO ocdm_user;
GRANT ALTER SESSION TO ocdm_user;

GRANT EXECUTE ON ocdm_sys.PKG_INTRA_ETL_PROCESS TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.PKG_INTRA_ETL_UTIL TO ocdm_user;

GRANT SELECT,UPDATE ON ocdm_sys.DWC_ETL_PARAMETER TO ocdm_user;
GRANT SELECT ON ocdm_sys.DWC_INTRA_ETL_ACTIVITY TO ocdm_user;
GRANT SELECT ON ocdm_sys.DWC_INTRA_ETL_PROCESS TO ocdm_user;
GRANT SELECT,UPDATE ON ocdm_sys.DWC_OLAP_ETL_PARAMETER TO ocdm_user;
GRANT SELECT ON ocdm_sys.DWC_OLAP_ACTIVITY TO ocdm_user;
GRANT SELECT ON ocdm_sys.DWC_MESSAGE TO ocdm_user;
```

Use ocdm_user user to update ETL parameter tables and execute intra-ETL workflow. In a SQLPLUS session, connect to ocdm_user user:

```
sqlplus ocdm_user/ocdm_user@SID
```

Update ETL parameter tables:

```
SQL> UPDATE DWC_ETL_PARAMETER
SET from_date_etl = < The beginning date of the ETL period >,
    to_date_etl   = < The ending date of the ETL period >
```

```
WHERE process_name = 'OCDM-INTRA-ETL'
;
/
SQL> commit;

SQL> UPDATE DWC_ETL_PARAMETER
SET from_date_etl = < The beginning date of the ETL period >,
    to_date_etl   = < The ending date of the ETL period >
WHERE process_name = 'DWD_CUST_DNA'
;
/
SQL> commit;

SQL> UPDATE DWC_ETL_PARAMETER
SET from_date_etl = < The beginning date of the ETL period >,
    to_date_etl   = < The ending date of the ETL period >
WHERE process_name = 'DWD_CNT_DAY'
;
/
SQL> commit;

SQL> UPDATE DWC_ETL_PARAMETER
SET from_date_etl = < The beginning date of the ETL period >,
    to_date_etl   = < The ending date of the ETL period >
WHERE process_name = 'OCDM-DWA-MV-DATE'
;
/
SQL> commit;

SQL> UPDATE DWC_ETL_PARAMETER
SET from_date_etl = < The beginning date of the ETL period >,
    to_date_etl   = < The ending date of the ETL period >
WHERE process_name = 'BUILD-MINING-MODELS'
;
/
SQL> commit;

SQL> UPDATE DWC_OLAP_ETL_PARAMETER
SET build_method = <>,
    cubename   = <>,
  .
  .
  .
  .
fcst_st_yr = <>,
fcst_end_yr = <>
;
/
SQL> commit;
```

Run the following command to execute intra-ETL workflow:

```
SQL> BEGIN
OCDM_SYS.PKG_INTRA_ETL_PROCESS.Run;
END;
/
```

The status of each activity is tracked using DWC_INTRA_ETL_ACTIVITY table. The status of each cube data loading is tracked using DWC_OLAP_ACTIVITY table. The status of the entire intra-ETL workflow process is tracked using DWC_INTRA_ETL_PROCESS table. See

"Monitoring the Execution of the Intra-ETL Process" for more information on these tables.

Refreshing the Data in an Oracle Communications Data Model Warehouse

The section, "Performing an Initial Load of the Access Layer" describes how to perform an initial load of an Oracle Communications Data Model data warehouse. After this initial load, you must load new data into your Oracle Communications Data Model data warehouse regularly so that it can serve its purpose of facilitating business analysis.

To load new data into your Oracle Communications 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 Oracle Communications Data Model warehouse, or you can refresh the data sequentially, as follows:

1. Refreshing the Foundation Layer of Oracle Communications Data Model Warehouse
2. Refreshing the Access Layer of an Oracle Communications Data Model Warehouse

In either case, you can manage errors during the execution of the intra-ETL as described in .

Refreshing the Foundation Layer of Oracle Communications Data Model Warehouse

You can refresh the foundation layer of an Oracle Communications Data Model warehouse (that is, the reference, lookup, and base tables) in the following ways:

- If an application adapter for Oracle Communications Data Model is available for the system from which you want to refresh the foundation layer of an Oracle Communications Data Model warehouse, you can use that adapter to refresh the foundation layer.
- You can refresh the foundation layer using source-ETL scripts that you wrote using Oracle Warehouse Builder or another ETL tool. For more information on creating source-ETL, see "Writing Your Own Source-ETL".

Refreshing the Access Layer of an Oracle Communications Data Model Warehouse

Refreshing the access layer of an Oracle Communications Data Model is a multi-step process. You can do a full incremental load of the access layer all at one time, or you can refresh the data sequentially, as follows:

- Refreshing Oracle Communications Data Model Derived Tables
- Refreshing Oracle Communications Data Model Aggregate Materialized Views
- Refreshing Oracle Communications Data Model OLAP Cubes
- Refreshing Oracle Communications 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 Communications Data Model Intra-ETL Execution".

To accomplish incremental loading of Oracle Communications Data Model data warehouse, ask your DBA to grant execute privilege on Derived intra-ETL, OLAP ETL, and Mining PL/SQL packages:

```
GRANT EXECUTE ON ocdm_sys.DWD_ACCT_BAL_MO_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_ACCT_DEBT_MO_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_ACCT_FRST_ACTVTY_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_ACCT_LAST_ACTVTY_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_ACCT_PYMT_DAY_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_ACCT_PMT_MTD_STAT_HST_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_AGRMNT_CHNG_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_AGRMNT_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_CANBLZTN_DTL_DAY_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_CMPGN_HIST_DAY_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_CNT_DAY_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_CNTCT_CNTR_DAY_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_CUST_EQPMNT_INSLTN_DAY_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_CUST_ORDR_DAY_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_CUST_ORDR_LN_ITEM_DAY_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_CUST_RFMP_SCR_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_CUST_SKU_SL_RETRN_DAY_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_DATA_USG_DAY_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_GIVE_AWAY_ITEM_DAY_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_INV_ADJ_ITEM_DAY_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_INV_POSN_ITEM_DAY_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_INV_RCPT_ITEM_DAY_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_INV_UNAVL_ITEM_DAY_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_INV_XFER_ITEM_DAY_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_INVC_AGNG_DAY_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_INVC_DAY_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_LYLTY_MBR_PNT_DAY_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_NBR_PRT_DAY_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_POS_TNDR_FLOW_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_PRCS_INVC_DAY_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_RTL_SL_RETRN_ITEM_DAY_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_SPLMNTR_SRVC_USG_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_SRVC_PRBLM_DAY_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_STORE_EFFNCY_DAY_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_VAS_SBRP_QCK_SUMM_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_VAS_USG_DAY_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_VOI_CALL_DAY_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_AGRMNT_RVN_DAY_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_PRPD_ACCT_STTSTC_DAY_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_RVN_DAY_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.DWD_CUST_DNA_PKG TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.PKG_MINING_ETL TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.PKG_OCDM_MINING TO ocdm_user;
GRANT EXECUTE ON ocdm_sys.PKG_OCDM_OLAP_ETL_AW_LOAD TO ocdm_user;
```

Refreshing Oracle Communications Data Model Derived Tables

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

1. Refresh the foundation layer of the Oracle Communications Data Model warehouse (that is, the reference, lookup, and base tables) with operational system data by executing the source-ETL that you have written.
2. Update the parameters of the DWC_ETL_PARAMETER control table for three processes('OCDM-INTRA-ETL', 'DWD_CUST_DNA', 'DWD_CNT_DAY'). Please refer to "Performing an Initial Load of an Oracle Communications Data Model Warehouse" for more information on the DWC_ETL_PARAMETER table. For an incremental load of an Oracle Communications Data Model warehouse, specify the values shown in the following table (that is, the beginning and end date of the ETL period) for all three processes

Columns	Value
FROM_DATE_ETL	The beginning date of the ETL period.
TO_DATE_ETL	The ending date of the ETL period.

For more information on DWC_ETL_PARAMETER control table, see *Oracle Communications Data Model Reference*.

3. Create a session by connecting ocdm_user user through SQLPLUS. Then, start an intra-ETL process. Make sure the previous process ended with 'COMPLETED-SUCCESS' status before starting a new process:

```
sqlplus ocdm_user/ocdm_user@SID

SQL> DECLARE
  l_process_type  OCDM_SYS.DWC_INTRA_ETL_PROCESS.PROCESS_TYPE%TYPE;
  l_error_text    OCDM_SYS.DWC_MESSAGE.MESSAGE_TEXT%TYPE;
  l_process_no    NUMBER;
BEGIN
  l_process_no := OCDM_SYS.PKG_INTRA_ETL_UTIL.Start_Process(l_process_type,l_
error_text);
END;
/
```

4. Refresh Oracle Communications Data Model derived tables by executing following commands:

```
SQL> DECLARE
  p_process_no    NUMBER;
  l_status        VARCHAR2(20);
BEGIN
  l_status := OCDM_SYS.DWD_ACCT_BAL_MO_PKG.Load('DWD_ACCT_BAL_MO',p_process_no);
  l_status := OCDM_SYS.DWD_ACCT_DEBT_MO_PKG.Load('DWD_ACCT_DEBT_MO',p_process_no);
  l_status := OCDM_SYS.DWD_ACCT_FRST_ACTVTY_PKG.Load('DWD_ACCT_FRST_ACTVTY',p_process_no);
  l_status := OCDM_SYS.DWD_ACCT_LAST_ACTVTY_PKG.Load('DWD_ACCT_LAST_ACTVTY',p_process_no);
  l_status := OCDM_SYS.DWD_ACCT_PYMT_DAY_PKG.Load('DWD_ACCT_PYMT_DAY',p_process_no);
  l_status := OCDM_SYS.DWD_ACCT_PMT_MTD_STAT_HST_PKG.Load('DWD_ACCT_PYMT_MTHD_STAT_HIST',p_process_
no);
  l_status := OCDM_SYS.DWD_AGRMNT_CHNG_PKG.Load('DWD_AGRMNT_CHNG',p_process_no);
  l_status := OCDM_SYS.DWD_AGRMNT_PKG.Load('DWD_AGRMNT',p_process_no);
  l_status := OCDM_SYS.DWD_CANBLZTN_DTL_DAY_PKG.Load('DWD_CANBLZTN_DTL_DAY',p_process_no);
  l_status := OCDM_SYS.DWD_CMPGN_HIST_DAY_PKG.Load('DWD_CMPGN_HIST_DAY',p_process_no);
  l_status := OCDM_SYS.DWD_CNT_DAY_PKG.Load('DWD_CNT_DAY',p_process_no);
  l_status := OCDM_SYS.DWD_CNTCT_CNTR_DAY_PKG.Load('DWD_CNTCT_CNTR_DAY',p_process_no);
  l_status := OCDM_SYS.DWD_CUST_EQPMNT_INSLTN_DAY_PKG.Load('DWD_CUST_EQPMNT_INSTLTN_DAY',p_process_
no);
  l_status := OCDM_SYS.DWD_CUST_ORDR_DAY_PKG.Load('DWD_CUST_ORDR_DAY',p_process_no);
```

```
  l_status := OCDM_SYS.DWD_CUST_ORDR_LN_ITEM_DAY_PKG.Load('DWD_CUST_ORDR_LN_ITEM_DAY',p_process_
no);
  l_status := OCDM_SYS.DWD_CUST_RFMP_SCR_PKG.Load('DWD_CUST_RFMP_SCR',p_process_no);
  l_status := OCDM_SYS.DWD_CUST_SKU_SL_RETRN_DAY_PKG.Load('DWD_CUST_SKU_SL_RETRN_DAY',p_process_
no);
  l_status := OCDM_SYS.DWD_DATA_USG_DAY_PKG.Load('DWD_DATA_USG_DAY',p_process_no);
  l_status := OCDM_SYS.DWD_GIVE_AWAY_ITEM_DAY_PKG.Load('DWD_GIVE_AWAY_ITEM_DAY',p_process_no);
  l_status := OCDM_SYS.DWD_INV_ADJ_ITEM_DAY_PKG.Load('DWD_INV_ADJ_ITEM_DAY',p_process_no);
  l_status := OCDM_SYS.DWD_INV_POSN_ITEM_DAY_PKG.Load('DWD_INV_POSN_ITEM_DAY',p_process_no);
  l_status := OCDM_SYS.DWD_INV_RCPT_ITEM_DAY_PKG.Load('DWD_INV_RCPT_ITEM_DAY',p_process_no);
  l_status := OCDM_SYS.DWD_INV_UNAVL_ITEM_DAY_PKG.Load('DWD_INV_UNAVL_ITEM_DAY',p_process_no);
  l_status := OCDM_SYS.DWD_INV_XFER_ITEM_DAY_PKG.Load('DWD_INV_XFER_ITEM_DAY',p_process_no);
  l_status := OCDM_SYS.DWD_INVC_AGNG_DAY_PKG.Load('DWD_INVC_AGNG_DAY',p_process_no);
  l_status := OCDM_SYS.DWD_INVC_DAY_PKG.Load('DWD_INVC_DAY',p_process_no);
  l_status := OCDM_SYS.DWD_LYLTY_MBR_PNT_DAY_PKG.Load('DWD_LYLTY_MBR_PNT_DAY',p_process_no);
  l_status := OCDM_SYS.DWD_NBR_PRT_DAY_PKG.Load('DWD_NBR_PRT_DAY',p_process_no);
  l_status := OCDM_SYS.DWD_POS_TNDR_FLOW_PKG.Load('DWD_POS_TNDR_FLOW',p_process_no);
  l_status := OCDM_SYS.DWD_PRCS_INVC_DAY_PKG.Load('DWD_PRCS_INVC_DAY',p_process_no);
  l_status := OCDM_SYS.DWD_RTL_SL_RETRN_ITEM_DAY_PKG.Load('DWD_RTL_SL_RETRN_ITEM_DAY',p_process_
no);
  l_status := OCDM_SYS.DWD_SPLMNTR_SRVC_USG_PKG.Load('DWD_SPLMNTR_SRVC_USG',p_process_no);
  l_status := OCDM_SYS.DWD_SRVC_PRBLM_DAY_PKG.Load('DWD_SRVC_PRBLM_DAY',p_process_no);
  l_status := OCDM_SYS.DWD_STORE_EFFNCY_DAY_PKG.Load('DWD_STORE_EFFNCY_DAY',p_process_no);
  l_status := OCDM_SYS.DWD_VAS_SBRP_QCK_SUMM_PKG.Load('DWD_VAS_SBRP_QCK_SUMM',p_process_no);
  l_status := OCDM_SYS.DWD_VAS_USG_DAY_PKG.Load('DWD_VAS_USG_DAY',p_process_no);
  l_status := OCDM_SYS.DWD_VOI_CALL_DAY_PKG.Load('DWD_VOI_CALL_DAY',p_process_no);
END;
/

SQL> DECLARE
  p_process_no    NUMBER;
  l_status        VARCHAR2(20);
BEGIN
  l_status := OCDM_SYS.DWD_AGRMNT_RVN_DAY_PKG.Load('DWD_AGRMNT_RVN_DAY',p_process_no);
  l_status := OCDM_SYS.DWD_PRPD_ACCT_STTSTC_DAY_PKG.Load('DWD_PRPD_ACCT_STTSTC_DAY',p_process_no);
  l_status := OCDM_SYS.DWD_RVN_DAY_PKG.Load('DWD_RVN_DAY',p_process_no);
END;
/

SQL> DECLARE
  p_process_no    NUMBER;
  l_status        VARCHAR2(20);
BEGIN
  l_status := OCDM_SYS.DWD_CUST_DNA_PKG.Load('DWD_CUST_DNA',p_process_no);
END;
/
```

Refreshing Oracle Communications Data Model Aggregate Materialized Views

Refreshing the Aggregate Materialized Views in an Oracle Communications Data Model is a multi-step process:

1. Refresh the foundation layer of the Oracle Communications Data Model warehouse (that is, the reference, lookup, and base tables) with operational system data by executing the source-ETL that you have written.
2. Refresh Oracle Communications Data Model derived tables as explained in Refreshing Oracle Communications Data Model Derived Tables.
3. Update the parameters of the `DWC_ETL_PARAMETER` control table for `OCDM-DWA-MV-DATE` process. Please refer to Performing an Initial Load of an Oracle

Communications Data Model Warehouse section to know how to update DWC_ETL_PARAMETER table. For an incremental load of an Oracle Communications Data Model warehouse, specify the values shown in the following table (that is, the beginning and end date of the ETL period) for OCDM-DWA-MV-DATE process.

Columns	Value
FROM_DATE_ETL	The beginning date of the ETL period.
TO_DATE_ETL	The ending date of the ETL period.

For more information on DWC_ETL_PARAMETER control table, see *Oracle Communications Data Model Reference*.

4. Create a session by connecting ocdm_user user through SQLPLUS. An intra-ETL process created in Refreshing Oracle Communications Data Model Derived Tables must be in 'RUNNING' status now:

```
sqlplus ocdm_user/ocdm_user@SID
```

5. Refresh Oracle Communications Data Model aggregate materialized views by executing following commands:

```
SQL> DECLARE
  p_process_no    NUMBER;
  l_status        VARCHAR2(20);
BEGIN
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_ACCT_DEBT_MO',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_ACCT_PYMT_MO',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_ACCT_STTSTC_MO',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_AGRMNT_ACCT_SBRP_PROD',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_ARPU_BASE_CUST_TYP',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_BER_FER_ERR_RATIO_MO',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_CALL_CNTR_CALL_MO',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_CALL_CNTR_CASE_MO',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_CELL_STTSTC_MO',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_CMISN_MO',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_CNT_MO',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_COST_CNTR_MO',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_CUST_ACQSTN_SUMM_MO',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_CUST_CHRN_MO',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_CUST_COST_MO',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_CUST_DEBT_COLLCTN_MO',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_CUST_EQPMNT_INSTLTN_MO',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_CUST_ORDR_MO',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_DATA_USG_MO',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_INVC_ADJ_MO',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_INVC_MO',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_INV_POSN_DEPT_DAY',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_INV_POSN_SBC_MO',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_IN_PLTFRM_MO',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_LYLTY_PROG_MO',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_MKT_SHARE',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_MSC_TRFC_MO',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_NBR_PRT_MO',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_NTWK_AVLBLTY_MO',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_NTWK_TCHPNT_MO',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_PRPD_ALWNCE_MO',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_PRTNR_STLMNT_MO',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_RDMPTN_MO',p_process_no);
```

```
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_RF_NTWK_CPCTY_MO',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_RVN_MO',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_SBSCBR_STTSTC_MO',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_SL_CMPGN_SUMM_MO',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_SPLMNTR_SRVC_USG_MO',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_STORE_EFFNCY_MO',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_VAS_SBRP_QCK_SUMM_MO',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_VAS_USG_MO',p_process_no);
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_VOI_CALL_MO',p_process_no);
END;
/


SQL> DECLARE
  p_process_no    NUMBER;
  l_status        VARCHAR2(20);
BEGIN
  l_status := OCDM_SYS.PKG_INTRA_ETL_UTIL.Refresh_MV('DWA_CUST_GROSS_ORDRS_QTR',p_process_no);
END;
/
```

Refreshing Oracle Communications 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 Communications Data Model data warehouse since the initial load of the OLAP cubes. Refreshing the OLAP Cubes in an Oracle Communications Data Model is a multi-step process:

1. Refresh the foundation layer of the Oracle Communications Data Model warehouse (that is, the reference, lookup, and base tables) with operational system data by executing the source-ETL that you have written.
2. Refresh Oracle Communications Data Model derived tables as explained in Refreshing Oracle Communications Data Model Derived Tables.
3. Refresh Oracle Communications Data Model aggregate materialized views as explained in Refreshing Oracle Communications Data Model Aggregate Materialized Views.
4. Update the parameters of the `DWC_OLAP_ETL_PARAMETER` control table. Please refer to Performing an Initial Load of an Oracle Communications Data Model Warehouse section to know how to update `DWC_OLAP_ETL_PARAMETER` table.

   For more information on `DWC_OLAP_ETL_PARAMETER` control table, see *Oracle Communications Data Model Reference*.
5. Create a session by connecting `ocdm_user` user through SQLPLUS. An intra-ETL process created in Refreshing Oracle Communications Data Model Derived Tables must be in `'RUNNING'` status now:

   ```
   sqlplus ocdm_user/ocdm_user@SID
   ```
6. Refresh Oracle Communications Data Model OLAP cubes by executing following commands:

   ```
   SQL> DECLARE
     l_build_methd OCDM_SYS.DWC_OLAP_ETL_PARAMETER.BUILD_METHOD%TYPE;
     l_cube_nm OCDM_SYS.DWC_OLAP_ETL_PARAMETER.CUBENAME%TYPE;
     l_maxjobques OCDM_SYS.DWC_OLAP_ETL_PARAMETER.MAXJOBQUEUES%TYPE;
     l_calc_fcst  OCDM_SYS.DWC_OLAP_ETL_PARAMETER.CALC_FCST%TYPE;
     l_no_fcst_yrs OCDM_SYS.DWC_OLAP_ETL_PARAMETER.NO_FCST_YRS%TYPE;
   ```

```
  l_fcst_mthd  OCDM_SYS.DWC_OLAP_ETL_PARAMETER.FCST_MTHD%TYPE;
  l_fcst_st_yr OCDM_SYS.DWC_OLAP_ETL_PARAMETER.FCST_ST_YR%TYPE;
  l_fcst_end_yr OCDM_SYS.DWC_OLAP_ETL_PARAMETER.FCST_END_YR%TYPE;
  l_status VARCHAR2(20);
BEGIN
  /***************  Fetching the values of the OLAP ETL parameters variable
used in this procedure  ****************/
      SELECT
         BUILD_METHOD l_build_methd,
        CUBENAME l_cube_nm,
        MAXJOBQUEUES l_maxjobques,
        CALC_FCST l_calc_fcst,
        NO_FCST_YRS l_no_fcst_yrs,
        FCST_MTHD l_fcst_mthd,
        FCST_ST_YR l_fcst_st_yr,
        FCST_END_YR l_fcst_end_yr
      INTO
        l_build_methd,
        l_cube_nm,
        l_maxjobques,
        l_calc_fcst,
        l_no_fcst_yrs,
        l_fcst_mthd,
        l_fcst_st_yr,
        l_fcst_end_yr
      FROM
      OCDM_SYS.DWC_OLAP_ETL_PARAMETER;
   l_status := OCDM_SYS.PKG_OCDM_OLAP_ETL_AW_LOAD.olap_etl_aw_build(l_build_
methd,l_cube_nm,l_maxjobques,l_calc_fcst,l_no_fcst_yrs,l_fcst_mthd,l_fcst_st_
yr,l_fcst_end_yr,null,null);
END;
/
```

7. If there is requirement to refresh only Oracle Communications Data Model OLAP cubes, the same can be achieved with step 6, but before that make sure an intra-ETL process is already running. If no intra-ETL process is running, start one:

```
sqlplus ocdm_user/ocdm_user@SID

SQL> DECLARE
  l_process_type  OCDM_SYS.DWC_INTRA_ETL_PROCESS.PROCESS_TYPE%TYPE;
  l_error_text    OCDM_SYS.DWC_MESSAGE.MESSAGE_TEXT%TYPE;
  l_process_no    NUMBER;
BEGIN
  l_process_no := OCDM_SYS.PKG_INTRA_ETL_UTIL.Start_Process(l_process_type,l_
error_text);
END;
/
```

Refreshing Oracle Communications Data Model Data Mining Models

Refreshing of data mining models is integrated into intra-ETL workflow. Data mining models get refreshed whenever intra-ETL workflow is executed. Data mining models trained using training data collected based on from_date_etl parameter and scored on apply data collected based on from_date_etl parameter in `DWC_ETL_PARAMETER` table for `BUILD-MINING-MODELS` process. You can also refresh all data mining models together or refresh each data mining model individually.

Refreshing the Data Mining Models in an Oracle Communications Data Model is a multi-step process:

1. Refresh the foundation layer of the Oracle Communications Data Model warehouse (that is, the reference, lookup, and base tables) with operational system data by executing the source-ETL that you have written.
2. Refresh Oracle Communications Data Model derived tables as explained in "Refreshing Oracle Communications Data Model Derived Tables".
3. Update the parameters of the DWC_ETL_PARAMETER control table for BUILD-MINING-MODELS process. See "Performing an Initial Load of an Oracle Communications Data Model Warehouse" for information on updating the DWC_ETL_PARAMETER table.

   For more information on DWC_ETL_PARAMETER control table, see *Oracle Communications Data Model Reference*.
4. Create a session by connecting ocdm_user user through SQLPLUS. An intra-ETL process created in "Refreshing Oracle Communications Data Model Derived Tables" must be in 'RUNNING' status now:

```
sqlplus ocdm_user/ocdm_user@SID
```

5. Refresh Oracle Communications Data Model data mining models by executing following commands:

```
SQL> DECLARE
  l_trnng_day       DATE;
  l_apply_day       DATE;
  l_trnng_day_key   NUMBER(30);
  l_apply_day_key   NUMBER(30);
  l_status          VARCHAR2(500);

BEGIN

  SELECT from_date_etl, to_date_etl  INTO l_trnng_day, l_apply_day
  FROM ocdm_sys.dwc_etl_parameter
  WHERE process_name = 'BUILD-MINING-MODELS';

  l_trnng_day_key := TO_CHAR(l_trnng_day,'YYYYMMDD');
  l_apply_day_key := TO_CHAR(l_apply_day,'YYYYMMDD');

    -- Create/refresh mining source views  OCDM_SYS.PKG_MINING_ETL.refresh_
mining_views(l_trnng_day_key,l_apply_day_key);

  -- Build mining models
    l_status := OCDM_SYS.PKG_OCDM_MINING.REFRESH_MODEL(l_apply_day_key,NULL);

END;
/
```

6. You can also refresh data mining models individually. To refresh Prepaid SVM Churn model, execute the following command (make sure you are connected as ocdm_user user):

```
SQL> exec ocdm_sys.pkg_ocdm_mining.create_prpd_churn_svm_model(training_day_
key);
```

Managing Errors During Oracle Communications 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

Three ocdm_sys schema control tables, DWC_INTRA_ETL_PROCESS, DWC_INTRA_ETL_ACTIVITY, DWC_OLAP_ACTIVITY monitor the execution of the intra-ETL process. These tables are documented in Oracle Communications Data Model Reference. You can access these three tables from ocdm_user user.

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'.
5. For OLAP cubes loading, a record is inserted into DWC_OLAP_ACTIVITY table with CUBENAME as cube name, status as 'RUNNING', and LOAD_START_DT as SYSDATE for each cube. It updates the record upon the completion of cube loading. It updates STATUS column to 'COMPLETED-SUCCESS' if cube loading is successful, otherwise 'COMPLETE-ERROR' and updates LOAD_END_DT column to SYSDATE. In case of 'COMPLETED-ERROR' cubes, it also updates ERROR_DTL column with error details.

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, DWC_INTRA_ETL_ACTIVITY, and DWC_OLAP_ACTIVITY tables. In DWC_INTRA_ETL_ACTIVITY table, see the records of currently running process. 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. Identify errors of OLAP cubes loading for individual cubes in DWC_OLAP_ACTIVITY table.
3. Correct the causes of the errors.
4. Re-invoke the intra-ETL process.

The intra-ETL workflow process identifies whether it is a normal run or recovery run by referring the DWC_INTRA_ETL_ACTIVITY table. During a recovery run, the intra-ETL workflow executes only the necessary programs. For example, for 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 refreshes aggregate materialized views 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.

5

Report and Query Customization

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

Reporting Approaches in Oracle Communications Data Model

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

Relational Reporting

With relational reporting, you create reports 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_ and DWA_ tables). However in some cases, you may need to use the base entities (that is, DWB_ tables) along with the reference tables to generate more detailed reports.

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_DAY and DWR_BSNS_MO, DWR_BSNS_QTR, DWR_BSNS_YR tables to query against a DAY level CDR Wireless entity such as DWD_VOI_CALL_DAY. On the other hand, you need to use the higher level snowflakes at Month level and above such as DWR_BSNS_MO, DWR_BSNS_QTR, DWR_BSNS_YR in order to query against the MONTH level CDR Wireless entity such as DWA_VOI_CALL_MO.

The lookup tables (that is tables, with the DWL_ prefix) represent the simpler dimensions comprising a single level containing a flat list of values. Typically, most reporting tools add a superficial top level to the dimension. These could be individual tables starting with DWL_ or views (also named DWL_) on DWL_CODE_MASTER that break out different code types into separate dimensions.

Note: The use of numbers as text in Lookup code allows you to group them by using only the first character of the lookup value code. This could provide an artificial hierarchy level.

OLAP Reporting

With OLAP reporting, you access Oracle OLAP cubes 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 entities stored in DWR_ tables, the OLAP dimension views contains information relating to the whole dimension including all the levels of the hierarchy logically partitioned on the basis of 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 11*g* 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 Communications Data Model reports. For examples of Oracle Communications Data Model reports, see:

- "Writing As Is and As Was Queries" on page 5-8
- "Tutorial: Creating a New Oracle Communications Data Model Dashboard" on page 5-13
- "Tutorial: Creating a New Oracle Communications Data Model Report" on page 5-17
- The sample reports provided with Oracle Communications Data Model that are documented in *Oracle Communications Data Model Reference*.

Customizing Oracle Communications Data Model Sample Reports

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

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

The sample 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.

You can use Oracle Business Intelligence Suite Enterprise Edition Answers and Dashboard presentation tools to customize the predefined sample dashboard reports:

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

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

Writing Your Own Queries and Reports

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

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

> **See also:** "Reporting Approaches in Oracle Communications Data Model" on page 5-1, "Oracle OLAP Cube Views" on page 3-19, and the discussion on querying dimensional objects in *Oracle OLAP User's Guide*.

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

For example, assume that you want to know the total call minutes for the top ten customers in the San Francisco area for March 2012. To answer this question, you might have to query the tables described in the following table.

Entity Name	Table Name	Description
WIRELESS CALL EVENT	`DWB_WRLS_CALL_EVT`	Occurrences of the wireless call.
CUSTOMER	`DWR_CUST`	Individual customers

Entity Name	Table Name	Description
ADDRESS LOCATION	DWR_ADDR_LOC	All addresses. The table has levels as country, state, city, address, and so on.
GEOGRAPHY CITY	DWR_GEO_CITY	The CITY level of the GEOGRAPHY hierarchy.

To perform this query, you execute the following SQL statement.

```
SELECT cust_key, tot_call_min FROM
(select round(sum(call.call_drtn)/60,2) tot_call_min , call.cust_key
from DWB_WRLS_CALL_EVT call,
DWR_CUST     cust,
DWR_ADDR_LOC addr,
DWR_GEO_CITY city
         Where to_date(to_char(call.evt_begin_dt,'MON-YY'),'MON-YY') like
to_date('MAR-12','MON-YY')
and cust.cust_key = call.cust_key
and cust.addr_loc_key = addr.addr_loc_key
and addr.geo_city_key = city.geo_city_key
and initcap(city.geo_city_name)='San Francisco'
group by call.cust_key
order by 1 desc) WHERE ROWNUM < 10;
```

The result of this query is shown below.

```
  CUST_KEY TOT_CALL_MIN
---------- ------------
      3390        101.6
      4304       100.25
      4269        97.37
      4152        93.02
      4230        92.97
      4157        92.95
      3345        91.62
      4115        48.43
      4111        44.48
```

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 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.

This 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 3–1, "Star Schema Diagram" on page 3-17 would be "What was the total number of umbrellas sold in Boston during the month of May 2008?"

1. The original query.

```
select SUM(quantity_sold) total_umbrellas_sold_in_Boston
From Sales s, Customers c, Products p, Times t
Where s.cust_id=cust_id
And s.prod_id = p.prod_id
And s.time_id=t.time_id
And c.cust_city='BOSTON'
And p.product='UMBRELLA'
And t.month='MAY'
And t.year=2012;
```

   As you can see all of the where clause predicates are on the dimension tables and the fact table (Sales) 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(quantity_sold
From Sales
Where cust_id IN
(select c.cust_id From Customers c Where c.cust_city='BOSTON')
And s.prod_id IN
(select p.prod_id From Products p Where  p.product='UMBRELLA')
And s.time_id IN
(select t.time_id From Times(Where t.month='MAY' And t.year=2012);
```

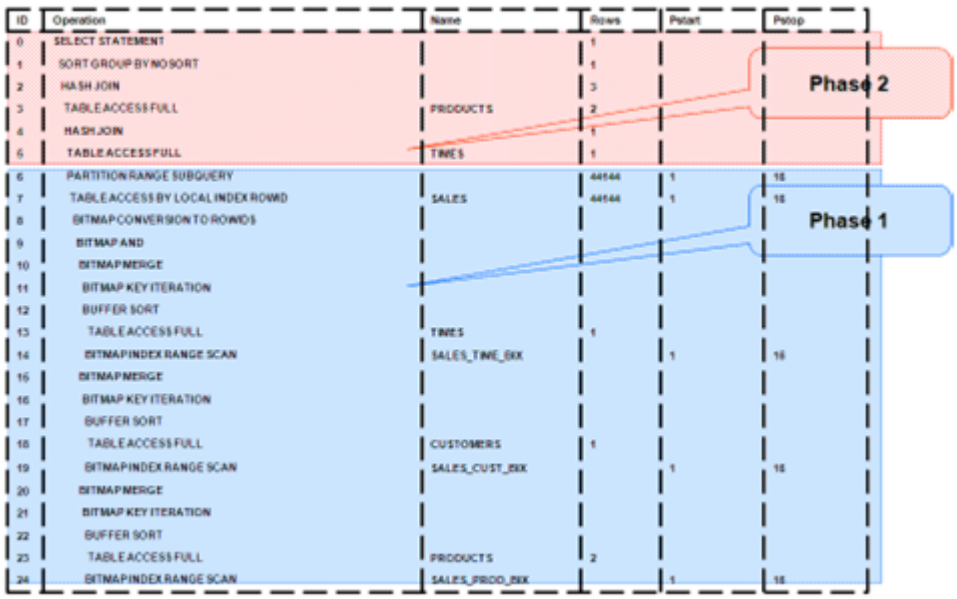
   By rewriting the query in this fashion you can now leverage the strengths of bitmap indexes. Bitmap indexes provide set based processing within the database, allowing you to use various fact methods for set operations such as AND, OR, MINUS, and COUNT. So, you use the bitmap index on time_id to identify 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 umbrellas and another is accessed for sales made in Boston. 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 following figure shows the typical execution plan for a star query when `STAR_TRANSFORMATION` has kicked in. The execution plan may not look exactly as you expected. There is no join back to the customer table after the rows have been successfully retrieved from the `Sales` table. If you look closely at the select list, you can see that there is not anything actually selected from the `Customers` 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 Communications 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 **ocdm**.

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

2. Open the Oracle Communications 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 an Oracle Communications Data Model Report"
- Example 5–4, "Troubleshooting a Report: A Table Does Not Exist"
- Example 5–5, "Troubleshooting a Report: When the Database is Not Connected"

Example 5–3 Troubleshooting an Oracle Communications Data Model Report

Assume the log file contains the following error.

```
Query Status: Query Failed: [nQSError: 15018] Incorrectly
defined logical table source (for fact table Customer Mining)
does not contain mapping for [Customer.Cell Phone Number,
Customer.Customer Segment Name, Customer.Party Name].
```

This error occurs when there is a problem in the Business layer in your Oracle Business Intelligence Suite Enterprise Edition repository.

In this case, you need to check the mapping for `Customer.Cell Phone Number`, `Customer.Customer Segment Name`, and `Customer.Party Name`.

Example 5–4 Troubleshooting a Report: A Table Does Not Exist

Assume the log file contains 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 has the table which actually does not exist in the Database.

To find out which table has problem:

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

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

Example 5–5 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 Against Oracle Communications Data Model

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 and 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 Against Oracle Communications Data Model

Based on the Data used for the examples on page 5-9, the following examples illustrate the characteristics of As Is and As Was queries:

- Example 5–6, "As Is Query for Tax Collection Split by Marital Status"
- Example 5–7, "As Was Queries for Tax Collection Split by Marital Status"
- Example 5–8, "As Is Query for Tax Collection Data Split by County"
- Example 5–9, "As Was Queries for Tax Collection Data Split by County"

Data used for the examples

Assume that your data warehouse has a `Customer` table, a `County`, and a `TaxPaid` fact table. As of January 1, 2012, these tables include the values shown below.

Customer Table

Cust Id	Cust Cd	Cust Nm	Gender	M Status	County Id	County Cd	Country Nm	...	Eff Frm	Eff To
101	JoD	John Doe	Male	Single	5001	SV	Sunnyvale	...	1-Jan-12	31-Dec-99
102	JaD	Jane Doe	Female	Single	5001	SV	Sunnyvale	...	1-Jan-12	31-Dec-99
103	JiD	Jim Doe	Male	Married	5002	CU	Cupertino	...	1-Jan-12	31-Dec-99

County Table

County Id	County CD	County Nm	Population	...	Eff Frm	Eff To
5001	SV	Sunnyvale	Very High	...	1-Jan-12	31-Dec-99
5002	CU	Cupertino	High	...	1-Jan-12	31-Dec-99

TaxPaid Table

Cust Id	Day	Tax Type	Tax
101	1-Jan-12	Professional Tax	100
102	1-Jan-12	Professional Tax	100
103	1-Jan-12	Professional Tax	100

Assume that the following events occurred in January 2012:

- On January 20, 2012, Jane Doe marries.
- On Jan 29, 2012, John Doe moves from Sunnyvale to Cupertino.

Consequently, as shown below, on February 1, 2012, the `Customer` and `TaxPaid` tables have new data while the values in the `County` table stay the same.

Customer table

Cust Id	Cust Cd	Cust Nm	Gender	M Status	County Id	County Cd	Country Nm	...	Eff Frm	Eff To
101	JoD	John Doe	Male	Single	5001	SV	Sunnyvale	...	1-Jan-12	29-Jan-12
102	JaD	Jane Doe	Female	Single	5001	SV	Sunnyvale	...	1-Jan-12	20-Jan-12
103	JiD	Jim Doe	Male	Married	5002	CU	Cupertino	...	1-Jan-12	31-Dec-99
104	JaD	Jane Doe	Female	Married	5001	SV	Sunnyvale	...	21-Jan-12	31-Dec-99

Cust Id	Cust Cd	Cust Nm	Gender	M Status	County Id	County Cd	Country Nm	...	Eff Frm	Eff To
105	JoD	John Doe	Male	Single	5002	CD	Cupertino	...	30-Jan-12	31-Dec-99

County table

County Id	County CD	County Nm	Population	...	Eff Frm	Eff To
5001	SV	Sunnyvale	Very High	...	1-Jan-12	31-Dec-99
5002	CU	Cupertino	High	...	1-Jan-12	31-Dec-99

TaxPaid Table

Cust Id	Day	Tax Type	Tax
101	1-Jan-12	Professional Tax	100
102	1-Jan-12	Professional Tax	100
103	1-Jan-12	Professional Tax	100
105	1-Feb-12	Professional Tax	100
104	1-Feb-12	Professional Tax	100
103	1-Feb-12	Professional Tax	100

Example 5–6 As Is Query for Tax Collection Split by Marital Status

Assuming the Data used for the examples on page 5-9, to show the tax collection data split by martial status, the following SQL statement that joins the `TaxPaid` fact table and the `Customer` dimension table on the `cust_id` surrogate key and the `Customer` and `County` snowflakes on the `cnty_id` surrogate key.

```
SELECT cust.cust_nm, cust.m_status, SUM(fct.tx)
FROM taxpaid fct, customer cust, county cnty
WHERE fct.cust_id = cust.cust_id
AND cust.cnty_id = cnt.cnt_id
GROUP BY cust.cust_nm, cust.m_status
ORDER BY 1,2,3;
```

The results of this query are shown below. Note that there are two rows for Jane Doe; one row for a marital status of Married and another for a marital status of Single.

Cust Nm	M Status	Tax
Jane Doe	Married	100
Jane Doe	Single	100
Jim Doe	Married	200
John Doe	Single	200

Example 5–7 As Was Queries for Tax Collection Split by Marital Status

Assuming the Data used for the examples on page 5-9, issue the following SQL statement to show the tax collection data split by marital status using an analysis date of January 15, 2012.

```
select
   cust.cust_nm, cust.m_status, sum(fct.tax)
```

```
from
   TaxPaid fct,
   (
      select
         cust_act.cust_id, cust_pit.cust_cd, cust_pit.cust_nm,
         cust_pit.m_status, cust_pit.gender,
         cust_pit.cnty_id, cust_pit.cnty_cd, cust_pit.cnty_nm
      from Customer cust_act
      inner join (
         select
            cust_id, cust_cd, cust_nm,
            m_status, gender,
            cnty_id, cnty_cd, cnty_nm
         from Customer cust_all
         where to_date('15-JAN-2012', 'DD-MON-YYYY') between eff_from and eff_to
      ) cust_pit
      on (cust_act.cust_cd = cust_pit.cust_cd)
   ) cust,
   (
      select
         cnty_act.cnty_id, cnty_pit.cnty_cd, cnty_pit.cnty_nm
      from County cnty_act
      inner join (
         select
            cnty_id, cnty_cd, cnty_nm
         from County cnty_all
         where to_date('15-JAN-2012', 'DD-MON-YYYY') between eff_from and eff_to
      ) cnty_pit
      on (cnty_act.cnty_cd = cnty_pit.cnty_cd)
   ) cnty
where fct.cust_id = cust.cust_id
and cust.cnty_id = cnty.cnty_id
GROUP BY cust.cust_nm, cust.m_status
order by 1,2,3;
```

The results of this query are shown below. Since Jane Doe was single on January 15, 2012 (the analysis date), all tax for Jane Doe is accounted under her Single status.

Cust Nm	M Status	Tax
Jane Doe	Single	200
Jim Doe	Married	200
John Doe	Single	200

Assume instead that you issued the exact same query except that for the `to_date` phrase you specify `09-FEB-2012` rather than `15-JAN-2012`. Since Jane Doe was married on February 9, 2012, then, as shown below all tax for Jane Doe would be accounted under her Married status.

Cust Nm	M Status	Tax
Jane Doe	Married	200
Jim Doe	Married	200
John Doe	Single	200

Example 5–8 As Is Query for Tax Collection Data Split by County

Assuming the Data used for the examples on page 5-9, issue the following SQL statement to show the tax collection data split by county.

```
SELECT cust.cust_nm, cnty.cnty_nm, SUM(fct.tax)
FROM TaxPaid fct, customer cust, county cnty
WHERE fct.cust_id = cust.cust_id
AND cust.cnty_id = cnty.cnty_ID
GROUP BY cut.cust_nm, cnty.cnty_nm
ORDER BY 1,2,3;
```

The results of this query are shown below. Note that since John Doe lived in two different counties, there are two rows of data for John Doe.

Cust Nm	County Nm	Tax
Jane Doe	Sunnyvale	200
Jim Doe	Cupertino	200
John Doe	Cupertino	100
John Doe	Sunnyvale	100

Example 5–9 As Was Queries for Tax Collection Data Split by County

Assuming the Data used for the examples on page 5-9, issue the following SQL statement to show the tax collection data split by county using an analysis date of January 15, 2012.

```
select
   cust.cust_nm, cnty.cnty_nm, sum(fct.tax)
from
  TaxPaid fct,
   (
      select
         cust_act.cust_id, cust_pit.cust_cd, cust_pit.cust_nm,
         cust_pit.m_status, cust_pit.gender,
         cust_pit.cnty_id, cust_pit.cnty_cd, cust_pit.cnty_nm
      from Customer cust_act
      inner join (
         select
            cust_id, cust_cd, cust_nm,
            m_status, gender,
            cnty_id, cnty_cd, cnty_nm
         from Customer cust_all
         where to_date('15-JAN-2012', 'DD-MON-YYYY') between eff_from and eff_to
      ) cust_pit
      on (cust_act.cust_cd = cust_pit.cust_cd
   ) cust,
   (
      select
         cnty_act.cnty_id, cnty_pit.cnty_cd, cnty_pit.cnty_nm
      from County cnty_act
      inner join (
         select
            cnty_id, cnty_cd, cnty_nm
         from County cnty_all
         where to_date('15-JAN-2012', 'DD-MON-YYYY') between eff_from and eff_to
      ) cnty_pit
      on (cnty_act.cnty_cd = cnty_pit.cnty_cd)
```

```
   ) cnty
where fct.cust_id = cust.cust_id
and cust.cnty_id = cnty.cnty_id
GROUP BY cust.cust_nm, cnty.cnty_nm
order by 1,2,3;
```

The results of this query are shown below. Note that since John Doe was in Sunnyvale as of the analysis date of January 15, 2012, all tax for John Doe is accounted for under the Sunnyvale county.

Cust Nm	County Nm	Tax
Jane Doe	Sunnyvale	200
Jim Doe	Cupertino	200
John Doe	Sunnyvale	200

Assume instead that you issued the exact same query except that for the `to_date` phrase you specify `09-FEB-2012` rather than `15-JAN-2012`. Since John Doe lived in Cupertino on February 9, 2012, then, as shown below all tax for John Doe would be accounted under Cupertino.

Cust Nm	County Nm	Tax
Jane Doe	Sunnyvale	200
Jim Doe	Cupertino	200
John Doe	Cupertino	200

Tutorial: Creating a New Oracle Communications Data Model Dashboard

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

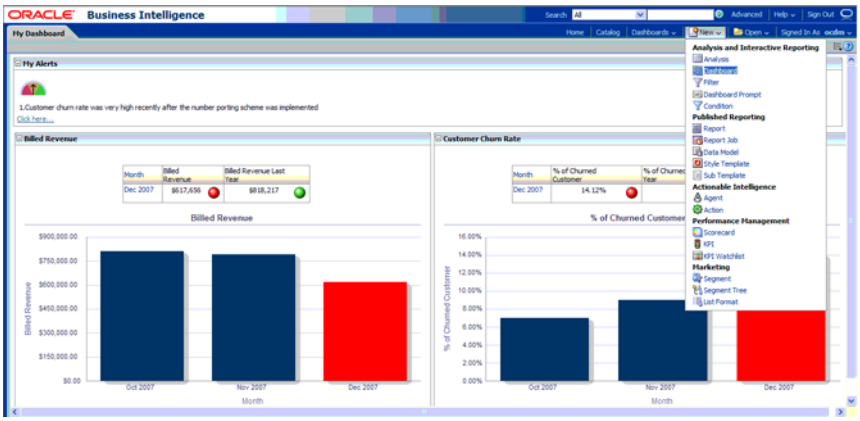
> **See:** *Oracle Communications Data Model Installation Guide* for more information on installing the sample reports and deploying the Oracle Communications Data Model RPD and webcat on the Business Intelligence Suite Enterprise Edition instance.

In this example assume that you want to create a dashboard named "Dropped call and Failed Call", and put both "Dropped Call Rate Report" and "Failed Call Rate Report" into this new dashboard.

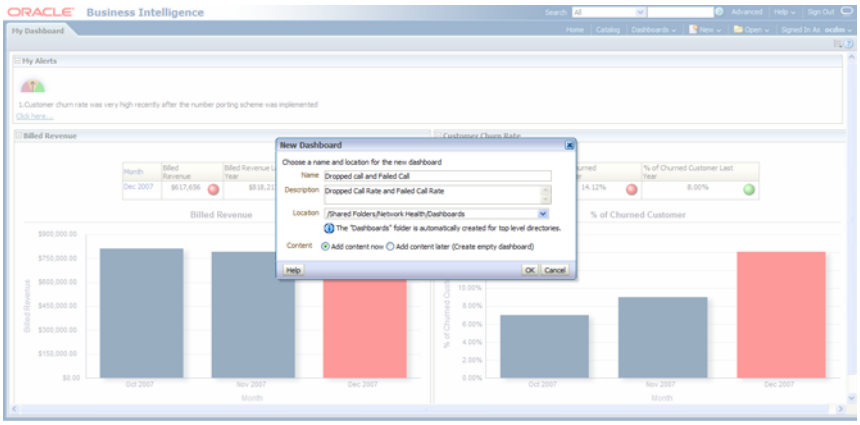
To create a dashboard, take the following steps:

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 `ocdm`, and provide the password.

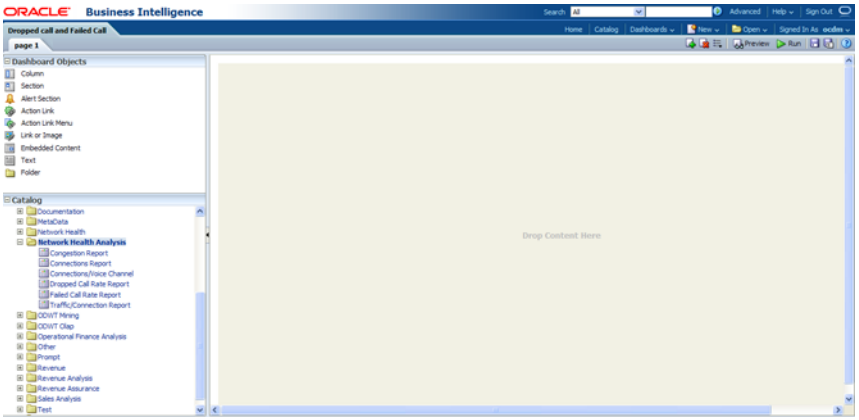
   Then, click **newDashboard** to create an Oracle Business Intelligence Suite Enterprise Edition dashboard.



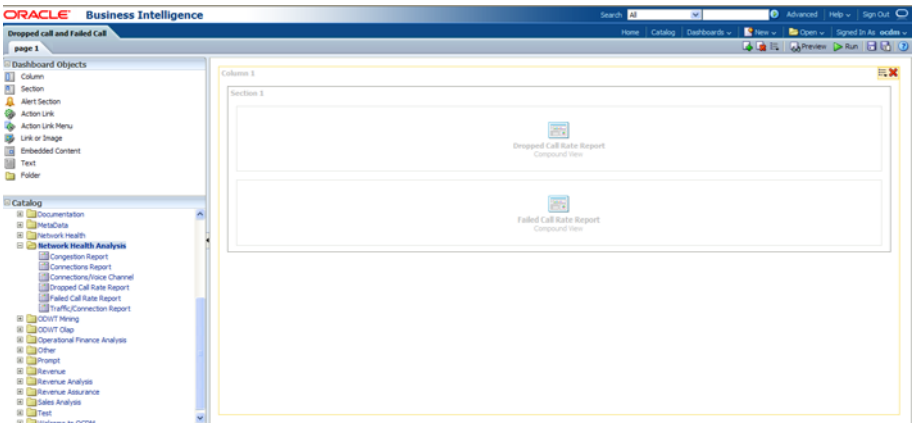
3. Input name and description, save it to the Network Health folder. Click **OK**.



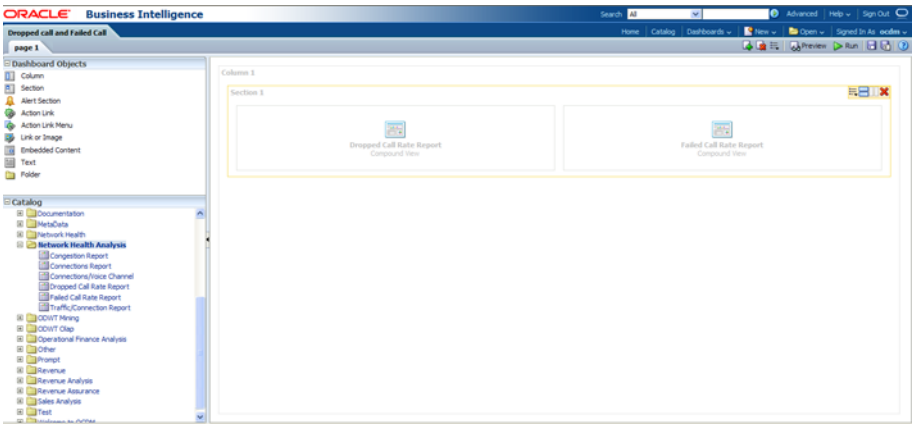
4. In the Catalog view, expand the Network Health Analysis folder. You can see the Failed Call Rate Report and Dropped Call Rate Report.



5. Drag the Failed Call Rate Report and Dropped Call Rate Report from the Catalog view into the right panel.



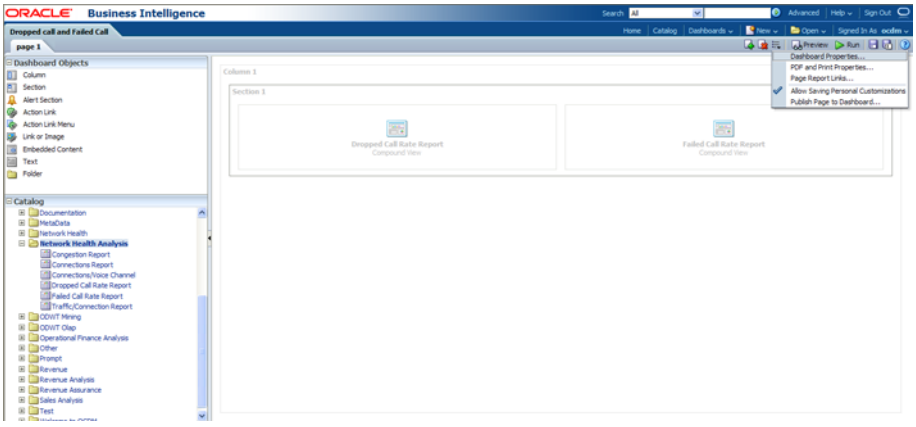
6. You can change the layout of this section to organize the two reports by horizontal or vertical.



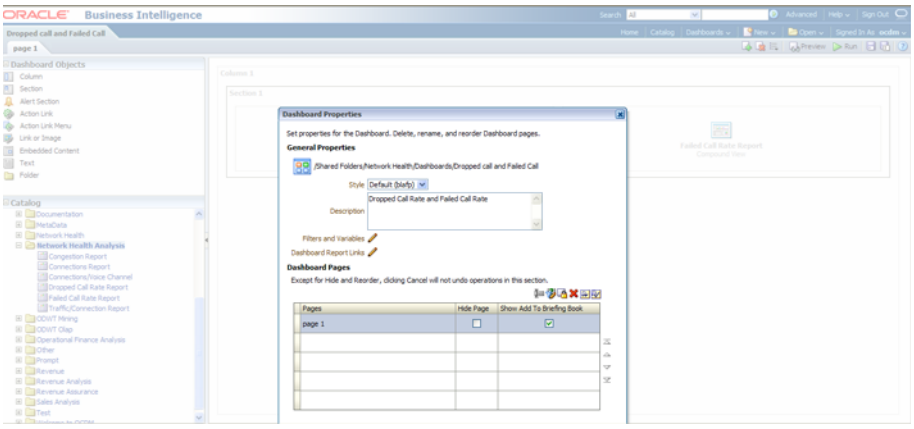
Note that the page name is still "Page1" so you must change it.

7. To change the page name:

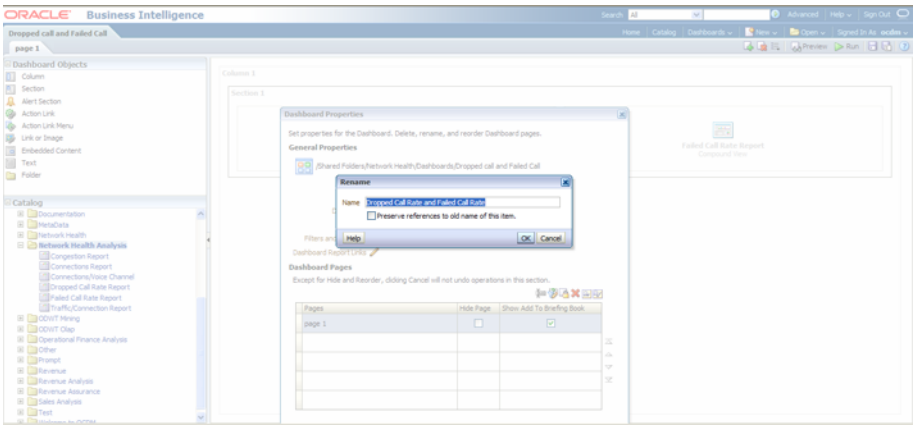
   a. Select the Dashboard.



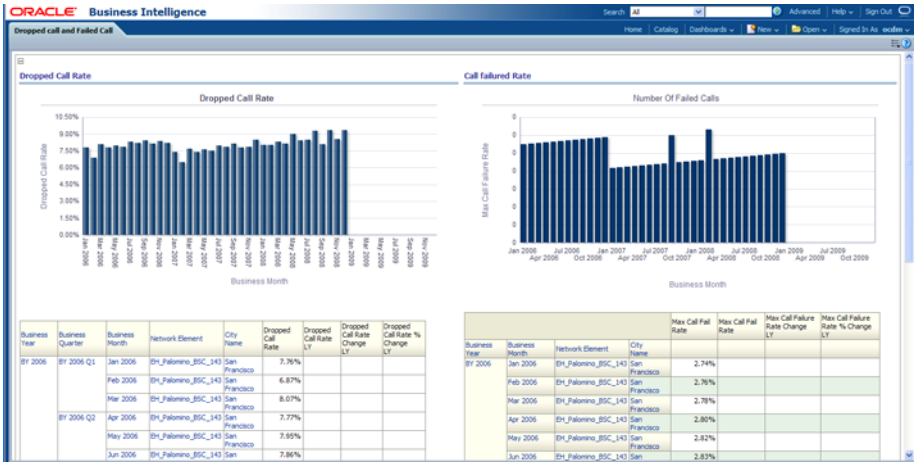
   b. In Dashboard Properties window, click **Change Name**.



c. Change the name to "Dropped Call Rate and Failed Call Rate", then click **OK.**



8. Click **Save** on the top of the dashboard. Now you have a new dashboard.



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 Communications Data Model Report

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

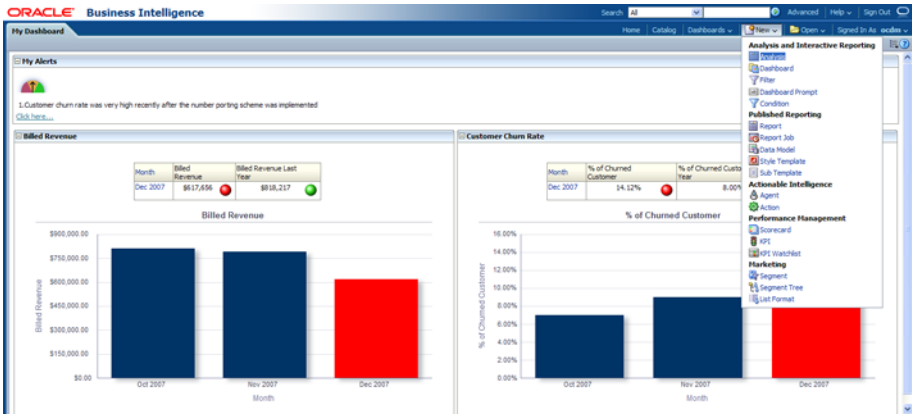
> **See:** *Oracle Communications Data Model Installation Guide* for more information on installing the sample reports and deploying the Oracle Communications Data Model RPD and webcat on the Business Intelligence Suite Enterprise Edition instance.

In this example, assume that you want to create a report named "Dropped call vs. Failed Call" to put both dropped call rate and failed call rate in one report.

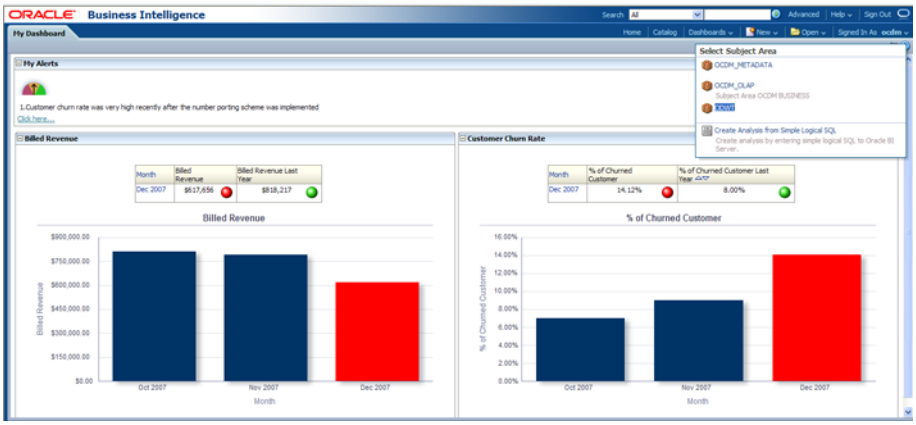
To create a this new report, take the following steps:

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 `ocdm`, and provide the password.

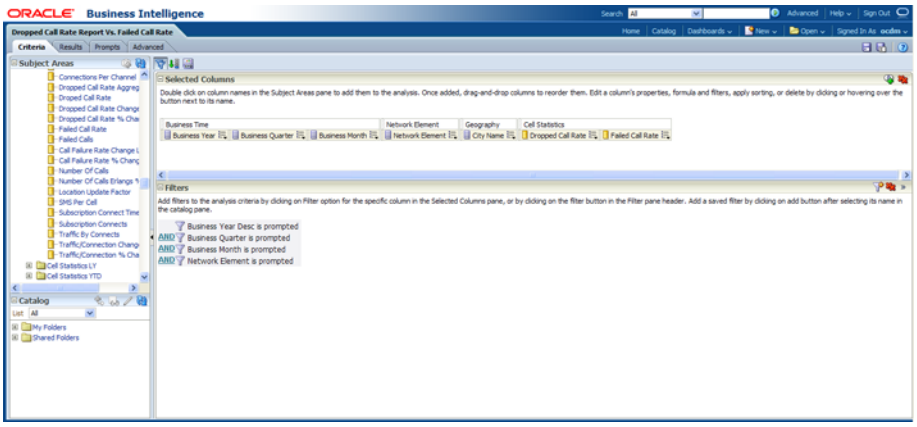
   Then, click **newAnalysis** to create an Oracle Business Intelligence Suite Enterprise Edition report.



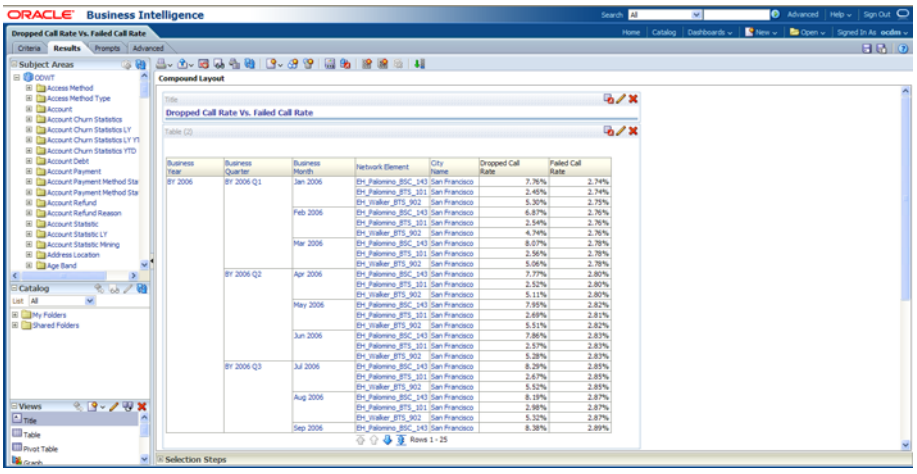
3. Select **Subject Area**, then select **ODWT** to create a relational report.



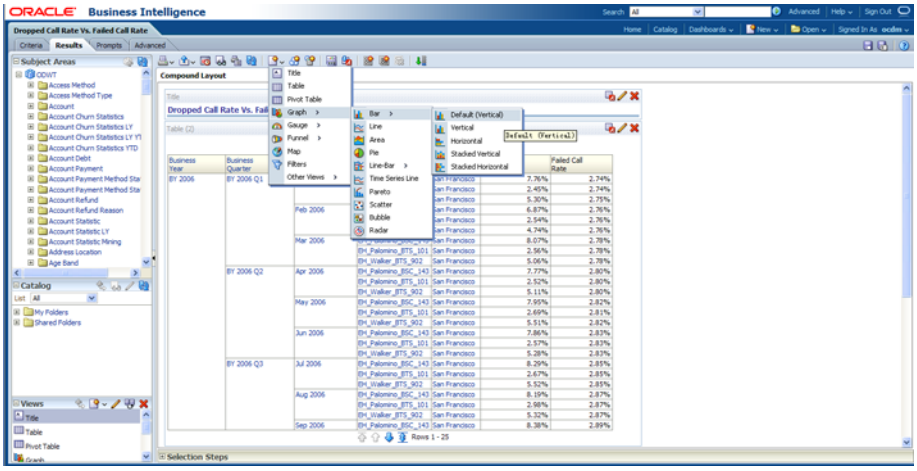
4. Drag and put the dimension and fact columns into the Select Columns panel.



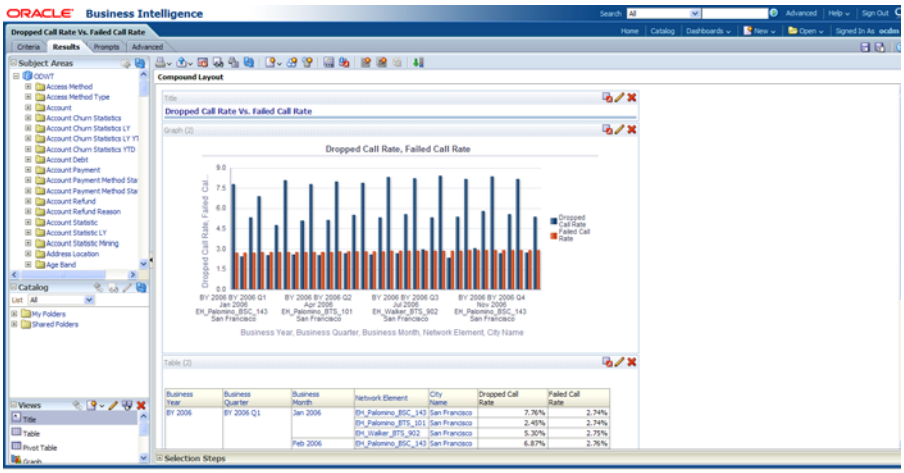
5. Select the Results tab to view the report



6. Click **New View** to add a chart into report.



7. Click **Save** to save this report to the Network Health Analysis folder



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

This chapter includes the following sections:

Overview of Managing Metadata for Oracle Communications 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 at a higher business object level.

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 Communications 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

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. Business metadata gathers this mapping information, business definitions, and rules information.
- **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. Physical layer: this metadata layer identifies the source data.
2. 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. 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
- 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 keeps on changing in your data warehouse day by day, update the Metadata Repository. When you want to change business rules, definitions, formulas or process (especially when customizing the Oracle Communications 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 Metadata Reports and Dashboard

To customize the Oracle Communications Data Model model, you must understand the dependencies among Oracle Communications Data Model components, especially how the report KPIs are mapped to the physical tables and columns. Oracle Communications Data Model provides a tool, the OCDM Metadata browser that helps you discover these dependencies. When you install Oracle Communications Data Model with its sample reports, the metadata browser is delivered as a sample Dashboard in the webcat.

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

There are four tabs (reports) in the Oracle Communications Data Model Metadata browser. To browse the metadata repository:

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 `ocdm`, and provide the password.
3. Select the Metadata Browser dashboard.
4. Use the tabs in the Metadata browser to explore the metadata.
   - **Measure-Entity tab**

     On the Measure-Entity tab you can see the business areas (relational, OLAP, mining), the measures description, corresponding formula, responsible entities, and attributes for the measure.
   - **Entity-Measure tab**

     Using the Entity-Measure tab, you can discover the mappings between entities, attributes, supported measures, and calculations of the measures. You can discover information about particular entities and attributes.
   - **Program-Table tab**

     Using the Program-Table tab you can browse for information on the intra-ETL mappings and report information. Take the following steps:
   - **Table-Program tab**

     By default when you go to the Table-Program tab you see all of the tables used for all the reports.

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

Using the Measure-Entity Tab Business Areas and Measures Attributes and Entities

The **Measure-Entity** tab provides information on the measure descriptions, computational formulas with physical columns, physical tables, and corresponding entities by Business Area.

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

Using the 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.

To view the **Entity-Measure** tab perform the following steps to learn more about an entity:

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

Using the Program-Table Tab

The **Program-Table** tab displays the input and output tables used in the selected programs.

To use the Program-Table tab, perform 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**.

Using the 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, move a particular table from the right pane to the left (Selected) pane.

To see the reports that use a particular table, perform 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.

Collecting and Populating Metadata

The Oracle Communications Data Model metadata browser generation packages generate and update the Oracle Communications Data Model metadata. The metadata generation package contains four main tables and several staging tables and views. The metadata generation tables are:

- MD_ENTY
- MD_PRG
- MD_KPI
- MD_REF_ENTY_KPI

Use the following steps to collect and populate the metadata.

1. Collect LDM Metadata:

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.

a. Start Oracle SQL Developer Data Modeler

b. Open Logical Data Model

c. Select **File**.

d. Select **Export**.

e. Select **To Reporting Schema**.

2. Collect Sample Dashboard Metadata:

   Extract the BIEE dashboard metadata from webcat to csv file.

   Using OBIEE catalog manager open the SQL Developer sample report webcat:

   Tools -> create Report -> Select type to report on -> select dashboard

   Select columns one by one as shown in 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 Communications Data Model sample report webcat.

   - Tools -> create Report -> Select type to report on -> select Analysis -> select columns one by one as shown in 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 Communications 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 Communications Data Model Physical Data Model naming convention information from csv into a staging table. Use `sqlloader` to load data from `name_conversion.csv` into MD_NAME_CONVERSION table. The `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. The `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. The `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.

   Note: If the OLAP part of the RPD is populated by the BIEE native OLAP import. Then the metadata of this part will not be shown in `md_rpd.csv`. You need to manually populate this part of metadata from the RPD.

   Use `sqlloader` to load data from `md_rpd.csv` into MD_RPD table. The `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 6-8.

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 6-10

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 6-11.

Load LDM/PDM Metadata (Table MD_ENTY)

If you want to get the mapping between a business area and an entity, you have to manually populate this information. You can only get this information from the metadata report for those entities which are used in the report, for those entities which are not used in report, you have to manually map them to the correct business area.

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 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 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	MD_RPD_RPT
MD_REPORT1	

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	MD_RPD_RPT_DASH
MD_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	MD_RPD_RPT
MD_REPORT1	

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

7

Working with User Privileges in Oracle Communications Data Model

This chapter provides information about managing user privileges in Oracle Communications Data Model. It contains the following topics:

- Accounts Created for Oracle Communications Data Model
- When You Must Consider User Privileges in an Oracle Communications Data Model
- Granting Only Select Privileges to Database Users of the Sample Reports

Accounts Created for Oracle Communications Data Model

Installing the Oracle Communications Data Model component creates the account: ocdm_sys. Installing the Oracle Communications Data Model sample reports create the ocdm_sample account. Ensure you unlock these accounts with new passwords following the postinstallation steps provided in *Oracle Communications Data Model Installation Guide*.

> **See:** *Oracle Communications Data Model Installation Guide* for information on installing Oracle Communications Data Model and for unlocking the ocdm_sys account.

The ocdm_sys accounts includes the following:

- **`ocdm_sys`** is the main schema for Oracle Communications Data Model. This schema contains all the relational and OLAP components of Oracle Communications Data Model.

  The Oracle Communications Data Model data mining tables are also in this schema.

When You Must Consider User Privileges in an Oracle Communications Data Model

The installation process grants the necessary privileges required for users of the default accounts (`ocdm_sys` and `ocdm_sample`). After installing the product, you only need to consider user privileges for the following:

- The intra-ETL programs run inside the `ocdm_sys` schema, therefore, these programs require the full access to the `ocdm_sys` schema. By default, the PL/SQL

intra-ETL packages for Oracle Communications Data Model connect to the ocdm_sys schema for intra-ETL execution. For security reasons, you may want to grant different privilages, for different purposes, to users of the ocdm_sys schema by following the steps outline in "Granting Only Required Privileges to Database Users of OCDM_SYS" on page 7-2.

- By default, the Oracle Communications Data Model sample reports connect to the ocdm_sys schema directly. For security reasons, you may want to grant only select privileges to users of the sample reports by following the steps outlined in "Granting Only Select Privileges to Database Users of the Sample Reports" on page 7-2.
- By default, you connect as ocdm in OBIEE to access the reports. For security reasons, you may want to create different users in OBIEE for different purposes by following the steps outlined in "Granting Permission Privileges of the OBIEE reports to BI Users and Roles" on page 7-2.

Granting Only Required Privileges to Database Users of OCDM_SYS

To grant only select privileges to users of the ocdm_sys schema, take the following steps:

- Create another role for a different purpose (for example, OCDM_developer for Oracle Communications Data Model customization for a developer who can execute packages and do some dml/ddl operations. And create OCDM_Viewer for a report viewer who wants to view data but cannot modify and object or data. Then create the user and grant proper roles.).
- Grant required privilege to different roles (For example, OCDM_developer needs execute privilege on etl packages but ocdm_viewer does not).
- Create users and grant required roles.
- Create a view (or synonym) in user schema that points to the ocdm_sys tables.

Granting Only Select Privileges to Database Users of the Sample Reports

To grant only select privileges to users of the sample reports, take the following steps:

1. Create a dedicated reporting user (for example, OCDM_Report).
2. Grant select privilege for all Oracle Communications Data Model tables required for reporting to OCDM_Report. (The easiest way to select privileges for these tables is to grant all Oracle Communications Data Model tables that start with a prefix of DWA_, DWB_, DWD_, DWR_, or DWL_.)
3. Create a view (or synonym) in OCDM_Report schema that points to the ocdm_sys tables.
4. In the Oracle Business Intelligence Suite Enterprise Edition repository for Oracle Communications Data Model, change the connection information to point to the new schema.

Granting Permission Privileges of the OBIEE reports to BI Users and Roles

To grant permission privileges to users of the OBIEE reports, take the following steps:

1. Create a dedicated report user (for example, market_manager).

2. Grant required group membership for user `market_manager`.
3. Create a role or manage the existing roles and add the user `market_manager` in referenced roles.
4. Configure permission privileges of the related reports or dashboards to user market_manager or the referenced roles.
5. Apply and refresh the OBIEE server.

A

Sizing and Configuring an Oracle Communications Data Model Warehouse

This appendix provides information about sizing and configuring an Oracle Communications Data Model warehouse. It contains the following topics:

Sizing an Oracle Communications 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. In order 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 stage layer database space in data warehouse system. Because this value is determined by your unique OLTP system, you must calculate this information yourself.
- The space needed for data stored to support the objects defined in the default Oracle Communications 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 Communications 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 Communications Data Model Warehouse

Use the following formula, based on the factors outlined in "Calculation Factors When Making a Data Volume Calculation for an Oracle Communications Data Model Warehouse" on page A-1, to determine the minimum disk space requirements for an Oracle Communications Data Model warehouse.[1]

```
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 ]`

[1] Carefully review whether these factors apply in your environment. These factors may not apply or may change in your environment, especially when using pretuned Exadata hardware.)

- `Compress factor` depends how you apply the compress function. If you are executing on an Exadata Database machine, it has a huge savings in disk space by using compression.

Configuring a Balanced System for Oracle Communications 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. In order 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 warehouse do not have the same memory requirements as mission-critical OLTP 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 Communications Data Model:

- Maintaining High Throughput in an Oracle Communications Data Model Warehouse
- Configuring I/O in an Oracle Communications Data Model for Bandwidth not Capacity
- Planning for Growth of Your Oracle Communications Data Model
- Testing the I/O System Before Building the Warehouse
- Balanced Hardware Configuration Guidelines for Oracle Communications Data Model

Maintaining High Throughput in an Oracle Communications 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 Communications 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 200 MB 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 Communications 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 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 Communications 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 adapter (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.

B

Upgrading Oracle Communications Data Model

This appendix provides information about how to upgrade Oracle Communications Data Model. It contains the following topics:

- Considerations for Upgrading Oracle Communications Data Model
- Upgrade Paths for Oracle Communications Data Model
- Regression Testing for Oracle Communications Data Model

> **Note:** This appendix provides general information about how to upgrade Oracle Communications Data Model. For specific information on changes from the previous release, see *Oracle Communications Data Model Release Notes*.

Considerations for Upgrading Oracle Communications Data Model

With the next release of Oracle Communications Data Model, identify the components in Oracle Communications Data Model that must be upgraded and analyze the impact that the upgrade will have on your existing data warehouse architecture and applications. The Oracle Communications Data Model upgrade focuses on Oracle Communications Data Model itself -- it does not include considerations of upgrading a hardware platform or related software.

You should consider the following points before you begin your upgrade:

- Review the System Requirements and Supported Platforms for Oracle Communications Data Model to make sure your database platform versions are supported.
- The upgrade process is not only a technical process. It requires significant planning and involvement from many teams, including development, database administrators, business analysts, QA, and so on.
- You should determine what customization was made to your existing Oracle Communications Data Model system before you begin the upgrade process. It is recommended that you budget sufficient time (in the upgrade project) for detailed review of current customizations and relevance of the customizations for the new version of the Oracle Communications Data Model system to which you plan to upgrade.
- You will also need to analyze the impact of the schema changes on your current custom implementation. The extent of your customizations will have an impact on the length of time required for the upgrade.

- Depending on the release from which you are upgrading, moving the customizations in your existing repositories, reports, and dashboards to the new version may require a multi-step process and may involve manual processes at some stages.
- It is highly recommended that you use side-by-side environments when performing each stage of the upgrade process. Enabling side-by-side-instances of the entire Oracle Communications Data Model environment is a critical success factor for upgrade. For some stages of the upgrade, you can upgrade your environments in place. However, for comparison and benchmarking purposes, it is recommended that you upgrade using side-by-side environments.
- For each stage of the upgrade process, you need to allocate a reasonable amount of time to validate the results of that stage and address any problems. In addition, final user acceptance testing must confirm that the entire upgrade process was successful. You may temporarily use the `TODEL_` tables, when defined, to help you in migrating data step by step.

Upgrade Paths for Oracle Communications Data Model

There ares two different path to upgrade Oracle Communications Data Model, depending on whether Oracle Communications Data Model has been customized.

Upgrade Path A: Based on an Earlier Default Oracle Communications Data Model

Upgrade Oracle Communications Data Model based on an earlier default Oracle Communications Data Model.

Upgrade Path B: Based on a Customized Oracle Communications Data Model

Take the following steps to upgrade your Oracle Communications Data Model based on a customized Oracle Communications Data Model:

1. Review your customization patch and upgrade it if necessary.
2. Execute the (new) customization patch.
3. Migrate the data.
4. Additions, if necessary, to modify the:
   - Derived, aggregate layer
   - Source-ETL and intra-ETL mapping
   - Workflow
   - Business intelligence applications
   - Metadata Repository

Regression Testing for Oracle Communications Data Model

After you upgrade Oracle Communications Data Model, perform regression testing. The major tasks to perform are:

- Create a test plan.
- Execute a test case.
- Test error handling.
- Create a final report.

Index

M

N

O

P

Q

R

S

T

U