Oracle® Airlines Data Model

Implementation and Operations Guide

11*g* Release 2 (11.2)

E26211-02

December 2011

Oracle Airlines Data Model Implementation and Operations Guide, 11*g* Release 2 (11.2)

E26211-02

Contents

List of Examples

List of Figures

Preface

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

This preface contains the following topics:

- Audience
- Documentation Accessibility
- Related Oracle Resources
- Conventions

Audience

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

Oracle Airlines Data Model Documentation Set

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

- *Oracle Airlines Data Model Installation Guide*
- *Oracle Airlines Data Model Reference*
- *Oracle Airlines 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.

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 Airlines Data Model Customization

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

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

What is the Oracle Airlines Data Model?

Oracle Airlines Data Model is a pre-built approach to airline data warehousing enabling an airline company to realize the power of insight more quickly. Oracle Airlines 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 an airline specific data model. You can use Oracle Airlines Data Model in any application environment. Also, you can easily extend the model.

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

Oracle Airlines Data Model provides much of the data modeling work that you must do for an airline business intelligence solution. The Oracle Airlines Data Model logical and physical data models were designed following best practices for airlines.

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

Components of the Oracle Airlines Data Model

Oracle Airlines Data Model includes the following components:

- Logical model which is a third normal form (3NF) entity-object model. The logical model follows aviation industry standards and is described in *Oracle Airlines Data Model Reference*.
- Physical model defined as an Oracle Database schema.

- Intra-ETL database packages and SQL scripts to extract, transform, and load (ETL) data from the Oracle Airlines Data Model 3NF physical tables to the derived and aggregate tables in Oracle Airlines 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 Airlines Data Model, you have the choice of performing two different types of installations:

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

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

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

Oracle Products That Make Up Oracle Airlines Data Model

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

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

Oracle Database with OLAP, Data Mining and Partitioning Option

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

Oracle Development Tools

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

Table 1–1 Oracle Development Tools Used with Oracle Airlines 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 Airlines Data Model.

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

Steps for Implementing an Oracle Airlines Data Model Warehouse

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

The reasons that you might customize Oracle Airlines Data Model include: your business does not have a business area that is in the logical model of Oracle Airlines Data Model, or you must apply a new or different business rule.

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

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

1. Perform the organizational tasks outlined in "Before You Begin Customizing the Oracle Airlines Data Model" on page 1-3.
2. Create a fit-gap analysis report by following the process outlined "Performing Fit-Gap Analysis for Oracle Airlines Data Model" on page 1-8.
3. In a development environment, install a copy of the Oracle Airlines Data Model.
4. Customize Oracle Airlines 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 Airlines 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 the Oracle Airlines Data Model" on page 4-8.
5. In a test environment, make a copy of your customized version of Oracle Airlines Data Model. Then, following the documentation you created in Step 2, test the customized version of Oracle Airlines Data Model
6. Following your typical procedures, roll the tested customized version of Oracle Airlines Data Model out into pre-production and, then, production.

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

Before You Begin Customizing the Oracle Airlines Data Model

Before you begin customizing Oracle Airlines 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 1-4.

- Implementation team. This team consists of IT engineers who have the expertise outlined in "Prerequisite Knowledge for Implementors" on page 1-4. This team has the responsibilities outlined in "Steps for Implementing an Oracle Airlines Data Model Warehouse" on page 1-3.
- Fit-gap analysis team. This team consists of business analysts who can identify the business requirements and scope of the Oracle Airlines 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 Airlines Data Model" on page 1-8.

After these teams and committees are formed:

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

Prerequisite Knowledge for Implementors

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

- Experience performing information and data analysis and data modeling. (Experience using Oracle SQL Data Modeler, is a plus.)
- 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 Airlines 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 covers the follow 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.
- The life cycle of data warehouse component management.

Data Warehouse Governance Committee: Data Governance Responsibilities

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

- Approving the data modeling standards, metadata standards and other related standards. This includes determining a metadata strategy as discussed in "Managing the Metadata for Oracle Airlines Data Model" on page 1-5' and identifying the data modeling tools to use that support these standards.
- Determining the data retention policy.
- Designing a data access policy based on legal restrictions and data security rules.
- Designing a data backup strategy that aligns with the impact analysis to the business unit.
- Monitoring and reporting on data usage, activity, and alerts.

Managing the Metadata for Oracle Airlines Data Model

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

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

- Metadata Categories and Standards
- Working with a Metadata Repository
- Browsing the Metadata Repository Supplied With Oracle Airlines Data Model

Metadata Categories and Standards

Metadata is organized into three major categories:

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

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

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

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

Working with a Metadata Repository

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

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

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

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

Then:

- Decide how important each part is.
- Assign responsibility to someone for each piece.
- Decide what constitutes a consistent and working set of metadata
- 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 Airlines Data Model), your first step is to survey the metadata and do an impact analysis to list all of the attributes in the data warehouse environment that would be affected by a proposed change.

Browsing the Metadata Repository Supplied With Oracle Airlines Data Model

To customize the Oracle Airlines Data Model model, you must understand the dependencies among Oracle Airlines Data Model components -- especially how the report KPIs are mapped to the physical tables and columns.

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

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

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 `oadm`, 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.

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

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

1. Select the entity.

2. Click **GO**.

- **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:

  1. Select the program type (that is, intra-ETL or report) and program name.

  2. Select **GO**.

- **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.

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

  1. In the right pane of the Table-Program tab, select the table.

  2. Move the table to the Selected list on the left by clicking on < (left arrow).

  3. Click **OK**.

  4. Select **GO**.

  The reports for the selected table are displayed.

Performing Fit-Gap Analysis for Oracle Airlines Data Model

Fit-gap analysis is where you compare your information needs and airline business requirements with the structure that is available with Oracle Airlines Data Model. You identify any required functionality that is not included in the logical model and the default schema, as well as 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 Airlines Data Model to fit your airline environment.

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

1. If you have performed previous evaluations, review the documentation from the previous phases, and if necessary add team members with the required business and technical expertise.
2. Review the data and map your logical entities and data structure with the Oracle Airlines Data Model logical model and schema:
   - Starting from business requirements, questions, and rules, identify any entities and attributes that are *not* in the Oracle Airlines Data Model.

- Compare the Oracle Airlines Data Model to your existing application model if have one.
- Compare the Oracle Airlines Data Model to the OLTP data that you are using as a data source to the Oracle Airlines Data Model warehouse.

3. Determine the differences between your needs and Oracle Airlines 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 Airlines 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 Airlines 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 Airlines Data Model.

   Tip: When identifying changes, ensure that the changes meet your security and metadata requirements.
5. Write the customization report, detailing what changes are required to make the Oracle Airlines 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 Airlines Data Model Warehouse" on page 1-3.

2 Physical Model Customization

This chapter provides general information about customizing the physical model of Oracle Airlines 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 data model of Oracle Airlines Data Model defines approximately:

370 tables and 8,500 columns
220 industry-specific measures and KPIs
6 pre-built data mining models
7 pre-built OLAP cubes

The default physical model of the Oracle Airlines 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 Airlines 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 OLTP 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 that much of the complex data transformation and data quality processing occurs. The most basic approach for the design of the staging layer is as a schema identical to the one that exists in the source operational system.

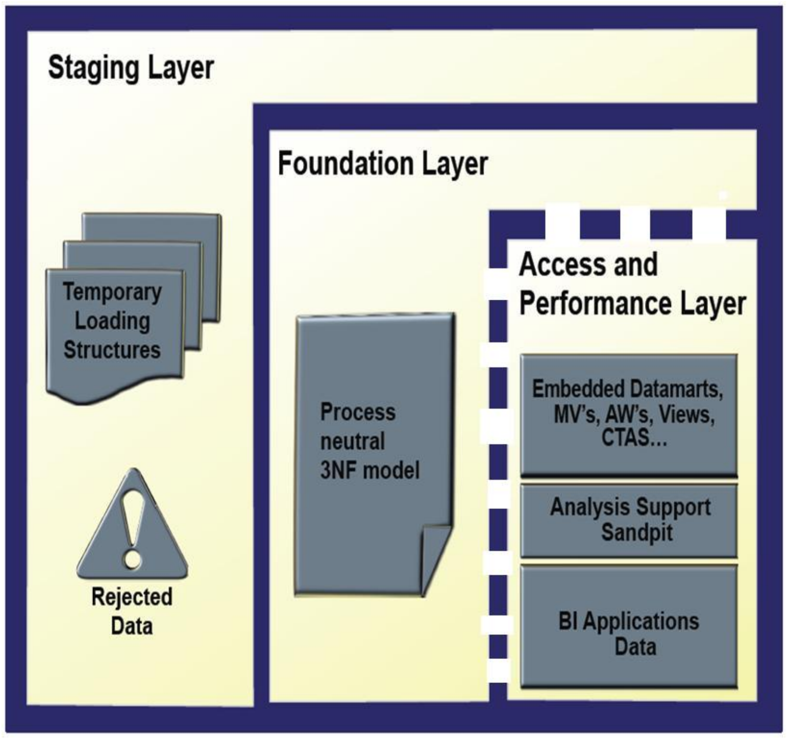
Note: In some implementations this layer is not necessary, because all data transformation processing is done "on the fly" as data is extracted from the source system before it is inserted directly into the foundation layer.

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

Layers in the Default Oracle Airlines Data Model Warehouse

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

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


Staging Layer
Foundation Layer
Access and Performance Layer
Temporary Loading Structures
Rejected Data
Process neutral 3NF model
Embedded Datamarts, MV's, AW's, Views, CTAS...
Analysis Support Sandpit
BI Applications Data

The layers in the Oracle Airlines Data Model warehouse are:

- **Staging layer**. As in a "traditional" data warehouse environment, an Oracle Airlines 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 Airlines Data Model.
- **Foundation and Access layers.** The physical objects for these layers are defined in a single schema, the `oadm_sys` schema:
  - **Foundation layer.** The foundation layer of the Oracle Airlines 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 `oadm_sys` schema (that is, the tables that have the `DWR_` , `DWL_`, `DWC_` prefixes).
  - **Access layer.** The access layer of Oracle Airlines Data Model is defined by dimension tables (defined with a `DWM_` prefix), derived and aggregate tables (defined with `DWD_` and `DWA_` prefixes), cubes (defined with a `CB$` prefix), and views (that is, views defined with the `_VIEW` suffix). These structures provide a summarized or "flattened" 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.

    See: *Oracle Airlines Data Model Reference* for detailed information on the `oadm_sys` schema.

Customizing the Oracle Airlines Data Model Physical Model

The starting point for the Oracle Airlines 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, views).

To customize the default physical model of the Oracle Airlines 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 Airlines Data Model as outlined in"Characteristics of the Default Physical Model" on page 2-1 and presented in detail in *Oracle Airlines Data Model Reference*.
3. Modify the foundation level of your physical model of Oracle Airlines Data Model, as needed. See "Common Change Scenarios When Customizing the Foundation Layer of Oracle Airlines Data Model" on page 2-7 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 2-9 to guide you when designing the physical objects.
   - Follow the conventions used when creating the default physical model of Oracle Airlines 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 `oadm_sys` schema.

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

Questions to Answer Before You Customize the Physical Model

When designing the physical model, remember that the logical data model is not one-to-one with the physical data model. Consider the load, query, and maintenance requirements when you 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 When Customizing the Foundation Layer of Oracle Airlines Data Model" on page 2-7 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 Airlines 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 Airlines 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
- Use of History (_H) Tables
- Domain Definition Standards

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 `oadm_sys` schema delivered with Oracle Airlines Data Model uses the prefixes and suffixes shown in the following table to identify object types.

Prefix or Suffix	Used for Name of These Objects
`CB$`	Materialized view of an OLAP cube. This materialized view is automatically created by the OLAP server. **Note:** Do not report or query against this object. Instead access the corresponding `_VIEW` object.
`DMV_`	Materialized views used for as the source data of data mining model.
`DWA_`	Aggregate tables.
`DWB_`	Base transaction data (3NF) tables.
`DWC_`	Control tables.
`DWD_`	Derived tables -- including data mining result tables.
`DWL_`	Lookup tables.
`DWM_`	Dimension tables in an access layer fact table (that is, for a `DWD_` or a `DWA_` table).
`DWV_`	Relational view of time dimension.
`DWR_`	Reference data tables used as dimension tables in a foundation layer fact table (that is, for a `DWB_` table).
`_H`	"Classic" data warehouse table that is used to store both the most recent data and the historical data of a certain entity. (See "Use of History (_H) Tables" on page 2-5.)
`_VIEW`	Relational views of OLAP cubes, dimensions, or hierarchies.

Use these prefixes and suffixes for any new tables, views, and cubes that you define.

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

Use of History (_H) Tables

The physical data model of the default Oracle Airlines Data Model was designed to function both as an Operational Data Store (ODS) and a data warehouse. To support this dual functionality, the logical entities are implemented as two different types of physical tables:

- *`tablename`*`_H` tables are "classic" data warehouse tables. These tables are designed to store both the most recent data and the historical data for an entity. These are the tables that provide the data for the access layer objects (that is, derived and aggregate tables, and OLAP cubes). When `_H` tables are populated with new data, the data is added to the table; data is never overwritten.
- *`tablename`* tables are ODS tables. These tables are designed to store *only* the most recent data for an entity. These are the tables that an application accesses when making real-time (or near real-time) queries. When these tables are populated with new data, the new data overwrites data already in the table.

For example, when you look at the physical tables that represent the logical entity Service, you see that there are two tables: `DWR_SERVICE` and `DWR_SERVICE_H`. The function of the `DWR_SERVICE` table is to hold the most recent data; while the function of `DWR_SERVICE_H` is to hold both the most recent data and the historical data.

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'`.
- Numeric value in different situations.
- Character string length in different situations.
- Coded value definition such as key or description.

Foundation Layer Customization

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

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

- When changing the foundation layer objects to reflect your logical model design, make as few changes as possible. "Common Change Scenarios When Customizing the Foundation Layer of Oracle Airlines Data Model" on page 2-7 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 2-9 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 Airlines Data Model with caution. The foundation layer of the physical model of the Oracle Airlines Data Model has (at its core) a set of generic structures that allow it to be flexible and extensible. Before making extensive additions, deletions, or changes, ensure that you understand the full range of capabilities of Oracle Airlines Data Model and that you cannot handle your requirements using the default objects in the foundation layer

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

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

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

- **Additions to Existing Structures**

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

  Carefully study the default foundation layer of the physical data model of Oracle Airlines 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 Airlines Data Model Development Team for possible inclusion in future releases of Oracle Airlines 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 it. Some tables may not be needed during the initial implementation, however 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 Airlines 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 Airlines Data Model. Also identify the impact on any applications based on the new design.

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

Example: Changing the Foundation Layer of the Oracle Airlines Data Model

As an example, let's take a look how Oracle Airlines Data Model supports the various airlines services, what a sample customer might discover during fit-gap analysis, and how that customer might extend Oracle Airlines Data Model to fit the discovered gaps.

Entities supporting airline services

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

- AIRPORT: Airport means an IATA recognized location that serves as an origin or destination of one or more flights. This entity stores the details of the airport like city, country, region of the airport.

- FLIGHT: This entity stores information about the flight (for example, the carrier to which it belongs, and the flight number).
- AIRCRAFT: This entity stores the type of aircraft (for example. Boeing 737).
- AIRCRAFT VERSION: This entity stores the details of the aircraft version used for flights. For example, if the aircraft type is Boeing 737 then the version will be Boeing 737-800.
- CARRIER: This entity stores the details about the carrier (for example, carrier code and description).
- CARRIER TYPE: This entity stores Carrier type details (for example, airline, railway, on road transport, and ship).

The differences discovered during fit-gap analysis

Assume that during the fit-gap analysis, you discover the following needs that are not supported by the logical model delivered with Oracle Airlines Data Model:

- Your company serves more than one service in airline arena. In other words, you discover that you track not only airports as represented in the standard logical model of Oracle Airlines Data Model, but also you need to define airports by categories of airport activities (that is, airport categories). These airport categories are, for example, commercial service, primary, cargo service, reliever, and general aviation airports.
- In the flight operations, the airline needs to maintain a history of airport categories used to operate the various lines of services (for example, commercial, cargo, reliever or general aviation).

Extending the logical and physical model to support the differences

To support the differences, you need to extend the logical and physical model in the following ways:

- To store details about category of the airport you will need to modify the logical model. The classic way to do this is to add another entity to hold this information. For example, you can add an entity named AIRPORT CATEGORY that has a 1:M relationship with the AIRPORT entity. Then, you extend the physical data model in a corresponding manner. The specific steps to extend the physical model are given in Example 2–1, "Extending the Physical Data Model of Oracle Airlines Data Model to Support Multiple Categories for AIRPORT".
- To maintain the history of flight operations which are operated from respective airport categories, you can find the same based on the existing relationship between flight schedule and AIRPORT entity for corresponding Leg entity.

Example 2–1 Extending the Physical Data Model of Oracle Airlines Data Model to Support Multiple Categories for AIRPORT

To extend the design of the foundation layer of Oracle Airlines Data Model to support multiple categories for `AIRPORT`, take the following steps:

1. Create a new table named `DWL_ARPRT_CTGRS` to hold the multiple categories information for airport by executing the following statements.

```
CREATE TABLE DWL_ARPRT_CTGRS
(
    ARPRT_CTGRS_ID          INTEGER NOT NULL ,
    ARPRT_CTGRS_NM          VARCHAR2(50) NULL ,
    ARPRT_CTGRS_DESC        VARCHAR2(500) NULL ,
    ARPRT_CTGRS_CD          CHAR(18) NULL ,
    DWFEED_ID                INTEGER NULL ,
    SRC_SYS_ID               VARCHAR2(30) NULL ,
    SRC_SYS_CRTD_TMSTMP      TIMESTAMP NULL ,
    SRC_SYS_UPD_TMSTMP       TIMESTAMP NULL ,
    SRC_SYS_DEL_IND          VARCHAR2(1) NULL
);


ALTER TABLE DWL_ARPRT_CTGRS
    ADD CONSTRAINT  XPKARPRT_CTGRS PRIMARY KEY (ARPRT_CTGRS_ID);
```

2. Create a new table named `DWL_ARPRT_CTGRS_H` to hold the history information for `DWL_ARPRT_CTGRS` by executing the following statement.

```
CREATE TABLE DWL_ARPRT_CTGRS_H
(
    ARPRT_CTGRS_ID           INTEGER NOT NULL ,
    ARPRT_CTGRS_NM           VARCHAR2(50) NULL ,
    ARPRT_CTGRS_DESC         VARCHAR2(500) NULL ,
    ARPRT_CTGRS_CD           CHAR(18) NULL ,
    DWFEED_ID                INTEGER NULL ,
    SRC_SYS_ID               VARCHAR2(30) NULL ,
    SRC_SYS_CRTD_TMSTMP      TIMESTAMP NULL ,
    SRC_SYS_UPD_TMSTMP       TIMESTAMP NULL ,
    SRC_SYS_DEL_IND          VARCHAR2(1) NULL ,
    ETL_BATCH_ID             INTEGER NULL ,
    ETL_BATCH_CRTD_BY        VARCHAR2(60) NULL ,
    ETL_BATCH_CRTD_TMSTMP    TIMESTAMP NULL ,
    ETL_BATCH_UPD_BY         VARCHAR2(60) NULL ,
    ETL_BATCH_UPD_TMSTMP     TIMESTAMP NULL ,
    DATA_MVT_STS_CD          VARCHAR2(25) NULL ,
    VLD_FRM                  TIMESTAMP NULL ,
    VLD_UPTO                 TIMESTAMP NULL ,
    CURR_STS                 VARCHAR2(1) NULL ,
    DWL_ARPRT_CTGRS_H_SKEY   INTEGER NOT NULL
);
```

3. Add a column named `ARPRT_CTGRS_ID` to the `DWR_AIP` table by issuing the following statement.

```
ALTER TABLE DWR_AIP ADD COLUMN ARPRT_CTGRS_ID INTEGER NULL;
```

General Recommendations When Designing Physical Structures

The `oadm_sys` schema delivered with Oracle Airlines 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 Airlines Data Model:

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

- Partitioned Tables in the Oracle Airlines Data Model
- Parallel Execution in the Oracle Airlines Data Model

Tablespaces in the Oracle Airlines 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 Airlines Data Model warehouse.

Changing the Tablespace and Partitions Used by Tables

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

  Consequently, for Oracle Airlines 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 Airlines 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 Airlines 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. Most of the base tables in the Oracle Airlines Data Model are compressed tables.

Recommendations: Data Compression

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

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

> **See also:** "Types of Data Compression Available" on page 2-11 and "Compressing Materialized Views" on page 3-24.

Types of Data Compression Available

Oracle Database offers the following types of compression:

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

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

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

Oracle Airlines 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;
```

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

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

> **Oracle by Example:** For more information on Oracle Advanced Compression, see the "Using Table Compression to Save Storage Costs" OBE tutorial.
>
> To access the tutorial, open the Oracle Learning Library in your browser by following the instructions in "Oracle Technology Network" on page xii; and, then, search for the tutorial by name.

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

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

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

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

Tables for Supertype and Subtype Entities in Oracle Airlines 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

For example, in the default Oracle Airlines Data Model, the table `DWB_PRTY_INTRATN` (Party Interaction) is a supertype that has a number of subtypes including `DWB_PRTY_INTRATN_CALL` (Party Interaction Call) and `DWB_PRTY_INTRATN_EML` (Party Interaction Email).

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 Airlines Data Model

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

The most common types of constraints include:

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

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

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

Indexes and Partitioned Indexes in Oracle Airlines 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 Airlines 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 Airlines Data Model" on page 2-14, "Choosing Indexes for Materialized Views" on page 3-22, "Choosing a Cube Partitioning Strategy for Oracle Airlines Data Model" on page 3-18, and "Partitioning and Materialized Views" on page 3-23.

Partitioned Tables in the Oracle Airlines 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 non-partitioned 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 Airlines Data Model for Manageability
- Partitioning the Oracle Airlines Data Model for Easier Data Access
- Partitioning the Oracle Airlines Data Model for Join Performance

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

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

See also: "Indexes and Partitioned Indexes in Oracle Airlines Data Model" on page 2-14, "Choosing a Cube Partitioning Strategy for Oracle Airlines Data Model" on page 3-18, and "Partitioning and Materialized Views" on page 3-23.

Partitioning the Oracle Airlines 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.

Partitioning the Oracle Airlines Data Model for Easier Data Access

Range partitioning also helps ensure that only the necessary data to answer a query is scanned. Consider the case where business users predominately accesses the sales data on a weekly basis (for example, total sales per week) then range partitioning this table by day ensures that the data is accessed in the most efficient manner, as only seven partitions must be scanned to answer the business users query instead of the entire table. The ability to avoid scanning irrelevant partitions is known as partition pruning.

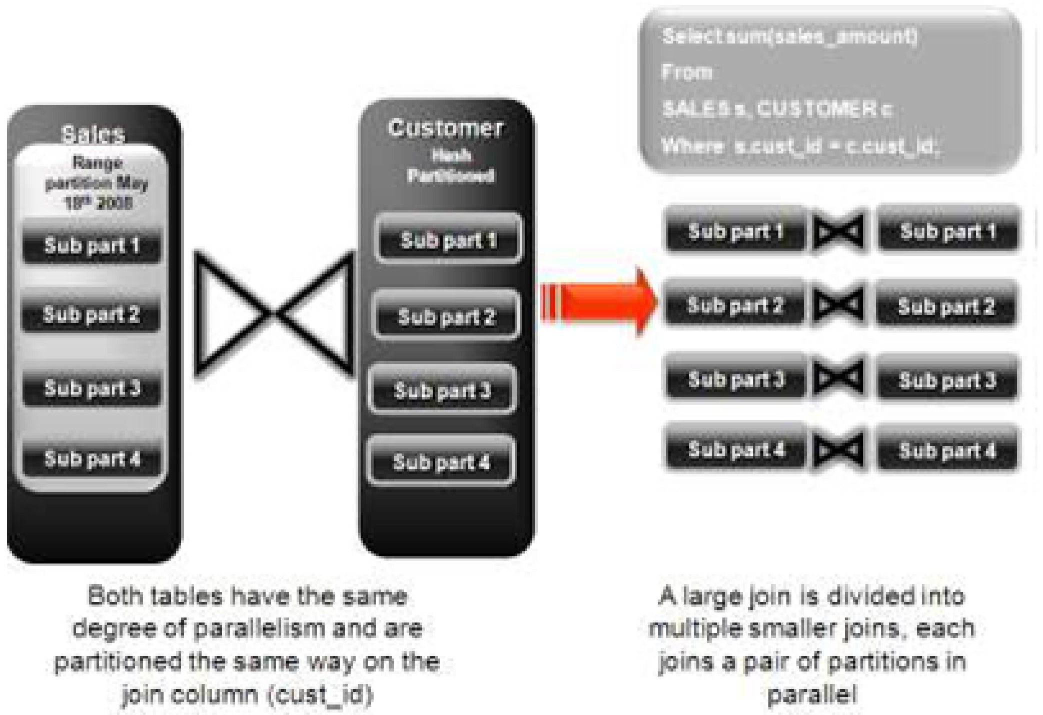
Partitioning the Oracle Airlines Data Model for Join Performance

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

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

As illustrated by Figure 2–2, "Partitioning for Join Performance", 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


Sales
Range partition May 18th 2008
Sub part 1
Sub part 2
Sub part 3
Sub part 4
Customer
Hash Partitioned
Sub part 1
Sub part 2
Sub part 3
Sub part 4
Select sum(sales_amount)
From
SALES s, CUSTOMER c
Where s.cust_id = c.cust_id;
Sub part 1
Sub part 1
Sub part 2
Sub part 2
Sub part 3
Sub part 3
Sub part 4
Sub part 4
Both tables have the same degree of parallelism and are partitioned the same way on the join column (cust_id)
A large join is divided into multiple smaller joins, each joins a pair of partitions in parallel

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 16MB in size. 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.

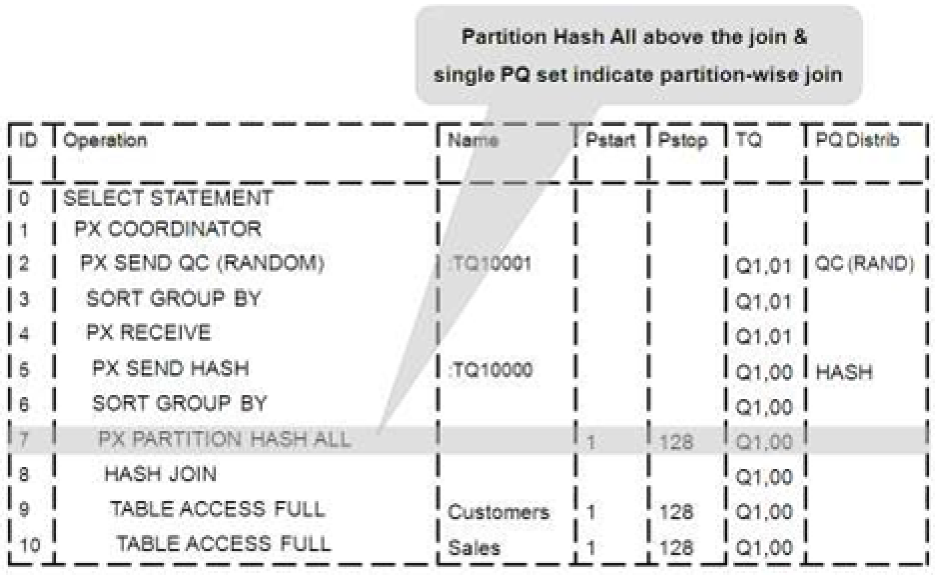
Parallel Execution in the Oracle Airlines Data Model

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, "Parallel Execution of a Full Partition-Wise Join Between Two Tables" 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


Partition Hash All above the join &
single PQ set indicate partition-wise join

```
| ID | Operation                | Name      | Pstart | Pstop | TQ    | PQ Distrib |
|----|--------------------------|-----------|--------|-------|-------|------------|
| 0  | SELECT STATEMENT         |           |        |       |       |            |
| 1  |  PX COORDINATOR          |           |        |       |       |            |
| 2  |   PX SEND QC (RANDOM)    | :TQ10001  |        |       | Q1,01 | QC (RAND)  |
| 3  |    SORT GROUP BY         |           |        |       | Q1,01 |            |
| 4  |    PX RECEIVE            |           |        |       | Q1,01 |            |
| 5  |     PX SEND HASH         | :TQ10000  |        |       | Q1,00 | HASH       |
| 6  |     SORT GROUP BY        |           |        |       | Q1,00 |            |
| 7  |      PX PARTITION HASH ALL|          | 1      | 128   | Q1,00 |            |
| 8  |       HASH JOIN          |           |        |       | Q1,00 |            |
| 9  |        TABLE ACCESS FULL | Customers | 1      | 128   | Q1,00 |            |
| 10 |        TABLE ACCESS FULL | Sales     | 1      | 128   | Q1,00 |            |
```

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 Partial Partition-Wise Join

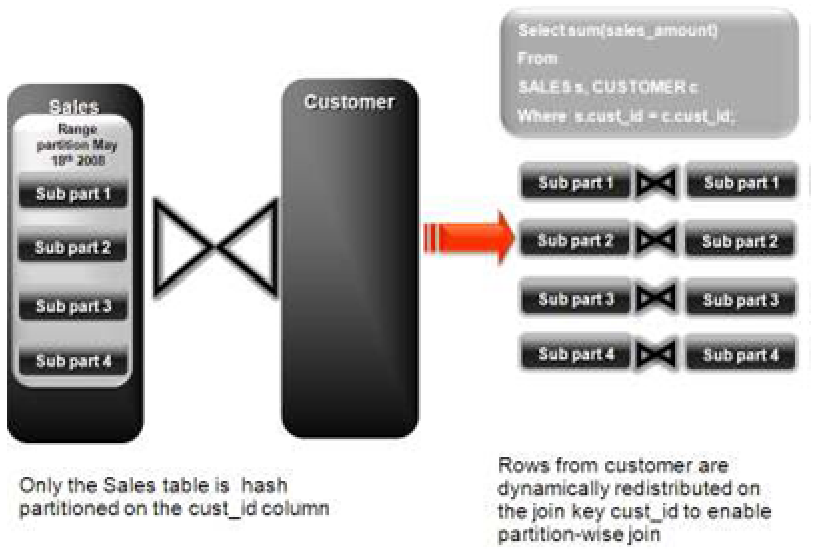

Sales
Range partition May 18th 2008
Sub part 1
Sub part 2
Sub part 3
Sub part 4
Customer
Select sum(sales_amount)
From
SALES s, CUSTOMER c
Where s.cust_id = c.cust_id;
Sub part 1
Sub part 1
Sub part 2
Sub part 2
Sub part 3
Sub part 3
Sub part 4
Sub part 4
Only the Sales table is hash partitioned on the cust_id column
Rows from customer are dynamically redistributed on the join key cust_id to enable partition-wise join

Figure 2–4, "Partial Partition-Wise Join" 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.

- Enabling Parallel Execution for a Session
- Enabling Parallel Execution of DML Operations
- Enabling Parallel Execution at the Table Level

Enabling Parallel Execution for a Session

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

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

Enabling Parallel Execution of DML Operations

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

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

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

- The following rules apply when determining how to parallelize DML `UPDATE` and `DELETE` statements:
  - Oracle Database can parallelize `UPDATE` and `DELETE` statements on partitioned tables, but only when multiple partitions are involved.

  - You cannot parallelize UPDATE or DELETE operations on a non-partitioned 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 Airlines Data Model. It includes the following topics:

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

See also: Chapter 2, "Physical Model Customization"

Introduction to Customizing the Access Layer of Oracle Airlines Data Model

The access layer of Oracle Airlines 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 `oadm_sys` schema include: derived and aggregate tables, tables that provide the dimension for the derived and aggregate tables, OLAP cubes, and materialized views. This layer also contains data mining models. The results of the these models are stored in derived tables.

When designing and customizing access layer objects:

- Follow the general guidelines for customizing physical objects given in "General Recommendations When Designing Physical Structures" on page 2-9.
- Design the access layer objects to support the business intelligence reports and queries that your site makes. See Chapter 5, "Report and Query Customization."

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

- Dimension Tables in the Oracle Airlines Data Model
- Derived Tables in the Oracle Airlines Data Model
- Aggregate Tables in the Oracle Airlines Data Model
- Dimensional Components in the Oracle Airlines Data Model

- Materialized Views in the Oracle Airlines Data Model

Dimension Tables in the Oracle Airlines Data Model

Dimension tables are tables that store the dimension data for fact tables. There are two types of tables that act as dimensions to other tables in the Oracle Airlines Data Model:

- In the foundation layer, `DWR_` tables (also known as reference tables) act as dimension tables to the base (`DWB_`) tables.
- In the access layer, `DWM_` tables (simply called "dimension" tables) act as dimension tables to derived (`DWD_`) and aggregate (`DWA_`) tables. `DWM_` tables provide a view of the data similar to the view provided by a dimension table in a star schema.

The default Oracle Airlines Data Model defines several dimension tables. For example, the `DWR_SEG` and `DWM_SEG` tables store the values of the segment information in different layers.

Derived Tables in the Oracle Airlines Data Model

Derived tables are tables that have as values the result of a non-aggregate calculation against the data in the foundation layer tables. Derived tables have a `DWD_` prefix.

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

- Tables that hold the results of a calculation. 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 information on customizing data mining models, see "Customizing Data Mining Models in the Oracle Airlines Data Model" on page 3-2.

See: The Derived Tables topic in *Oracle Airlines Data Model Reference* for a list of all of the derived tables in the default Oracle Airlines 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 Airlines Data Model Reference*.

Creating New Derived Tables for Calculated Data

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

Customizing Data Mining Models in the Oracle Airlines Data Model

Some derived (`DWD_`) tables define in the `oadm_sys` schema are the result tables for the data mining models defined in the Oracle Airlines Data Model. All Oracle Airlines Data Model data mining models use materialized views as source input. Those materialized views are defined in the `oadm_mining_init.sql` script in `$ORACLE_HOME/oadm/pdm/mining`. Different data mining models use different source materialized views.

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

- Create a new data mining model as discussed in "Creating a New Data Mining Model for Oracle Airlines Data Model" on page 3-3.
- Modify an existing data mining model as discussed in "Modifying a Data Mining Model in the Oracle Airlines Data Model" on page 3-3.

See also: "Tutorial: Customizing the Customer Life Time Value Prediction Data Mining Model" on page 3-4.

Creating a New Data Mining Model for Oracle Airlines Data Model

To create a new data mining model:

1. Ensure that the `oadm_sys` schema includes a definition for materialized view that you can use as input to the new data mining model. Define a new materialized view, if necessary.
2. Create the new data mining model as you would any data mining model. Follow the instructions given in *Oracle Data Mining Concepts*.
3. Add any physical tables needed by the data mining model into the `oadm_sys` schema. When naming a data mining source table, use the prefix `DM_` . When naming a data mining results table, use the prefix `DWD_` .

Modifying a Data Mining Model in the Oracle Airlines Data Model

To customize an Oracle Airlines Data Model data mining model, take the following steps:

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

Example 3–1 Adding a New Column to the create_cust_ltv_svm_rgrsn_model Data Mining Model

The `create_cust_ltv_svm_rgrsn_model` data mining model uses the `dmv_cust_ltv_src` materialized view as source input. To customize the `create_cust_ltv_svm_rgrsn_model` data mining model by adding a new column to the model, take the following steps:

1. Modify the following materialized views:
   - `dmv_bkg_fact_src`
   - `dmv_lylty_acct_bal_src`
   - `dmv_cust_profile_src`
2. Train customer life time value regression data mining model, execute the following statement.

   ```
   pkg_oadm_mining.create_cust_ltv_svm_rgrsn
   ```
3. Issue the following statement to query the result table and ensure the new column is included in the query result.

```
SELECT attribute_name FROM TABLE( SELECT attribute_set FROM
TABLE(DBMS_DATA_MINING.GET_MODEL_DETAILS_SVM('CUST_LTV_SVM_
RGRSN')));
```

See also: "Refreshing Oracle Airlines Data Model Data Mining Models" on page 4-13, and "Troubleshooting Data Mining Model Creation" on page 4-15.

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

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

This tutorial shows how to predict the Life Time Value of customers, who are frequent flier passengers, for next 6 months based on populated Oracle Airlines Data Model warehouse. Using prebuilt Oracle Airlines Data Model data mining models, you can easily and very quickly see the prediction results of your customers, without having to go through all of the data preparation, training, testing, and applying process that you must perform in a traditional from-scratch mining project.

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

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

This tutorial consists of the following:

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

See: *Oracle Data Mining Concepts* for more information about the Oracle Database data mining model training and scoring (applying) process.

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 Airlines Data Model.
3. Populate the base, reference and lookup tables.
4. Execute the intra-ETL.

   Ensure that at least the following tables contain valid data:

   `DWB_LYLTY_ACCT_BAL_HIST_H`

```
DWD_BKG_FACT
DWM_CLNDR
DWM_FRQTFLR
DWM_GEOGRY
```

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

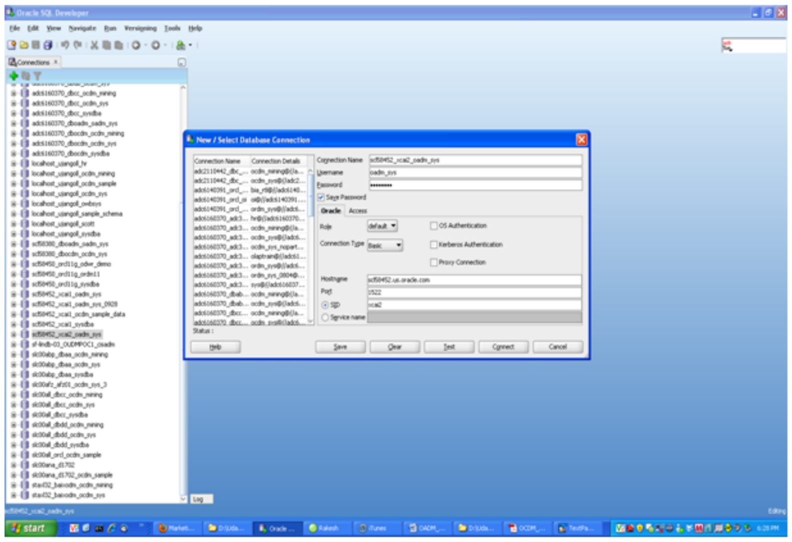
1. Ensure that during the install, you generated the calendar data covering range of 2005~2011. For example, the parameters of starting from `20050101` for 7 years satisfy this condition.
2. Download the sample data (`oadm_sample_mining.dmp.zip`) and import the data into your new `oadm_sys` schema.

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

To prepare the environment, take the following steps:

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

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

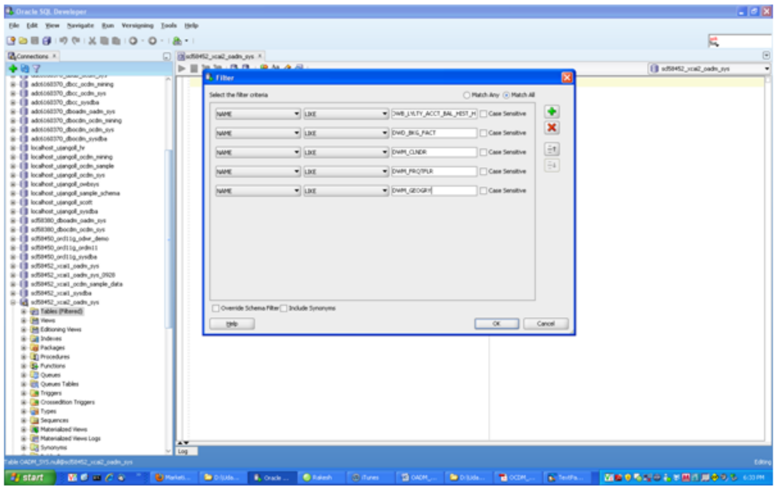

Oracle SQL Developer
File Edit View Navigate Run Versioning Tools Help
Connections
New / Select Database Connection

Connection Name: scf50452_xcal2_oadm_sys
Username: oadm_sys
Password: **********
[✓] Save Password
Oracle | Access
Role: default
Connection Type: Basic
[ ] OS Authentication
[ ] Kerberos Authentication
[ ] Proxy Connection
Hostname: scf50452.us.oracle.com
Port: 1522
(●) SID: xcal2
( ) Service name
Status :
Help | Save | Clear | Test | Connect | Cancel

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.

---

2. After you connect to the `oadm_sys` schema, you can see all of the tables that have been created. You can narrow down the list by right clicking "Tables" and then applying filters.



NAME	LIKE	DWB_LYLTY_ACCT_BAL_HIST_H	Case Sensitive
NAME	LIKE	DWD_BKG_FACT	Case Sensitive
NAME	LIKE	DWM_CLNDR	Case Sensitive
NAME	LIKE	DWM_FRQTFLR	Case Sensitive
NAME	LIKE	DWM_GEOGRY	Case Sensitive

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

   a. Grant dba to `oadm_sys` by issuing the following statement:

   ```
   grant dba to oadm_sys
   ```

   b. Disable all foreign keys on those tables required by the tutorial.

   First, issue the following statement that generates SQL statements.

   ```
   SELECT 'ALTER TABLE ' || table_name || ' DISABLE CONSTRAINT ' ||
     CONSTRAINT_NAME ' CASCADE;' FROM all_constraints
    WHERE status='ENABLED' AND owner='OADM_SYS'
      AND constraint_type=
       'R' and table_name IN
         ('DWB_LYLTY_ACCT_BAL_HIST_H','DWD_BKG_FACT','DWM_CLNDR',
           'DWM_FRQTFLR','DWM_GEOGRY') ;
   ```

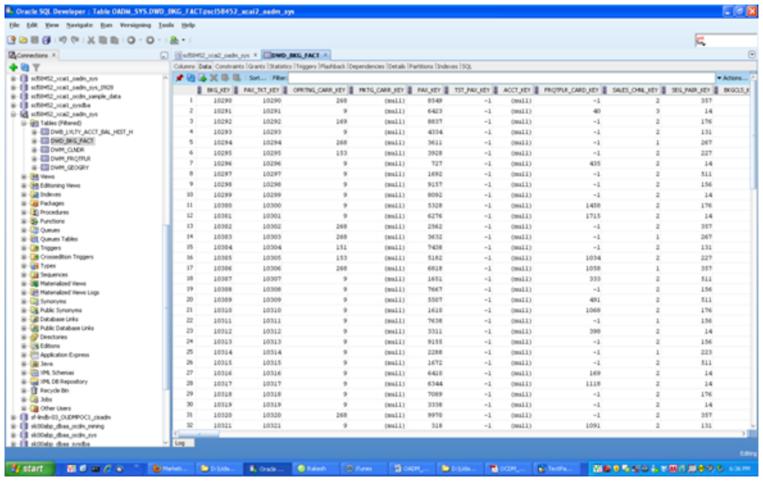
   Then, to actually disable the foreign keys for the following tables, execute the SQL statements generated by the previous SELECT statement:

   ```
   DWB_LYLTY_ACCT_BAL_HIST_H
   DWD_BKG_FACT
   DWM_CLNDR
   DWM_FRQTFLR
   DWM_GEOGRY
   ```

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

   ```
   impdp oadm_sys/password directory=DATA_PUMP_DIR dumpfile=oadm_sample.dmp
      content=DATA_ONLY table_exists_action=truncate
      TABLES=oadm_sample.DWB_LYLTY_ACCT_BAL_HIST_H,
      oadm_sample.DWD_BKG_FACT,oadm_sample.DWM_CLNDR,
      oadm_sample.DWM_FRQTFLR,oadm_sample.DWM_GEOGRY
   ```

4. Review the tables to ensure that they contain valid data (either your own customer data or sample mining data).


Oracle SQL Developer : Table OADM_SYS.DWD_BKG_FACT@scl58452_xcal2_oadm_sys
File Edit View Navigate Run Versioning Tools Help
Connections
DWD_BKG_FACT
Columns Data Constraints Grants Statistics Triggers Flashback Dependencies Details Partitions Indexes SQL
Sort... Filter:
Actions...
Log
Editing
start

	BKG_KEY	PAX_TKT_KEY	OPRTNG_CARR_KEY	MKTG_CARR_KEY	PAX_KEY	TST_PAX_KEY	ACCT_KEY	FRQTFLR_CARD_KEY	SALES_CHNL_KEY	SEG_PAIR_KEY	BKGCLS_K
1	10290	10290	268	(null)	8549	-1	(null)	-1	2	357	
2	10291	10291	9	(null)	6423	-1	(null)	40	3	14	
3	10292	10292	169	(null)	8837	-1	(null)	-1	2	176	
4	10293	10293	9	(null)	4334	-1	(null)	-1	2	131	
5	10294	10294	268	(null)	3611	-1	(null)	-1	1	267	
6	10295	10295	153	(null)	3928	-1	(null)	-1	2	227	
7	10296	10296	9	(null)	727	-1	(null)	435	2	14	
8	10297	10297	9	(null)	1692	-1	(null)	-1	2	511	
9	10298	10298	9	(null)	9157	-1	(null)	-1	2	156	
10	10299	10299	9	(null)	8092	-1	(null)	-1	2	14	
11	10300	10300	9	(null)	5328	-1	(null)	1458	2	176	
12	10301	10301	9	(null)	6276	-1	(null)	1715	2	14	
13	10302	10302	268	(null)	2562	-1	(null)	-1	2	357	
14	10303	10303	268	(null)	3632	-1	(null)	-1	1	267	
15	10304	10304	151	(null)	7438	-1	(null)	-1	2	131	
16	10305	10305	153	(null)	5182	-1	(null)	1034	2	227	
17	10306	10306	268	(null)	6818	-1	(null)	1058	1	357	
18	10307	10307	9	(null)	1651	-1	(null)	333	2	511	
19	10308	10308	9	(null)	7667	-1	(null)	-1	2	156	
20	10309	10309	9	(null)	5507	-1	(null)	491	2	511	
21	10310	10310	9	(null)	1618	-1	(null)	1068	2	176	
22	10311	10311	9	(null)	7638	-1	(null)	-1	1	156	
23	10312	10312	9	(null)	3311	-1	(null)	398	2	14	
24	10313	10313	9	(null)	9155	-1	(null)	-1	2	156	
25	10314	10314	9	(null)	2288	-1	(null)	-1	1	223	
26	10315	10315	9	(null)	1672	-1	(null)	-1	2	511	
27	10316	10316	9	(null)	6418	-1	(null)	169	2	14	
28	10317	10317	9	(null)	6344	-1	(null)	1118	2	14	
29	10318	10318	9	(null)	7089	-1	(null)	-1	2	176	
30	10319	10319	9	(null)	3338	-1	(null)	-1	2	14	
31	10320	10320	268	(null)	9978	-1	(null)	-1	2	357	
32	10321	10321	9	(null)	318	-1	(null)	1091	2	131	

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

Generating the Model This tutorial uses two procedures from Oracle Airlines Data Model Mining APIs:

- `pkg_oadm_mining.refresh_mining_source` that refreshes all mining source materialized views.
- `pkg_oadm_mining.create_cust_ltv_svm_rgrsn` that generates the Customer Life Time Value Prediction Model.

Take the following steps to use the procedures:

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

   ```
   SELECT count(*) FROM dmv_cust_ltv_src;
   exec pkg_oadm_mining.refresh_mining_source;
   SELECT count(*)FROM dmv_cust_ltv_src;
   ```

   These statements:

   a. Displays the number of records in `dmv_cust_ltv_src` materialized view before materialized view refresh.

   b. Refreshes mining source materialized views.

   c. Displays the number of records in `dmv_cust_ltv_src` materialized view after materialized view refresh.

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

   ```
   SELECT count(*) FROM dwd_cust_mnng;
   SELECT count(*) FROM dwd_cust_ltv_svm_factor;
   EXEC pkg_oadm_mining. create_cust_ltv_svm_rgrsn;
   SELECT count(*) FROM dwd_cust_mnng;
   SELECT count(*) FROM dwd_cust_ltv_svm_factor;
   ```

   These statements:

   a. Shows the records count in `dwd_cust_mnng` table before data mining model build.

b. Shows the records counts in the `dwd_cust_ltv_svm_factor` table before data mining model build.

c. Trains data mining model.

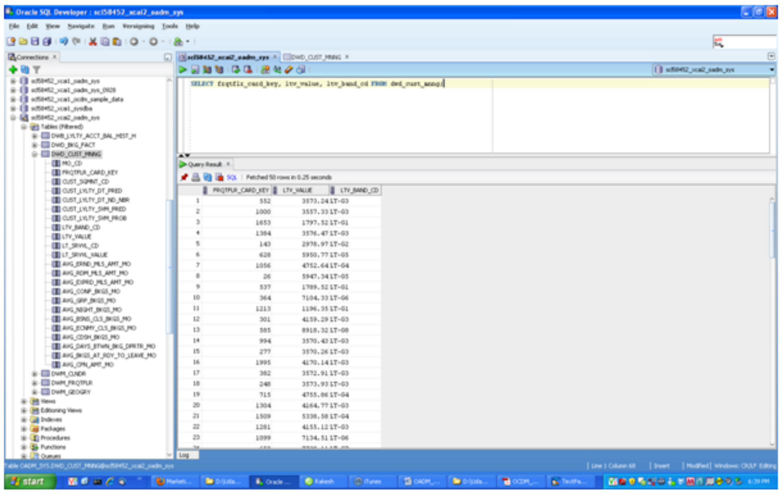
d. Shows the records count in the `dwd_cust_mnng` table after data mining model build.

e. Shows the records count in the `dwd_cust_ltv_svm_factor` table after data mining model build.

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

1. Issue the following query.

```
SELECT frqtflr_card_key, ltv_value, ltv_band_cd FROM dwd_cust_mnng;
```



```
SELECT frqtflr_card_key, ltv_value, ltv_band_cd FROM dwd_cust_mnng;
```

Fetched 50 rows in 0.25 seconds

```
     FRQTFLR_CARD_KEY  LTV_VALUE  LTV_BAND_CD
 1                552    3573.24  LT-G3
 2               1000    3557.33  LT-G3
 3               1653    1797.52  LT-G1
 4               1384    3576.47  LT-G3
 5                143    2978.97  LT-G2
 6                628    5950.77  LT-G5
 7               1056    4752.64  LT-G4
 8                 26    5947.34  LT-G5
 9                537    1789.52  LT-G1
10                364    7104.33  LT-G6
11               1213    1196.35  LT-G1
12                301    4159.29  LT-G3
13                585    8918.32  LT-G8
14                994    3570.43  LT-G3
15                277    3570.26  LT-G3
16               1995    4170.14  LT-G3
17                382    3572.91  LT-G3
18                248    3573.93  LT-G3
19                715    4755.86  LT-G4
20               1304    4164.77  LT-G3
21               1509    5338.58  LT-G4
22               1281    4155.12  LT-G3
23               1099    7134.51  LT-G6
```

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

Aggregate Tables in the Oracle Airlines Data Model

Aggregate tables are tables that aggregate or "roll up" the data to one level higher than a base or derived table. The aggregate tables in the default Oracle Airlines Data Model 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 while the dimensions of that fact table are `DWM_` tables.

The default Oracle Airlines Data Model defines several aggregate tables. For example, the `DWA_DLY_BKG_FACT` table aggregates the values of the `DWD_BKG_FACT` table to the day, segment, and traffic category levels.

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

If, during fit-gap analysis, you identified a need for 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 Airlines Data Model" on page 3-20 and "Defining New Oracle OLAP Cubes for Oracle Airlines Data Model" on page 3-16.

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

Dimensional Components in the Oracle Airlines 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 Airlines Data Model. The foundation layer of the Oracle Airlines Data Model is a 3NF model. The default Oracle Airlines 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 Airlines 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 Airlines Data Model
- Defining New Oracle OLAP Cubes for Oracle Airlines Data Model
- Changing an Oracle OLAP Cube in Oracle Airlines Data Model
- Creating a Forecast Cube for Oracle Airlines Data Model
- Choosing a Cube Partitioning Strategy for Oracle Airlines Data Model
- Choosing a Cube Data Maintenance Method for Oracle Airlines 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 Booking or Partner Earning). Measures are sometimes called "facts". Measures are organized by one or more dimensions and may be stored or calculated at query time:
  - **Stored Measures**. Stored measures are loaded and stored at the leaf level. Commonly, there is also a percentage of summary data that is stored. Summary data that is not stored is dynamically aggregated when queried.
  - **Calculated Measures.** Calculated measures are measures whose values are calculated dynamically at query time. Only the calculation rules are stored in the database. Common calculations include measures such as ratios, differences, moving totals, and averages. Calculations do not require disk storage space, and they do not extend the processing time required for data maintenance.
- **Dimensions.** A dimension is a structure that categorizes data to enable users to answer business questions. Commonly used dimensions are Account, Airport, and Flight, Geography, and Calendar. A dimension's structure is organized hierarchically based on parent-child relationships. These relationships enable:
  - Navigation between levels.

    Hierarchies on dimensions enable drilling down to lower levels or navigation (rolling up) to higher levels. Drilling down on a Calendar Year dimension member 2005 typically navigates you to quarters Q1 2005 through Q4 2005. In a calendar year hierarchy, drilling down on Q1 2005 would navigate you to the months, January 05 through March 05. These kinds of relationships make it easy for users to navigate large volumes of multidimensional data.
  - Aggregation from child values to parent values.

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

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

    Share and index calculations take advantage of hierarchical relationships (for example, the percentage of total ticket booking count contributed by each frequent flyer member level, or the percentage share of flown revenue for a certain segment, or sales revenue as a percentage of the geographical region for a sales agent 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 3-11.
- **Oracle OLAP Cubes**. The physical model provided with Oracle Airlines Data Model provides a dimensional perspective of the data using Oracle OLAP cubes. This dimensional model is discussed in "Characteristics of the OLAP Dimensional Model" on page 3-12.

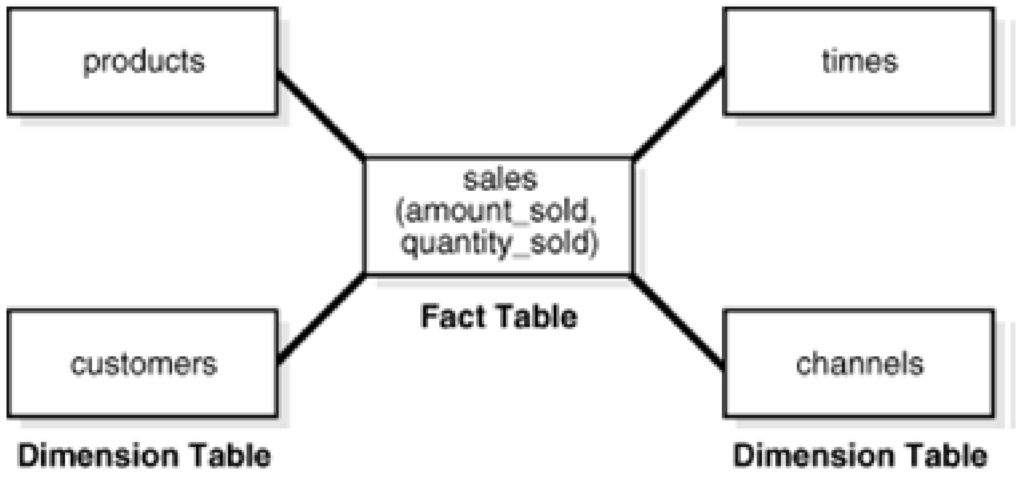
Characteristics of Relational Star and Snowflake Tables

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

A star schema borders on a physical model, as drill paths, hierarchy and query profile are embedded in the data model itself rather than the data. This in part at least, is what makes navigation of the model so straightforward for end users. Star schemas usually have a large fact table surrounded by smaller dimension tables. Dimension tables do not change very much. Most of the information that the users need are in the fact tables. Therefore, star schemas have fewer table joins than do 3NF models.

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

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 3-12 and "Validating Relational Dimension Tables" on page 3-12.

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.

Note: In the default Oracle Airlines Data Model, relational tables used as dimension tables are not defined using the `CREATE DIMENSION` command.

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.

Note: In the ETL delivered with the default Oracle Airlines Data Model, relational dimension tables are not validated.

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 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 3-14 and Chapter 5, "Report and Query Customization."
- Oracle OLAP cubes can be enhanced so that they are materialized views as described in "Cube Materialized Views" on page 3-14.

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

Benefits of Using Oracle OLAP

The benefits to your organization are significant. Oracle OLAP offers the power of simplicity: 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.

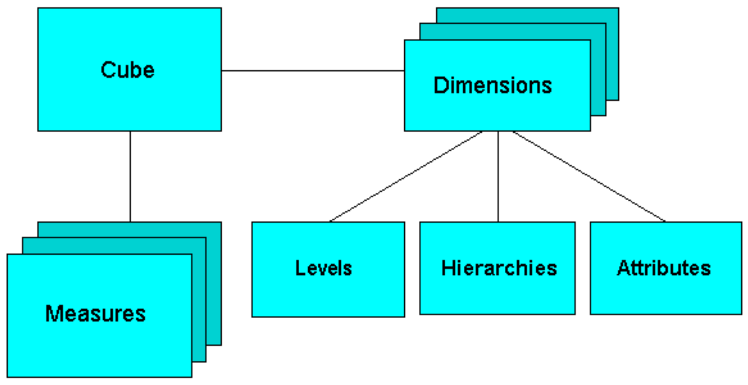
The OLAP engine manipulates dimensional objects in the same way that the SQL engine manipulates relational objects. However, because the OLAP engine is optimized to calculate analytic functions, and dimensional objects are optimized for analysis, analytic and row functions can be calculated much faster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

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

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

- All of the default OLAP cubes are loaded with data from `DWA_` tables.
- 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 Airlines Data Model are are ready to be enabled as cube materialized views (that is, `CB$` objects).

  Tip: 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 3-15.

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

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

Defining New Oracle OLAP Cubes for Oracle Airlines Data Model

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

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 Airlines Data Model" on page 3-8 for information on creating new tables.
2. Use the AWM to define new Cubes for a customized version of Oracle Airlines Data Model. Follow the instructions given for creating cubes and dimensions in *Oracle OLAP User's Guide*.

   Use the information provided in "Characteristics of the OLAP Dimensional Model" on page 3-12. and the Oracle OLAP User's Guide to guide you when you design and define new OLAP cubes. Also, if you are familiar with a relational star schema design as outlined in "Characteristics of Relational Star and Snowflake Tables" on page 3-11, then you can use this understanding to help you design an OLAP Cube as described below:

   - 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 Airlines Data Model.

See: *Oracle Airlines Data Model Installation Guide* for more information on installing the sample reports and deploying the Oracle Airlines 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 3-15.

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

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

Since all Oracle Airlines 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 Airlines 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 Airlines Data Model

To create a forecast cube for Oracle Airlines Data Model:

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

Note: You cannot enable materialized views for an Oracle Airlines 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 Airlines 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 specific 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 calculation 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 Oracle Airlines Data Model" on page 2-14, and "Partitioning and Materialized Views" on page 3-23.

Choosing a Cube Data Maintenance Method for Oracle Airlines 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 3-21

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 Airlines Data Model

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

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

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

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

Prefix	Description
`CB$`	An OLAP cube enhanced with materialized view capabilities. **See:** OLAP cube materialized views in *Oracle Airlines Data Model Reference* for a list of these objects in the default data model. "Characteristics of the OLAP Cubes in Oracle Airlines Data Model" on page 3-15 for information on OLAP cubes. **Note:** Do `not` report or query against these objects. Instead access the relational view of an OLAP cube (that is, the object with the `_VIEW` suffix).
`DMV_`	Data mining source materialized views. **See:** *Oracle Airlines 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 Airlines 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 Airlines 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 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 Oracle Airlines Data Model" on page 2-14

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 Airlines Data Model" on page 2-14, "Indexes and Partitioned Indexes in Oracle Airlines Data Model" on page 2-14, and "Choosing a Cube Partitioning Strategy for Oracle Airlines Data Model" on page 3-18

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 Airlines Data Model" on page 2-10, and "Aggregate Tables in the Oracle Airlines Data Model" on page 3-8.

4 ETL Implementation and Customization

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

The Role of ETL in the Oracle Airlines Data Model

Figure 2–1, "Layers of an Oracle Airlines Data Model Warehouse" on page 2-2 illustrated the three layers in Oracle Airlines Data Model warehouse environment: the optional staging layer, the foundation layer, and the access layer. 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 OLTP system is known as source ETL.

  Oracle Airlines Data Model does *not* include source-ETL scripts. You must create source-ETL yourself using your understanding of your OLTP system and your customized Oracle Airlines Data Model. See "ETL for the Foundation Layer of an Oracle Airlines 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 Airlines Data Model *does* include intra-ETL. You can modify the default intra-ETL to populate a customized access layer from a customized foundation layer. See "Customizing Intra-ETL for the Oracle Airlines Data Model" on page 4-8 for more information on the intra-ETL.

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

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

You populate the foundation layer of an Oracle Airlines Data Model warehouse by writing your own source-ETL scripts using Oracle Warehouse Builder or another ETL tool and then use those scripts to populate the foundation layer.

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

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

Source-ETL Design Considerations

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

- You can populate the calendar data by using the calendar population scripts provided with Oracle Airlines Data Model and described in *Oracle Airlines 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 lookup tables before loading the reference 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 Airlines Data Model Source-ETL

ETL 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 sin data warehouse implementation. The difference between them is where the transformation proceed in or out 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.

When designing source-ETL for Oracle Airlines 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.

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.

Then, 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 regard to 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 Airlines 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 Golden Gate)
- 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 Airlines 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 Airlines Data Model warehouse.

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

Assume that there is a large table called `DWB_PNR_H`, which is range partitioned by day. At the end of each business day, data from the online airlines system is loaded into the `DWB_PNR_H` 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_pnr` that has the same column structure as `DWB_PNR_H` table
3. Build any indexes that are on the `DWB_PNR_H` table on the `tmp_pnr` table
4. Issue the `EXCHANGE PARTITION` command.

   ```
   Alter table dwb_pnr_h exchange partition SYS_P1926 with
       table tmp_pnr 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_pnr 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 the Oracle Airlines Data Model

Intra-ETL is delivered as a component of Oracle Airlines Data Model. This intra-ETL is delivered as a PL/SQL package named PKG_INTRA_ETL_PROCESS which is a complete intra-ETL process that has procedures that populate the access layer.

The PKG_INTRA_ETL_PROCESS package is composed of individual sub-process procedures and functions that respect the dependency of each individual program. The main procedures execute in the following order:

1. Populate_Dimension - Populates the dimension (DWM_) tables based on the content of the reference (DWR_) tables.
2. Populate_Derived - Populates the derived (DWD_) tables based on the content of the base (DWB_) tables.
3. Populate_Aggregate - Refreshes all of the aggregate (DWA_) tables using data from the dimension (DWM_) and derived (DWD_) tables.
4. Populate_Aw - Loads data from Oracle Airlines Data Model aggregate (DWA_) tables into the Oracle Airlines Data Model Analytical Workspace and calculates the forecast data. It reads OLAP ETL parameters from DWC_OLAP_ETL_PARM table.
5. Populate_MINING - Triggers the data mining models.

The PKG_INTRA_ETL_PROCESS package is designed to work with the default Oracle Airlines Data Model. If you have customized the model, you also need to customize the intra-ETL. To customize the intra-ETL delivered with Oracle Airlines Data Model, take the following steps:

1. Familiarize yourself with the PKG_INTRA_ETL_PROCESS package. In particular, understand the procedures, data flows, and maps as described in *Oracle Airlines Data Model Reference*, and how the procedures use the DWC_ETL_PARAMETER and DWC_OLAP_ETL_PARM control tables, and the DWC_INTRA_ETL_PROCESS and DWC_INTRA_ETL_ACTIVITY control tables. Review the following topics: "Performing an Initial Load of the Access Layer" on page 4-9, "Refreshing the Access Layer of an Oracle Airlines Data Model Warehouse" on page 4-11, and "Managing Errors During Oracle Airlines Data Model Intra-ETL Execution" on page 4-13.
2. Identify the changes that you have made to the Oracle Airlines Data Model and what changes need to be made to the intra-ETL needs to support those model changes.
3. Make a copy of the packages that you need to change. Give the copies a different name.
4. Make the changes you identified in Step 2 to the packages you created in Step 3.

Optionally, you can create new intra-ETL from scratch - either by writing your own PL/SQL code or by using an ETL tool such as Oracle Warehouse Builder.

Performing an Initial Load of an Oracle Airlines Data Model Warehouse

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

1. Load the foundation layer of the Oracle Airlines Data Model warehouse (that is, the reference, lookup, and base tables) as described in "Performing an Initial Load of the Foundation Layer" on page 4-9.
2. Load the access layer of the Oracle Airlines Data Model warehouse (that is, the derived and aggregate tables, the tables that dimension the derived and aggregate tables, materialized views, OLAP cubes, and data mining models) as described in "Performing an Initial Load of the Access Layer" on page 4-9.

Performing an Initial Load of the Foundation Layer

You perform the initial load of the foundation layer using source-ETL that you create. See "ETL for the Foundation Layer of an Oracle Airlines Data Model Warehouse" on page 4-2 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 Airlines 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 `oadm_sys` schema so that the ETL can use this information (that is, the beginning and end date of the ETL period) when loading the data into the access layer.

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

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

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

2. Update the Oracle Airlines Data Model OLAP ETL parameters in `DWC_OLAP_ETL_PARM` control table in the `oadm_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 Oracle Airlines Data Model OLAP ETL Parameters in the DWC_ OLAP_ETL_PARM Table for Initial Load

Column Name	Value
`PROCESS_NAME`	`'OADM-INTRA-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 Airlines 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.
`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 *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 *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 in one of the ways described in "Executing the Default Oracle Airlines Data Model Intra-ETL" on page 4-10.

   See also: "Refreshing the Data in an Oracle Airlines Data Model Warehouse" on page 4-11

Executing the Default Oracle Airlines Data Model Intra-ETL

You execute the `PKG_INTRA_ETL_PROCESS` process flow from an Oracle client tool (for example, ex: SQL Plus) by issuing the following statement.

```
EXEC PKG_INTRA_ETL_PROCESS.RUN( )
```

The PL/SQL code executed by `PKG_INTRA_ETL_PROCESS`:

- Reads the values from the `DWC_ETL_PARAMETER` and `DWC_OLAP_ETL_PARM` control tables in the `oadm_sys` schema before executing the mappings in the correct order.

- The result of each table loading are tracked in the `DWC_INTRA_ETL_PROCESS` and `DWC_INTRA_ETL_ACTIVITY` control tables.

Executing the `PKG_INTRA_ETL_PROCESS` from within an Oracle client provides the ability to monitor the execution of the process. However, you can simply execute the `RUN` procedure within the `PKG_INTRA_ETL_PROCESS` PL/SQL package.

In either case, you can execute the intra-ETL explicitly or invoke its execution in some other program or process (for example, the source-ETL process after its successful execution) or through a predefined schedule (for example, using Oracle Job Scheduling feature and so on).

> **See:** "Monitoring the Execution of the Intra-ETL Process" on page 4-13, "Recovering an Intra ETL Process" on page 4-14, and "Troubleshooting Intra-ETL Performance" on page 4-14.

Refreshing the Data in an Oracle Airlines Data Model Warehouse

"Performing an Initial Load of the Access Layer" on page 4-9 describes how to perform an initial load of an Oracle Airlines Data Model data warehouse. After this initial load, you must load new data into your Oracle Airlines Data Model data warehouse regularly so that it can serve its purpose of facilitating business analysis.

To load new data into your Oracle Airlines 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 and are 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 Airlines Data Model warehouse, or you can refresh the data sequentially.:

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

In either case, you can manage errors during the execution of the intra-ETL as described in "Managing Errors During Oracle Airlines Data Model Intra-ETL Execution" on page 4-13.

Refreshing the Foundation Layer of Oracle Airlines Data Model Warehouse

You 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 "ETL for the Foundation Layer of an Oracle Airlines Data Model Warehouse" on page 4-2.

Refreshing the Access Layer of an Oracle Airlines Data Model Warehouse

Refreshing the access layer of an Oracle Airlines Data Model is a multi-step process. You can do a full incremental load of the access layer all at one time by executing the `PKG_INTRA_ETL_PROCESS` package as described in"Executing the Default Oracle Airlines Data Model Intra-ETL" on page 4-10, or you can refresh the data sequentially:

1. Refreshing the Access Layer Relational Tables in the Oracle Airlines Data Model

2. Refreshing Oracle Airlines Data Model OLAP Cubes
3. Refreshing Oracle Airlines 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 Airlines Data Model Intra-ETL Execution" on page 4-13.

Refreshing the Access Layer Relational Tables in the Oracle Airlines Data Model

After you have refreshed the foundation layer of the Oracle Airlines Data Model. You can refresh only the relational tables and views in the access layer of an Oracle Airlines Data Model by taking the following steps:

1. Update the parameters of the `DWC_ETL_PARAMETER` control table in the `oadm_sys` schema. For an incremental load of an Oracle Airlines Data Model warehouse, specify the values shown in the following table (that is, the beginning and end date of the ETL period).

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

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

2. Refresh the tables by executing the following procedures in the `PKG_INTRA_ETL_PROCESS` PL/SQL package as described in "Executing the Default Oracle Airlines Data Model Intra-ETL" on page 4-10. Execute the procedures in the following order:
   a. `Populate_Dimension`
   b. `Populate_Derived`
   c. `Populate_Aggregate`

Refreshing Oracle Airlines 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 Airlines Data Model data warehouse since the initial load of the OLAP cubes.

Take these steps to refresh *only* the OLAP cube data in the Oracle Airlines Data Model warehouse:

1. Ensure that the underlying aggregate tables are refreshed. See "Refreshing the Access Layer Relational Tables in the Oracle Airlines Data Model" on page 4-12 for more information.
2. Execute the `PKG_INTRA_ETL_PROCESS.Populate_Aw` procedure to load the cube data.
3. If necessary, recover from errors that happen during the execution of `Populate_Aw` by taking the following steps.
   a. Change the value of the `BUILD_METHOD` column of the `DWC_OLAP_ETL_PARM` table to `"C"`.

b. Re-execute `PKG_INTRA_ETL_PROCESS.Populate_Aw`.

Refreshing Oracle Airlines Data Model Data Mining Models

The `PKG_INTRA_ETL_PROCESS.Populate_Mining` procedure triggers the data mining model refreshment as part of the initial load of the warehouse. After the initial load of the warehouse, it is recommended that you refresh the data mining models monthly.

After you have refreshed the OLAP cubes, you can also refresh the data mining models. In this case, the way you refresh a data mining model varies depending on whether you want to refresh all of the data mining models or only one data mining model:

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

  ```
  oadm_sys.pkg_oadm_mining.refresh_model(p_month_code,p_process_no)
  ```

  This procedure performs the following tasks for each data mining model:

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

- To manually re-create only *one* data mining model, you can call the corresponding `oadm_sys.pkg_oadm_mining.create_` procedure.

  For example, to re-create the Customer Life Time Value regression data mining model, call the following procedure.

  ```
  oadm_sys.pkg_oadm_mining.create_cust_ltv_svm_rgrsn(p_month_cd);
  ```

  "Tutorial: Customizing the Customer Life Time Value Prediction Data Mining Model" on page 3-4 provides detailed instructions for refreshing a single data mining model.

  See also: "Troubleshooting Data Mining Model Creation" on page 4-15

Managing Errors During Oracle Airlines Data Model Intra-ETL Execution

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

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

Monitoring the Execution of the Intra-ETL Process

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

Each normal run of the `PKG_INTRA_ETL_PROCESS` package (as opposed to an error-recovery run) 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, input date range in the fields FROM_DATE_ETL and TO_DATE_ETL.

2. Invokes each of the individual PKG_INTRA_ETL_PROCESS procedures 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 a system generated unique activity key, the process key value corresponding to the Intra-ETL process, individual program name as the Activity Name, a suitable activity description, SYSDATE as activity start time, RUNNING as the activity status.
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'. If 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. If all the individual programs succeed, the procedure updates the status to 'COMPLETED-SUCCESS', otherwise it updates the status to 'COMPLETED-ERROR'.

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

Recovering an Intra ETL Process

To recover run of the PKG_INTRA_ETL_PROCESS package:

1. Identify the errors by looking at the corresponding error details that are tracked against the individual programs in the DWC_INTRA_ETL_ACTIVITY table.
2. Correct the causes of the errors.
3. Re-execute the PKG_INTRA_ETL_PROCESS package.

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

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

Troubleshooting Intra-ETL Performance

To troubleshoot the performance of the intra-ETL:

- Check the execution plan as described in "Checking the Execution Plan" on page 4-15.
- Monitor parallel DML executions as described in "Monitoring PARALLEL DML Executions" on page 4-15.
- Check that data mining models were created correctly as described in "Troubleshooting Data Mining Model Creation" on page 4-15.

Checking the Execution Plan

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

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

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

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

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

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

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

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

Monitoring PARALLEL DML Executions

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

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

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

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

> **See also:** "Parallel Execution in the Oracle Airlines Data Model" on page 2-16

Troubleshooting Data Mining Model Creation

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

```
set line 160
```

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

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

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

- Example 4–2, "Troubleshooting the "Message not available ... [Language=ZHS]" Error"
- Example 4–3, "Troubleshooting ORA-40112: insufficient number of valid data rows,"
- Example 4–4, "Troubleshooting ORA-40113: insufficient number of distinct target values"

Example 4–2 Troubleshooting the "Message not available ... [Language=ZHS]" Error

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

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

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

Example 4–3 Troubleshooting ORA-40112: insufficient number of valid data rows,

Assume that the returned error is `ORA-40112: insufficient number of valid data rows, for "create_cust_ltv_svm_rgrsn" model`.

For this model, the target column is `oadm_sys.dmv_cust_ltv_src.tot_cpn_amt`.

To troubleshoot this error:

1. Execute the following SQL statements.

   ```
   SELECT count(tot_cpn_amt) FROM dmv_cust_ltv_src;
   SELECT count(frqtflr_card_key) FROM dmv_cust_ltv_src;
   ```

2. Check that the values returned by above two queries are same and greater than 0 (zero). If value returned by the first `SELECT` statement is smaller than value returned by the second `SELECT` statement, then check the source tables and materialized views of `dmv_cust_ltv_src`.

Example 4–4 Troubleshooting ORA-40113: insufficient number of distinct target values

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

For a two-class classification model, the target column should have two distinct values. This error happens when the target column for the training model contains only one value or no value when it is expecting two values.

For example, for frequent fliers prediction among non-frequent fliers svm model, the target column is `oadm_sys.dmv_ffp_pred_src.ff_ind`.

To troubleshoot this error:

1. Execute a SQL query to check if there are enough values in this column. Using the frequent flier prediction svm model as an example, issue the following statement.

   ```
   SELECT ff_ind, count(*) FROM dmv_ffp_pred_src GROUP BY ff_ind;
   ```

   The result of the query is shown below.

   ```
   FF_IND       COUNT(*)
   ------       --------
        1           1296
        0            990
   ```

2. Check the following tables to make sure that there are both frequent fliers and non-frequent fliers by issuing the following query.

   ```
   SELECT NVL2(frqtflr_nbr,'FFP','Non_FFP') AS ffp_ind, count(*)
      FROM oadm_sys.dwd_bkg_fact GROUP BY NVL2(frqtflr_nbr,'FFP','Non_FFP');
   ```

   The result of the query is shown below

   ```
   FFP_IND       COUNT(*)
   -------       --------
   Non-FFP          77353
   FFP              42647
   ```

3. Execute the following statement to refresh the mining source materialized views:

   ```
   exec pkg_oadm_mining.refresh_mining_source;
   ```

5 Report and Query Customization

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

Reporting Approaches in Oracle Airlines Data Model

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

- **Relational Reporting**

  With relational reporting, you create reports against the analytical layer entities using the fact tables as the center of the star with the dimension or lookup tables acting as the dimensions of the star. Typically the facts are derived (`DWD_`) and aggregate (`DWA_`) tables with `DWM_` dimension tables. However in some cases, you may need to use base (`DWB_`) tables along with reference tables (`DWR_`) or lookup (`DWL_`) tables to generate more detailed reports.

  – The dimension (`DWM_`) 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 you to attach the appropriate set of reference tables for the multiple subject area and fact entities composed of differing granularity. For example, you can use the time dimension table `DWM_CLNDR` to query against a `DAY` level Passenger Name Record (PNR) data such as `DWD_PNR`.

  – Reference tables (`DWR_`) and lookup (`DWL_`) tables represent the simpler dimensions comprising 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.

- **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 views of the dimension and cube views are relational views with names ending with `_VIEW`:

  – OLAP dimension views contain 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`). You can think of OLAP dimensions as "flattened" views of snowflake dimension tables. Typically, in Oracle Airlines Data Model, a dimension view is named *`dimension_hierarchy_`*`VIEW`

  – Cube views 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. Typically, in Oracle Airlines Data Model, a cube view is named *`cube`*`_VIEW`.

  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 Airlines Data Model reports. For examples of Oracle Airlines Data Model reports, see:

- "Writing As Is and As Was Queries" on page 5-7
- "Tutorial: Creating a New Oracle Airlines Data Model Dashboard" on page 5-13
- "Tutorial: Creating a New Oracle Airlines Data Model Report" on page 5-17
- The sample reports provided with Oracle Airlines Data Model that are documented in *Oracle Airlines Data Model Reference*.

Customizing Oracle Airlines Data Model Sample Reports

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

> **See:** *Oracle Airlines Data Model Installation Guide* for more information on installing the sample reports and deploying the Oracle Airlines 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 Airlines Data Model Reference* for detailed information on the sample reports.

Note: The reports and dashboards that are used in examples and delivered with Oracle Airlines Data Model are provided only for demonstration purposes. They are not supported by Oracle.

Writing Your Own Queries and Reports

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

Oracle Airlines Data Model also supports On Line Analytic Processing (OLAP) reporting using OLAP cubes defined in the `oadm_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 Airlines Data Model" on page 5-1, "Oracle OLAP Cube Views" on page 3-14, and the discussion on querying dimensional objects in *Oracle OLAP User's Guide*.

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

For example, assume that you want to know the total booking count for the business class for February 2010. To answer this question, you might have to query the tables described in the following table..

Entity Name	Table Name	Description
Daily Booking Fact	`DWA_DLY_BKG_FACT`	Stores the booking related fact data.
Booking Class Fact	`DWM_BKG_CLS_TYP`	The booking class type like business or economy.
Calendar	`DWM_CLNDR`	Stores the calendar information.

To make the query, you execute the following SQL statement.

```
select DWM_CLNDR.CLNDR_MONTH_CD,DWM_BKG_CLS_TYP.SVC_CLS_DESC,sum(DWA_DLY_BKG_
FACT.BKD) as Booking_Count
```

```
from DWA_DLY_BKG_FACT,DWM_CLNDR,DWM_BKG_CLS_TYP
where DWM_CLNDR.CLNDR_KEY=DWA_DLY_BKG_FACT.BKG_DT_KEY
and DWM_BKG_CLS_TYP.BKG_CLS_KEY=DWA_DLY_BKG_FACT.BKG_CLS_KEY
and DWM_BKG_CLS_TYP.SVC_CLS_DESC='BUSINESS'
and DWM_CLNDR.CLNDR_MONTH_KEY=20100201
group by DWM_CLNDR.CLNDR_MONTH_CD, DWM_BKG_CLS_TYP.SVC_CLS_DESC
```

The result of this query is shown below.

```
CLNDR_MONTH_CD            SVC_CLS_DESC            BOOKING_COUNT
------------------------- --------------------    ----------------------
20100201                  BUSINESS                61
```

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.

In order to optimize, you simply:

- 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 3-11 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=2008;
```

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

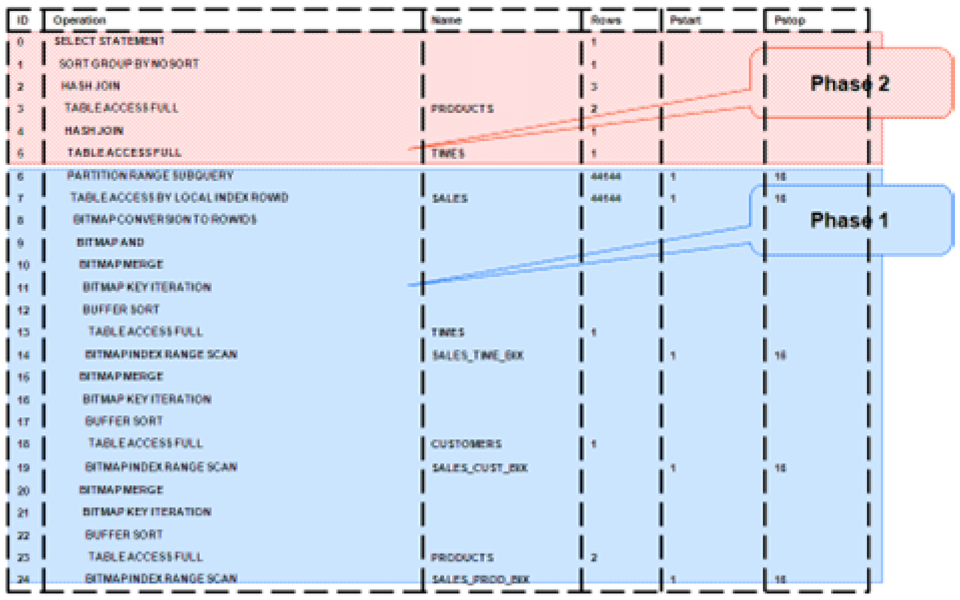
```
select SUM(quantity_sold
From Sales
Where cust_id IN
(select c.cust_id From Customers c Where c.cust_city='BOSTON')
And s.prod_id IN
(select p.prod_id From Products p Where  p.product='UMBRELLA')
And s.time_id IN
(select t.time_id From Times(Where t.month='MAY' And tyear=2008);
```

   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 how you imagined it. You may have noticed that you can see that 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.


Phase 2
Phase 1

ID	Operation	Name	Rows	Pstart	Pstop
0	SELECT STATEMENT		1		
1	SORT GROUP BY NOSORT		1		
2	HASH JOIN		3		
3	TABLE ACCESS FULL	PRODUCTS	2		
4	HASH JOIN		1		
5	TABLE ACCESS FULL	TIMES	1		
6	PARTITION RANGE SUBQUERY		44544	1	16
7	TABLE ACCESS BY LOCAL INDEX ROWID	SALES	44544	1	16
8	BITMAP CONVERSION TO ROWIDS				
9	BITMAP AND				
10	BITMAP MERGE				
11	BITMAP KEY ITERATION				
12	BUFFER SORT				
13	TABLE ACCESS FULL	TIMES	1		
14	BITMAP INDEX RANGE SCAN	SALES_TIME_BIX		1	16
15	BITMAP MERGE				
16	BITMAP KEY ITERATION				
17	BUFFER SORT				
18	TABLE ACCESS FULL	CUSTOMERS	1		
19	BITMAP INDEX RANGE SCAN	SALES_CUST_BIX		1	16
20	BITMAP MERGE				
21	BITMAP KEY ITERATION				
22	BUFFER SORT				
23	TABLE ACCESS FULL	PRODUCTS	2		
24	BITMAP INDEX RANGE SCAN	SALES_PROD_BIX		1	16

Troubleshooting Oracle Airlines 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 **oadm**.

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

2. Open the Oracle Airlines 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 Airlines 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 Airlines 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 Booking Segment Departure Fact) does not contain mapping
for [Booking Segment Departure Fact.Confirmed Count, Booking Segment Departure
Fact.Belly Cargo Revenue].
```

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 mappings for `Booking Segment Departure Fact.Confirmed Count` and `Booking Segment Departure Fact.Belly Cargo Revenue`.

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

Characteristics of an As Is Query

An As Is query has the following characteristics:

- The resulting report shows the data as it happened.
- The snowflake dimension tables are also joined using the surrogate key columns (that is the primary key and foreign key columns).
- The fact table is joined with the dimension tables (at leaf level) using the surrogate key column.
- Slowly-changing data in the dimensions are joined with their corresponding fact records and are presented individually.

- It is possible to add up the components if the different versions share similar characteristics.

Characteristics of an As Was Query

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

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

Examples: As Is and As Was Queries

Based on the Data used for the examples on page 5-8, 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, 2011, 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-11	31-Dec-99
102	JaD	Jane Doe	Female	Single	5001	SV	Sunnyvale	...	1-Jan-11	31-Dec-99
103	JiD	Jim Doe	Male	Married	5002	CU	Cupertino	...	1-Jan-11	31-Dec-99

County Table

County Id	County CD	County Nm	Population	...	Eff Frm	Eff To
5001	SV	Sunnyvale	Very High	...	1-Jan-11	31-Dec-99
5002	CU	Cupertino	High	...	1-Jan-11	31-Dec-99

TaxPaid Table

Cust Id	Day	Tax Type	Tax
101	1-Jan-11	Professional Tax	100
102	1-Jan-11	Professional Tax	100
103	1-Jan-11	Professional Tax	100

Assume that the following events occurred in January 2011:

- On January 20, 2011, Jane Doe marries.
- On Jan 29, 2011, John Doe moves from Sunnyvale to Cupertino.

Consequently, as shown below, on February 1, 2011, 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-11	29-Jan-11
102	JaD	Jane Doe	Female	Single	5001	SV	Sunnyvale	...	1-Jan-11	20-Jan-11
103	JiD	Jim Doe	Male	Married	5002	CU	Cupertino	...	1-Jan-11	31-Dec-99
104	JaD	Jane Doe	Female	Married	5001	SV	Sunnyvale	...	21-Jan-11	31-Dec-99
105	JoD	John Doe	Male	Single	5002	CD	Cupertino	...	30-Jan-11	31-Dec-99

County table

County Id	County CD	County Nm	Population	...	Eff Frm	Eff To
5001	SV	Sunnyvale	Very High	...	1-Jan-11	31-Dec-99
5002	CU	Cupertino	High	...	1-Jan-11	31-Dec-99

TaxPaid Table

Cust Id	Day	Tax Type	Tax
101	1-Jan-11	Professional Tax	100
102	1-Jan-11	Professional Tax	100
103	1-Jan-11	Professional Tax	100
105	1-Feb-11	Professional Tax	100
104	1-Feb-11	Professional Tax	100
103	1-Feb-11	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 5-8, 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 5-8, issue the following SQL statement to show the tax collection data split by marital status using an analysis date of January 15, 2011.

```
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-2011', '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-2011', '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, 2011 (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-2011` rather than `15-JAN-2011`. Since Jane Doe was married on February 9, 2011, 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 5-8, 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 5-8, issue the following SQL statement to show the tax collection data split by county using an analysis date of January 15, 2011.

```
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-2011', '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-2011', '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, 2011, 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-2011` rather than `15-JAN-2011`. Since John Doe lived in Cupertino on February 9, 2011, 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

Cust Nm	County Nm	Tax
John Doe	Cupertino	200

Tutorial: Creating a New Oracle Airlines Data Model Dashboard

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

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

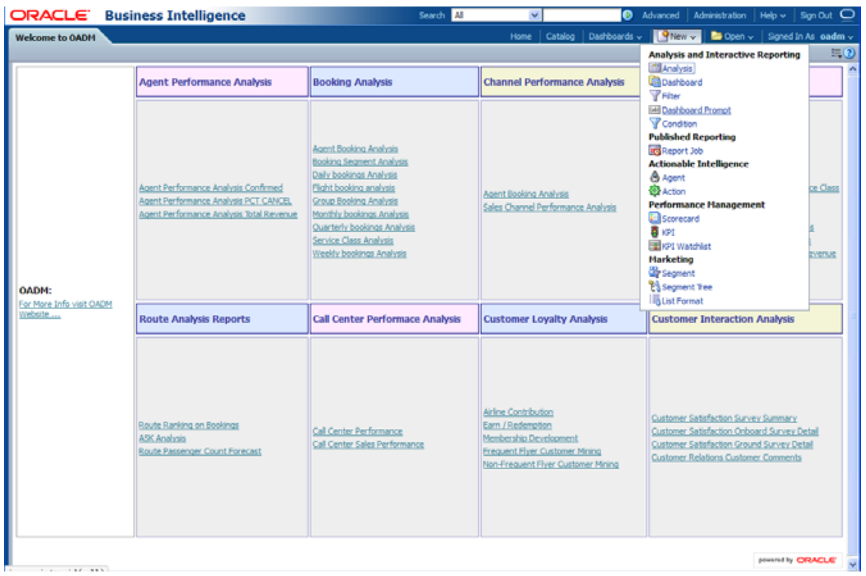
In this example assume that you want to create a dashboard named "Agent Revenue and Booking Analysis", and put both "Agent Revenue Analysis in USD" and "Agent Performance Analysis PCT CANCEL" into this new dashboard.

To create a dashboard, take the following steps:

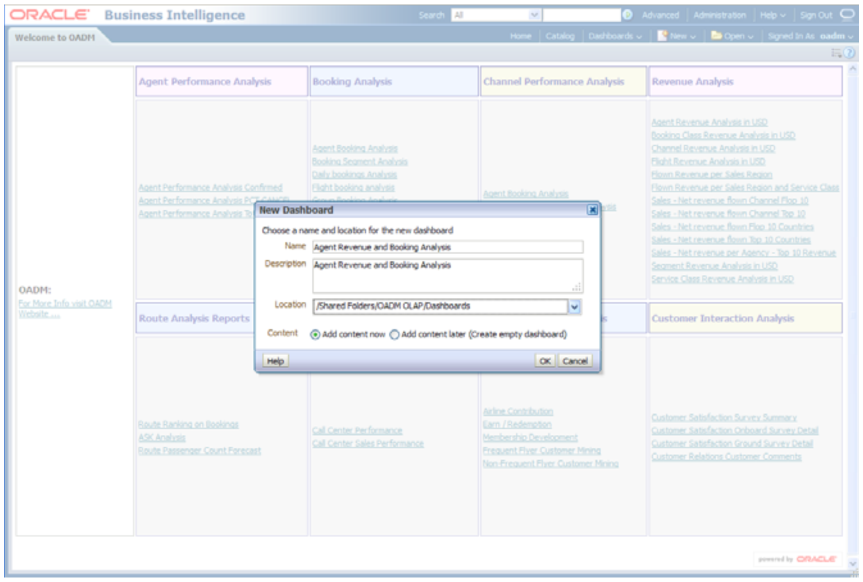
1. In the browser, open the login page at `http://`*`servername`*`:9704/analytics` where *`servername`* is the server on which the webcat is installed.

   Login with username of `oadm`, and provide the password.

2. Select **New**, and then select **Dashboard** to create an Oracle Business Intelligence Suite Enterprise Edition dashboard.


ORACLE Business Intelligence
Search All
Advanced
Administration
Help
Sign Out
Welcome to OADM
Home
Catalog
Dashboards
New
Open
Signed In As oadm
Agent Performance Analysis
Agent Performance Analysis Confirmed
Agent Performance Analysis PCT CANCEL
Agent Performance Analysis Total Revenue
Booking Analysis
Agent Booking Analysis
Booking Segment Analysis
Daily bookings Analysis
Flight booking analysis
Group Booking Analysis
Monthly bookings Analysis
Quarterly bookings Analysis
Service Class Analysis
Weekly bookings Analysis
Channel Performance Analysis
Agent Booking Analysis
Sales Channel Performance Analysis
Analysis and Interactive Reporting
Analysis
Dashboard
Filter
Dashboard Prompt
Condition
Published Reporting
Report Job
Actionable Intelligence
Agent
Action
Performance Management
Scorecard
KPI
KPI Watchlist
Marketing
Segment
Segment Tree
List Format
OADM:
For More Info visit OADM Website ...
Route Analysis Reports
Route Ranking on Bookings
ASK Analysis
Route Passenger Count Forecast
Call Center Performace Analysis
Call Center Performance
Call Center Sales Performance
Customer Loyalty Analysis
Airline Contribution
Earn / Redemption
Membership Development
Frequent Flyer Customer Mining
Non-Frequent Flyer Customer Mining
Customer Interaction Analysis
Customer Satisfaction Survey Summary
Customer Satisfaction Onboard Survey Detail
Customer Satisfaction Ground Survey Detail
Customer Relations Customer Comments
powered by ORACLE

3. Enter a name and description. Then save the dashboard to the Dashboards folder. Click OK.


New Dashboard

Choose a name and location for the new dashboard

Name: Agent Revenue and Booking Analysis

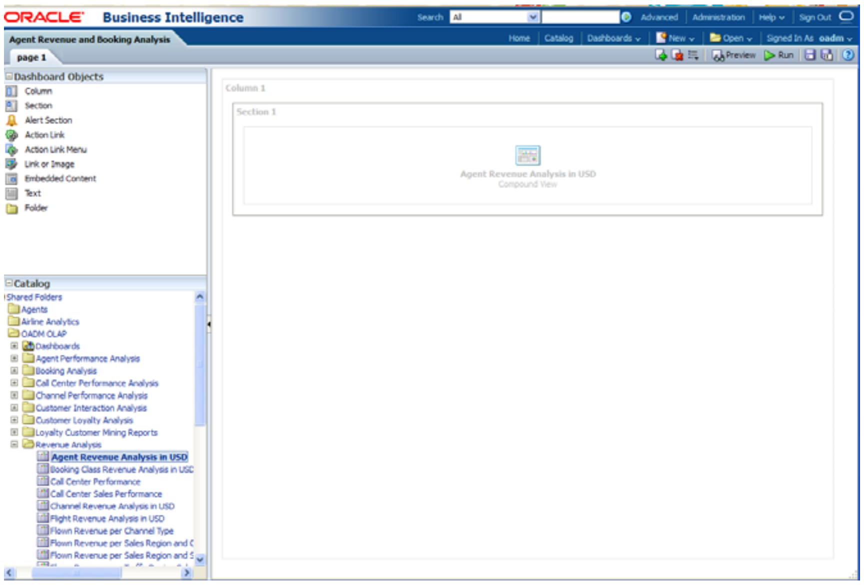
Description: Agent Revenue and Booking Analysis

Location: /Shared Folders/OADM OLAP/Dashboards

Content: [✓ Add content now] Add content later (Create empty dashboard)

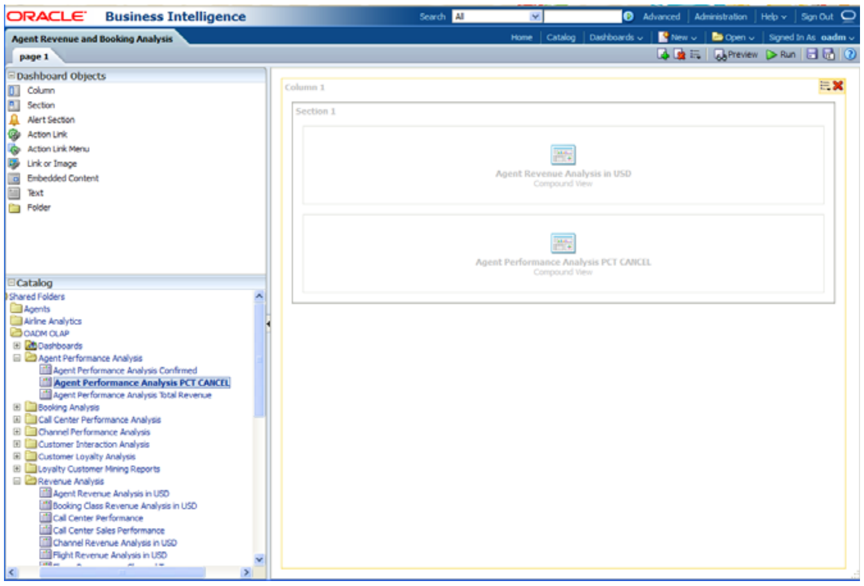
ORACLE Business Intelligence
Welcome to OADM
Agent Performance Analysis
Booking Analysis
Channel Performance Analysis
Revenue Analysis
Route Analysis Reports
Customer Interaction Analysis
Help
OK
Cancel

4. In the Catalog view, expand the Revenue Analysis folder. You can see the Agent Revenue Analysis in USD Report, drag it from catalog view into the right panel.

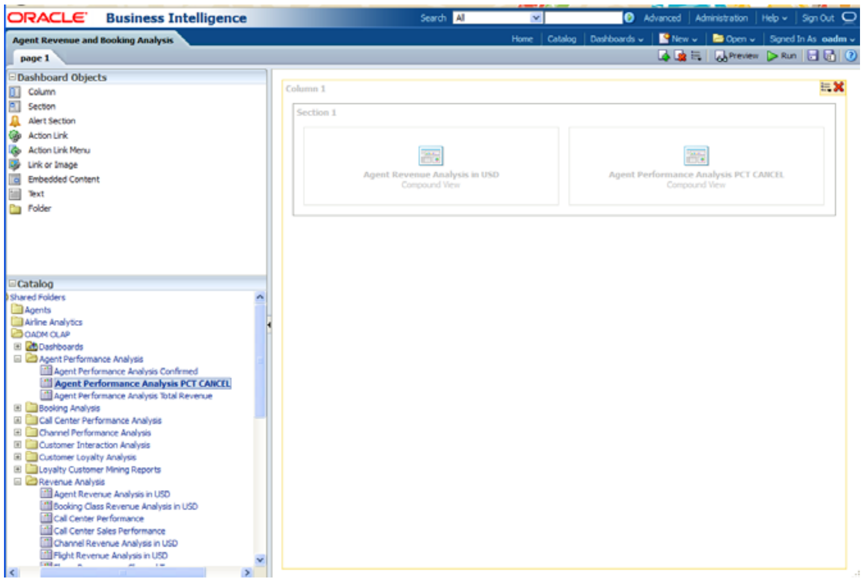


ORACLE Business Intelligence
Agent Revenue and Booking Analysis
page 1
Dashboard Objects
Column
Section
Alert Section
Action Link
Action Link Menu
Link or Image
Embedded Content
Text
Folder
Catalog
Shared Folders
Agents
Airline Analytics
OADM OLAP
Dashboards
Agent Performance Analysis
Booking Analysis
Call Center Performance Analysis
Channel Performance Analysis
Customer Interaction Analysis
Customer Loyalty Analysis
Loyalty Customer Mining Reports
Revenue Analysis
Agent Revenue Analysis in USD
Booking Class Revenue Analysis in USD
Call Center Performance
Call Center Sales Performance
Channel Revenue Analysis in USD
Flight Revenue Analysis in USD
Flown Revenue per Channel Type
Column 1
Section 1
Agent Revenue Analysis in USD
Compound View
Preview
Run

5. In the Catalog view, expand the Revenue Analysis folder. You can see the Agent Performance Analysis PCT CANCEL Report. Drag that report from the Catalog view into the right panel.


ORACLE Business Intelligence
Agent Revenue and Booking Analysis
page 1
Dashboard Objects
Column
Section
Alert Section
Action Link
Action Link Menu
Link or Image
Embedded Content
Text
Folder
Catalog
Shared Folders
Agents
Airline Analytics
OADM OLAP
Dashboards
Agent Performance Analysis
Agent Performance Analysis Confirmed
Agent Performance Analysis PCT CANCEL
Agent Performance Analysis Total Revenue
Booking Analysis
Call Center Performance Analysis
Channel Performance Analysis
Customer Interaction Analysis
Customer Loyalty Analysis
Loyalty Customer Mining Reports
Revenue Analysis
Agent Revenue Analysis in USD
Booking Class Revenue Analysis in USD
Call Center Performance
Call Center Sales Performance
Channel Revenue Analysis in USD
Flight Revenue Analysis in USD
Column 1
Section 1
Agent Revenue Analysis in USD
Compound View
Agent Performance Analysis PCT CANCEL
Compound View

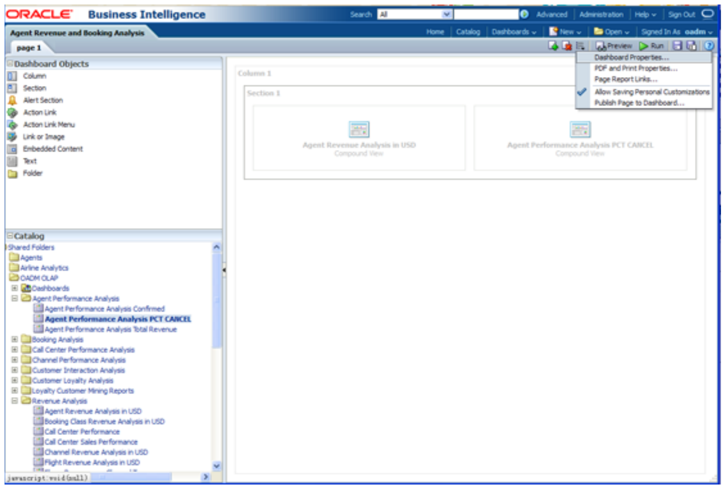
6. You can change the layout of this section to organize the two reports by horizontal or vertical.


ORACLE Business Intelligence
Agent Revenue and Booking Analysis
page 1
Dashboard Objects
Catalog
Column 1
Section 1
Agent Revenue Analysis in USD
Compound View
Agent Performance Analysis PCT CANCEL
Compound View

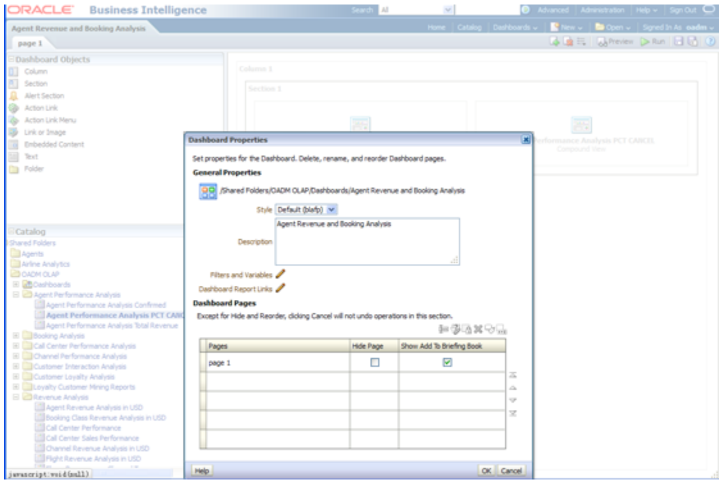
Note that the page name is still "Page1" so you must change it.

7. To change the page name:

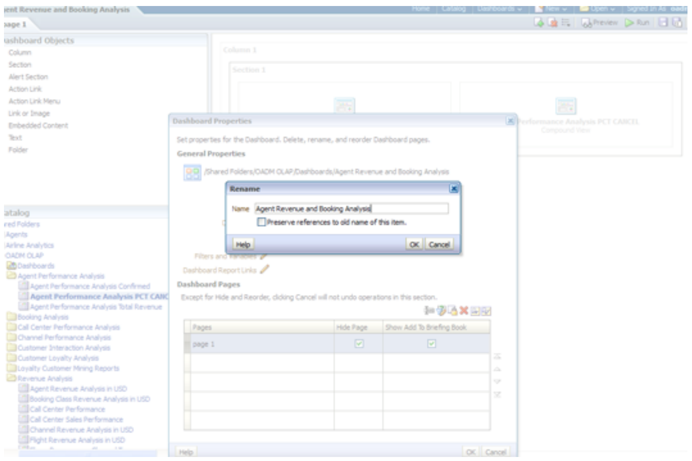
   a. Select the Dashboard.


ORACLE Business Intelligence
Agent Revenue and Booking Analysis
page 1
Dashboard Objects
Column
Section
Alert Section
Action Link
Action Link Menu
Link or Image
Embedded Content
Text
Folder
Dashboard Properties...
PDF and Print Properties...
Page Report Links...
Allow Saving Personal Customizations
Publish Page to Dashboard...
Column 1
Section 1
Agent Revenue Analysis in USD
Agent Performance Analysis PCT CANCEL
Catalog

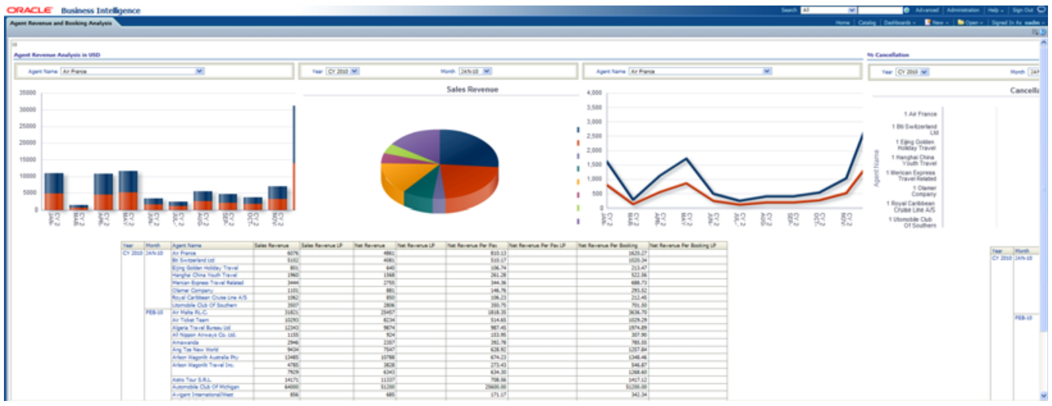
b. In Dashboard Properties window, click Change Name.


Dashboard Properties
Set properties for the Dashboard. Delete, rename, and reorder Dashboard pages.
General Properties
/Shared Folders/OADM OLAP/Dashboards/Agent Revenue and Booking Analysis
Style Default (blafp)
Description Agent Revenue and Booking Analysis
Filters and Variables
Dashboard Report Links
Dashboard Pages
Except for Hide and Reorder, clicking Cancel will not undo operations in this section.
Pages
Hide Page
Show Add To Briefing Book
page 1
Help
OK
Cancel

c. Change the name to "Agent Revenue and Booking Analysis", then click OK.


Rename
Name Agent Revenue and Booking Analysis
Preserve references to old name of this item.
Help
OK
Cancel

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


ORACLE Business Intelligence
Agent Revenue and Booking Analysis
Agent Revenue Analysis in USD
Sales Revenue
% Cancellation

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.

Tutorial: Creating a New Oracle Airlines Data Model Report

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

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

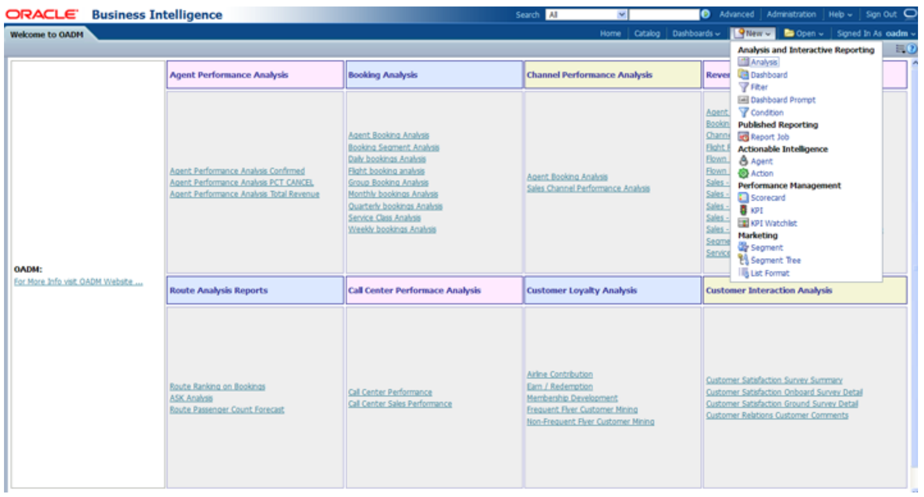
In this example, assume that you want to create a report named "Agent Booking and Flown Revenue" to put both booking count and flown revenue in one report.

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

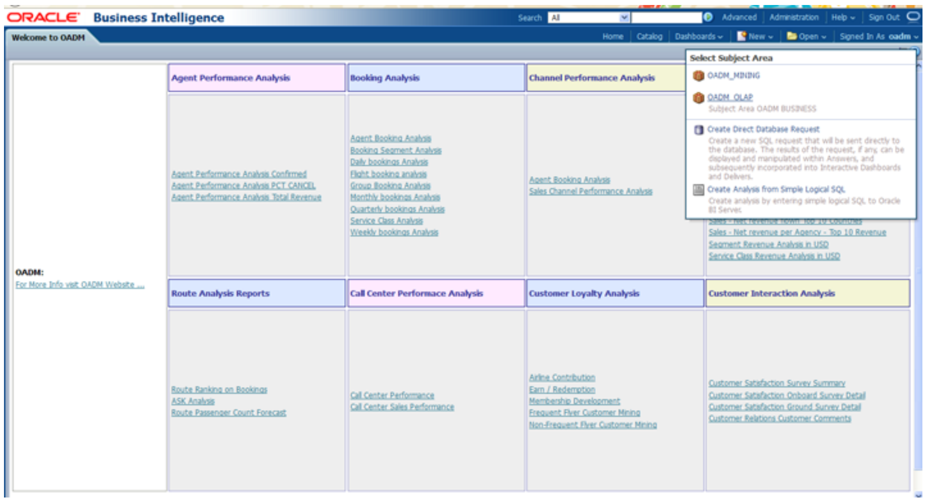
1. In the browser, open the login page at `http://`*`servername`*`:9704/analytics` where *`servername`* is the server on which the webcat is installed.

   Login with username of `oadm`, and provide the password.

2. Select **New**, and then select **Analysis** to create an Oracle Business Intelligence Suite Enterprise Edition report.


ORACLE Business Intelligence
Welcome to OADM
Search All
Advanced
Administration
Help
Sign Out
Home
Catalog
Dashboards
New
Open
Signed In As oadm
Analysis and Interactive Reporting
Analysis
Dashboard
Filter
Dashboard Prompt
Condition
Published Reporting
Report Job
Actionable Intelligence
Agent
Action
Performance Management
Scorecard
KPI
KPI Watchlist
Marketing
Segment
Segment Tree
List Format
OADM:
For More Info visit OADM Website ...
Agent Performance Analysis
Agent Performance Analysis Confirmed
Agent Performance Analysis PCT CANCEL
Agent Performance Analysis Total Revenue
Booking Analysis
Agent Booking Analysis
Booking Segment Analysis
Daily bookings Analysis
Flight booking analysis
Group Booking Analysis
Monthly bookings Analysis
Quarterly bookings Analysis
Service Class Analysis
Weekly bookings Analysis
Channel Performance Analysis
Agent Booking Analysis
Sales Channel Performance Analysis
Route Analysis Reports
Route Ranking on Bookings
ASK Analysis
Route Passenger Count Forecast
Call Center Performace Analysis
Call Center Performance
Call Center Sales Performance
Customer Loyalty Analysis
Airline Contribution
Earn / Redemption
Membership Development
Frequent Flyer Customer Mining
Non-Frequent Flyer Customer Mining
Customer Interaction Analysis
Customer Satisfaction Survey Summary
Customer Satisfaction Onboard Survey Detail
Customer Satisfaction Ground Survey Detail
Customer Relations Customer Comments

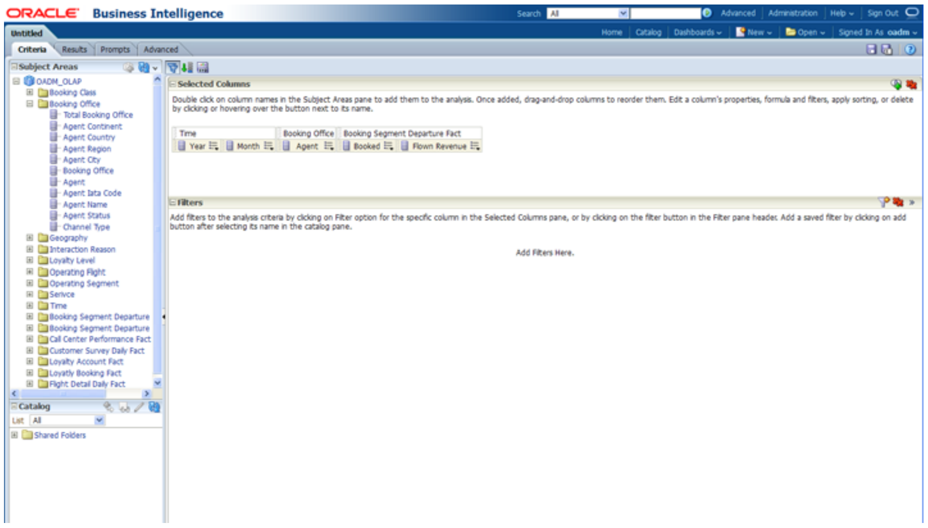
3. Select **Subject Area**, then select **OADM_OLAP** to create a relational report.


ORACLE Business Intelligence
Welcome to OADM
Search All
Advanced
Administration
Help
Sign Out
Home
Catalog
Dashboards
New
Open
Signed In As oadm
Select Subject Area
OADM_MINING
OADM_OLAP
Subject Area OADM BUSINESS
Create Direct Database Request
Create Analysis from Simple Logical SQL
OADM:
For More Info visit OADM Website ...
Agent Performance Analysis
Agent Performance Analysis Confirmed
Agent Performance Analysis PCT CANCEL
Agent Performance Analysis Total Revenue
Booking Analysis
Agent Booking Analysis
Booking Segment Analysis
Daily bookings Analysis
Flight booking analysis
Group Booking Analysis
Monthly bookings Analysis
Quarterly bookings Analysis
Service Class Analysis
Weekly bookings Analysis
Channel Performance Analysis
Agent Booking Analysis
Sales Channel Performance Analysis
Sales - Net revenue per Agency - Top 10 Revenue
Segment Revenue Analysis in USD
Service Class Revenue Analysis in USD
Route Analysis Reports
Route Ranking on Bookings
ASK Analysis
Route Passenger Count Forecast
Call Center Performace Analysis
Call Center Performance
Call Center Sales Performance
Customer Loyalty Analysis
Airline Contribution
Earn / Redemption
Membership Development
Frequent Flyer Customer Mining
Non-Frequent Flyer Customer Mining
Customer Interaction Analysis
Customer Satisfaction Survey Summary
Customer Satisfaction Onboard Survey Detail
Customer Satisfaction Ground Survey Detail
Customer Relations Customer Comments

Create a new SQL request that will be sent directly to the database. The results of the request, if any, can be displayed and manipulated within Answers, and subsequently incorporated into Interactive Dashboards and Delivers.

Create analysis by entering simple logical SQL to Oracle BI Server.

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


Selected Columns

Double click on column names in the Subject Areas pane to add them to the analysis. Once added, drag-and-drop columns to reorder them. Edit a column's properties, formula and filters, apply sorting, or delete by clicking or hovering over the button next to its name.

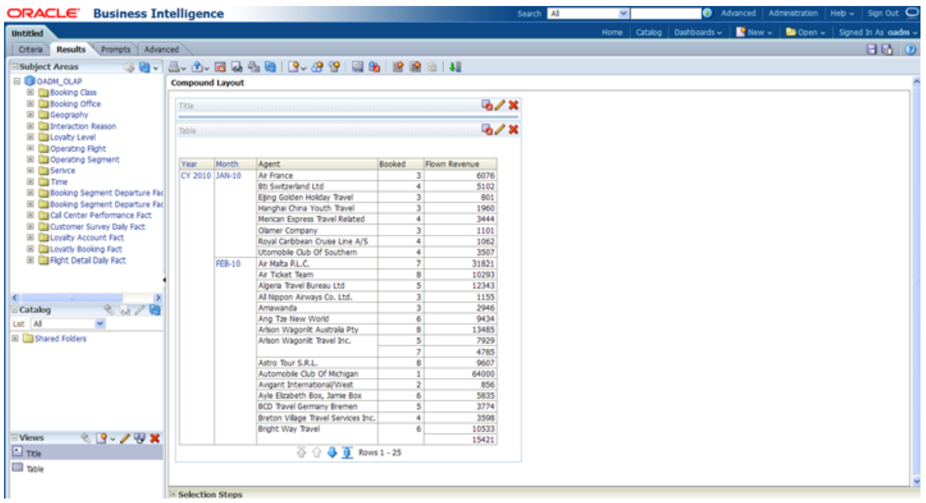
Time		Booking Office	Booking Segment Departure Fact	
Year	Month	Agent	Booked	Flown Revenue

Filters

Add filters to the analysis criteria by clicking on Filter option for the specific column in the Selected Columns pane, or by clicking on the filter button in the Filter pane header. Add a saved filter by clicking on add button after selecting its name in the catalog pane.

Add Filters Here.

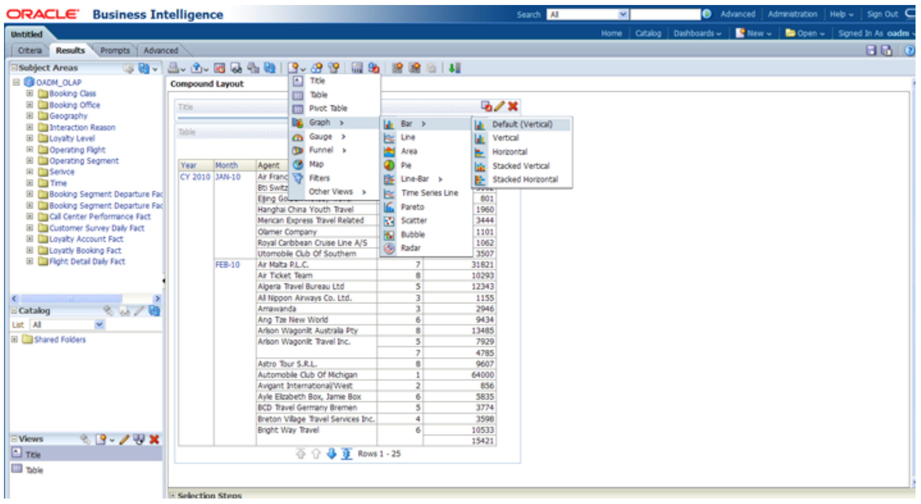
5. Select the **Results** tab to view the report.


Compound Layout

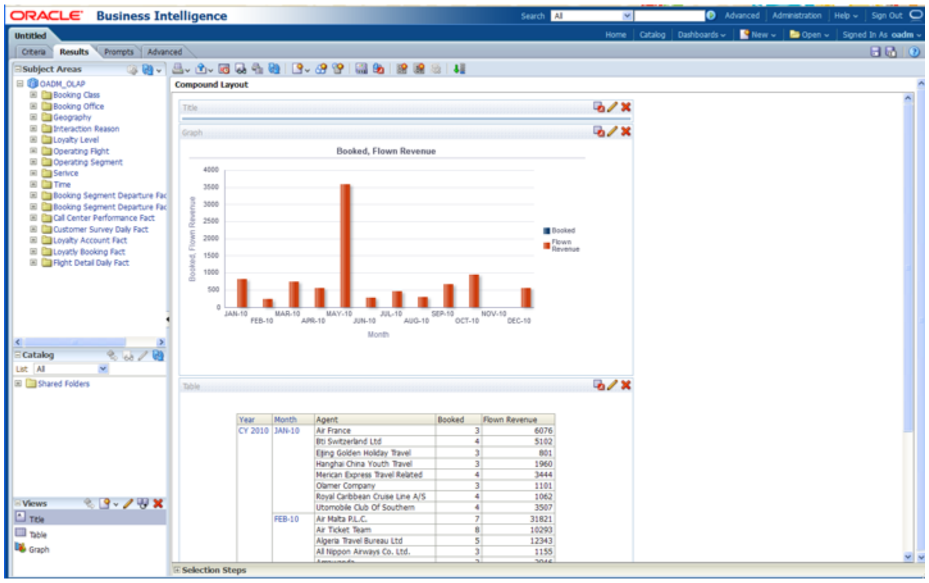
Year	Month	Agent	Booked	Flown Revenue
CY 2010	JAN-10	Air France	3	6076
		Bti Switzerland Ltd	4	5102
		Ejing Golden Holiday Travel	3	801
		Hanghai China Youth Travel	3	1960
		Merican Express Travel Related	4	3444
		Olamer Company	3	1101
		Royal Caribbean Cruise Line A/S	4	1062
		Utomobile Club Of Southern	4	3507
	FEB-10	Air Malta P.L.C.	7	31821
		Air Ticket Team	8	10293
		Algeria Travel Bureau Ltd	5	12343
		All Nippon Airways Co. Ltd.	3	1155
		Amawanda	3	2946
		Ang Tze New World	6	9434
		Arlson Wagonlit Australia Pty	8	13485
		Arlson Wagonlit Travel Inc.	5	7929
			7	4785
		Astro Tour S.R.L.	8	9607
		Automobile Club Of Michigan	1	64000
		Avigant International/West	2	856
		Ayle Elizabeth Box, Jamie Box	6	5835
		BCD Travel Germany Bremen	5	3774
		Breton Village Travel Services Inc.	4	3598
		Bright Way Travel	6	10533
				15421

Rows 1 - 25

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



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



Year	Month	Agent	Booked	Flown Revenue
CY 2010	JAN-10	Air France	3	6076
		Bti Switzerland Ltd	4	5102
		Eijing Golden Holiday Travel	3	801
		Hanghai China Youth Travel	3	1960
		Merican Express Travel Related	4	3444
		Olamer Company	3	1101
		Royal Caribbean Cruise Line A/S	4	1062
		Utomobile Club Of Southern	4	3507
	FEB-10	Air Malta P.L.C.	7	31821
		Air Ticket Team	8	10293
		Algeria Travel Bureau Ltd	5	12343
		All Nippon Airways Co. Ltd.	3	1155

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.

A

Sizing and Configuring an Oracle Airlines Data Model Warehouse

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

- Sizing an Oracle Airlines Data Model Warehouse
- Configuring a Balanced System for Oracle Airlines Data Model

Sizing an Oracle Airlines 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 Airlines 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 Airlines 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 Airlines Data Model Warehouse

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

```
Disk Space Minimum Requirements = Raw data size * Database space
factor * (1+GrthperY)nY*OS and File system factor * Compress
Factor * Storage Redundant factor
```

where:

- `Raw data size = (reference and lookup data per year + base/transaction data per year + derived and summary data per year +staging data +other data(OLAP/Data Mining))`
- `Database space factor = Indexes + Temporary Tables + Logs]`
- `GrthperY` = growth rate per year
- `OS and File system factor` is the install and configuration and maintain space for OS and Database
- `Redundant factor= ASM disk space and RAID factor. [RAID 1=2, RAID 5=1.25 or 1.33]`
- `Compress factor` depends how you apply the compress function. If you using HCC, it has a huge savings in disk space by using compression.

Configuring a Balanced System for Oracle Airlines 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 Airlines Data Model:

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

Maintaining High Throughput in an Oracle Airlines 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 Airlines Data Model for Bandwidth not Capacity

Based on the data volume calculated and the highest throughput requirement, you can estimate the I/O throughput along with back-end ETL process and front end business intelligence applications by time unit. Typically, a value of approximately 200MB per second I/O throughput per core is a good planning number for designing a balanced system. All subsequent critical components on the I/O path - the Host Bus Adapters, fiber channel connections, the switch, the controller, and the disks - have to be sized appropriately.

When running a data warehouse on an Oracle Real Application Cluster (Oracle RAC) it is just as important to size the cluster interconnect with the same care and caution you would use for the I/O subsystem throughput.

When configuring the storage subsystem for a data warehouse, it should be simple, efficient, highly available and very scalable. An easy way to achieve this is to apply the S.A.M.E. methodology (Stripe and Mirror Everything). S.A.M.E. can be implemented at the hardware level or by using Oracle ASM (Automatic Storage Management) or by using a combination of both. There are many variables in sizing the I/O systems, but one basic rule of thumb is that the data warehouse system has multiple disks for each CPU (at least two disks for each CPU at a bare minimum) to achieve optimal performance.

Planning for Growth of Your Oracle Airlines Data Model

A data warehouse designer plans for future growth of a data warehouse. There are several approaches to handling the growth in a system, and the key consideration is to be able to grow the I/O system without compromising on the I/O bandwidth. You cannot, for example, add four disks to an existing system of 20 disks, and grow the database by adding a new tablespace striped across only the four new disks. A better solution would be to add new tablespaces striped across all 24 disks, and over time also convert the existing tablespaces striped across 20 disks to be striped across all 24 disks.

Testing the I/O System Before Building the Oracle Airlines Data Model Warehouse

When creating a data warehouse on a new system, test the I/O bandwidth before creating all of the database data files to validate that the expected I/O levels are being achieved. On most operating systems, you can perform the test using simple scripts to measure the performance of reading and writing large test files.

Balanced Hardware Configuration Guidelines for Oracle Airlines Data Model

You can reference the follow tips for a balanced hardware configuration:

- Total throughput = `#cores X 100-200MB` (depends on the chip set)
- Total host bus adaptor (HBA) throughput = Total core throughput

Note: If total core throughput is 1.6 GB, you need four 4 Gbit HBAs.

- Use one disk controller per HBA port (throughput capacity must be equal).
- Switches must have the capacity as HBAs and disk controllers.
- Use a maximum of ten physical disk per controller (that is, use smaller drives: 146 or 300 GB).
- Use a minimum of 4 GB of memory per core (8 GB if using compress).
- Interconnect bandwidth equals I/O bandwidth (InfiniBand).

Oracle now provides the Oracle Database Machine, Exadata which combines industry-standard hardware from Oracle, Oracle Database 11*g* Release 2, and Oracle Exadata Storage Server Software to create a faster, more versatile database machine.

It's a completely scalable and fault tolerant package for all data management, especially for data warehousing.

Oracle also has a series of Optimized Warehouse Reference configurations that help customers take the risk out of designing and deploying Oracle data warehouses. Using extensive field experience and technical knowledge, Oracle and its hardware partners have developed a choice of data warehouse reference configurations that can support various sizes, user populations and workloads. These configurations are fast, reliable and can easily scale from 500 GB to over 100 TB on single and clustered servers to support tens to thousands of users.

Index