Oracle® OLAP

User's Guide

11*g* Release 1 (11.1)

B28124-03

June 2008

ORACLE®

Oracle OLAP User's Guide, 11*g* Release 1 (11.1)

B28124-03

Contents

5 Enhancing Your Database With Analytic Content

6 Developing Reports and Dashboards

7 Administering Oracle OLAP

8 Security

9 Advanced Aggregations

A Designing a Dimensional Model

Glossary

Index

Preface

The *Oracle OLAP User's Guide* explains how SQL applications can extend their analytic processing capabilities and manage summary data by using the OLAP option of Oracle Database. It also provides information about managing resources for OLAP.

The preface contains these topics:

Audience

This manual is intended for DBAs who need to perform these tasks:

- Develop and manage a data warehouse
- Create and maintain dimensional data objects
- Administer Oracle Database with the OLAP option

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at `http://www.oracle.com/accessibility/`.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, 7 days a week. For TTY support, call 800.446.2398. Outside the United States, call +1.407.458.2479.

Related Documents

For more information about the OLAP option, see the following manuals in the Oracle Database 11*g* documentation set:

- *Oracle OLAP Java API Developer's Guide*

  Introduces the Oracle OLAP API, a Java application programming interface for Oracle OLAP, which is used for defining, building, and querying dimensional objects in the database.

- *Oracle OLAP Java API Reference*

  Describes the classes and methods in the Oracle OLAP Java API for defining, building, and querying dimensional objects in the database.

- *Oracle OLAP DML Reference*

  Contains a complete description of the OLAP Data Manipulation Language (OLAP DML) used to define and manipulate analytic workspace objects.

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.

What's New in Oracle OLAP?

This preface identifies the major enhancements to the OLAP option of Oracle Database.

Oracle Database 11*g* Release 11.1 Oracle OLAP

The OLAP Option to Oracle Database 11*g* continues the development trends of Oracle9*i* and Oracle Database 10*g*, especially in deepening integration with the database and enhancing SQL access to cubes, security, and metadata. The power of OLAP is easily accessible to SQL applications. Oracle Database 11*g* also introduces the cube as a summary management solution for relational OLAP (ROLAP) implementations.

OLAP Metadata Integration

All metadata for cubes and dimensions is stored in the Oracle database and revealed in the data dictionary views, so that you can query the entire business model in SQL. Use of the data dictionary to store the metadata officially codifies the dimensional model in the database, provides significant improvements for metadata queries, and supports other new features such as SQL object security for cubes and dimensions.

See Also:

- Chapter 4, "Querying Dimensional Objects"
- *Oracle Database Reference*

Automatic Maintenance of Cube and Dimension Views

Oracle Database 11*g* automatically creates and maintains relational views for every cube, dimension, and hierarchy in the database. If you modify a dimensional object, such as adding a calculated measure to a cube, the view is immediately re-created to reflect the change. Oracle Database defines these views using the new `CUBE_TABLE` function, which enables the SQL Optimizer enhancements.

See Also:

- Chapter 4, "Querying Dimensional Objects"
- *Oracle Database SQL Language Reference*

Cube Scripts

A cube script is an ordered list of commands that prepare a cube for querying, such as Clear Data, Load Data, Aggregate, Execute PL/SQL, and Execute OLAP DML. For many applications, cube scripts will eliminate the need to use procedural programs for processing cubes.

See Also: Chapter 3, "Creating Dimensions and Cubes"

Cost-Based Aggregation

Fast updates and uniform querying performance are two hallmarks of the OLAP option. Cost-based aggregation enhances performance in both areas by executing a fine-grained pre-aggregation strategy and storing sparse data sets very efficiently.

See Also: Chapter 3, "Creating Dimensions and Cubes"

Calculation Expression Syntax

OLAP calculation expressions extend the syntax of the SQL analytic functions. This syntax is already familiar to SQL developers and DBAs, so that it is easier for them to adopt than proprietary OLAP languages and APIs.

This syntax is used to define calculations that are embedded in the cube, such as dynamically calculated facts or measures.

See Also: Chapter 5, "Enhancing Your Database With Analytic Content"

Cube Materialized Views

Cube materialized views are cubes that have been enhanced to use the automatic refresh and query rewrite features of Oracle Database.

Cube materialized views bring the fast update and fast query capabilities of the OLAP option to applications that query detail relational tables. Summary data is generated and stored in a cube, and query rewrite automatically redirects queries to the cube materialized views. Applications experience excellent query performance.

See Also:

- Chapter 3, "Creating Dimensions and Cubes"
- Chapter 7, "Administering Oracle OLAP"

Object and Data Security

Oracle Database 11*g* introduces both object security and data security to OLAP cubes and dimensions. Both types of security are granted to database users and roles.

Object security controls access to analytic workspaces, cubes, and dimensions using standard SQL `GRANT` and `REVOKE` syntax.

Data security controls access to the data in a cube or a dimension. You can grant `SELECT`, `INSERT`, `UPDATE`, and `DELETE` privileges to dimension members (keys) either globally or in the context of a particular cube to control access to the data in a cube.

See Also: Chapter 8, "Security"

1

Overview

This chapter introduces the powerful analytic resources available in the Oracle Database with the OLAP option. It consists of the following topics:

OLAP Technology in the Oracle Database

Oracle Database offers the industry's first and only embedded OLAP server. 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 operators. These capabilities make the Oracle database a complete analytical platform, capable of supporting the entire spectrum of business intelligence and advanced analytical applications.

Full Integration of Multidimensional Technology

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.
- Applications can query dimensional objects using SQL.

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.

Ease of Application Development

Oracle OLAP makes it easy to enrich your database and your applications with interesting analytic content. Native SQL access to Oracle multidimensional objects and calculations greatly eases the task of developing dashboards, reports, business intelligence (BI) and analytical applications of any type compared to systems that offer proprietary interfaces. Moreover, SQL access means that the power of Oracle OLAP analytics can be used by *any* database application, not just by the traditional limited collection of OLAP applications.

Ease of Administration

Because Oracle OLAP is completely embedded in the Oracle database, there is no administration learning curve as is typically associated with stand-alone OLAP servers. You can leverage your existing DBA staff, rather than invest in specialized administration skills.

One major administrative advantage of Oracle's embedded OLAP technology is automated cube maintenance. With stand-alone OLAP servers, the burden of refreshing the cube is left entirely to the administrator. This can be a complex and potentially error-prone job. The administrator must create procedures to extract the changed data from the relational source, move the data from the source system to the system running the stand-alone OLAP server, load and rebuild the cube. The DBA must take responsibility for the security of the deltas (changed values) during this process as well.

With Oracle OLAP, in contrast, cube refresh is handled entirely by the Oracle database. The database tracks the staleness of the dimensional objects, automatically keeps track of the deltas in the source tables, and automatically applies only the changed values during the refresh process. The DBA simply schedules the refresh at appropriate intervals, and Oracle Database takes care of everything else.

Security

With Oracle OLAP, standard Oracle Database security features are used to secure your multidimensional data.

In contrast, with a stand-alone OLAP server, administrators must manage security twice: once on the relational source system and again on the OLAP server system. Additionally, they must manage the security of data in transit from the relational system to the stand-alone OLAP system.

Unmatched Performance and Scalability

Business intelligence and analytical applications are dominated by actions such as drilling up and down hierarchies and comparing aggregate values such as period-over-period, share of parent, projections onto future time periods, and a myriad of similar calculations. Often these actions are essentially random across the entire space of potential hierarchical aggregations. Because Oracle OLAP precomputes or efficiently computes on the fly all aggregates in the defined multidimensional space, it delivers unmatched performance for typical business intelligence applications.

Oracle OLAP queries take advantage of Oracle shared cursors, dramatically reducing memory requirements and increasing performance.

When Oracle Database is installed with Real Application Clusters (RAC), OLAP applications receive the same benefits in performance, scalability, fail over, and load balancing as any other application.

Reduced Costs

All these features add up to reduced costs. Administrative costs are reduced because existing personnel skills can be leveraged. Moreover, the Oracle database can manage the refresh of dimensional objects, a complex task left to administrators in other systems. Standard security reduces administration costs as well. Application development costs are reduced because the availability of a large pool of application developers who are SQL knowledgeable, and a large collection of SQL-based development tools means applications can be developed and deployed more quickly. Any SQL-based development tool can take advantage of Oracle OLAP. Hardware costs are reduced by Oracle OLAP's efficient management of aggregations, use of shared cursors, and Oracle RAC, which enables highly scalable systems to be built from low-cost commodity components.

Developing Reports and Dashboards Using SQL Tools and Application Builders

Analysts can choose any SQL query and analysis tool for selecting, viewing, and analyzing the data. You can use your favorite tool or application, or use one of the tools supplied with Oracle Database.

Figure 1–1 displays a portion of a dashboard created in Oracle Application Express, which is distributed with Oracle Database. Application Express generates HTML reports that display the results of SQL queries. It only understands SQL; it has no special knowledge of dimensional objects.

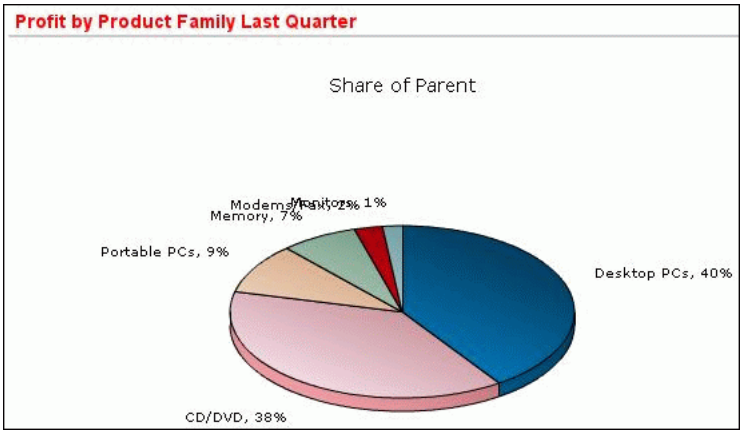
This dashboard demonstrates information-rich calculations such as ratio, share, prior period, and cumulative total. Separate tabs on the dashboard present Profitability Analysis, Sales Analysis, and Product Analysis. Each tab presents the data in dials, bar charts, horizontal bar charts, pie charts, and cross-tabular reports. A drop-down list in the upper left corner provides a choice of Customers.

The dial displays the quarterly profit margin. To the right is a bar chart that compares current profits with year-ago profits.

Figure 1–1 Dashboard Created in Oracle Application Express

The pie chart in Figure 1–2 displays the percent share that each product family contributed to the total profits in the last quarter.

Figure 1–2 Contributions of Product Families to Total Profits



The horizontal bar chart in Figure 1–3 displays ranked results for locations with the largest gains in profitability from a year ago. Decision makers can see at a glance how each location improved by the last quarter.

Figure 1–3 Ranking of Percent Change in Year-to-Date Profits From Year Ago

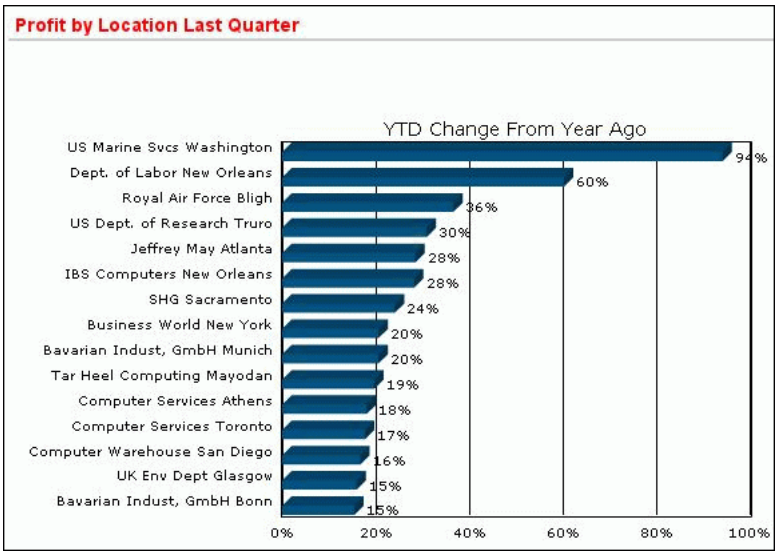


Figure 1–4 compares current profits with year-to-date, year-to-date year ago, the change between year-to-date and year-to-date year ago, and percent change between year-to-date and year-to-date year-ago profits. The cross-tabular report features interactive drilling, so that decision makers can easily see the detailed data that contributed to a parent value of interest.

Figure 1–4 Year-to-Date Profits Compared to Year Ago

Profit Reporting

Time	Product	Customer	Profit	YTD	YTD Yr Ago	YTD Chg Yr Ago	YTD % Chg Yr Ago
1998	Total Product	Total Customer	7,249,296	7,249,296			
1999	Total Product	Total Customer	9,190,282	9,190,282	7,249,296	1,940,986	2677
2000	Total Product	Total Customer	8,880,369	8,880,369	9,190,282	-309,913	-337
2001	Total Product	Total Customer	8,658,271	8,658,271	8,880,369	-222,098	-250
2002	Total Product	Total Customer	6,854,325	6,854,325	8,658,271	-1,803,945	-2083
2003	Total Product	Total Customer	8,730,695	8,730,695	6,854,325	1,876,370	2737
2004	Total Product	Total Customer	11,175,647	11,175,647	8,730,695	2,444,952	2800
2005	Total Product	Total Customer	10,544,532	10,544,532	11,175,647	-631,115	-565
2006	Total Product	Total Customer	11,024,547	11,024,547	10,544,532	480,015	455

Overview of the Dimensional Data Model

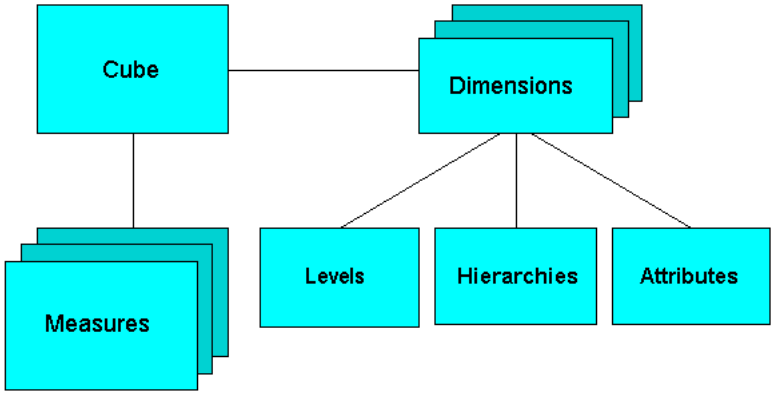
Dimensional objects are an integral part of OLAP. Because OLAP is on-line, it must provide answers quickly; analysts pose iterative queries during interactive sessions, not in batch jobs that run overnight. And because OLAP is also analytic, the queries are complex. The dimensional objects and the OLAP engine are designed to solve complex queries in real time.

The dimensional objects include cubes, measures, dimensions, attributes, levels, and hierarchies. The simplicity of the 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.

Figure 1–5 shows the general relationships among dimensional objects.

Figure 1–5 Diagram of the OLAP Dimensional Model



The 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, OracleBI Suite Enterprise Edition, BusinessObjects Enterprise, and Cognos ReportNet.

Cubes

Cubes provide a means of organizing measures that have the same shape, that is, they have the exact same dimensions. Measures in the same cube can easily be analyzed and displayed together.

A cube usually corresponds to a single fact table or view.

Measures

Measures populate the cells of a cube with the facts collected about business operations. Measures are organized by dimensions, which typically include a Time dimension.

An analytic database contains snapshots of historical data, derived from data in a transactional database, legacy system, syndicated sources, or other data sources. Three

years of historical data is generally considered to be appropriate for analytic applications.

Measures are static and consistent while analysts are using them to inform their decisions. They are updated in a batch window at regular intervals: weekly, daily, or periodically throughout the day. Some administrators refresh their data by adding periods to the time dimension of a measure, and may also roll off an equal number of the oldest time periods. Each update provides a fixed historical record of a particular business activity for that interval. Other administrators do a full rebuild of their data rather than performing incremental updates.

A critical decision in defining a measure is the lowest level of detail. Users may never view this detail data, but it determines the types of analysis that can be performed. For example, market analysts (unlike order entry personnel) do not need to know that Beth Miller in Ann Arbor, Michigan, placed an order for a size 10 blue polka-dot dress on July 6, 2006, at 2:34 p.m. But they might want to find out which color of dress was most popular in the summer of 2006 in the Midwestern United States.

The base level determines whether analysts can get an answer to this question. For this particular question, Time could be rolled up into months, Customer could be rolled up into regions, and Product could be rolled up into items (such as dresses) with an attribute of color. However, this level of aggregate data could not answer the question: At what time of day are women most likely to place an order? An important decision is the extent to which the data has been aggregated before being loaded into a data warehouse.

Dimensions

Dimensions contain a set of unique values that identify and categorize data. They form the edges of a cube, and thus of the measures within the cube. Because measures are typically multidimensional, a single value in a measure must be qualified by a member of each dimension to be meaningful. For example, the Sales measure has four dimensions: Time, Customer, Product, and Channel. A particular Sales value (43,613.50) only has meaning when it is qualified by a specific time period (Feb-06), a customer (Warren Systems), a product (Portable PCs), and a channel (Catalog).

Base-level dimension values correspond to the unique keys of a fact table.

Hierarchies and Levels

A **hierarchy** is a way to organize data at different levels of aggregation. In viewing data, analysts use dimension hierarchies to recognize trends at one level, drill down to lower levels to identify reasons for these trends, and roll up to higher levels to see what affect these trends have on a larger sector of the business.

Level-Based Hierarchies

Each **level** represents a position in the hierarchy. Each level above the base (or most detailed) level contains aggregate values for the levels below it. The members at different levels have a one-to-many **parent-child relation**. For example, `Q1-05` and `Q2-05` are the children of `2005`, thus `2005` is the parent of `Q1-05` and `Q2-05`.

Suppose a data warehouse contains snapshots of data taken three times a day, that is, every 8 hours. Analysts might normally prefer to view the data that has been aggregated into days, weeks, quarters, or years. Thus, the Time dimension needs a hierarchy with at least five levels.

Similarly, a sales manager with a particular target for the upcoming year might want to allocate that target amount among the sales representatives in his territory; the

allocation requires a dimension hierarchy in which individual sales representatives are the child values of a particular territory.

Hierarchies and levels have a many-to-many relationship. A hierarchy typically contains several levels, and a single level can be included in more than one hierarchy.

Each level typically corresponds to a column in a dimension table or view. The base level is the primary key.

Value-Based Hierarchies

Although hierarchies are typically composed of named levels, they do not have to be. The parent-child relations among dimension members may not define meaningful levels. For example, in an employee dimension, each manager has one or more reports, which forms a parent-child relation. Creating levels based on these relations (such as individual contributors, first-level managers, second-level managers, and so forth) may not be meaningful for analysis. Likewise, the line item dimension of financial data does not have levels. This type of hierarchy is called a **value-based hierarchy.**

Attributes

An **attribute** provides additional information about the data. Some attributes are used for display. For example, you might have a product dimension that uses Stock Keeping Units (SKUs) for dimension members. The SKUs are an excellent way of uniquely identifying thousands of products, but are meaningless to most people if they are used to label the data in a report or a graph. You would define attributes for the descriptive labels.

You might also have attributes like colors, flavors, or sizes. This type of attribute can be used for data selection and answering questions such as: Which colors were the most popular in women's dresses in the summer of 2005? How does this compare with the previous summer?

Time attributes can provide information about the Time dimension that may be useful in some types of analysis, such as identifying the last day or the number of days in each time period.

Each attribute typically corresponds to a column in dimension table or view.

2

Getting Started with Oracle OLAP

This chapter describes the preliminary steps you need to take to use Oracle OLAP. It assumes that you have already installed Oracle Database 11*g* Enterprise Edition. The OLAP option is installed automatically as part of a Basic installation of Oracle Database.

> **Note:** To start querying dimensional objects immediately, install the Global analytic workspace, as described in "Installing the Sample Schema". Then follow the instructions in Chapter 4.

This chapter includes the following topics:

- Installing the Sample Schema
- Database Management Tasks
- Granting Privileges to DBAs and Application Developers
- Getting Started with Analytic Workspace Manager

Installing the Sample Schema

You can download and install the sample Global schema from the Oracle Web site and use it to try the examples shown throughout this guide:

`http://www.oracle.com/technology/products/bi/olap/doc_sample_schemas/sampleschemasfordoc11g.html`

Instructions for installing the schema are provided in the `README` file.

Database Management Tasks

You should create undo, permanent, and temporary tablespaces that are appropriate for use by dimensional objects. Follow the recommendations in "Storage Management" on page 7-2.

Granting Privileges to DBAs and Application Developers

Anyone who needs to create or manage dimensional objects in Oracle Database must have the necessary privileges. These privileges are different from those needed just to query the data stored in dimensional objects. The security system is discussed in Chapter 8.

DBAs and application developers need the following roles and privileges.

To create dimensional objects in the user's own schema:

- `OLAP_USER` role
- `CREATE SESSION` privilege

To create dimensional objects in different schemas:

- `OLAP_DBA` role
- `CREATE SESSION` privilege

To administer data security:

- `OLAP_XS_ADMIN` role

To create cube materialized views in the user's own schema:

- `CREATE MATERIALIZED VIEW` privilege
- `CREATE DIMENSION` privilege
- `ADVISOR` privilege

To create cube materialized views in different schemas:

- `CREATE ANY MATERIALIZED VIEW` privilege
- `CREATE ANY DIMENSION` privilege
- `ADVISOR` privilege

Users also need an unlimited quota on the tablespace in which the dimensional objects will be stored. The tablespaces should be defined specifically for OLAP use, as described in Chapter 7.

If the source tables are in a different schema, then the owner of the dimensional objects needs `SELECT` object privileges on those tables.

Example 2–1 shows the SQL statements for creating the `GLOBAL` user.

Example 2–1 SQL Statements for Creating the GLOBAL User

```
CREATE USER "GLOBAL" IDENTIFIED BY password
   DEFAULT TABLESPACE glo
   TEMPORARY TABLESPACE glotmp
   QUOTA UNLIMITED ON glo
   PASSWORD EXPIRE;

GRANT OLAP_USER TO GLOBAL;
GRANT CREATE SESSION TO GLOBAL;
GRANT OLAP_XS_ADMIN TO GLOBAL;
```

Getting Started with Analytic Workspace Manager

In this section, you will learn how to install Analytic Workspace Manager software and make a connection to Oracle Database.

Installing Analytic Workspace Manager

Analytic Workspace Manager is distributed on the Oracle Database Client installation disk.

If you are installing on the same system as the database, then choose a **Custom** installation and install into the same Oracle home directory as the database. Select **OLAP Analytic Workspace Manager and Worksheet** from the list of components.

If you are installing on a remote system, then choose either an **Administrator** or a **Custom** installation. The Administrator choice automatically installs Analytic Workspace Manager on the client.

> **See Also:** An installation guide for your client platform, such as the *Oracle Database Client Quick Installation Guide for Microsoft Windows (32-Bit)*.

Opening Analytic Workspace Manager

On Windows, to open Analytic Workspace Manager:

- From the Start menu, choose **Oracle -** *Oracle_home*, then **Integrated Management Tools**, and then **OLAP Analytic Workspace Manager and Worksheet**.

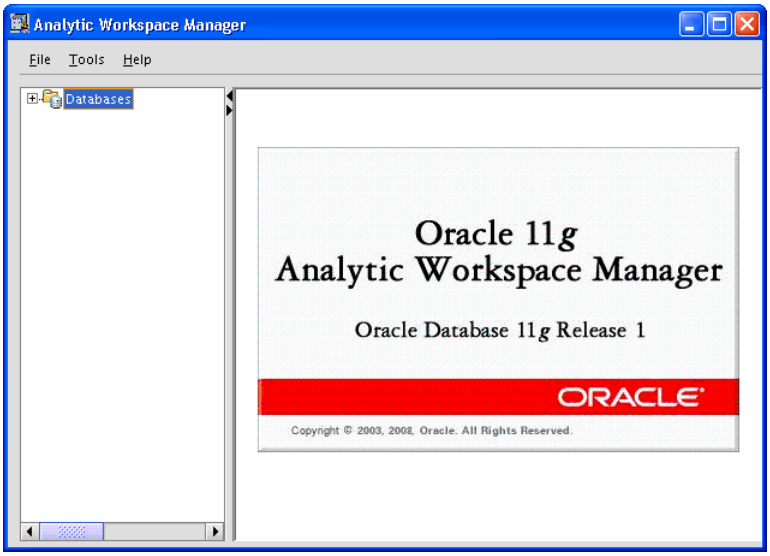
On Linux, to open Analytic Workspace Manager:

- From the shell command line, enter this command:

```
$ORACLE_HOME/olap/awm/awm.sh
```

Figure 2–1 shows the initial display.

Figure 2–1 Opening Analytic Workspace Manager



Defining a Database Connection

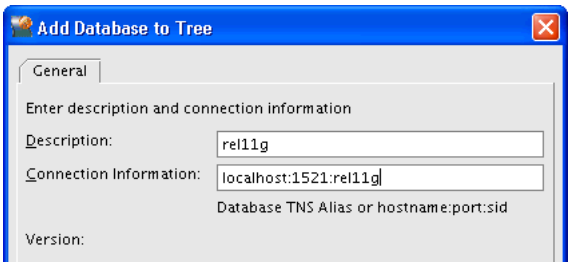
You can define a connection to each database that you use for OLAP. After you define a connection, the database instance is listed in the navigation tree for you to access at any time.

To define a database connection:

1. Right-click the top Databases folder in the navigation tree, then choose **Add Database to Tree** from the pop-up menu.
2. Complete the Add Database to Tree dialog box.

Figure 2–2 shows the connection information on the General tab of the Add Database to Tree dialog box.

Figure 2–2 Defining a Database Connection



Opening a Database Connection

To connect to a database:

1. Click the plus icon (+) next to a database in the navigation tree.
2. Supply your database user name and password in the Connect to Database dialog box.

Installing Plugins

Plugins extend the functionality of Analytic Workspace Manager. Any Java developer can create a plugin. Plugins are distributed as JAR files. The developer should provide information about what the plugin does and how to use it.

If you have one or more plugins, then you only need to identify their location to Analytic Workspace Manager.

To use plugins:

1. Create a local directory for storing plugins for Analytic Workspace Manager.
2. Copy the JAR files to that directory.
3. Open Analytic Workspace Manager.
4. Choose **Configuration** from the Tools menu.

   The Configuration dialog box opens.
5. Select **Enable Plugins** and identify the plugin directory. Click **OK**.
6. Close and reopen Analytic Workspace Manager.

   The new functionality provided by the plugins is available in the navigator.

> **See Also:** *Developing Analytic Workspace Manager Plug-ins*, which you can download from the Oracle Technology Network at
>
> `http://www.oracle.com/technology/products/bi/olap`

3

Creating Dimensions and Cubes

This chapter explains how to design a data model and create dimensions and cubes using Analytic Workspace Manager. It contains the following topics:

Designing a Dimensional Model for Your Data

Chapter 1 introduced the dimensional objects: Cubes, measures, dimensions, levels, hierarchies, and attributes. In this chapter, you will learn how to define them in Oracle Database, but first you need to decide upon the dimensional model you want to create. What are your measures? What are your dimensions? How can you distinguish between a dimension and an attribute in your data? You can design a dimensional model using pencil and paper, a database design software package, or any other method that suits you.

If your source data is already in a star or snowflake schema, then you already have the elements of a dimensional model:

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

You can also get insights into the dimensional model by looking at the reports currently being generated from the source data. The reports will identify the levels of

aggregation that interest the report consumers, as well as the attributes used to qualify the data.

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

See Also:

"Overview of the Dimensional Data Model" on page 1-5 for an introduction to dimensional objects

Appendix A, "Designing a Dimensional Model" for a case study of developing a dimensional model for the Global analytic workspace

Introduction to Analytic Workspace Manager

Analytic Workspace Manager is the primary tool for creating, developing, and managing dimensional objects in Oracle Database. Your goal in using Analytic Workspace Manager is to create a dimensional data store that supports business analysis. This data store can stand alone or store summary data as part of a relational data warehouse.

Populating dimensional objects involves a physical transformation of the data. The first step in that transformation is defining the cubes, measures, dimensions, levels, hierarchies, and attributes. Afterward, you can map these dimensional objects to their relational data sources. The data loading process transforms the data from a relational format into a dimensional format.

Using Analytic Workspace Manager, you can:

- Develop a dimensional model of your data.
- Instantiate that model as dimensional objects.
- Load data from relational tables into those objects.
- Define information-rich calculations.
- Create materialized views that can be used by the database refresh system.
- Automatically generate relational views of the dimensional objects.

You can load data from these sources in the database:

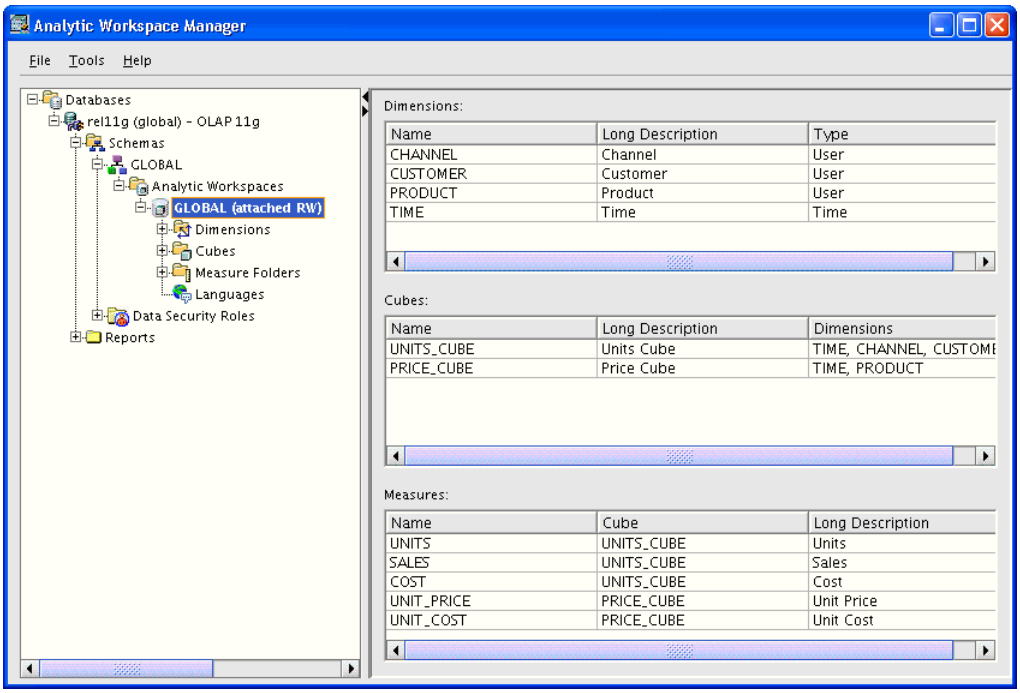
- Tables
- Views
- Synonyms

You must have `SELECT` privileges on the relational data sources so you can load the data into the dimensions and cubes. This chapter assumes that you have a star, snowflake, or other relational schema that supports dimensional objects.

Figure 3–1 shows the main window of Analytic Workspace Manager. It contains menus, a toolbar, a navigation tree, and property sheets. When you select an object in the navigation tree, the property sheet to the right provides detailed information about that object. When you right-click an object, you get a choice of menu items with appropriate actions for that object.

Analytic Workspace Manager has a full online Help system, which includes context-sensitive Help.

Figure 3–1 Analytic Workspace Manager Main Window



Creating a Dimensional Data Store Using Analytic Workspace Manager

An analytic workspace is a container for storing related cubes. You create dimensions, cubes, and other dimensional objects within the context of an analytic workspace.

Basic Steps for Creating an Analytic Workspace

To create an analytic workspace:

1. Open Analytic Workspace Manager and connect to your database instance as the user defined for this purpose.
2. Create a new analytic workspace in the database:
   a. In the navigation tree, expand the folders until you see the schema where you want to create the analytic workspace.
   b. Right-click Analytic Workspaces, then choose **Create Analytic Workspace**.
   c. Complete the Create Analytic Workspace dialog box, then choose **Create**.

   The new analytic workspace appears in the Analytic Workspaces folder for the schema.
3. Define the dimensions for the data.

   See "Creating Dimensions" on page 3-4.
4. Define the cubes for the data.

   See "Creating Cubes" on page 3-14.
5. Load data into the cubes and dimensions.

   See "Loading Data Into a Cube" on page 3-19

When you have finished, you will have an analytic workspace populated with the detail data fetched from relational tables or views. You may also have summarized data and calculated measures.

Adding Functionality to Dimensional Objects

In addition to the basic steps, you can add functionality to the cubes in these ways:

- Develop custom cube scripts to customize the builds.

  See "Creating and Executing Custom Cube Scripts" on page 3-22.

- Generate materialized views that support automatic refresh and query rewrite.

  See "Adding Materialized View Capability to a Cube" on page 3-24.

- Support multiple languages by adding translations of metadata and attribute values.

  See "Supporting Multiple Languages" on page 3-26.

- Define measure folders to simplify access for end users.

  See "Defining Measure Folders" on page 3-26.

How Analytic Workspace Manager Saves Changes

Analytic Workspace Manager saves changes automatically that you make to the analytic workspace. You do not explicitly save your changes.

Saves occur when you take an action such as these:

- Click **OK** or the equivalent button in a dialog box.

  For example, when you click **Create** in the Create Dimension dialog box, the new dimension is committed to the database.

- Click **Apply** in a property sheet.

  For example, when you change the labels on the General property page for an object, the change takes effect when you click **Apply**.

Creating Dimensions

Dimensions are lists of unique values that identify and categorize data. They form the edges of a cube, and thus of the measures within the cube. In a report, the dimension values (or their descriptive attributes) provide labels for the rows and columns.

You can define dimensions that have any of these common forms:

- Level-based dimensions that use parent-child relationships to group members into levels. Most dimensions are level-based.
- Value-based dimensions that have parent-child relationships among their members, but these relationships do not form meaningful levels.
- List or flat dimensions that have no levels or hierarchies.

Dimension Members Must Be Unique

Every dimension member must be a unique value. Depending on your data, you can create a dimension that uses either natural keys or surrogate keys from the relational sources for its members. If you have any doubt that the values are unique across all levels, then keep the default choice of surrogate keys.

- **Source keys** are read from the relational sources without modification. To use the same exact keys as the source data, the values must be unique across levels. Because each level may be mapped to a different relational column, this uniqueness may not be enforced in the source data. For example, a dimension table might have a Day column with values of `1` to `366` and a Week column with values of `1` to `52`. Unless you take steps to assure uniqueness, the values from the Week column will overwrite the first 52 Day values.
- **Surrogate keys** ensure uniqueness by adding a level prefix to the members while loading them into the analytic workspace. For the previous example, surrogate keys create two dimension members named `DAY_1` and `WEEK_1`, instead of a single member named `1`. A dimension that has surrogate keys must be defined with at least one level-based hierarchy.

Analytic Workspace Manager creates surrogate keys unless you specify otherwise.

Time Dimensions Have Special Requirements

You can define dimensions as either User or Time dimensions. Business analysis is performed on historical data, so fully defined time periods are vital. A time dimension table must have columns for period end dates and time span. These required attributes support comparisons with earlier or later time periods. If this information is not available, then you can define Time as a User dimension, but it will not support time-based analysis.

You must define a Time dimension with at least one level to support time-based analysis, such as a custom measure that calculates the difference from the prior period.

To create a dimension:

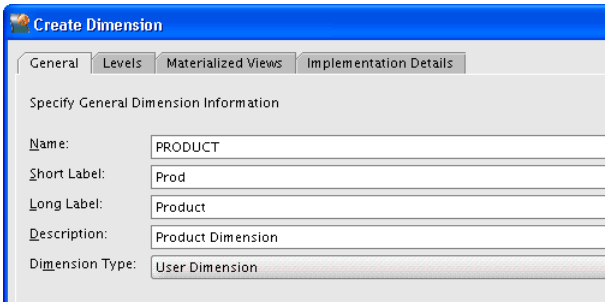
1. Expand the folder for the analytic workspace.
2. Right-click **Dimensions**, then choose **Create Dimension**.

   The Create Dimension dialog box is displayed.
3. Complete the General tab.
4. If the keys in the source table are unique across levels, you can change the default setting on the Implementation Details tab.
5. Click **Create**.

   The new dimension appears as a subfolder under Dimensions.

Figure 3–2 shows the creation of the Product dimension.

Figure 3–2 Creation of the Product Dimension



Creating Levels

For business analysis, data is typically summarized by level. For example, your database may contain daily snapshots of a transactional database. Days are the base level. You might summarize this data at the weekly, quarterly, and yearly levels.

Levels have parent-child or one-to-many relationships, which form a **level-based hierarchy**. For example, each week summarizes seven days, each quarter summarizes 13 weeks, and each year summarizes four quarters. This hierarchical structure enables analysts to detect trends at the higher levels, then drill down to the lower levels to identify factors that contributed to a trend.

For each level that you define, you must identify a data source for dimension members at that level. Members at all levels are stored in the same dimension. In the previous example, the Time dimension contains members for weeks, quarters, and years.

To create a level:

1. Expand the folder for the dimension.
2. Right-click **Levels**, then choose **Create Level**.

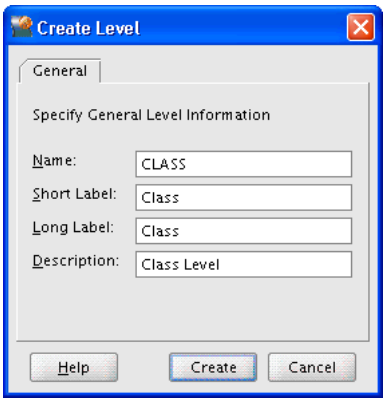
   The Create Level dialog box is displayed.
3. Complete the General tab of the Create Level dialog box.
4. Click **Create**.

   The new level appears as an item in the Levels folder.

   Tip: Alternatively, you can create levels in the Create Dimension dialog box Levels tab.

Figure 3–3 shows the creation of the Class level for the Product dimension.

Figure 3–3 Creation of the Class Level



Creating Hierarchies

Dimensions can have one or more hierarchies. They can be level-based or value-based.

Level-Based Hierarchies

Most hierarchies are level-based. Analytic Workspace Manager supports these common types of level-based hierarchies:

- **Normal hierarchies** consist of one or more levels of aggregation. Members roll up into the next higher level in a many-to-one relationship, and these members roll up into the next higher level, and so forth to the top level.
- **Ragged hierarchies** contain at least one member with a different base, creating a "ragged" base level for the hierarchy. Ragged hierarchies are not supported for cube materialized views.
- **Skip-level hierarchies** contain at least one member whose parents are more than one level above it, creating a hole in the hierarchy. An example of a skip-level hierarchy is City-State-Country, where at least one city has a country as its parent (for example, Washington D.C. in the United States).

  In relational source tables, a skip-level hierarchy may contain nulls in the level columns. Skip-level hierarchies are not supported for cube materialized views.

Multiple hierarchies for a dimension typically share the base-level dimension members and then branch into separate hierarchies. They can share the top level if they use all the same base members and use the same aggregation operators. Otherwise, they need different top levels to store different aggregate values. For example, a Customer dimension may have multiple hierarchies that include all base-level customers and are summed to a shared top level. However, a Time dimension with calendar and fiscal hierarchies must aggregate to separate Calendar Year (January to December) and Fiscal Year (July to June) levels, because they use different selections of base-level members.

Value-Based Hierarchies

You may also have dimensions with parent-child relations that do not support levels. For example, an employee dimension might have a parent-child relation that identifies each employee's supervisor. However, levels that group together first-, second-, and third-level supervisors and so forth may not be meaningful for analysis. Similarly, you might have a line-item dimension with members that cannot be grouped into meaningful levels. In this situation, you can create a **value-based hierarchy** defined by the parent-child relations, which does not have named levels. You can create value-based hierarchies only for dimensions that use the source keys, because surrogate keys are formed with the names of the levels.

To create a hierarchy:

1. Expand the folder for the dimension.
2. Right-click **Hierarchies**, then choose **Create Hierarchy**.

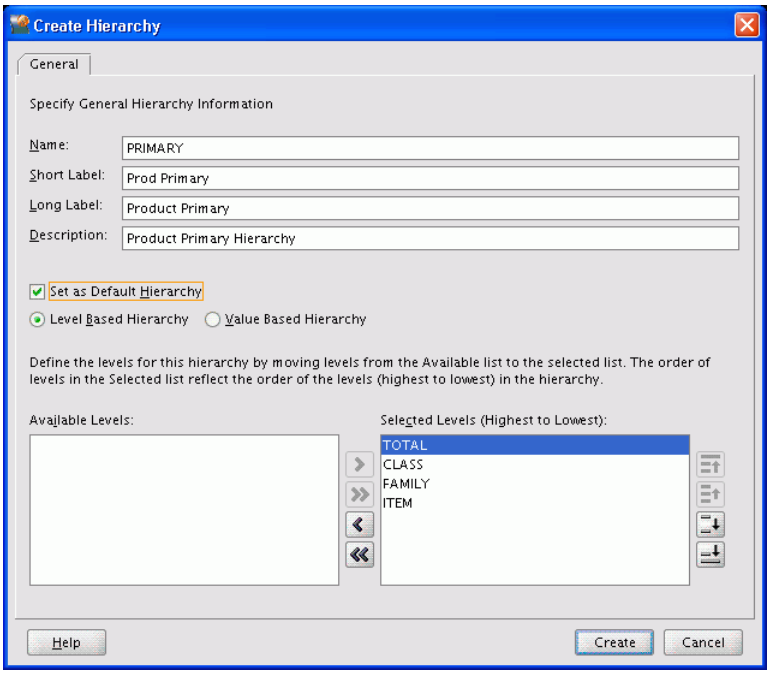
   The Create Hierarchy dialog box is displayed.
3. Complete the General tab of the Create Hierarchy dialog box.

   Click **Help** for information about these choices.
4. Click **Create**.

   The new hierarchy appears as an item in the Hierarchies folder.

Figure 3–4 shows the creation of the Primary hierarchy for the Product dimension.

Figure 3–4 Creation of the Product Primary Hierarchy



Creating Attributes

Attributes provide information about the individual members of a dimension. They are used for labeling crosstabular and graphical data displays, selecting data, organizing dimension members, and so forth.

Automatically Defined Attributes

Analytic Workspace Manager creates some attributes automatically when creating a dimension. These attributes have a unique type, such as "Long Description."

All dimensions are created with long and short description attributes. If your source tables include long and short descriptions, then you can map the attributes to the appropriate columns. However, if your source tables include only one set of descriptions, then you can map the long description attributes. If you map the short description attributes to the same column, the data will be loaded twice.

Time dimensions are created with time-span and end-date attributes. This information must be provided for all Time dimension members.

User-Defined Attributes

You can create additional "User" attributes that provide supplementary information about the dimension members, such as the addresses and telephone numbers of customers, or the color and sizes of products.

To create an attribute:

1. Expand the folder for the dimension.
2. Right-click **Attributes**, then choose **Create Attribute**.

   The Create Attribute dialog box is displayed.

3. Complete the General tab of the Create Attribute dialog box.

   Some attributes apply to all dimension members, and others apply to only one level. Your selection in the Apply Attributes To box controls the mapping of the attribute to one column or to more than one.

   Click **Help** for information about these choices.

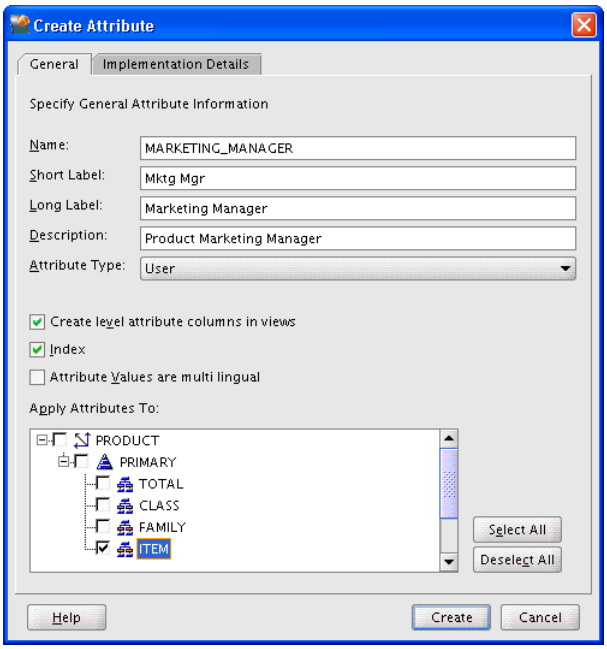
4. To change the data type from the default choice of `VARCHAR2`, complete the Implementation Details tab.

5. Click **Create**.

   The new attribute appears as an item in the Attributes folder.

Figure 3–5 shows the creation of the Marketing Manager attribute for the Product dimension. Notice that this attribute applies only to the Item level.

Figure 3–5 Creation of the Product Marketing Manager Attribute



Unique Key Attributes

Materialized views require that each dimension of the cube have unique key attributes. These attributes store the original key values of the source dimensions, which may have been changed when creating the embedded total dimensions of the cubes.

Analytic Workspace Manager automatically creates unique key attributes for the dimensions of a cube materialized view. You do not create them manually.

Mapping Dimensions

Mapping identifies the relational data source for each dimensional object. After mapping a dimension to a column of a relational table or view, you can load the data. You can create, map, and load each dimension individually, or perform each step for all dimensions before proceeding to the next step.

Mapping Window

The mapping window has a tabular view and a graphical view. You can switch between the two views, using the icons at the top of the canvas.

- **Tabular view**: Drag-and-drop the names of individual columns from the schema navigation tree to the rows for the dimensional objects.
- **Graphical view**: Drag-and-drop icons, which represent tables and views, from the schema navigation tree onto the mapping canvas. Then draw lines from the columns to the dimensional objects.

You can use expressions when mapping dimensions in the tabular view. This capability enables you to create the top level of a dimension without having a source column in the dimension table.

Click **Help** on the Mapping window for more information.

To map a dimension:

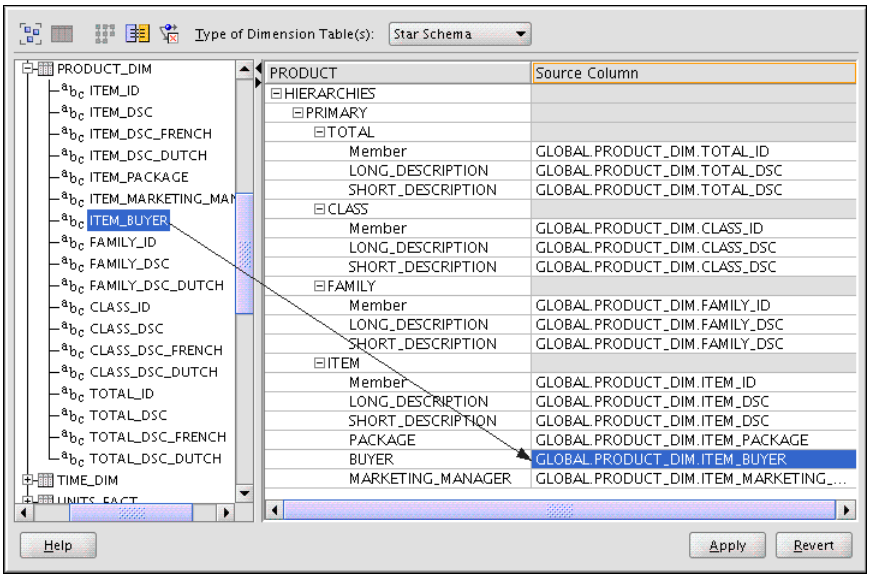
1. In the navigation tree, expand the dimension folder and click **Mappings**.

   The Mapping window contains a schema navigation tree on the left and a mapping table for the dimension with rows for the levels and their attributes. This is the tabular view.

2. To enlarge the Mapping Window, drag the divider to the left.
3. In the schema tree, expand the tables, views, or synonyms that contain the dimension members and attributes.
4. Drag-and-drop the source columns onto the appropriate cells in the mapping table for the dimension.
5. After you have mapped all levels and attributes, click **Apply**.
6. Drag the divider back to the right to reveal the navigation tree.

Figure 3–6 shows the Product dimension mapped in the tabular view. The arrow highlights how the `PRODUCT_DIM.ITEM_BUYER` column maps to the `PRODUCT.ITEM.BUYER` attribute.

Figure 3–6 Product Dimension Mapped in Tabular View



Source Data Query

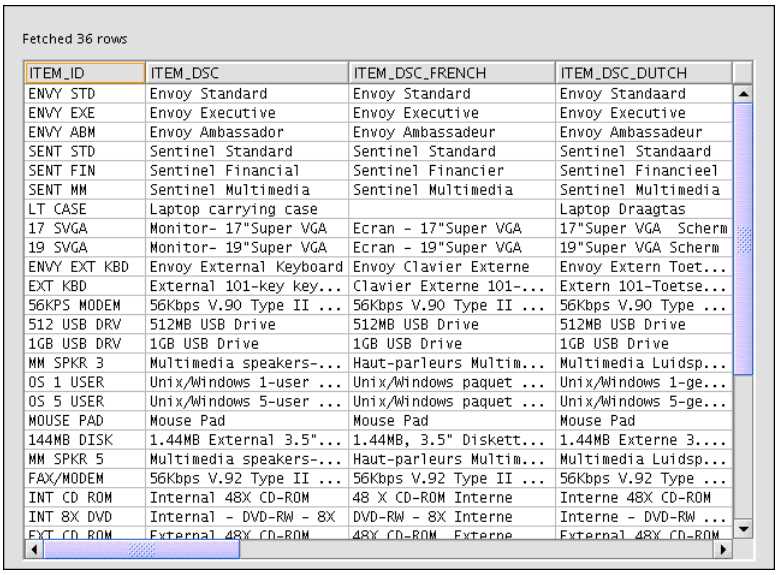
You can view the contents of a particular source column without leaving the mapping window. The information is readily available, eliminating the guesswork when the names are not adequately descriptive.

To see the values in a particular source table or view:

1. Right-click the source object in either the schema tree or the graphical view of the mapping canvas.
2. Choose **View Data** from the pop-up menu.

Figure 3–7 shows the data stored in the `PRODUCT_DIM` table.

Figure 3–7 Data in the PRODUCT_DIM Table



Fetched 36 rows

ITEM_ID	ITEM_DSC	ITEM_DSC_FRENCH	ITEM_DSC_DUTCH
ENVY STD	Envoy Standard	Envoy Standard	Envoy Standaard
ENVY EXE	Envoy Executive	Envoy Executive	Envoy Executive
ENVY ABM	Envoy Ambassador	Envoy Ambassadeur	Envoy Ambassadeur
SENT STD	Sentinel Standard	Sentinel Standard	Sentinel Standaard
SENT FIN	Sentinel Financial	Sentinel Financier	Sentinel Financieel
SENT MM	Sentinel Multimedia	Sentinel Multimedia	Sentinel Multimedia
LT CASE	Laptop carrying case		Laptop Draagtas
17 SVGA	Monitor- 17"Super VGA	Ecran - 17"Super VGA	17"Super VGA Scherm
19 SVGA	Monitor- 19"Super VGA	Ecran - 19"Super VGA	19"Super VGA Scherm
ENVY EXT KBD	Envoy External Keyboard	Envoy Clavier Externe	Envoy Extern Toet...
EXT KBD	External 101-key key...	Clavier Externe 101-...	Extern 101-Toetse...
56KPS MODEM	56Kbps V.90 Type II ...	56Kbps V.90 Type II ...	56Kbps V.90 Type ...
512 USB DRV	512MB USB Drive	512MB USB Drive	512MB USB Drive
1GB USB DRV	1GB USB Drive	1GB USB Drive	1GB USB Drive
MM SPKR 3	Multimedia speakers-...	Haut-parleurs Multim...	Multimedia Luidsp...
OS 1 USER	Unix/Windows 1-user ...	Unix/Windows paquet ...	Unix/Windows 1-ge...
OS 5 USER	Unix/Windows 5-user ...	Unix/Windows paquet ...	Unix/Windows 5-ge...
MOUSE PAD	Mouse Pad	Mouse Pad	Mouse Pad
144MB DISK	1.44MB External 3.5"...	1.44MB, 3.5" Diskett...	1.44MB Externe 3....
MM SPKR 5	Multimedia speakers-...	Haut-parleurs Multim...	Multimedia Luidsp...
FAX/MODEM	56Kbps V.92 Type II ...	56Kbps V.92 Type II ...	56Kbps V.92 Type ...
INT CD ROM	Internal 48X CD-ROM	48 X CD-ROM Interne	Interne 48X CD-ROM
INT 8X DVD	Internal - DVD-RW - 8X	DVD-RW - 8X Interne	Interne - DVD-RW ...
EXT CD ROM	External 48X CD-ROM	48X CD-ROM Externe	External 48X CD-ROM

Loading Data Into Dimensions

Analytic Workspace Manager provides several ways to load data into dimensional objects. The quickest way when developing a data model is using the default choices of the Maintenance Wizard. Other methods may be more appropriate in a production environment than the one shown here. They are discussed in "Choosing a Data Maintenance Method" on page 3-22.

To load data into the dimensions:

1. In the navigation tree, right-click the Dimensions folder or the folder for a particular dimension.
2. Choose **Maintain Dimension**.

   The Maintenance Wizard opens on the Select Objects page.
3. Select one or more dimensions from Available Target Objects and use the shuttle buttons to move them to Selected Target Objects.

4. Click **Finish** to load the dimension values immediately.

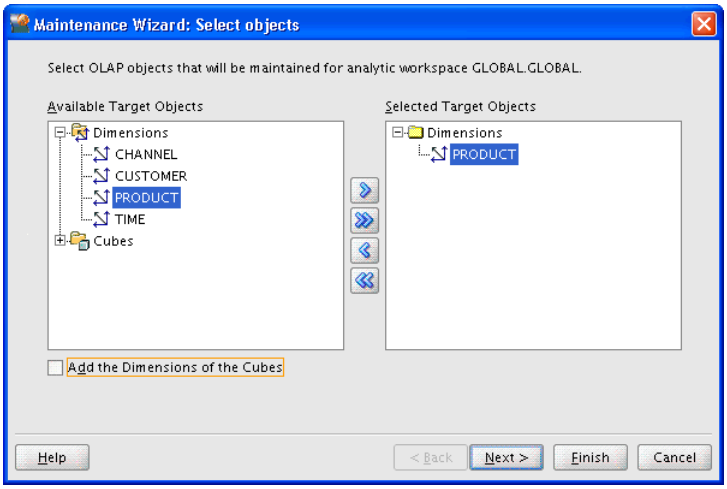
   The additional pages of the wizard enable you to create a SQL script or submit the load to the Oracle job queue. To use these options, click **Next** instead.

5. Review the build log, which appears when the build is complete. If the log shows that errors occurred, then fix them and run the Maintenance Wizard again.

   Errors are typically caused by problems in the mapping. Check for incomplete mappings or changes to the source objects.

Figure 3–8 shows the first page of the Maintenance Wizard. Only the Product dimension has been selected for maintenance. All the Product dimension members and attributes will be fetched from the mapped relational sources.

Figure 3–8 Loading Dimension Values into the Product Dimension

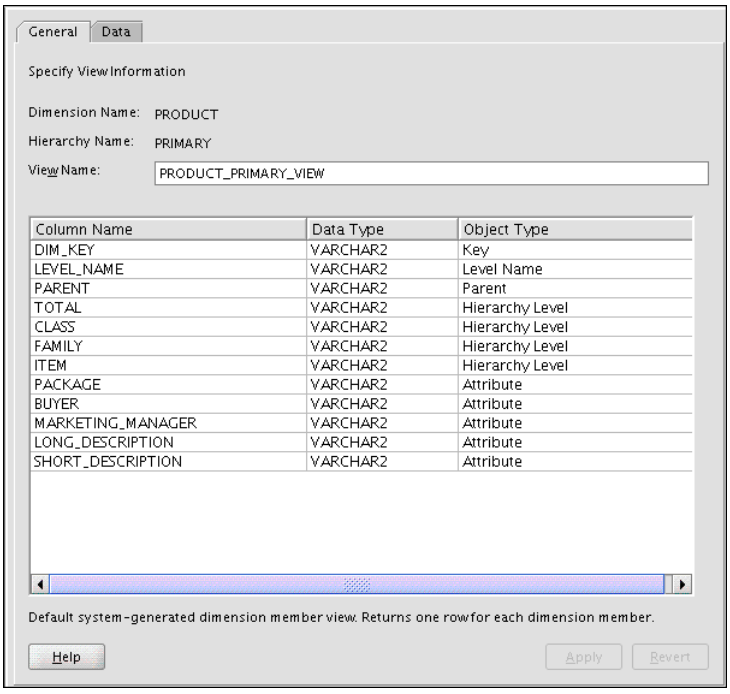


Displaying the Dimension View

The Maintenance Wizard automatically generates relational views of dimensions and hierarchies. Chapter 4 describes these views and explains how to query them.

Figure shows the description of the relational view of the Product Primary hierarchy. You can view the data on the Data tab.

Product Primary Hierarchy View



Displaying the Default Hierarchy

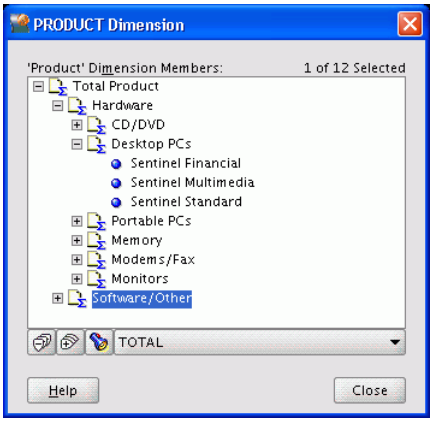
After loading a dimension, you can display the default hierarchy.

To display the default hierarchy:

1. In the navigation tree, right-click the name of a dimension.
2. Choose **View Data**.

Figure 3–9 shows the Primary hierarchy of the Product dimension.

Figure 3–9 Displaying the Product Primary Hierarchy



Creating Cubes

Cubes are informational objects that identify measures with the exact same dimensions and thus are candidates for being processed together at all stages: data loading, aggregation, storage, and querying.

Cubes define the shape of your business measures. They are defined by a set of ordered dimensions. The dimensions form the edges of a cube, and the measures are the cells in the body of the cube.

To create a cube:

1. Expand the folder for the analytic workspace.
2. Right-click **Cubes**, then choose **Create Cube**.

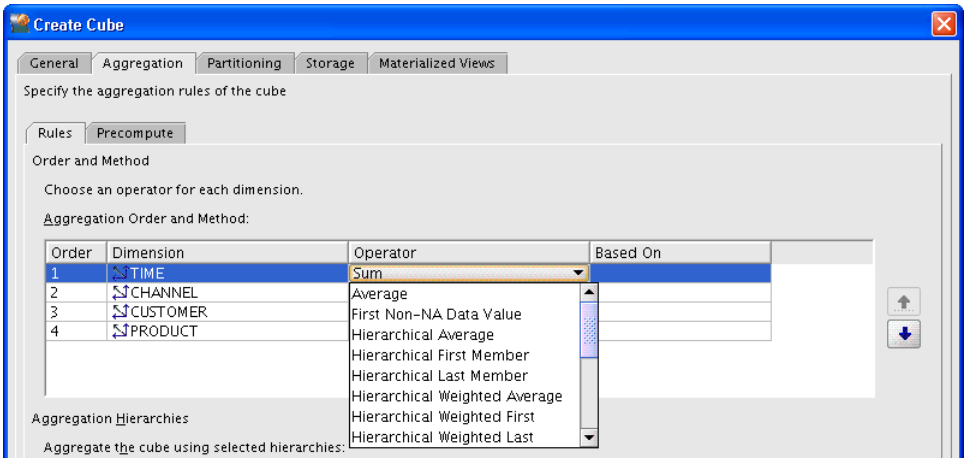
   The Create Cube dialog box is displayed.
3. On the General tab, enter a name for the cube and select its dimensions.
4. On the Aggregation tab, click the Rules subtab and select an aggregation method for each dimension. If the cube uses more than one method, then you may need to specify the order in which the dimensions are aggregated to get the desired results.

   You can ignore the bottom of the tab, unless you want to exclude a hierarchy from the aggregation.
5. If you run the advisors after mapping the cube, Oracle OLAP can determine the best partitioning and storage options. However, if you want to define these options yourself, then complete the Partitioning and Storage tabs before creating the cube.
6. Click **Create**. The new cube appears as a subfolder under **Cubes**.

Figure 3–10 shows the Rules subtab for the Units cube with the list of operators displayed.

> **See Also:** "Aggregation Operators" on page 9-3 for descriptions of the aggregation operators

Figure 3–10 Selecting an Aggregation Operator



Creating Measures

Measures store the facts collected about your business. Each measure belongs to a particular cube, and thus shares particular characteristics with other measures in the cube, such as the same dimensions. The default characteristics of a measure are inherited from the cube.

To create a measure:

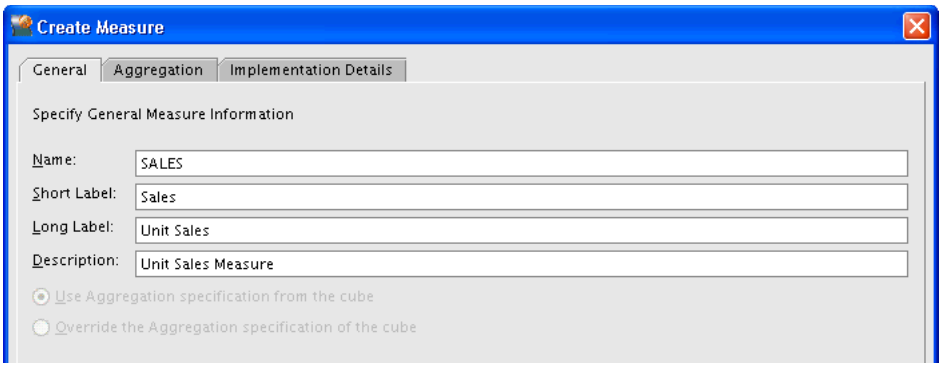
1. Expand the folder for the cube that has the dimensions of the new measure.
2. Right-click **Measures**, then choose **Create Measure**.

   The Create Measure dialog box is displayed.
3. On the General tab, enter a name for the measure.
4. Click **Create**.

   The new measure appears in the navigation tree as an item in the Measures folder.

Figure 3–11 shows the General tab of the Create Measure dialog box.

Figure 3–11 Creating the Sales Measure



Mapping Cubes

You use the same interface to map cubes as you did to map dimensions, as described in "Mapping Dimensions" on page 3-9.

You can use expressions when mapping cubes in the tabular view. This capability enables you to perform tasks like these as part of data maintenance, without any intermediate staging of the data:

- Perform calculations on the relational data using any combination of functions and operators available in the OLAP expression syntax.
- Create measures that are more aggregate than their relational sources. For example, suppose the Time dimension has columns for Day, Month, Quarter, and Year, and the fact table for Sales is related to Time by the Day foreign key column. In a basic mapping, you would store data in the cube at the Day level. However, you could aggregate it to the Month level during the data refresh. Using a technique called one-up mapping, you would map the cube to the Month column for Time, and specify a join between the dimension table and the fact table on the Day columns.

Click **Help** on the Mapping window for more information about these techniques.

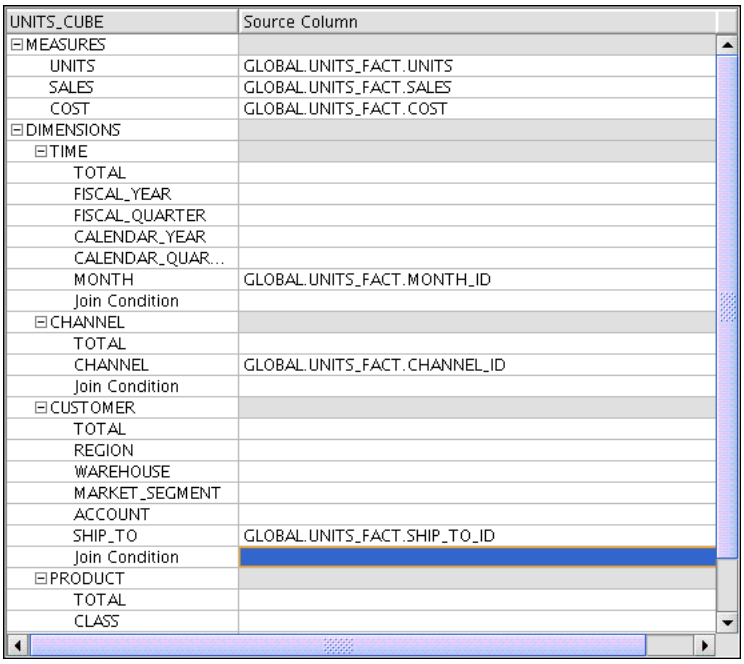
To map a cube:

1. In the navigation tree, expand the cube folder and click **Mappings**.

   The Mapping window contains a schema navigation tree on the left and a mapping table for the cube and its dimensions. This is the tabular view.
2. To enlarge the Mapping window, drag the divider to the left.
3. In the schema tree, expand the tables, views, or synonyms that contain the data for the measures.
4. Drag-and-drop the source columns onto the appropriate cells in the mapping table for the cube.
5. After you have mapped all dimensions and measures, click **Apply**.
6. Drag the divider back to the right to reduce the size of the Mapping window.

Figure 3–12 shows the mapping canvas with the Units cube mapped to columns in the `UNITS_FACT` table. After you save the mappings, Analytic Workspace Manager provides the join conditions for base-level mappings such as the ones shown here.

Figure 3–12 Units Cube Mapped in the Tabular View



Choosing a Partitioning Strategy

Partitioning is a method of physically storing the measures in a cube. It improves the performance of large measures 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.

Note: Run the Cube Partitioning Advisor after mapping the cube to a data source and before loading the data. You can change the partitioning strategy at any time using the Cube Partitioning Advisor, but you will need to reload the data afterward.

You can specify your own partitioning strategy only when creating the cube.

Choosing a Dimension for 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. In Figure 3–14, for instance, the Quarter level of the Time dimension is used as the partitioning key. The Cube Partitioning Advisor has a Time option, which will recommend a hierarchy and a level in the Time dimension for partitioning.

If life-cycle management is not a primary consideration, then run the Cube Partitioning Advisor and choose the Statistics option. The Cube Partitioning Advisor will develop a strategy designed to achieve optimal build and query performance.

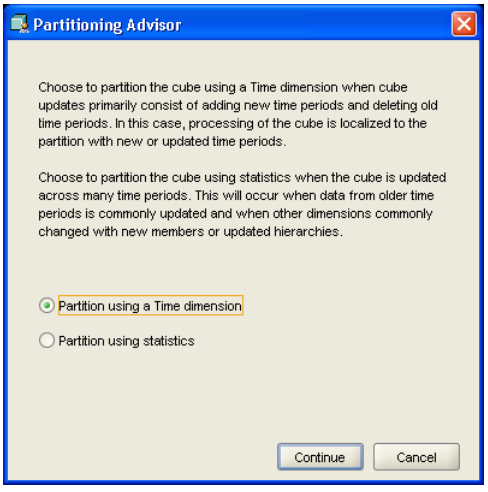
To run the Cube Partitioning Advisor:

1. Map the cube to its data source, if you have not done so already.
2. On the navigation tree, select the cube to display its property pages.
3. On the Partitioning tab, click **Cube Partitioning Advisor**.
4. Choose **Partition Using a Time Dimension** or **Partition Using Statistics.**

   Wait while the Cube Partitioning Advisor analyzes the cube. When it is done, the Cube Partitioning Advisor displays its recommendations.
5. Evaluate the recommendations of the Cube Partitioning Advisor.
   - Select **Accept Partition Advice** to accept the recommendations. The cube will be re-created with the new partitions.
   - Clear the **Accept Partition Advice** box to reject the recommendations.
6. Click **OK**.

Figure 3–13 shows the Cube Partitioning Advisor dialog box.

Figure 3–13 Partitioning a Cube



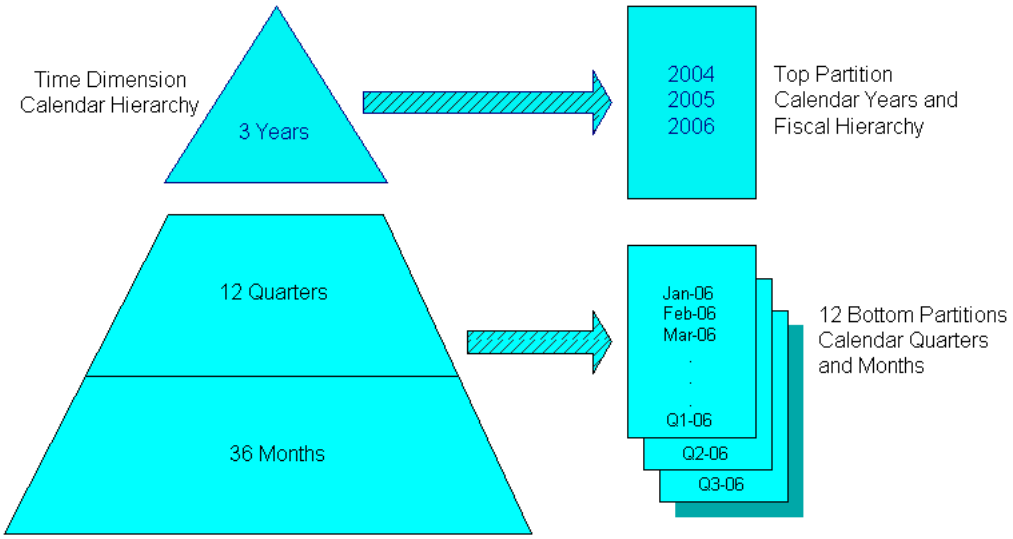
Example of a Partitioned Dimension

The Cube Partitioning Advisor might recommend partitioning at the Quarter level of the Calendar hierarchy of the Time dimension. Each Quarter and its descendants are stored in a separate partition. If there are three years of data in the analytic workspace, then partitioning on Quarter produces 12 bottom partitions, in addition to the default top partition. The top partition contains all remaining levels, that is, those above

Quarter (such as Year) and those in other hierarchies (such as Fiscal Year or Year-to-Date).

Figure 3–14 illustrates a Time dimension partitioned by Quarter.

Figure 3–14 Partitioning Time by Quarter



Loading Data Into a Cube

You load data into cubes using the same methods as dimensions. However, loading and aggregating the data for your business measures typically takes more time to complete. Unless you are developing a dimensional model using a small sample of data, you may prefer to run the build in one or more background processes.

1. In the navigation tree, right-click the Cubes folder or the name of a particular cube.
2. Choose **Maintain Cube**.

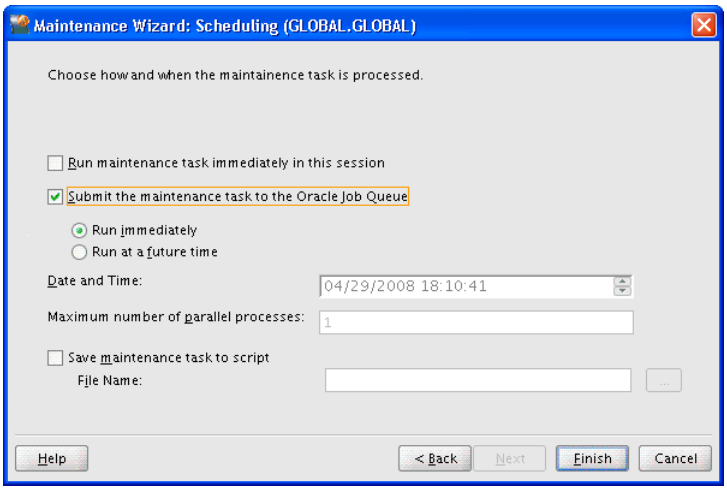
   The Maintenance Wizard opens on the Select Objects page.
3. Select one or more cubes from Available Target Objects and use the shuttle buttons to move them to Selected Target Objects. If the dimensions are already loaded, you can omit them from Selected Target Objects.
4. On the Dimension Data Processing Options page, you can keep the default values.
5. On the Task Processing Options page, you can submit the build to the Oracle job queue or create a SQL script that you can run outside of Analytic Workspace Manager.

   You can also select the number of processes to dedicate to this build. The number of parallel processes is limited by the smallest of these numbers: The number of partitions in the cube, the number of processes dedicated to the build, and the setting of the `JOB_QUEUE_PROCESSES` initialization parameter.

   Click **Help** for information about these choices.
6. Click **Finish**.

Figure 3–15 shows the build submitted immediately to the Oracle job queue.

Figure 3–15 Selecting the Scheduling Options



Displaying the Data in a Cube

After loading a cube, you can display the data for your business measures in Analytic Workspace Manager.

To display the data in a cube:

1. In the navigation tree, right-click the cube.
2. Choose **View Data** from the pop-up menu.

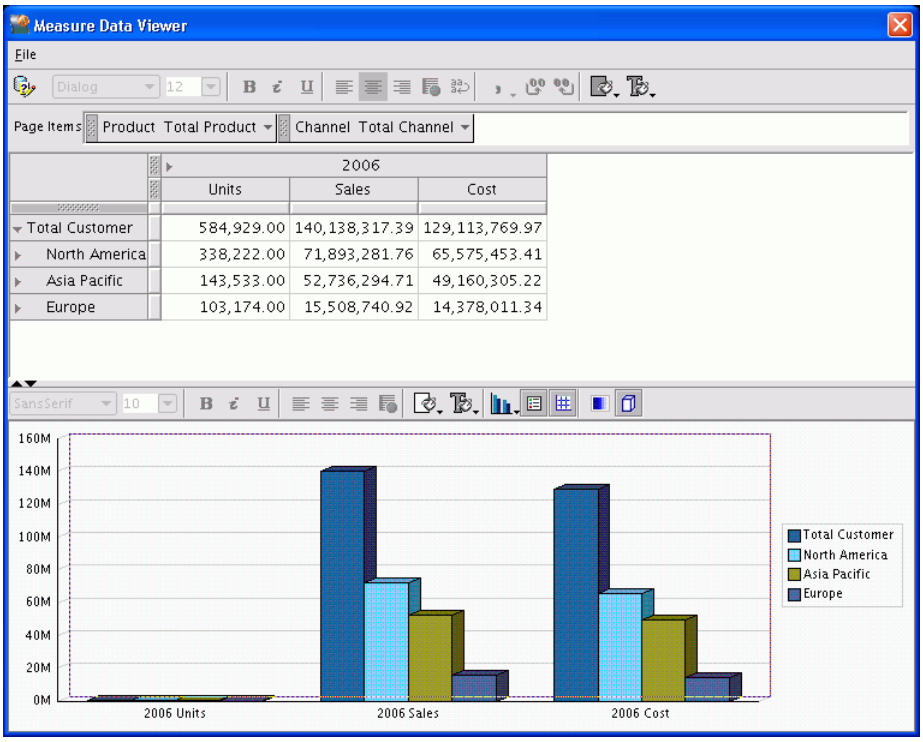
The Measure Data Viewer displays the selected measure in a crosstab at the top of the page and a graph at the bottom of the page. On the crosstab, you can expand and collapse the dimension hierarchies that label the rows and columns. You can also change the location of a dimension by pivoting or swapping it. If you wish, you can use more than one dimension to label the columns and rows, by nesting one dimension under another.

- To pivot, drag a dimension from one location and drop it at another location, usually above or below another dimension.
- To swap dimensions, drag and drop one dimension directly over another dimension, so they exchange locations.

To make extensive changes to the selection of data, choose **Query Builder** from the File menu.

Figure 3–16 shows the Units cube in the Measure Viewer.

Figure 3–16 Displaying the Units Cube

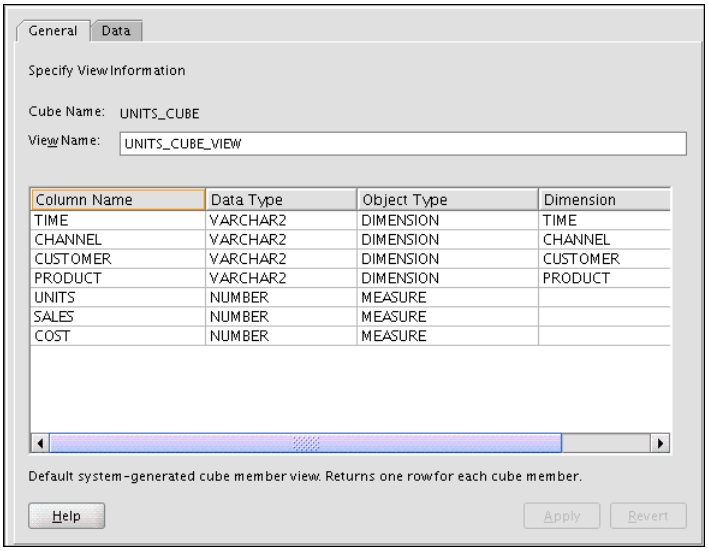


Displaying the Cube View Descriptions

The Maintenance Wizard automatically generates relational views of a cube. Chapter 4 describes these views and explains how to query them.

Figure 3–17 shows the description of the relational view of the Units cube.

Figure 3–17 Description of the Units Cube View



Choosing a Data Maintenance Method

While developing a dimensional model of your data, mapping and loading each object immediately after you create it is a good idea. That way, you can 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 scratch using this package.

The data for a partitioned cube will be loaded and aggregated in parallel when multiple processes have been allocated to the build. You will be 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 will use the materialized views as an integral part of data maintenance. Materialized view capabilities restrict the types of analytics that can be performed by a custom cube script.

See Also:

- "Build Logs" on page 7-5
- "Parallelism" on page 7-6

Creating and Executing Custom Cube Scripts

A cube script is an ordered list of steps that prepare a cube for querying. Each step represents a particular data transformation. By specifying the order in which these steps are performed, you can allow for interdependencies.

Each cube automatically has a default cube script named Load_and_Aggregate that loads the data and aggregates it using the rules defined on the cube. You can define any number of additional scripts and designate one of them as the default cube script. All methods of refreshing a cube execute the default cube script. You can execute other cube scripts manually using the Maintenance Wizard.

You can choose from these step types:

- **Clear Data**: Clears the data from the entire cube, from selected measures, or from selected portions of the cube. You can clear just the detail data (called **leaves**) for a fast refresh, just the aggregate data, or both for a complete refresh. Clearing old data values is typically done before loading new values.
- **Load**: Loads the data from the source tables into the cube. You can load all measures in the cube or just selected measures.

- **Aggregation**: Generates aggregate values using the rules defined for the cube. You can aggregate the entire cube, selected measures, or selected portions of the cube.
- **Analyze**: Generates optimizer statistics, which can improve the performance of some types of queries. For more information, see "Analyzing Cubes and Dimensions" on page 7-9. Generating statistics is typically done at the end of data maintenance.
- **OLAP DML**: Executes a command or program in the OLAP DML.
- **PL/SQL**: Executes a PL/SQL command or script. You can run a PL/SQL script, for example, at the beginning of data maintenance to initiate a refresh of the relational source tables.

If a cube is used to support advanced analytics in a cube script, then it cannot be enhanced as a cube materialized view, as described in "Adding Materialized View Capability to a Cube" on page 3-24. In this case, you are responsible for detecting when the data in the cube is stale and needs to be refreshed.

To create a cube script:

1. Expand the folder for a cube.
2. Right-click **Cube Scripts**, then choose **Create Cube Script**.

   The Create Cube Script dialog box is displayed.
3. On the General tab, enter a name for the cube script.
4. To create a new step, click **New Step**.
5. Choose the type of step.

   The New Step dialog box is displayed for that type of step.
6. Complete the tabs, then click **OK**.

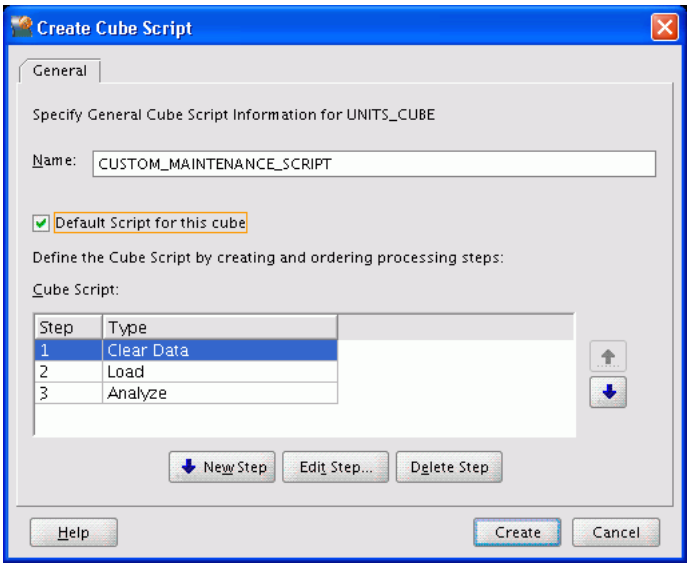
   The new step is listed on the Cube Script General tab.
7. Click **Create**.

   The new cube script appears as an item in the Cube Script folder.
8. To run the cube script:
   a. Right-click the cube script on the navigation tree, and choose **Run Cube Script**.

      The Maintenance Wizard opens.
   b. Follow the steps of the wizard.
   c. To view the results, right-click the cube and choose **View Data**.

Figure 3–18 shows the Create Cube Script dialog box, in which several steps have already been defined.

Figure 3–18 Creating a Cube Script



Adding Materialized View Capability to a Cube

Oracle OLAP cubes can be enhanced with materialized view capabilities. They 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. A cube that has been enhanced in this way is called a **cube materialized view**.

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

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 hierarchy. Ragged and skip-level hierarchies are not supported.
- All dimensions of the cube use the same aggregation operator, which is either `SUM`, `MIN`, or `MAX`.
- 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 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.

See Also: "Cube Materialized Views" on page 7-15

To add materialized view capabilities:

1. In the navigation tree, select a cube.

   The property sheets for the cube are displayed.

2. Choose the Materialized Views tab.

3. Review the check list and, if some tests failed, fix the cause of the problem.

   You cannot define a cube materialized view until the cube is valid.

4. For automatic refresh, complete just the top half of the page. For query rewrite, complete the entire page.

   Click **Help** for information about the choices on this page.

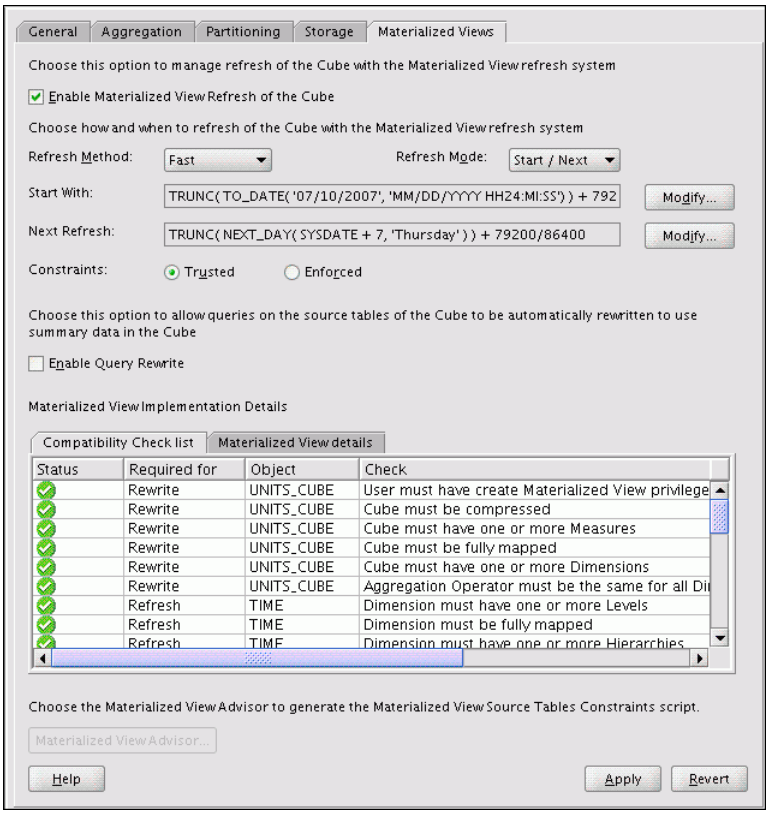
5. Click **Apply**.

The cube materialized views appear in the same schema as the analytic workspace. A materialized view is created for the cube and each of its dimensions. Cube materialized views do not store data; the data is stored in the cube. A `CB$` prefix identifies the tables as cube materialized views.

The initial state of a new materialized view is stale, so it will not support query rewrite until after it is refreshed. You can specify the first refresh time on the Materialized View tab of the cube, or you can run the Maintenance Wizard.

Figure 3–19 shows the Materialized View tab of the Units Cube. It specifies an automatic refresh of the data every Thursday at 10:00 P.M.

Figure 3–19 Defining a Materialized View



Supporting Multiple Languages

A single analytic workspace can support multiple languages. This support enables users of OLAP applications and tools to view the metadata in their native languages. For example, you can provide translations for the display names of measures, cubes, and dimensions. You can also map attributes to multiple columns, one for each language.

The number and choice of languages is restricted only by the database character set and your ability to provide translated text. Languages can be added or removed at any time.

To add support for multiple languages:

1. In the navigation tree, expand the folder for the analytic workspace.
2. Select Languages to display its property page.
3. On the General tab, click **Modify Languages**.
4. On the Modify Languages dialog box, select the languages that you want the analytic workspace to support. Use the shuttle keys to move them to the Selected Languages box.
5. Click **OK** to return to the Languages property page.
6. Enter the translations of the various labels and descriptions. Each language has a column where you can enter this information.
7. For each dimension, open the Mappings window. Map the attributes to the source columns for each language.

Defining Measure Folders

Measure folders organize and label groups of measures. Users may have access to several analytic workspaces or relational schemas with measures named Sales or Costs, and measure folders provide a way for applications to differentiate among them.

To create a measure folder:

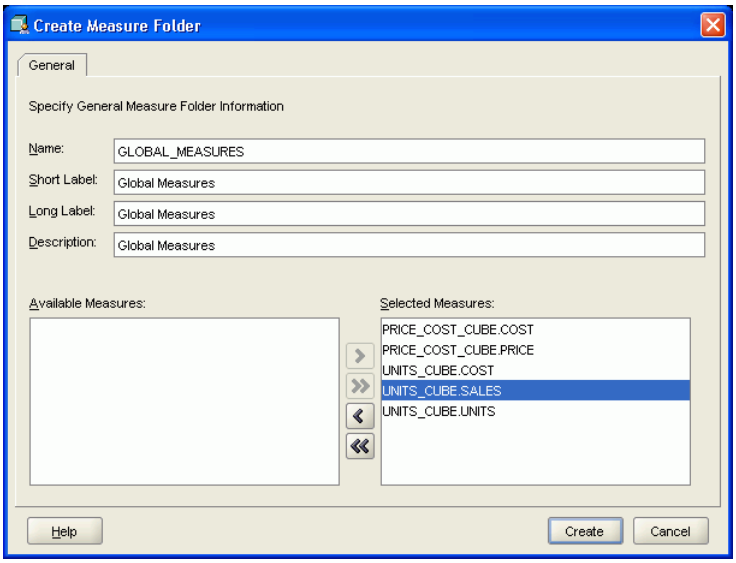
1. Expand the folder for the analytic workspace.
2. Right-click Measure Folders, then choose **Create Measure Folder** from the pop-up menu.
3. Complete the General tab of the Create Measure Folder dialog box.

   Click **Help** for specific information about these choices.

The new measure folder appears in the navigation tree under Measure Folders. You can also create subfolders.

Figure 3–20 shows creation of a measure folder.

Figure 3–20 Creating a Measure Folder



Using Templates to Re-Create Dimensional Objects

Analytic Workspace Manager enables you to save all or part of the data model as a text file. This text file contains the XML definitions of the dimensional objects, such as dimensions, levels, hierarchies, attributes, and measures. Only the metadata is saved, not the data. Templates are small files, so you can easily distribute them by email or on a Web site, just as the templates for Global and Sales History are distributed on the Oracle Web site. To re-create the dimensional objects, you simply identify the templates in Analytic Workspace Manager.

You can save the following types of objects as XML templates:

- **Analytic workspace**: Saves all dimensional objects. You can save measure folders only by saving the complete analytic workspace.
- **Dimension**: Saves the dimension and its levels, hierarchies, attributes, and mappings.
- **Cube**: Saves the cube and its measures, calculated measures, cube scripts, and mappings.

You can save the template in anywhere on your local system.

To create a template:

In the navigation tree, right-click an analytic workspace, a dimension, or a cube, and choose **Save *object* to Template**.

To re-create an analytic workspace from a template:

In the navigation tree, right-click **Analytic Workspaces** and choose **Create Analytic Workspace From Template**.

To re-create a dimension or a cube from a template:

1. Create or open an analytic workspace.
2. In the navigation tree, right-click **Dimensions** or **Cubes** and choose **Create *object* From Template**.

4

Querying Dimensional Objects

Oracle OLAP adds power to your SQL applications by providing extensive analytic content and fast query response times. A SQL query interface enables any application to query cubes and dimensions without any knowledge of OLAP.

The OLAP option automatically generates a set of relational views on cubes, dimensions, and hierarchies. 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 comply with the structure expected by your applications, using the system-generated views like base tables.

In this chapter, you will learn the basic methods for querying dimensional objects in SQL. It contains the following topics:

- Exploring the OLAP Views
- Creating Basic Queries
- Creating Hierarchical Queries
- Using Calculations in Queries
- Using Attributes for Aggregation
- Viewing Execution Plans
- Querying the Data Dictionary

See Also:

- "Developing Reports and Dashboards Using SQL Tools and Application Builders" on page 1-3 for a sample dashboard created using Oracle Application Express
- "Overview of the Dimensional Data Model" on page 1-5 for a discussion of cubes, dimensions, and hierarchies

Exploring the OLAP Views

The system-generated views are created in the same schema as the analytic workspace. Oracle OLAP provides three types of views:

- Cube views
- Dimension views
- Hierarchy views

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 will join a cube view with either a hierarchy view or a dimension view.

Cube Views

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.

Discovering the Names of the Cube Views

The default name for a cube view is *cube*_VIEW. To find the view for UNITS_CUBE in your schema, you might issue a query like this one:

```
SELECT view_name FROM user_views WHERE view_name LIKE 'UNITS_CUBE%';

VIEW_NAME
------------------------------
UNITS_CUBE_VIEW
```

The next query returns the names of all the cube views in your schema from USER_CUBE_VIEWS:

```
SELECT view_name FROM user_cube_views;

VIEW_NAME
------------------------------
UNITS_CUBE_VIEW
PRICE_CUBE_VIEW
```

Discovering the Columns of a Cube View

Like a fact table, a cube view contains a column for each measure, calculated measure, and dimension in the cube. In the following example, UNITS_CUBE_VIEW has columns for the SALES, UNITS, and COST measures, for several calculated measures on SALES, and for the TIME, CUSTOMER, PRODUCT, and CHANNEL dimensions.

```
DESCRIBE units_cube_view
Name                                      Null?    Type
----------------------------------------- -------- ----------------------------
SALES                                              NUMBER
UNITS                                              NUMBER
COST                                               NUMBER
SALES_PP                                           NUMBER
SALES_CHG_PP                                       NUMBER
SALES_PCTCHG_PP                                    NUMBER
SALES_PROD_SHARE_PARENT                            NUMBER
SALES_PROD_SHARE_TOTAL                             NUMBER
SALES_PROD_RANK_PARENT_PP                          NUMBER
TIME                                               VARCHAR2(100)
CUSTOMER                                           VARCHAR2(100)
PRODUCT                                            VARCHAR2(100)
CHANNEL                                            VARCHAR2(100)
```

The USER_CUBE_VIEW_COLUMNS data dictionary view describes the columns of a cube view, as shown by the following query.

```
SELECT column_name, column_type FROM user_cube_view_columns
     WHERE view_name = 'UNITS_CUBE_VIEW';

COLUMN_NAME                    COLUMN_TYPE
------------------------------ --------------
SALES                          MEASURE
UNITS                          MEASURE
COST                           MEASURE
SALES_PP                       MEASURE
SALES_CHG_PP                   MEASURE
SALES_PCTCHG_PP                MEASURE
SALES_PROD_SHARE_PARENT        MEASURE
SALES_PROD_SHARE_TOTAL         MEASURE
SALES_PROD_RANK_PARENT_PP      MEASURE
TIME                           KEY
CUSTOMER                       KEY
PRODUCT                        KEY
CHANNEL                        KEY

13 rows selected.
```

Displaying the Contents of a Cube View

You can display the contents of a cube view quickly with a query like this one. All levels of the data are contained in the cube, from the detail level to the top.

```
SELECT sales, units, time, customer, product, channel
     FROM units_cube_view WHERE ROWNUM < 15;

     SALES      UNITS TIME       CUSTOMER   PRODUCT    CHANNEL
---------- ---------- ---------- ---------- ---------- --------
1120292752    4000968 TOTAL      TOTAL      TOTAL      TOTAL
 134109248     330425 CY1999     TOTAL      TOTAL      TOTAL
 130276514     534069 CY2003     TOTAL      TOTAL      TOTAL
 100870877     253816 CY1998     TOTAL      TOTAL      TOTAL
 136986572     565718 CY2005     TOTAL      TOTAL      TOTAL
 140138317     584929 CY2006     TOTAL      TOTAL      TOTAL
 144290686     587419 CY2004     TOTAL      TOTAL      TOTAL
 124173522     364233 CY2000     TOTAL      TOTAL      TOTAL
  92515295     364965 CY2002     TOTAL      TOTAL      TOTAL
 116931722     415394 CY2001     TOTAL      TOTAL      TOTAL
31522409.5      88484 CY2000.Q1  TOTAL      TOTAL      TOTAL
27798426.6      97346 CY2001.Q2  TOTAL      TOTAL      TOTAL
29691668.2     105704 CY2001.Q3  TOTAL      TOTAL      TOTAL
32617248.6     138953 CY2005.Q3  TOTAL      TOTAL      TOTAL

14 rows selected.
```

Dimension and Hierarchy Views

Each dimension has one dimension view plus a hierarchy view for each hierarchy associated with the dimension. For example, a Time dimension might have these three views:

- Time dimension view
- Calendar hierarchy view
- Fiscal hierarchy view

You can use dimension views and hierarchy views like dimension tables in a star schema.

Discovering the Names of Dimension and Hierarchy Views

USER_CUBE_DIM_VIEWS identifies the dimension views for all dimensions. The default name for a dimension view is *dimension*_VIEW.

```
SELECT * FROM user_cube_dim_views;

DIMENSION_NAME                 VIEW_NAME
------------------------------ ------------------------------
PRODUCT                        PRODUCT_VIEW
CUSTOMER                       CUSTOMER_VIEW
CHANNEL                        CHANNEL_VIEW
TIME                           TIME_VIEW
```

USER_CUBE_HIER_VIEWS identifies the hierarchy views for all the dimensions. For a hierarchy view, the default name is *dimension_hierarchy*_VIEW. The following query returns the dimension, hierarchy, and view names.

```
SELECT * FROM user_cube_hier_views ORDER BY dimension_name;

DIMENSION_NAME  HIERARCHY_NAME  VIEW_NAME
--------------- --------------- ------------------------------
CHANNEL         PRIMARY         CHANNEL_PRIMARY_VIEW
CUSTOMER        SEGMENT         CUSTOMER_SEGMENT_VIEW
CUSTOMER        SHIPMENTS       CUSTOMER_SHIPMENTS_VIEW
PRODUCT         PRIMARY         PRODUCT_PRIMARY_VIEW
TIME            FISCAL          TIME_FISCAL_VIEW
TIME            CALENDAR        TIME_CALENDAR_VIEW
```

Discovering the Columns of a Dimension View

Like a dimension table, a dimension view contains a key column, level name, level keys for every level of every hierarchy associated with the dimension, and attribute columns. In the following example, TIME_VIEW has a column for the dimension keys, the level name, and the dimension attributes.

```
DESCRIBE time_view
Name                                    Null?    Type
--------------------------------------- -------- ----------------------------
DIM_KEY                                          VARCHAR2(100)
LEVEL_NAME                                       VARCHAR2(30)
DIM_ORDER                                        NUMBER
END_DATE                                         DATE
LONG_DESCRIPTION                                 VARCHAR2(100)
SHORT_DESCRIPTION                                VARCHAR2(100)
TIME_SPAN                                        NUMBER
```

USER_CUBE_DIM_VIEW_COLUMNS describes the information in each column, as shown in this query.

```
SELECT column_name, column_type FROM user_cube_dim_view_columns
     WHERE view_name ='TIME_VIEW';

COLUMN_NAME                    COLUMN_TYPE
------------------------------ --------------------
DIM_KEY                        KEY
LEVEL_NAME                     LEVEL_NAME
DIM_ORDER                      DIM_ORDER
END_DATE                       ATTRIBUTE
TIME_SPAN                      ATTRIBUTE
LONG_DESCRIPTION               ATTRIBUTE
SHORT_DESCRIPTION              ATTRIBUTE

7 rows selected.
```

Displaying the Contents of a Dimension View

The following query displays the level and attributes of each dimension key.

```
SELECT dim_key, level_name, long_description description, time_span, end_date
     FROM time_view WHERE dim_key LIKE '%2005%';

DIM_KEY      LEVEL_NAME          DESCRIPTION   TIME_SPAN END_DATE
------------ ------------------- ------------ ---------- ---------
CY2005       CALENDAR_YEAR       2005                365 31-DEC-05
CY2005.Q2    CALENDAR_QUARTER    Q2.05                91 30-JUN-05
CY2005.Q4    CALENDAR_QUARTER    Q4.05                92 31-DEC-05
CY2005.Q3    CALENDAR_QUARTER    Q3.05                92 30-SEP-05
CY2005.Q1    CALENDAR_QUARTER    Q1.05                90 31-MAR-05
2005.01      MONTH               JAN-05               31 31-JAN-05
2005.05      MONTH               MAY-05               31 31-MAY-05
2005.07      MONTH               JUL-05               31 31-JUL-05
2005.03      MONTH               MAR-05               31 31-MAR-05
2005.04      MONTH               APR-05               30 30-APR-05
2005.08      MONTH               AUG-05               31 31-AUG-05
2005.09      MONTH               SEP-05               30 30-SEP-05
2005.02      MONTH               FEB-05               28 28-FEB-05
2005.11      MONTH               NOV-05               30 30-NOV-05
2005.06      MONTH               JUN-05               30 30-JUN-05
2005.10      MONTH               OCT-05               31 31-OCT-05
2005.12      MONTH               DEC-05               31 31-DEC-05
FY2005       FISCAL_YEAR         FY2005              365 30-JUN-05
FY2005.Q4    FISCAL_QUARTER      Q4 FY-05             91 30-JUN-05
FY2005.Q1    FISCAL_QUARTER      Q1 FY-05             92 30-SEP-04
FY2005.Q2    FISCAL_QUARTER      Q2 FY-05             92 31-DEC-04
FY2005.Q3    FISCAL_QUARTER      Q3 FY-05             90 31-MAR-05

22 rows selected.
```

Discovering the Columns of a Hierarchy View

Like the dimension views, the hierarchy views also contain columns for the dimension key, level name, and level keys. However, all of the rows and columns are associated with the dimension keys that belong to the hierarchy.

```
DESCRIBE time_calendar_view
Name                                      Null?    Type
----------------------------------------- -------- ----------------------------
DIM_KEY                                            VARCHAR2(100)
LEVEL_NAME                                         VARCHAR2(30)
DIM_ORDER                                          NUMBER
HIER_ORDER                                         NUMBER
LONG_DESCRIPTION                                   VARCHAR2(100)
SHORT_DESCRIPTION                                  VARCHAR2(100)
END_DATE                                           DATE
TIME_SPAN                                          NUMBER
PARENT                                             VARCHAR2(100)
TOTAL                                              VARCHAR2(100)
CALENDAR_YEAR                                      VARCHAR2(100)
CALENDAR_QUARTER                                   VARCHAR2(100)
MONTH                                              VARCHAR2(100)
```

Displaying the Contents of a Hierarchy View

The following query displays the dimension keys, parent key, and the full ancestry for calendar year 2005.

```
SELECT dim_key, long_description description, parent, calendar_year year,
       calendar_quarter quarter, month FROM time_calendar_view
       WHERE calendar_year='CY2005'
       ORDER BY level_name, end_date;

DIM_KEY      DESCRIPTION  PARENT       YEAR         QUARTER      MONTH
------------ ------------ ------------ ------------ ------------ ------------
CY2005.Q1    Q1.05        CY2005       CY2005       CY2005.Q1
CY2005.Q2    Q2.05        CY2005       CY2005       CY2005.Q2
CY2005.Q3    Q3.05        CY2005       CY2005       CY2005.Q3
CY2005.Q4    Q4.05        CY2005       CY2005       CY2005.Q4
CY2005       2005         TOTAL        CY2005
2005.01      JAN-05       CY2005.Q1    CY2005       CY2005.Q1    2005.01
2005.02      FEB-05       CY2005.Q1    CY2005       CY2005.Q1    2005.02
2005.03      MAR-05       CY2005.Q1    CY2005       CY2005.Q1    2005.03
2005.04      APR-05       CY2005.Q2    CY2005       CY2005.Q2    2005.04
2005.05      MAY-05       CY2005.Q2    CY2005       CY2005.Q2    2005.05
2005.06      JUN-05       CY2005.Q2    CY2005       CY2005.Q2    2005.06
2005.07      JUL-05       CY2005.Q3    CY2005       CY2005.Q3    2005.07
2005.08      AUG-05       CY2005.Q3    CY2005       CY2005.Q3    2005.08
2005.09      SEP-05       CY2005.Q3    CY2005       CY2005.Q3    2005.09
2005.10      OCT-05       CY2005.Q4    CY2005       CY2005.Q4    2005.10
2005.11      NOV-05       CY2005.Q4    CY2005       CY2005.Q4    2005.11
2005.12      DEC-05       CY2005.Q4    CY2005       CY2005.Q4    2005.12

17 rows selected.
```

Creating Basic Queries

Querying a cube is similar to querying a star schema. In a star schema, you join a fact table to a dimension table. The fact table provides the numerical business measures, and the dimension table provides descriptive attributes that give meaning to the data.

Similarly, you join a cube view with either a dimension view or a hierarchy view to provide fully identified and meaningful data to your users.

For dimensions with no hierarchies, use the dimension views in your queries. For dimensions with hierarchies, use the hierarchy views, because they contain more information than the dimension views.

When querying a cube, remember these guidelines:

- Apply a filter to every dimension.

  The cube contains both detail level and aggregated data. A query with an unfiltered dimension typically returns more data than users need, which negatively impacts performance.

- Let the cube aggregate the data.

  Because the aggregations are already calculated in the cube, a typical query does not need a `GROUP BY` clause. Simply select the aggregations you want by using the appropriate filters on the dimension keys or attributes.

Applying a Filter to Every Dimension

To create a level filter, you must know the names of the dimension levels. You can easily acquire them by querying the dimension or hierarchy views:

```
SELECT DISTINCT level_name FROM time_calendar_view;

LEVEL_NAME
------------------------------
CALENDAR_YEAR
CALENDAR_QUARTER
MONTH
TOTAL
```

Several data dictionary views list the names of the levels. The following example queries `USER_CUBE_HIER_LEVELS`.

```
SELECT level_name FROM user_cube_hier_levels
     WHERE dimension_name = 'TIME' AND hierarchy_name ='CALENDAR';

LEVEL_NAME
--------------------
TOTAL
CALENDAR_YEAR
CALENDAR_QUARTER
MONTH
```

To see the importance of applying a filter to every dimension, consider the query in Example 4–1, which has no filter on the time dimension.

Example 4–1 Displaying Aggregates at All Levels of Time

```
/* Select key descriptions and facts */
SELECT t.long_description time,
     ROUND(f.sales) sales
/* From dimension views and cube view */
  FROM time_calendar_view t,
     product_primary_view p,
     customer_shipments_view cu,
     channel_primary_view ch,
     units_cube_view f
```

```
/* No filter on Time */
  WHERE p.level_name = 'TOTAL'
     AND cu.level_name = 'TOTAL'
     AND ch.level_name = 'TOTAL'
/* Join dimension views to cube view */
     AND t.dim_key = f.time
     AND p.dim_key = f.product
     AND cu.dim_key = f.customer
     AND ch.dim_key = f.channel
  ORDER BY t.end_date;
```

Without a filter on the Time dimension, the query returns values for every level of time. This is more data than users typically want to see, and the volume of data returned can negatively impact performance.

```
TIME            SALES
---------- ----------
JAN-98        8338545
FEB-98        7972132
Q1.98        24538588
MAR-98        8227911
APR-98        8470315
MAY-98        8160573
JUN-98        8362386
Q2.98        24993273
JUL-98        8296226
AUG-98        8377587
SEP-98        8406728
Q3.98        25080541
OCT-98        8316169
NOV-98        8984156
Q4.98        26258474
1998        100870877
                .
                .
                .
```

Now consider the results when a filter restricts Time to yearly data.

Example 4–2 shows a basic query. It selects the Sales measure from `UNITS_CUBE_VIEW`, and joins the keys from the cube view to the hierarchy views to get descriptions of the keys.

Example 4–2 Basic Cube View Query

```
/* Select key descriptions and facts */
SELECT t.long_description time,
     ROUND(f.sales) sales
/* From dimension views and cube view */
  FROM time_calendar_view t,
     product_primary_view p,
     customer_shipments_view cu,
     channel_primary_view ch,
     units_cube_view f
/* Create level filters */
  WHERE t.level_name = 'CALENDAR_YEAR'
     AND p.level_name = 'TOTAL'
     AND cu.level_name = 'TOTAL'
     AND ch.level_name = 'TOTAL'
```

```
/* Join dimension views to cube view */
     AND t.dim_key = f.time
     AND p.dim_key = f.product
     AND cu.dim_key = f.customer
     AND ch.dim_key = f.channel
  ORDER BY t.end_date;
```

Example 4–2 selects the following rows. For CUSTOMER, PRODUCT, and CHANNEL, only one value is at the top level. TIME has a value for each calendar year.

```
TIME          SALES
-------- ----------
1998      100870877
1999      134109248
2000      124173522
2001      116931722
2002       92515295
2003      130276514
2004      144290686
2005      136986572
2006      140138317

9 rows selected.
```

Dimension attributes also provide a useful way to select the data for a query. The WHERE clause in Example 4–3 uses attributes values to filter all of the dimensions.

Example 4–3 Selecting Data With Attribute Filters

```
/* Select key descriptions and facts */
SELECT t.long_description time,
     p.long_description product,
     cu.long_description customer,
     ch.long_description channel,
     ROUND(f.sales) sales
/* From dimension views and cube view */
FROM time_calendar_view t,
     product_primary_view p,
     customer_shipments_view cu,
     channel_primary_view ch,
     units_cube_view f
/* Create attribute filters */
WHERE t.long_description in ('2005', '2006')
    AND p.package = 'Laptop Value Pack'
    AND cu.long_description LIKE '%Boston%'
    AND ch.long_description = 'Internet'
/* Join dimension views to cube view */
    AND t.dim_key = f.time
    AND p.dim_key = f.product
    AND cu.dim_key = f.customer
    AND ch.dim_key = f.channel
ORDER BY time, customer;
```

The query select two calendar years, the products in the Laptop Value Pack, the customers in Boston, and the Internet channel.

```
TIME   PRODUCT                        CUSTOMER              CHANNEL       SALES
------ ------------------------------ --------------------- -------- ----------
2005   Laptop carrying case           KOSH Entrpr Boston    Internet       5936
2005   56Kbps V.92 Type II Fax/Modem  KOSH Entrpr Boston    Internet      45285
2005   Internal 48X CD-ROM            KOSH Entrpr Boston    Internet       2828
2005   Standard Mouse                 KOSH Entrpr Boston    Internet        638
2005   Envoy Standard                 Warren Systems Boston Internet      19359
2005   Laptop carrying case           Warren Systems Boston Internet      13434
2005   Standard Mouse                 Warren Systems Boston Internet        130
2006   Standard Mouse                 KOSH Entrpr Boston    Internet        555
2006   Laptop carrying case           KOSH Entrpr Boston    Internet       6357
2006   56Kbps V.92 Type II Fax/Modem  KOSH Entrpr Boston    Internet      38042
2006   Internal 48X CD-ROM            KOSH Entrpr Boston    Internet       3343
2006   Envoy Standard                 Warren Systems Boston Internet      24198
2006   Laptop carrying case           Warren Systems Boston Internet      13153
2006   Standard Mouse                 Warren Systems Boston Internet         83

14 rows selected.
```

Allowing the Cube to Aggregate the Data

A cube contains all of the aggregate data. As shown in this chapter, a query against a cube just needs to select the aggregate data, not calculate the values.

The following is a basic query against a fact table:

```
/* Querying a fact table */
SELECT t.calendar_year_dsc time,
     SUM(f.sales) sales
  FROM time_dim t, units_fact f
  WHERE t.calendar_year_dsc IN ('2005', '2006')
     AND t.month_id = f.month_id
  GROUP BY t.calendar_year_dsc;
```

The next query fetches the exact same results from a cube using filters:

```
/* Querying a cube */
SELECT t.long_description time, f.sales sales
  FROM time_calendar_view t,
     product_primary_view p,
     customer_shipments_view cu,
     channel_primary_view ch,
     units_cube_view f
/* Apply filters to every dimension */
  WHERE t.long_description IN ('2005', '2006')
     AND p.level_name = 'TOTAL'
     AND cu.level_name = 'TOTAL'
     AND ch.level_name = 'TOTAL'
/* Join dimension views to cube view */
     AND t.dim_key = f.TIME
     AND p.dim_key = f.product
     AND cu.dim_key = f.customer
     AND ch.dim_key = f.channel
  ORDER BY time;
```

Both queries return these results:

```
TIME       SALES
----- ----------
2005   136986572
2006   140138317
```

The query against the cube does not compute the aggregate values with a SUM operator and GROUP BY clause. Because the aggregates exist already in the cube, this would re-aggregate previously aggregated data. Instead, the query selects the aggregates directly from the cube and specifies the desired aggregates by applying the appropriate filter to each dimension.

Query Processing

The most efficient queries allow the OLAP engine to filter the data, so that the minimum number of rows required by the query are returned to SQL.

The following are among the WHERE clause operations that are pushed into the OLAP engine for processing:

- =
- !=
- >
- !>
- <
- !<
- IN
- NOT IN
- IS NULL
- LIKE
- NOT LIKE

The OLAP engine also processes nested character functions, including INSTR, LENGTH, NVL, LOWER, UPPER, LTRIM, RTRIM, TRIM, LPAD, RPAD, and SUBSTR.

SQL processes other operations and functions in the WHERE clause, and all operations in other parts of the SELECT syntax.

Creating Hierarchical Queries

Drilling is an important capability in business analysis. In a dashboard or an application, users click on a dimension key to change the selection of data. Decision makers frequently want to drill down to see the contributors to a data value, or drill up to see how a particular data value contributes to the whole. For example, the Boston regional sales manager might start at total Boston sales, drill down to see the contributions of each sales representative, then drill up to see how the Boston region contributes to the New England sales total.

The hierarchy views include a PARENT column that identifies the parent of every dimension key. This column encapsulates all of the hierarchical information of the dimension: If you know the parent of every key, then you can derive the ancestors, the children, and the descendants.

For level-based hierarchies, the LEVEL_NAME column supplements this information by providing a convenient way to identify all the keys at the same depth in the hierarchy, from the top to the base. For value-based hierarchies, the PARENT column provides all the information about the hierarchy.

> **See Also:** Chapter 6, "Developing Reports and Dashboards" about using bind variables to support drilling

Drilling Down to Children

You can use the PARENT column of a hierarchy view to select only the children of a particular value. The following WHERE clause selects the children of calendar year 2005.

```
/* Select children of calendar year 2005 */
WHERE  t.parent = 'CY2005'
   AND p.dim_key = 'TOTAL'
   AND cu.dim_key = 'TOTAL'
   AND ch.dim_key = 'TOTAL'
```

The query drills down from Year to Quarter. The four quarters Q1-05 to Q4-05 are the children of year CY2005 in the Calendar hierarchy.

```
TIME          SALES
-------- ----------
Q1.05      31381338
Q2.05      37642741
Q3.05      32617249
Q4.05      35345244
```

Drilling Up to Parents

The PARENT column of a hierarchy view identifies the parent of each dimension key. Columns of level keys identify the full heritage. The following WHERE clause selects the parent of a Time key based on its LONG_DESCRIPTION attribute.

```
/* Select the parent of a Time key*/
WHERE t.dim_key =
   (SELECT DISTINCT parent
      FROM time_calendar_view
      WHERE long_description='JAN-05')
   AND p.dim_key= 'TOTAL'
   AND cu.dim_key = 'TOTAL'
   AND ch.dim_key = 'TOTAL'
```

The query drills up from Month to Quarter. The parent of month JAN-05 is the quarter Q1-05 in the Calendar hierarchy.

```
TIME          SALES
-------- ----------
Q1.05      31381338
```

Drilling Down to Descendants

The following WHERE clause selects the descendants of calendar year 2005 by selecting the rows with a LEVEL_NAME of MONTH and a CALENDAR_YEAR of CY2005.

```
/* Select Time level and ancestor */
WHERE t.level_name = 'MONTH'
    AND t.calendar_year = 'CY2005'
    AND p.dim_key = 'TOTAL'
    AND cu.dim_key = 'TOTAL'
    AND ch.dim_key = 'TOTAL'
```

The query drills down two levels, from year to quarter to month. The 12 months Jan-05 to Dec-05 are the descendants of year 2005 in the Calendar hierarchy.

```
TIME          SALES
-------- ----------
JAN-05     12093518
FEB-05     10103162
MAR-05      9184658
APR-05      9185964
MAY-05     11640216
JUN-05     16816561
JUL-05     11110903
AUG-05      9475807
SEP-05     12030538
OCT-05     11135032
NOV-05     11067754
DEC-05     13142459
```

Drilling Up to Ancestors

The hierarchy views provide the full ancestry of each dimension key, as shown in "Displaying the Contents of a Hierarchy View" on page 4-6. The following WHERE clause uses the CALENDAR_YEAR level key column to identify the ancestor of a MONTH dimension key.

```
/* Select the ancestor of a Time key based on its Long Description attribute */
WHERE t.dim_key =
   (SELECT calendar_year
      FROM time_calendar_view
      WHERE long_description = 'JAN-05')
   AND p.dim_key = 'TOTAL'
   AND cu.dim_key = 'TOTAL'
   AND ch.dim_key = 'TOTAL'
```

The query drills up two levels from month to quarter to year. The ancestor of month Jan-05 is the year 2005 in the Calendar hierarchy.

```
TIME          SALES
-------- ----------
2005      136986572
```

Using Calculations in Queries

A DBA can create calculated measures in Analytic Workspace Manager, so they are available to all applications. This not only simplifies application development, but ensures that all applications use the same name for the same calculation.

Nonetheless, you may want to develop queries that include your own calculations. In this case, you can use an inner query to select aggregate data from the cube, then

perform calculations in an outer query. You can select data from cubes that use any type of aggregation operators, and you can use any functions or operators in the query. You only need to make sure that you select the data from the cube at the appropriate levels for the calculation, and that the combination of operators in the cube and in the query create the calculation you want.

Example 4–4 shows a query that answers the question, What was the average sales of Sentinel Standard computers to Government customers for the third quarter of fiscal year 2005. UNITS_CUBE is summed over all dimensions, so that FY2005.Q3 is a total for July, August, and September. The inner query extracts the data for these months, and the outer query uses the MIN, MAX, and AVG operator s and a GROUP BY clause to calculate the averages.

Example 4–4 Calculating Average Sales Across Customers

```
SELECT customer, ROUND(MIN(sales)) minimum, ROUND(MAX(sales)) maximum,
   ROUND(AVG(sales)) average
FROM
   (SELECT cu.long_description customer,
       f.sales sales
   FROM time_fiscal_view t,
     product_primary_view p,
     customer_segment_view cu,
     channel_primary_view ch,
     units_cube_view f
   WHERE t.parent = 'FY2005.Q3'
     AND p.dim_key = 'SENT STD'
     AND cu.parent = 'GOV'
     AND ch.level_name = 'TOTAL'
     AND t.dim_key = f.time
     AND p.dim_key = f.product
     AND cu.dim_key = f.customer
     AND ch.dim_key = f.channel
   )
GROUP BY customer
ORDER BY customer;
```

This is the data extracted from the cube by the inner query:

```
CUSTOMER                                TIME          SALES
--------------------------------------- -------- ----------
Dept. of Labor                          JAN-05      1553.26
Dept. of Labor                          MAR-05       1555.6
Ministry of Intl Trade                  JAN-05      1553.26
Ministry of Intl Trade                  FEB-05      1554.56
Ministry of Intl Trade                  MAR-05       1555.6
Royal Air Force                         JAN-05      1553.26
Royal Air Force                         FEB-05      6218.23
UK Environmental Department             JAN-05      4659.78
UK Environmental Department             FEB-05      3109.12
```

The outer query calculates the minimum, maximum, and average sales for each customer:

```
CUSTOMER                         MINIMUM    MAXIMUM    AVERAGE
------------------------------ ---------- ---------- ----------
Dept. of Labor                       1553       1556       1554
Ministry of Intl Trade               1553       1556       1554
Royal Air Force                      1553       6218       3886
UK Environmental Department          3109       4660       3884
```

Using Attributes for Aggregation

An OLAP cube aggregates the data within its hierarchies, using the parent-child relationships revealed in the hierarchy views. The OLAP engine does not calculate aggregates over dimension attribute values.

Nonetheless, you may want to aggregate products over color or size, or customers by age, zip code, or population density. This is the situation when you can use a `GROUP BY` clause when querying a cube. Your query can extract data from the cube, then use SQL to aggregate by attribute value.

The cube must use the same aggregation operator for all dimensions, and the aggregation operator in the `SELECT` list of the query must match the aggregation operator of the cube. You can use a `GROUP BY` clause to query cubes that use these operators:

- First Non-NA Value
- Last Non-NA Value
- Maximum
- Minimum
- Sum

Aggregating Measures Over Attributes

Example 4–5 shows a query that aggregates over an attribute named Package. It returns these results:

```
TIME   PACKAGE                 SALES
------ ------------------ ----------
2005   All                1809157.64
2005   Multimedia         18083256.3
2005   Executive            19836977
2005   Laptop Value Pack  9547494.81
```

Units Cube uses the `SUM` operator for all dimensions, and the query uses the `SUM` operator to aggregate over Sales. The Package attribute applies only to the Item level of the Product dimension, so the query selects the Item level of Product. It also eliminates nulls for Package, so that only products that belong to a package are included in the calculation. The `GROUP BY` clause breaks out Total Sales by Time and Package

Example 4–5 Aggregating Over an Attribute

```
SELECT t.long_description time,
     p.package package,
     SUM(f.sales) sales
  FROM time_calendar_view t,
     product_primary_view p,
     customer_shipments_view cu,
     channel_primary_view ch,
     units_cube_view f
/* Select Product by level and attribute */
  WHERE p.level_name = 'ITEM'
     AND p.package IS NOT NULL
     AND t.long_description = '2005'
     AND cu.level_name = 'TOTAL'
     AND ch.level_name = 'TOTAL'
```

```
/* Join dimensions and cube */
     AND t.dim_key = f.time
     AND p.dim_key = f.product
     AND cu.dim_key = f.customer
     AND ch.dim_key = f.channel
  GROUP BY t.long_description, p.package;
```

Aggregating Calculated Measures Over Attributes

Before using the technique described in "Aggregating Measures Over Attributes" on page 4-15, be sure that the calculation is meaningful. For example, the common calculation Percent Change might be defined as a calculated measure in a cube. Summing over Percent Change would produce unexpected results, because the calculation for Percent Change (*(a-b)/b*,) is not additive.

Consider the following rows of data. The correct Total Percent Change is `.33`, whereas the sum of the percent change for the first two rows is `.75`.

Row	Sales	Sales Prior Period	Percent Change
1	15	10	.50
2	25	20	.25
Total	**40**	**30**	**.33**

Example 4–6 shows a query that aggregates over the Package attribute and calculates Percent Change From Prior Period. The inner query aggregates Sales and Sales Prior Period over the attributes, and the outer query uses the results to compute the percent change. These are the results of the query, which show the expected results for `PCT_CHG_PP`:

```
TIME   PACKAGE                SALES PRIOR_PERIOD PCT_CHG_PP
------ ------------------ ---------- ------------ ----------
2005   All                1809157.64   1853928.06 -.02414895
2006   All                1720399.03   1809157.64 -.04906074
2005   Executive            19836977   20603879.8 -.03722128
2006   Executive          19580638.4     19836977 -.01292226
2005   Laptop Value Pack  9547494.81   10047298.6 -.04974509
2006   Laptop Value Pack  9091450.58   9547494.81 -.04776585
2005   Multimedia         18083256.3   19607675.5 -.07774604
2006   Multimedia         18328678.7   18083256.3 .013571806

8 rows selected.
```

Example 4–6 Querying Over Attributes Using Calculated Measures

```
/* Calculate Percent Change */
SELECT TIME, package, sales, prior_period,
     ((sales - prior_period) / prior_period) pct_chg_pp
FROM
/* Fetch data from the cube and aggregate over Package */
     (SELECT t.long_description time,
          p.package package,
          SUM(f.sales) sales,
          SUM(f.sales_pp) prior_period
```

```
      FROM time_calendar_view t,
         product_primary_view p,
         customer_shipments_view cu,
         channel_primary_view ch,
         units_cube_view f
/* Create filters */
      WHERE p.level_name = 'ITEM'
         AND p.package IS NOT NULL
         AND t.long_description IN ('2005', '2006')
         AND cu.level_name = 'TOTAL'
         AND ch.level_name = 'TOTAL'
/* Join dimension views to cube view */
         AND t.dim_key = f.time
         AND p.dim_key = f.product
         AND cu.dim_key = f.customer
         AND ch.dim_key = f.channel
      GROUP BY t.long_description, p.package
      ORDER BY p.package);
```

Viewing Execution Plans

You can generate and view execution plans for queries against cubes and dimensions the same as for those against relational tables.

The SQL `EXPLAIN PLAN` command creates a table with the content of the explain plan. The default table name is `PLAN_TABLE`.

Generating Execution Plans

The following command creates an execution plan for a basic query on a cube:

```
EXPLAIN PLAN FOR
  SELECT t.long_description time,
     p.long_description product,
     cu.long_description customer,
     ch.long_description channel,
     f.sales sales
  FROM time_calendar_view t,
     product_primary_view p,
     customer_shipments_view cu,
     channel_primary_view ch,
     units_cube_view f
  WHERE t.level_name = 'CALENDAR_YEAR'
     AND p.level_name = 'TOTAL'
     AND cu.level_name = 'TOTAL'
     AND ch.level_name = 'TOTAL'
     AND t.dim_key = f.TIME
     AND p.dim_key = f.product
     AND cu.dim_key = f.customer
     AND ch.dim_key = f.channel
  ORDER BY t.end_date;
```

Example 4–7 shows selected columns of the execution plan. A `CUBE SCAN` operation is performed. The plan option is `PARTIAL OUTER`, which is described in "Types of Execution Plans" on page 4-19.

Example 4–7 Selected Columns From PLAN_TABLE

```
SQL> SELECT operation, options, object_name FROM plan_table;

OPERATION            OPTIONS              OBJECT_NAME
-------------------- -------------------- ---------------
SELECT STATEMENT
SORT                 ORDER BY
JOINED CUBE SCAN     PARTIAL OUTER
CUBE ACCESS                               UNITS_CUBE
CUBE ACCESS                               CHANNEL
CUBE ACCESS                               CUSTOMER
CUBE ACCESS                               PRODUCT
CUBE ACCESS                               TIME

8 rows selected.
```

The `DISPLAY` table function of the `DBMS_XPLAN` PL/SQL package formats and displays information from an execution plan, as shown in Example 4–8.

Example 4–8 Formatted Execution Plan From DBMS_XPLAN

```
SQL> SELECT plan_table_output FROM TABLE(dbms_xplan.display());

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------
Plan hash value: 1667678335

-------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |            |     1 |   100 |   104   (3)| 00:00:02 |
|   1 |  SORT ORDER BY                 |            |     1 |   100 |   104   (3)| 00:00:02 |
|   2 |   JOINED CUBE SCAN PARTIAL OUTER|           |       |       |            |          |
|   3 |    CUBE ACCESS                 | UNITS_CUBE |       |       |            |          |
|   4 |    CUBE ACCESS                 | CHANNEL    |       |       |            |          |
|   5 |    CUBE ACCESS                 | CUSTOMER   |       |       |            |          |
|   6 |    CUBE ACCESS                 | PRODUCT    |       |       |            |          |
|*  7 |    CUBE ACCESS                 | TIME       |     1 |   100 |   103   (2)| 00:00:02 |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   7 - filter(SYS_OP_ATG(VALUE(KOKBF$),12,13,2)='CALENDAR_YEAR' AND
              SYS_OP_ATG(VALUE(KOKBF$),43,44,2)='TOTAL' AND
              SYS_OP_ATG(VALUE(KOKBF$),33,34,2)='TOTAL' AND
              SYS_OP_ATG(VALUE(KOKBF$),23,24,2)='TOTAL')

22 rows selected.
```

Types of Execution Plans

Table 4–1 describes the types of execution plans for cubes.

Table 4–1 Descriptions of Execution Plans for Cubes and Dimensions

Operation	Option	Description
CUBE SCAN	--	Uses inner joins for all cube access.
CUBE SCAN	PARTIAL OUTER	Uses an outer join for least one dimension, and inner joins for the other dimensions.
CUBE SCAN	OUTER	Uses outer joins for all cube access.

Querying the Data Dictionary

If you are developing a generic application -- that is, one where the names of the dimensional objects are not known -- then your application can retrieve this information from the data dictionary.

Among the static views of the database data dictionary are those that provide information about dimensional objects. All OLAP metadata is stored in the data dictionary. A few of the data dictionary views were introduced previously in this chapter.

Table 4–2 provides brief descriptions of the ALL views. There are corresponding DBA and USER views.

Table 4–2 Static Data Dictionary Views for OLAP

View	Description
ALL_CUBE_ATTR_VISIBILITY	Describes the visibility of the attributes for cube dimensions.
ALL_CUBE_ATTRIBUTES	Describes the attributes for cube dimensions.
ALL_CUBE_BUILD_PROCESSES	Describes the cube build processes and maintenance scripts.
ALL_CUBE_CALCULATED_MEMBERS	Describes the calculated members (keys) for cube dimensions.
ALL_CUBE_DIM_LEVELS	Describes the cube dimension levels.
ALL_CUBE_DIM_MODELS	Describes the models for cube dimensions.
ALL_CUBE_DIM_VIEW_COLUMNS	Describes the columns of the system-generated relational views of cube dimensions.
ALL_CUBE_DIM_VIEWS	Describes the system-generated relational views of OLAP dimensions.
ALL_CUBE_DIMENSIONALITY	Describes the dimension order of the OLAP cubes.
ALL_CUBE_DIMENSIONS	Describes the cube dimensions.
ALL_CUBE_HIER_LEVELS	Describes the hierarchy levels for cube dimensions.
ALL_CUBE_HIER_VIEW_COLUMNS	Describes the columns of relational hierarchy views of cube dimensions.
ALL_CUBE_HIER_VIEWS	Describes the hierarchies for cube dimensions.
ALL_CUBE_HIERARCHIES	Describes the OLAP dimension hierarchies.
ALL_CUBE_MEASURES	Describes the measures in the OLAP cubes.

Table 4–2 (Cont.) Static Data Dictionary Views for OLAP

View	Description
ALL_CUBE_VIEW_COLUMNS	Describes the columns of the relational views of OLAP cubes.
ALL_CUBE_VIEWS	Describes the system-generated relational views of OLAP cubes.
ALL_CUBES	Describes the OLAP cubes.
ALL_MEASURE_FOLDER_CONTENTS	Describes the contents of OLAP measure folders.
ALL_MEASURE_FOLDERS	Describes the OLAP measure folders.

See Also: *Oracle Database Reference* for full descriptions of data dictionary views.

5

Enhancing Your Database With Analytic Content

Oracle OLAP provides an extensive set of analytic functions for enhancing your database with information-rich content. This chapter explains how you can use Analytic Workspace Manager to create calculated measures using templates and free-form calculations.

This chapter contains the following topics:

What Is a Calculated Measure?

Calculated measures return values that are computed at run-time from data stored in one or more measures. Like relational views, calculated measures store queries against data stored in other objects. Because calculated measures do not store data, you can create dozens of them without increasing the size of the database. You can use them as the basis for defining other calculated measures, which adds depth to the types of calculations you can create using the templates in Analytic Workspace Manager.

As soon as you create a calculated measure, it appears as a column in a cube view. Because calculated measures do not contain data, they are not associated with a build process. You can create a calculated measure at any time, and it is available immediately for querying by SQL applications.

Functions for Defining Calculations

The library of functions for defining calculated measures contains these basic categories:

- Arithmetic Operators: Perform calculations on the values of two measures.
- Analytic Functions: Perform calculations on an ordered series or a range of values in a single measure or column.
- Single-Row Functions: Perform calculations on a value in a single row.

Arithmetic Operators

You can perform the following arithmetic operations using two measures. The calculations in the cube are performed on a cell-by-cell basis at all levels of the dimension hierarchies.

- **Addition**: Adds the values of two measures.
- **Subtraction**: Subtracts the values of one measure from the values of another measure.
- **Multiplication**: Multiplies the values of two measures.
- **Division** or **Ratio**: Divides the values of one measure by the values of another measure.
- **Percent Difference**: Calculates the percent difference between the values of two measures.

The arithmetic operations are available in Analytic Workspace Manager as templates. as described in "Using Calculation Templates" on page 5-5.

Analytic Functions

The analytic functions provide the most powerful computations and fuel the most useful queries for business intelligence and similar applications. They include a variety of rank, share, time series, and other single-column functions. The analytic functions enable analysts and decision makers to make comparisons and identify trends.

Analytic functions provided by Oracle OLAP leverage the knowledge associated with the dimensions about levels and family relationships. Time dimensions have additional information that enables them to support time series methods such as lags, leads, moving and cumulative calculations. Because the knowledge is stored with the dimension, you do not need to specify these relationships when creating a calculated measure.

The analytic functions are available in Analytic Workspace Manager as templates. They are described in "Using Calculation Templates" on page 5-5.

Single-Row Functions

Oracle OLAP supports most of the SQL single-row functions including:

- Numeric functions such as `ABS`, `CEIL`, `FLOOR`, `POWER`, `ROUND`, and `TRUNC`.
- Character functions such as `CONCAT`, `LPAD`, `RPAD`, `LTRIM`, `RTRIM`, `REPLACE`, and `SUBSTR`.
- Datetime functions such as `CURRENT_DAY`, `MONTHS_BETWEEN`, `NEXT_DAY`, and `SYSTIMESTAMP`.
- Comparison functions `GREATEST` and `LEAST`.
- Conversion functions such as `TO_CHAR`, `TO_DATE`, `TO_NUMBER`, and `TO_TIMESTAMP`.

You can use these functions to manipulate the data values in a measure, typically as part of a more complex calculation. These functions are not available as templates, but you can use them in free-form calculations, as described in "Creating User-Defined Expressions" on page 5-12.

Creating Calculated Measures

Analytic Workspace Manager provides easy-to-use templates for creating calculated measures. You can create them in the same cube with the source measures, or you can create them in a separate cube.

Calculated measures are available for querying as additional columns in a cube view (such as `UNITS_CUBE_VIEW`). They are not available through cube materialized views (such as `CB$UNITS_CUBE`).

To create a calculated measure:

1. In the navigation tree, expand an existing cube or create a new cube.
2. Right-click Calculated Measures, then choose **Create Calculated Measure** from the pop-up menu.

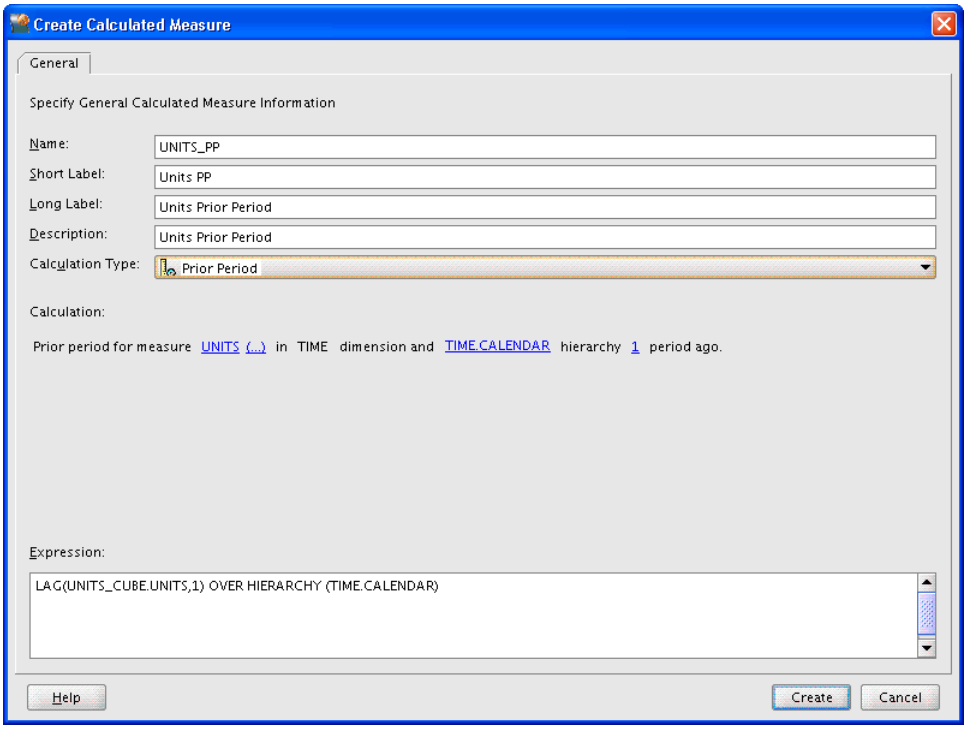
   The Create Calculated Measure dialog box is displayed.
3. Enter a descriptive name.
4. Choose a calculation method.

   Your choice of an arithmetic or analytic function dynamically changes the Calculation template.
5. Modify the calculation template.
6. Click **Create**.

   The new calculated measure appears in the navigation tree in the Calculated Measures folder.

Figure 5–1 displays the Create Calculated Measure dialog box.

Figure 5–1 Creating a Calculated Measure

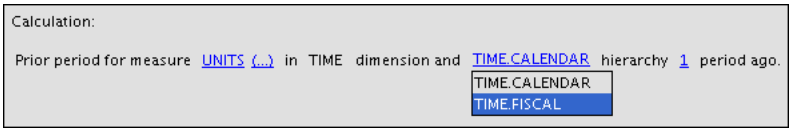


Modifying a Template

The calculation that you selected is presented as template, which is a description of the calculation with variable parts that enable you to customize it.

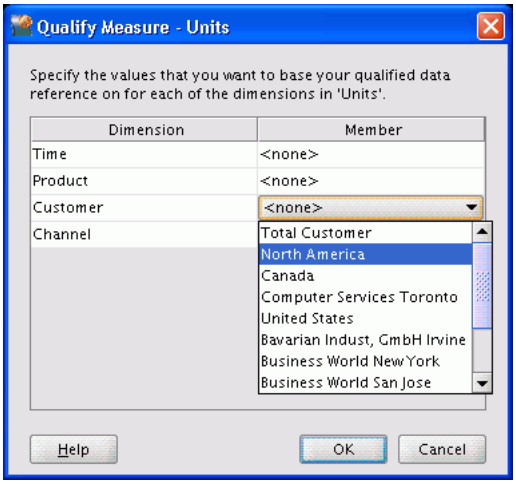
Figure 5–2 shows the template for calculating the prior period. You can view the choice lists by clicking the links.

Figure 5–2 Changing the Variable Parts of a Calculation



You can include all values of a measure in a calculation, or, for some types of calculations, you can filter the measure to include only a selection of values. To limit one or more dimensions to a single dimension member, click the ellipses (. . .) next to the measure. The Qualify Measure dialog box appears, as shown in Figure 5–3.

Figure 5–3 Limiting a Dimension to a Single Member



Choosing a Range of Time Periods

Many calculations are performed over time periods at the same level in the hierarchy. In some types of calculations, you can control the range of time periods that are used in the same calculation. For example, you might want to calculate a running total of months for each fiscal year, not a running total that begins with the first month stored in the cube.

You can use the following methods for identifying the range of time periods that you want calculated together:

- **Level**: Calculates all time periods at the same level, so that all months in the cube are included in one calculation, all quarters are included in another calculation, and so forth.
- **Parent**: Calculates all time periods with the same parent, so that all months in Q1-07 are included in one calculation, all months in Q2-07 are included in another calculation, and so forth.

- **Ancestor at level**: Calculates all time periods with the same ancestor at a specified level. For example, if the specified level is Year in a Year-Quarter-Month hierarchy, then Q1-06 to Q4-06 are included in one calculation, Q1-07 to Q4-07 are included in another calculation, Jan-06 to Dec-06 are included in a third calculation, and so forth. Any levels higher in the hierarchy are not calculated.
- **Gregorian periods**: The Gregorian periods -- Year, Quarter, Month, and Week -- impose the Gregorian calendar onto the selected hierarchy. This can be useful for analyzing data that uses non-standard calendar hierarchies. For example, if you use Gregorian Year on a fiscal hierarchy that begins July 1 and ends June 30, then the last half of one fiscal year and the first half of the next fiscal year are calculated together. Time periods higher in the hierarchy than the specified Gregorian period are not calculated.

Using Calculation Templates

Analytic Workspace Manager provides templates for all of the calculations typically in demand for business intelligence applications. The following topics describe the types of calculations available as calculation templates in Analytic Workspace Manager.

Arithmetic Calculations

Basic mathematical operations enable you to perform cell-by-cell calculations on two measures, as described in "Arithmetic Operators" on page 5-2.

Arithmetic Example

This template defines a calculated measure for the Global Price Cube using Percent Difference:

```
Percent difference between measure UNIT_PRICE and measure UNIT_COST.
```

A query against this calculated measure returns results like these. The `PCT_CHG` column shows the percent change between `PRICE` and `COST`, which is calculated as `PRICE-COST/COST`.

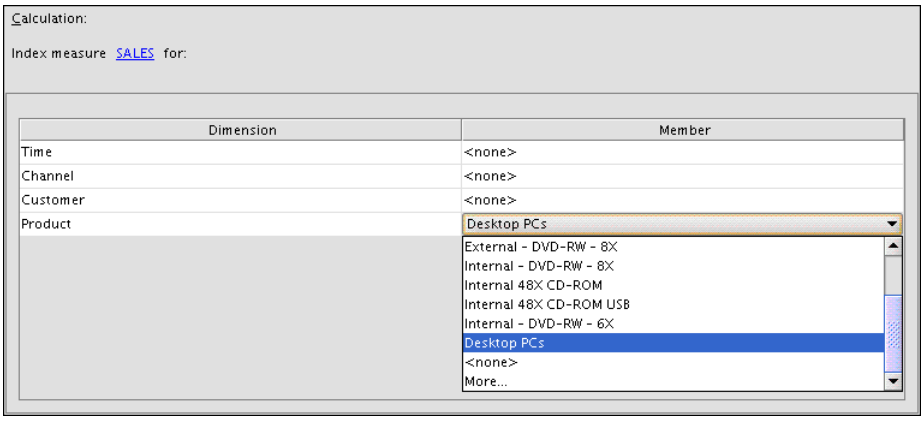
```
PRODUCT                   PRICE       COST   PCT_DIFF
-------------------- ---------- ---------- ----------
Envoy Ambassador            2892       2664        .09
Envoy Executive             2803       2644        .06
Envoy Standard              1662       1737       -.04
Sentinel Financial          1755       1658        .06
Sentinel Multimedia         1770       1813       -.02
Sentinel Standard           1552       1410         .1
```

Index

An index is a mathematical operation calculated on a single measure. An index calculates the percentage difference between the values of a measure and a selected value that serves as a base number.

An index does not use a calculation template. Instead, it provides a list of dimension members for each dimension of the cube, from which you can choose one to use as an index, as shown in Figure 5–4.

Figure 5–4 Calculating a Product Index



Index Example

This example creates an index on the Product dimension using `Desktop PCs` as the index.

```
PRODUCT                  SALES PROD_INDEX
-------------------- ---------- ----------
Desktop PCs            76682955        100
Portable PCs           18072328         24
CD/DVD                 17302122         23
Modems/Fax              5565552          7
Memory                  5347292          7
Monitors                3926632          5
```

Prior and Future Periods

Oracle OLAP provides several calculations for prior or future time periods:

- **Prior Period**: Returns the value of a measure at an earlier time period.
- **Difference From Prior Period**: Calculates the difference between values for the current time period and an earlier period.
- **Percent Difference From Prior Period**: Calculates the percent difference between the values for the current time period and an earlier period.
- **Future Period**: Returns the value of a measure at a later time period.
- **Difference From Future Period**: Calculates the difference between the values for the current time period and a later period.
- **Percent Difference From Future Period**: Calculates the percent difference between the values for the current time period and a later period.

When creating a calculation for prior or future time periods, you choose the measure, the time dimension, the hierarchy, and the number of periods from the current period.

Prior Period Example

This template defines a calculated measure using Prior Period:

```
Prior period for measure SALES in TIME dimension and TIME.CALENDAR hierarchy 1
period ago.
```

These are the results of a query against the calculated measure. The PRIOR_PERIOD column shows the value of Sales for the preceding period at the same level in the Calendar hierarchy.

```
TIME     TIME_LEVEL               SALES PRIOR_PERIOD
-------- -------------------- ---------- ------------
2005     CALENDAR_YEAR         136986572    144290686
2006     CALENDAR_YEAR         140138317    136986572
Q1.05    CALENDAR_QUARTER       31381338     41988687
Q2.05    CALENDAR_QUARTER       37642741     31381338
Q3.05    CALENDAR_QUARTER       32617249     37642741
Q4.05    CALENDAR_QUARTER       35345244     32617249
Q1.06    CALENDAR_QUARTER       36154815     35345244
Q2.06    CALENDAR_QUARTER       36815657     36154815
Q3.06    CALENDAR_QUARTER       32318935     36815657
Q4.06    CALENDAR_QUARTER       34848911     32318935
```

Period to Date

Period-to-date functions perform a calculation over time periods with the same parent up to the current period. These functions calculate period-to-date:

- **Period to Date**: Calculates the values up to the current time period.
- **Period to Date Period Ago**: Calculates the data values up to a prior time period.
- **Difference From Period to Date Period Ago**: Calculates the difference in data values up to the current time period compared to the same calculation up to a prior period.
- **Percent Difference From Period To Date Period Ago**: Calculates the percent difference in data values up to the current time period compared to the same calculation up to a prior period.

When creating a period-to-date calculation, you can choose from these aggregation methods:

- Sum
- Average
- Maximum
- Minimum

You also choose the measure, the time dimension, and the hierarchy.

Period to Date Example

This template defines a calculated measure using Period to Date.

```
Gregorian Year to date for SALES in the TIME dimension and TIME.CALENDAR
hierarchy. Aggregate using MINIMUM from the beginning of the period.
```

These are the results of a query against the calculated measure. The MIN_TO_DATE column displays the current minimum SALES value within the current level and year.

```
TIME     TIME_LEVEL               SALES MIN_TO_DATE
-------- -------------------- ---------- -----------
Q1.06    CALENDAR_QUARTER       36154815    36154815
Q2.06    CALENDAR_QUARTER       36815657    36154815
Q3.06    CALENDAR_QUARTER       32318935    32318935
Q4.06    CALENDAR_QUARTER       34848911    32318935
JAN-06   MONTH                  13119235    13119235
```

```
FEB-06   MONTH                11441738    11441738
MAR-06   MONTH                11593842    11441738
APR-06   MONTH                11356940    11356940
MAY-06   MONTH                13820218    11356940
JUN-06   MONTH                11638499    11356940
JUL-06   MONTH                 9417316     9417316
AUG-06   MONTH                11596052     9417316
SEP-06   MONTH                11305567     9417316
OCT-06   MONTH                11780401     9417316
NOV-06   MONTH                10653184     9417316
DEC-06   MONTH                12415325     9417316
```

Share

Share calculates the ratio of a measure's value for the current dimension member to the value for a related member of the same dimension. You can choose whether the related member is:

- **Top of hierarchy**: Calculates the ratio of each member to the total.
- **Member's parent**: Calculates the ratio of each member to its parent.
- **Member's ancestor at level**: Calculates the ratio of each member to its ancestor, that is, a member at a specified level higher in the hierarchy.

When creating a share calculation, you can choose the measure, dimension, and hierarchy. You also have the option of multiplying the results by 100 to get percentages instead of fractions.

Share Example

This template defines a calculated measure using SHARE:

```
Share of measure SALES in PRODUCT.PRIMARY hierarchy of the PRODUCT dimension as a
ratio of top of hierarchy.
```

These are the results of a query against the calculated measure. The TOTAL_SHARE column displays the percent share of the total for the selected products.

```
PRODUCT              PROD_LEVEL           SALES TOTAL_SHARE
-------------------- --------------- ---------- -----------
Total Product        TOTAL           144290686         100
Hardware             CLASS           130145388          90
Desktop PCs          FAMILY           78770152          55
Portable PCs         FAMILY           19066575          13
CD/DVD               FAMILY           16559860          11
Software/Other       CLASS            14145298          10
Accessories          FAMILY            6475353           4
Operating Systems    FAMILY            5738775           4
Memory               FAMILY            5430466           4
Modems/Fax           FAMILY            5844185           4
Monitors             FAMILY            4474150           3
Documentation        FAMILY            1931170           1
```

Rank

Rank orders the values of a dimension based on the values of the selected measure. When defining a rank calculation, you choose the dimension, a hierarchy, and the measure.

You can choose a method for handling identical values:

- **Rank**: Assigns the same minimum rank to identical values. For example, it may return `1, 2, 3, 3, 5` for a series of five dimension members.
- **Dense Rank**: Assigns the same rank to identical values, so there may be fewer ranks than there are members. For example, it may return `1, 2, 3, 3, 4` for a series of five dimension members.
- **Average Rank**: Assigns the same average rank to identical values. For example, it may return `1, 2, 3.5, 3.5, 5` for a series of five dimension members.

You can also choose the group in which the dimension members are ranked:

- **Member's level**: Ranks members at the same level.
- **Member's parent**: Ranks members with the same parent.
- **Member's ancestor at level**: Ranks members with the same ancestor at a specified level higher in the hierarchy.

Rank Example

This template defines a calculated measure using Rank:

```
Rank members of the PRODUCT dimension and PRODUCT.PRIMARY hierarchy based on
measure SALES. Calculate rank using RANK method with member's parent in order
lowest to highest. Rank NA (null) values nulls last.
```

These are the results of a query against the calculated measure in which the products are ordered by `RANK`:

```
PRODUCT                    SALES       RANK
-------------------- ---------- ----------
Monitors                4474150          1
Memory                  5430466          2
Modems/Fax              5844185          3
CD/DVD                 16559860          4
Portable PCs           19066575          5
Desktop PCs            78770152          6
```

Parallel Period

Parallel periods are at the same level as the current time period, but have different parents in an earlier period. For example, you may want to compare current sales with sales for the prior year.

Oracle OLAP provides several functions for parallel periods:

- **Parallel Period**: Calculates the value of the parallel period.
- **Difference From Parallel Period**: Calculates the difference in values between the current period and the parallel period.
- **Percent Difference From Parallel Period**: Calculates the percent difference in values between the current period and the parallel period.

To identify the parallel period, you specify a level and the number of periods before the current period. You can also decide what happens when two periods do not exactly match, such as comparing daily sales for February (28 days) with January (31 days).

You also choose the measure, the time dimension, and the hierarchy.

Parallel Period Example

This template defines a calculated measure using Parallel Period.

```
Parallel period for SALES in the TIME dimension and TIME.CALENDAR hierarchy 1
TIME.CALENDAR.QUARTER ago based on position from beginning to ending of period.
```

These are the results of a query against the calculated measure, which lists the months for two calendar quarters. The parallel month has the same position within the previous quarter. The prior period for JUL-06 is APR-06, for AUG-06 is MAY-06, and for SEP-06 is JUN-06.

```
TIME     PARENT          SALES   LAST_QTR
-------- ---------- ---------- ----------
APR-06   CY2006.Q2    11356940   13119235
MAY-06   CY2006.Q2    13820218   11441738
JUN-06   CY2006.Q2    11638499   11593842
JUL-06   CY2006.Q3     9417316   11356940
AUG-06   CY2006.Q3    11596052   13820218
SEP-06   CY2006.Q3    11305567   11638499
```

Moving Calculations

Moving calculations are performed over the time periods surrounding the current period. Oracle OLAP provides several aggregation methods for moving calculations:

- **Moving Average**: Calculates the average value for a measure over a fixed number of time periods.
- **Moving Maximum**: Calculates the maximum value for a measure over a fixed number of time periods.
- **Moving Minimum**: Calculates the minimum value for a measure over a fixed number of time periods.
- **Moving Total**: Returns the total value for a measure over a fixed number of time periods.

You can choose the measure, the time dimension, and the hierarchy. You can also select the range, as described in "Choosing a Range of Time Periods" on page 5-4, and the number of time periods before and after the current period to include in the calculation.

Moving Calculation Example

This template defines a calculated measure using Moving Minimum.

```
Moving minimum of SALES in the TIME dimension and TIME.CALENDAR hierarchy. Include
1 preceding and 1 following members within level.
```

These are the results of a query against the calculated measure, which displays values for the descendants of calendar year 2004. Each value of Minimum Sales is the smallest among the current value and the values immediately before and after it. The calculation is performed over all members of a level in the cube.

```
TIME     TIME_LEVEL               SALES  MIN_SALES
-------- -------------------- ---------- ----------
Q1.04    CALENDAR_QUARTER       32977874   32977874
Q2.04    CALENDAR_QUARTER       35797921   32977874
Q3.04    CALENDAR_QUARTER       33526203   33526203
Q4.04    CALENDAR_QUARTER       41988687   31381338
JAN-04   MONTH                  11477898   10982016
FEB-04   MONTH                  10982016   10517960
```

```
MAR-04   MONTH                 10517960   10517960
APR-04   MONTH                 11032057   10517960
MAY-04   MONTH                 11432616   11032057
JUN-04   MONTH                 13333248   11432616
JUL-04   MONTH                 12070352   11108893
AUG-04   MONTH                 11108893   10346958
SEP-04   MONTH                 10346958   10346958
OCT-04   MONTH                 14358605   10346958
NOV-04   MONTH                 12757560   12757560
DEC-04   MONTH                 14872522   12093518
```

Cumulative Calculations

Cumulative calculations start with the first time period and calculate up to the current member, or start with the last time period and calculate back to the current member. Oracle OLAP provides several aggregation methods for cumulative calculations:

- **Cumulative Average**: Calculates a running average across time periods.
- **Cumulative Maximum**: Calculates the maximum value across time periods.
- **Cumulative Minimum**: Calculates the minimum value across time periods.
- **Cumulative Total**: Calculates a running total across time periods.

You can choose the measure, the time dimension, and the hierarchy. You can also select the range, as described in "Choosing a Range of Time Periods" on page 5-4, and whether you want to start the calculation with the first period and calculate forward, or start with the last period and calculate back.

Cumulative Calculation Example

This template defines a calculated measure using Cumulative Minimum.

```
Cumulative minimum of SALES in the TIME dimension and TIME.CALENDAR hierarchy
within ancestor at level TIME.CALENDAR YEAR. Total from beginning to current
member.
```

These are the results of a query against the calculated measure, which displays values for the descendants of calendar year 2004. The minimum value for quarters begins with Q1-04 and ends with Q4-04, and for months begins with Jan-04 and ends with Dec-04.

```
TIME     TIME_LEVEL                SALES  MIN_SALES
-------- -------------------- ---------- ----------
Q1.04    CALENDAR_QUARTER       32977874   32977874
Q2.04    CALENDAR_QUARTER       35797921   32977874
Q3.04    CALENDAR_QUARTER       33526203   32977874
Q4.04    CALENDAR_QUARTER       41988687   32977874
JAN-04   MONTH                  11477898   11477898
FEB-04   MONTH                  10982016   10982016
MAR-04   MONTH                  10517960   10517960
APR-04   MONTH                  11032057   10517960
MAY-04   MONTH                  11432616   10517960
JUN-04   MONTH                  13333248   10517960
JUL-04   MONTH                  12070352   10517960
AUG-04   MONTH                  11108893   10517960
SEP-04   MONTH                  10346958   10346958
OCT-04   MONTH                  14358605   10346958
NOV-04   MONTH                  12757560   10346958
DEC-04   MONTH                  14872522   10346958
```

Nested Calculations

You can extend the variety of functions available through the templates by using a calculated measure as the basis for another calculated measure.

For example, Analytic Workspace Manager has templates for Moving Average and for Difference From Prior Period. You can create a calculated measure that calculates a moving average, then calculate the difference between the current and the previous moving averages.

Nested Calculations Example

This template creates a moving average for Units named UNITS_MOVING_AVG:

```
Moving average of UNITS in the TIME dimension and TIME.CALENDAR hierarchy. Include
1 preceding and 1 following members within level.
```

The next template creates a Difference From Prior Period calculation from UNITS_MOVING_AVG.

```
Difference from prior period for UNITS MOVING AVG in TIME dimension and
TIME.CALENDAR hierarchy 1 period ago.
```

These are the results of a query against the Units measure and the two calculated measures. The MOVING_AVG column shows the moving average, and the DIFF column shows the difference between the current moving average and the prior period's.

TIME	TIME_LEVEL	UNITS	MOVING_AVG	DIFF
JAN-06	MONTH	47776	48520	66
FEB-06	MONTH	47695	48940	419
MAR-06	MONTH	51348	48683	-257
APR-06	MONTH	47005	50387	1705
MAY-06	MONTH	52809	48411	-1976
JUN-06	MONTH	45419	48872	461
JUL-06	MONTH	48388	47546	-1326
AUG-06	MONTH	48830	47857	312
SEP-06	MONTH	46354	47532	-326
OCT-06	MONTH	47411	46869	-663
NOV-06	MONTH	46842	49768	2899
DEC-06	MONTH	55052	50947	1179
2006	CALENDAR_YEAR	584929	575324	-4032
Q1.06	CALENDAR_QUARTER	146819	145705	2093
Q2.06	CALENDAR_QUARTER	145233	145208	-497
Q3.06	CALENDAR_QUARTER	143572	146037	829
Q4.06	CALENDAR_QUARTER	149305	146439	402

Creating User-Defined Expressions

Among the calculation types is a user-defined expression. It enables you to create calculations using the OLAP expression syntax, which includes the analytic functions, arithmetic operators, and single-row functions described in this chapter. The OLAP syntax is an extension of the SQL syntax. If you have used SQL analytic functions or single-row functions, this syntax will be familiar to you.

The easiest way to formulate an expression is to let Analytic Workspace Manager do the work for you. You can use the templates to create a similar calculation, and cut-and-paste the syntax into an expression.

To create an expression:

1. Open the Create Calculated Measure dialog box.
2. Select the calculation type that most closely matches the one you want to define.
3. Modify the template as desired.
4. Cut-and-paste the calculation from the Calculation box into a text editor.
5. Repeat these steps if your calculation uses two or more functions.
6. Modify the calculation as desired in the text editor. You can combine numeric operators, analytic functions, and single-row functions in a single calculation.
7. From the Calculation Types list, select **Expression**.
8. Cut-and-paste the calculation from the text editor into the Calculation box.
9. Click **Create**.

> **See Also:** Analytic Workspace Manager Help for information about the OLAP expression syntax

Expression Example Using an Arithmetic Operator

This template for Multiplication generates a calculation using Units Sold and Unit Cost.

```
Multiply measure UNITS by measure UNIT_COST.
```

The template generates this calculation using the multiplication operator (*). It is displayed in the Calculation box. Note that UNITS is in the Units Cube and UNIT_COST is in the Price Cube.

```
UNITS_CUBE.UNITS * PRICE_CUBE.UNIT_COST
```

The syntax of this calculation is so simple that you only need the template to obtain the qualified name of the measure.

Following is a free-form calculation that calculates a 2% increase in units sold:

```
UNITS_CUBE.UNITS * 1.02
```

These are the results of a query against this calculated measure:

```
PRODUCT                   UNITS     TARGET
------------------- ---------- ----------
Envoy Ambassador           2116       2158
Envoy Executive            2481       2531
Envoy Standard             3300       3366
Sentinel Financial        30513      31123
Sentinel Multimedia        7948       8107
Sentinel Standard          7302       7448
```

Free-Form Calculation Example Using an Analytic Function

This template for Cumulative Average generates a calculation for the average number of units sold:

```
Cumulative average of UNITS in the TIME dimension and TIME.CALENDAR hierarchy
within level. Total from beginning to following member.
```

The template generates this calculation using the AVG function.

```
AVG(UNITS_CUBE.UNITS) OVER HIERARCHY (TIME.CALENDAR BETWEEN UNBOUNDED PRECEDING
AND UNBOUNDED FOLLOWING WITHIN LEVEL)
```

Following is a free-form calculation that computes the percent difference between current units sold and the cumulative average. It uses the AVG function and the subtraction (-), division (/) and multiplication (*) operators.

```
((UNITS_CUBE.UNITS - AVG(UNITS_CUBE.UNITS) OVER HIERARCHY (TIME.CALENDAR BETWEEN
UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING WITHIN LEVEL)) / AVG(UNITS_CUBE.UNITS)
OVER HIERARCHY (TIME.CALENDAR BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
WITHIN LEVEL)) * 100
```

These are the results of a query against this calculated measure.

```
TIME          UNITS    CUM_AVG   PCT_DIFF
-------- ---------- ---------- ----------
Q1.06        146819     107965         36
Q2.06        145233     109062         33
Q3.06        143572     110048         30
Q4.06        149305     111138         34
```

Note that you could create this calculation using templates:

1. Calculate the cumulative average of UNITS with the Cumulative Average template.
2. Calculate the percent difference between current UNITS and the cumulative average with the Percent Difference template.

Analytic Functions

Table 5–1 describes the analytic functions that you can use to create free-form calculations. For the syntax of these functions, refer to Analytic Workspace Manager Help.

Table 5–1 OLAP Analytic Functions

Function	Description
AVERAGE_RANK	Orders the members of a dimension based on the values of an expression. The function returns the sequence numbers of the dimension members, and assigns the same average rank to identical values.
AVG	Returns the average of a selection of values calculated over time.
COUNT	Tallies the number of data values identified by a selection of dimension members.
DENSE_RANK	Orders dimension members based on the values of an expression. The function returns the sequence numbers of the dimension members, and assigns the same minimum rank to identical values.
HIER_ANCESTOR	Returns an ancestor at a particular level of a hierarchy for either all members in the hierarchy or a particular member.
HIER_CHILD_COUNT	Returns the number of children of either all dimension members in a hierarchy or a particular member.
HIER_DEPTH	Returns a number representing the level depth of either all members of a hierarchy or a particular member, where 0 is the top level.
HIER_LEVEL	Returns the level of either all members of a hierarchy or a particular member.
HIER_PARENT	Returns the parent of either all dimension members in a hierarchy or a particular member.

Table 5–1 (Cont.) OLAP Analytic Functions

Function	Description
HIER_TOP	Returns the topmost ancestor of either all members of a hierarchy or a particular member.
LAG	Returns the value of an expression at a specified number of time periods before the current period.
LAG_VARIANCE	Returns the difference between values for the current time period and a prior period.
LAG_VARIANCE_PERCENT	Returns the percent different between values for the current time period and a prior period.
LEAD	Returns the value of an expression at a specified number of time periods after the current period.
LEAD_VARIANCE	Returns the difference between values for the current time period and a future period.
LEAD_VARIANCE_PERCENT	Returns the percent different between values for the current time period and a future period.
MAX	Returns the largest of a selection of data values calculated over a particular dimension.
MIN	Returns the smallest of a selection of data values calculated over a particular dimension.
OLAP_DML_EXPRESSION	Executes an expression in the OLAP DML language.
RANK	Orders the members of a dimension based on the values of an expression. The function returns the sequence numbers of the dimension members, and assigns the same rank to identical values.
ROW_NUMBER	Orders the members of a dimension based on the values of an expression. The function returns the sequence numbers of the dimension members, and assigns a unique and arbitrary rank to identical values.
SHARE	Calculates the ratio of an expression's value for the current dimension member to the value for a related member of the same dimension.
SUM	Returns the total of a selection of values calculated over a particular dimension.

6

Developing Reports and Dashboards

You can use any SQL development tool or application to create reports and dashboards populated with data from OLAP cubes. This chapter shows the basic steps for working with the tools provided with Oracle Database: Oracle Business Intelligence Publisher (BI Publisher) and Oracle Application Express. You can try these tools, or you can apply the methods shown here to your favorite SQL tool.

This chapter contains the following topics:

See Also: Chapter 4, "Querying Dimensional Objects"

Developing OLAP Applications

You can use any SQL query against a cube as the content for a report or dashboard. Both BI Publisher and Application Express contain a Query Builder, which you can use to develop queries against both relational and dimensional objects. You can also cut-and-paste queries from a SQL script or another source, which is the method used in this chapter.

If your goal is to create static reports and dashboards, then you do not need to read any further. You can start developing OLAP applications immediately using your favorite tool. This chapter explains how to create applications with dynamic content. It focuses on ways to leverage the unique capabilities of cubes and dimensions to create drillable reports and graphs using a single query. You will learn how to create two types of drillable interfaces:

- Choice Lists: You can create a drop-down list for each dimension to drill on the dimensions in a report or dashboard.
- Linked Dimension Columns: In Application Express, you can add links to the dimension columns of a crosstab to drill down to the bottom of a hierarchy, and use a Reset button to return to the top level.

These user interfaces set the values of bind variables in the `WHERE` clause of the source query. When a user changes the current selection in a choice list or clicks a link in a crosstab, that action dynamically changes the value of the variable. When the variable changes, so does the condition of the query and the contents of the report or dashboard.

When the variable sets the value of the `PARENT` column of the hierarchy views, users can drill on a parent to view its children.

Example 6–1 shows a basic SQL query against UNITS_CUBE_VIEW in the Global sample schema. The query selects the SALES measure and three calculated measures that use SALES as the basis for the calculations:

- SALES_PP: Sales from the prior period.
- SALES_CHG_PP: Difference in sales between the current period and the prior period.
- SALES_PCTCHG_PP: Percent difference in sales between the current period and the prior period.

This query is used in the sample applications developed in this chapter. The PARENT columns for the Product, Customer, and Time dimensions will support drilling in these applications. The CHANNEL dimension will remain anchored at the TOTAL level.

Example 6–1 SQL Query Against the Sales Cube

```
SELECT p.long_description "Product",
     cu.long_description "Customer",
     t.long_description "Time",
     ROUND(f.sales) "Sales",
     ROUND(f.sales_pp) "Prior Period",
     ROUND(f.sales_chg_pp) "Change",
     ROUND(f.sales_pctchg_pp * 100) "Percent Change"
/* From dimension views and cube view */
FROM product_primary_view p,
     customer_shipments_view cu,
     time_calendar_view t,
     channel_primary_view ch,
     units_cube_view f
/* Use parent columns to implement drilling */
WHERE p.parent = 'TOTAL'
    AND cu.parent = 'TOTAL'
    AND t.parent = 'CY2006'
    AND ch.level_name = 'TOTAL'
/* Join dimension views to cube view */
    AND p.dim_key = f.product
    AND cu.dim_key = f.customer
    AND t.dim_key = f.time
    AND ch.dim_key = f.channel
ORDER BY product, customer, t.end_date;

Product         Customer        Time            Sales Prior Period     Change Percent Change
--------------- --------------- ---------- ---------- ------------ ---------- --------------
Hardware        North America   Q1.06        16002175     14493426    1508749             10
Hardware        North America   Q2.06        16032643     16002175      30469              0
Hardware        North America   Q3.06        15698208     16032643    -334436             -2
Hardware        North America   Q4.06        15958791     15698208     260583              2
Hardware        Asia Pacific    Q1.06        13416447     14273900    -857453             -6
Hardware        Asia Pacific    Q2.06        14306431     13416447     889984              7
                 .
                 .
                 .
Software/Other  Asia Pacific    Q4.06          652300       647019       5281              1
Software/Other  Europe          Q1.06          737523       634293     103230             16
Software/Other  Europe          Q2.06          678391       737523     -59132             -8
Software/Other  Europe          Q3.06          499008       678391    -179383            -26
Software/Other  Europe          Q4.06          710796       499008     211788             42

24 rows selected.
```

Developing a Report Using BI Publisher

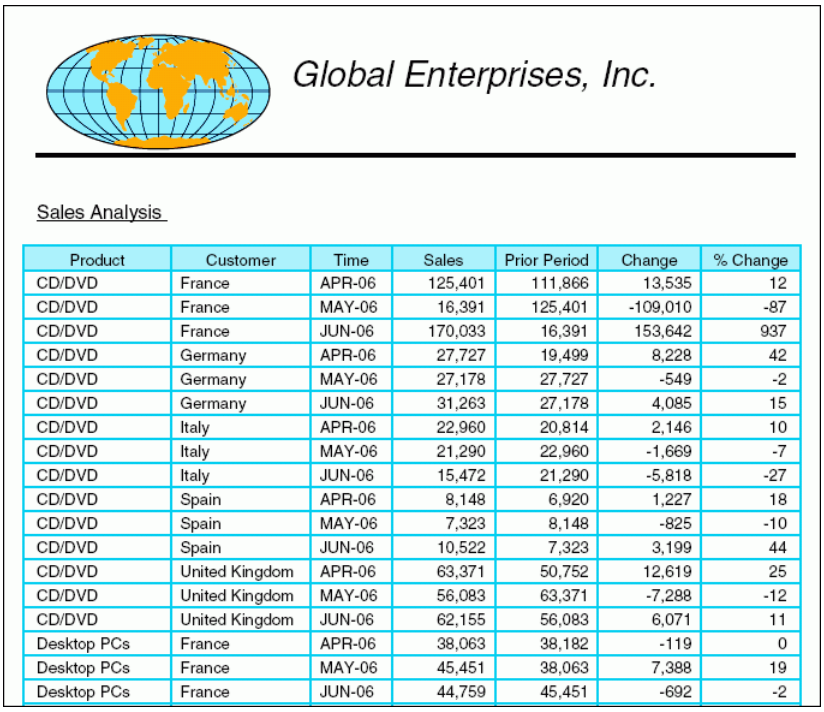
BI Publisher is an efficient, scalable reporting solution for generating and delivering information through a variety of distribution methods. It reduces the high costs associated with the development and maintenance of business documents, while increasing the efficiency of reports management. BI Publisher generates reports in a variety of formats, including HTML, PDF, and Excel.

If you have not used BI Publisher, you can download the software, tutorials, and full documentation from the Oracle Technology Network at

`http://www.oracle.com/technology/products/xml-publisher/index.html`

Figure 6–1 shows a report in PDF format based on the query shown in Example 6–1. When generating a report for distribution, you can select any combination of Products, Customers, and Time Periods from the choice lists. The selection for this report is Hardware products, customers in Europe, and months in Q2-06. This chapter explains how you can create a report like this one using drillable dimensions.

Figure 6–1 Sales Report in BI Publisher



Global Enterprises, Inc.

Sales Analysis

Product	Customer	Time	Sales	Prior Period	Change	% Change
CD/DVD	France	APR-06	125,401	111,866	13,535	12
CD/DVD	France	MAY-06	16,391	125,401	-109,010	-87
CD/DVD	France	JUN-06	170,033	16,391	153,642	937
CD/DVD	Germany	APR-06	27,727	19,499	8,228	42
CD/DVD	Germany	MAY-06	27,178	27,727	-549	-2
CD/DVD	Germany	JUN-06	31,263	27,178	4,085	15
CD/DVD	Italy	APR-06	22,960	20,814	2,146	10
CD/DVD	Italy	MAY-06	21,290	22,960	-1,669	-7
CD/DVD	Italy	JUN-06	15,472	21,290	-5,818	-27
CD/DVD	Spain	APR-06	8,148	6,920	1,227	18
CD/DVD	Spain	MAY-06	7,323	8,148	-825	-10
CD/DVD	Spain	JUN-06	10,522	7,323	3,199	44
CD/DVD	United Kingdom	APR-06	63,371	50,752	12,619	25
CD/DVD	United Kingdom	MAY-06	56,083	63,371	-7,288	-12
CD/DVD	United Kingdom	JUN-06	62,155	56,083	6,071	11
Desktop PCs	France	APR-06	38,063	38,182	-119	0
Desktop PCs	France	MAY-06	45,451	38,063	7,388	19
Desktop PCs	France	JUN-06	44,759	45,451	-692	-2

Creating an OLAP Report in BI Publisher

A report consists of a **report entry**, which you create in BI Publisher, and a **layout template**, which you create using an application such as Microsoft Word or Adobe Acrobat. You can organize your reports in folders.

BI Publisher is a middleware application and can derive data from multiple sources. These procedures assume that you can access one or more cubes from BI Publisher. If you cannot, contact your BI Publisher administrator about defining a new data source.

To create a report entry:

1. Open a browser to the BI Publisher home page and log in.
2. Click **My Folders**.
3. Open an existing folder.

   or

   To create a new folder:

   a. Click **Create a New Folder**.

   b. Enter a name for the folder in the text box, such as `OLAP Reports`.

   c. Click **Create**.
4. Click the new folder to open it.
5. Create a new report:

   a. Click **Create a New Report**.

   b. Enter a report name in the text box.

      This example creates a report named Global Sales.

   c. Click **Create**.

   The new report appears in the folder, as shown in Figure 6–2.

Figure 6–2 Creating a New Report in BI Publisher

To configure the report entry:

1. To define the contents of the report, click **Edit**.

   The Report Editor opens.
2. For General Settings, enter a description and select a default data source.

   If the list does not include a connection to the database and schema containing your cubes, contact your BI Publisher administrator.
3. Select Data Model, then click **New**.

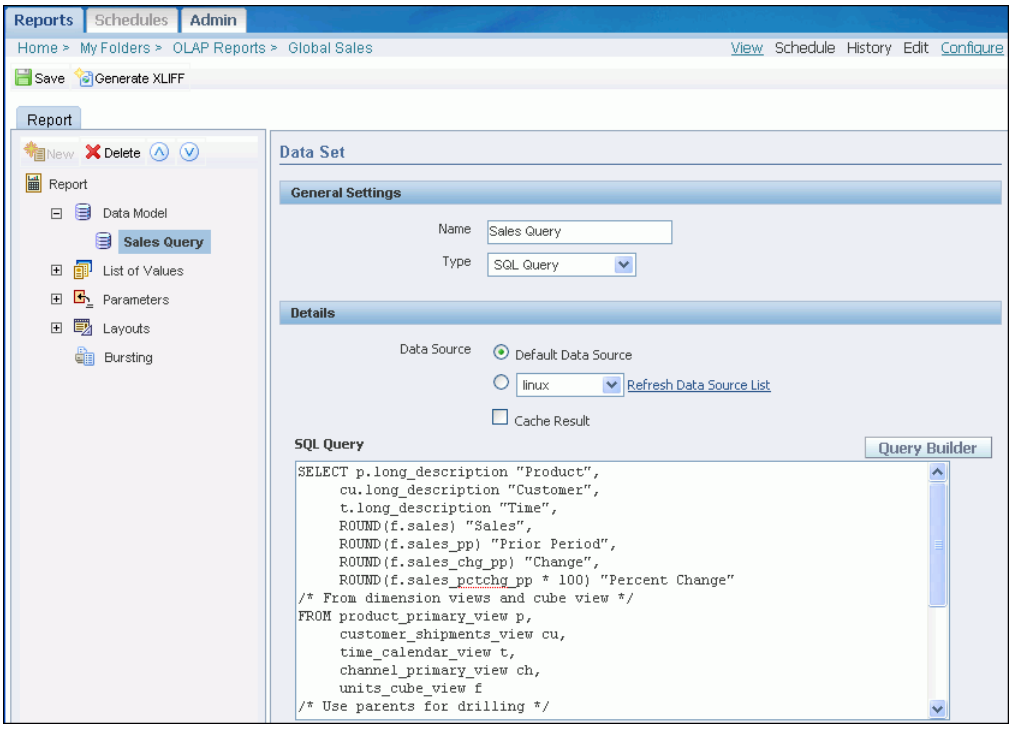
   The Data Set page opens.

4. Enter a name for the data set and enter a SQL query like the one shown in Example 6–1. Do not use a semicolon.
5. Click **Save**.
6. Click **View**.

   BI Publisher checks the report definition for errors. If there are none, then it generates the XML for the report.

Figure 6–3 shows the Report Editor with the Data Set page displayed.

Figure 6–3 Creating a Data Model in the BI Publisher Report Editor



Creating a Template in Microsoft Word

BI Publisher does not contain formatting tools. Instead, it enables you to design a report using familiar desktop applications. This example uses Microsoft Word. A report template can contain:

- Static text and graphics that you enter like any other Word document.
- Dynamic fields such as the date and time or page numbers, which are processed by Word.
- Codes that identify the XML tags for your data, which are processed by BI Publisher. When BI Publisher generates a report, it replaces the codes with the data identified by these tags.

You can format all parts of the report template in Word, selecting the fonts, text and background colors, table design, and so forth.

Example 6–2 shows the XML for a row of data returned by the sample query. The tags match the column names in the select list, except that underscores replace the spaces. The tags are `Product`, `Customer`, `Time`, `Sales`, `Prior_Period`, `Change`, and

`Percent_Change`. XML tags are case-sensitive. You use the HTML tag names as the codes in the Word document.

Example 6–2 XML for a SQL Query

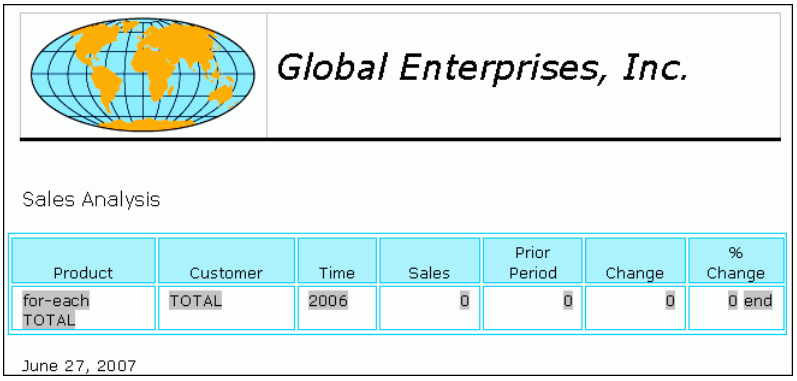
```
<ROW>
<Product>Hardware</Product>
<Customer>North America</Customer>
<Time>Q1.06</Time>
<Sales>16002175</Sales>
<Prior_Period>14493426</Prior_Period>
<Change>1508749</Change>
<Percent_Change>10</Percent_Change>
</ROW>
```

Figure 6–4 shows the Word document that will be used as the template for the sample report. It contains these elements:

- A table used to format the banner, which consists of a graphic, the company name, and a horizontal line. (Static)
- The name of the report. (Static)
- A table for the query results that contains two rows:
  - A heading row. (Static)
  - A body row containing text form fields, which identify the XML tags and the appropriate formatting for the data. BI Publisher will replace these fields with data from the query. Note that the first and last columns contain two fields. The first and last fields identify the range of repeating columns. (Dynamic)
- A date field. Word updates this field with the current date. (Dynamic)

This example uses a blank Word template, but you could use a template with, for example, the banner already defined.

Figure 6–4 Sample Report Template Created in Word for BI Publisher



The following procedure defines the template manually. Alternatively, you can use a Word plugin called Oracle BI Publisher Desktop. On the BI Publisher My Folders page, click **Template Builder** to download the plugin.

To create a BI Publisher template in Word:

1. Open a new document in Word.
2. Compose the page according to your preferences.
3. For the query results, create a table.

   The table shown in Figure 6–4 is very simple. You can use much more elaborate formatting if you wish, including nested columns and tables.
4. From the View menu, choose **Toolbars**, then **Forms**.

   The Forms toolbar opens.
5. Enter a field in the body row of each column:
   a. Position the cursor in the appropriate cell.
   b. On the Forms toolbar, click the **Text Form Field** icon.

      The Text Form Field Options dialog box opens.
   c. Choose an appropriate Type, generally **Regular Text** for dimension labels and **Number** for measures.
   d. Enter a default value and a format.
   e. Click **Add Help Text**.

      The Form Field Help Text dialog box opens.
   f. Type the appropriate XML tag in the Type Your Own box, using the format `<?tag?>`.

      Enter the tag name exactly as it appears in the XML report. For example, enter `<?Product?>` for the XML tag `<Product>`.
   g. Click **OK** to close the Form Field Help dialog box.
   h. Click **OK** to close the Text Form Field Options dialog box.
6. Insert an additional form field at the beginning of the first column:
   a. In the Text Form Field Options dialog box, enter any default value, such as `For-Each`.
   b. In the Form Field Help Text dialog box, enter this text:

      ```
      <?for-each:ROW?>
      ```
7. Insert an additional form field at the end of the last column:
   a. In the Text Form Field Options dialog box, enter any default value, such as `End`.
   b. In the Form Field Help Text dialog box, enter this text:

      ```
      <?end for-each?>
      ```
8. Make any additional formatting changes in Word, such as the appropriate justification of the table headings and data columns.
9. Save the document as an RTF file.

Generating a Formatted Report

After creating a report template in Word, you can upload it to BI Publisher and associate it with your report definition. Then you can generate reports in a variety of formats.

Create a Report Layout:

1. Open the report editor in BI Publisher.
2. Select **Layouts**.

   The Create Layouts page opens.
3. Click **New**.

   The Layout page opens.
4. Enter a name and select **RTF** for the template type.
5. Select **Layouts** again, and select the new layout as the default template for this report.
6. Under Manage Template Files, click **Browse**. Select the RTF file you created.
7. Click **Upload**.

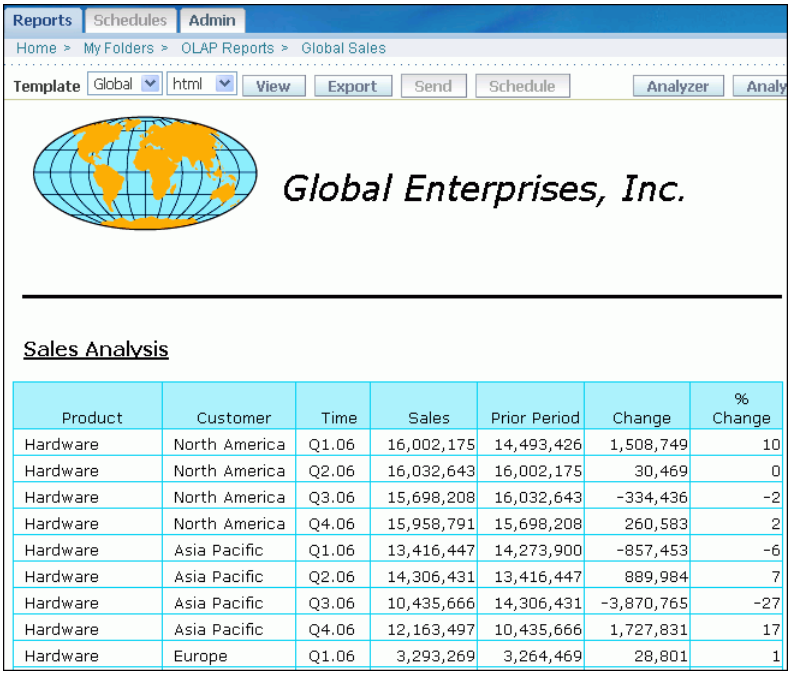
   The uploaded file will be listed under Manage Template Files. Whenever you change the file in Word, upload it again. Otherwise, BI Publisher will continue to use its copy of the previous version.
8. Click **Save**.
9. Click **View**.

   The report is displayed.
10. To change the format, select a new format from the list and click **View**.

    To see the XML, select **Data**.

Figure 6–5 shows the report in HTML format.

Figure 6–5 BI Publisher Report Displayed in HTML Format



Product	Customer	Time	Sales	Prior Period	Change	% Change
Hardware	North America	Q1.06	16,002,175	14,493,426	1,508,749	10
Hardware	North America	Q2.06	16,032,643	16,002,175	30,469	0
Hardware	North America	Q3.06	15,698,208	16,032,643	-334,436	-2
Hardware	North America	Q4.06	15,958,791	15,698,208	260,583	2
Hardware	Asia Pacific	Q1.06	13,416,447	14,273,900	-857,453	-6
Hardware	Asia Pacific	Q2.06	14,306,431	13,416,447	889,984	7
Hardware	Asia Pacific	Q3.06	10,435,666	14,306,431	-3,870,765	-27
Hardware	Asia Pacific	Q4.06	12,163,497	10,435,666	1,727,831	17
Hardware	Europe	Q1.06	3,293,269	3,264,469	28,801	1

Adding Dimension Choice Lists

You can add choice lists for the dimensions to a report. When generating a report, you can change the selection of data without changing the query. To add choice lists, take these steps:

- Create one or more Lists of Values (LOV) to be displayed in the menu.
- Create menus for displaying the LOVs.
- Edit the query to use the bind variables created for the menus.

Creating a List of Values

For an LOV, use a SQL query that selects the dimension keys that you want to display. Include the `LONG_DESCRIPTION` and `DIM_KEY` columns from the hierarchy view. This example creates a list for the Product Primary hierarchy:

```
SELECT long_description, dim_key
     FROM product_primary_view
     WHERE parent = 'TOTAL'
     OR dim_key = 'TOTAL'
     ORDER BY level_name, long_description

LONG_DESCRIPTION     DIM_KEY
-------------------- ------------
Hardware             HRD
Software/Other       SFT
Total Product        TOTAL
```

To create a list of values:

1. Open the Report Editor in BI Publisher.
2. Select **List of Values**, then click **New**.

   The List of Values page opens.
3. Define the list:
   a. Enter a name for the list, such as `Product_LOV`.
   b. For the type, select **SQL Query**.
   c. Enter a query against the dimension hierarchy view, as shown previously.
4. Click **Save**.

Repeat these steps for the other dimensions. This example uses lists for Product, Customer, and Time.

Creating a Menu

In BI Publisher, a menu is a type of parameter. Creating a parameter automatically creates a bind variable that you can use in the query for the report.

To create a menu:

1. Select Parameters, then click **New**.

   The Parameter page opens.
2. Define the parameter:
   a. For the Identifier, enter a name such as `product`.

      This is the case-sensitive name of the bind variable that you will use in the query.
   b. Select an appropriate data type, typically String.
   c. For the Default Value, enter the dimension key used in the `WHERE` clause of the LOV query.

      The menu will initially display this key.
   d. For the Parameter Type, select **Menu**.
   e. Select the appropriate List of Values.
   f. Clear all options.
3. Click **Save**.

Repeat these steps for the other dimensions. This example creates menus for Product, Customer, and Time.

Editing the Query

To activate the menus, you change the `WHERE` clause in the query for the report to use the bind variables. The value of a bind variable is the current menu choice.

This is the format for the conditions of the `WHERE` clause:

```
parent_column = :bind_variable
```

In this example, the WHERE clause uses the bind variables for Time, Product, and Customer:

```
WHERE  p.parent  = :product
   AND cu.parent = :customer
   AND t.parent  = :time
   AND ch.level_name = 'TOTAL'
```

To edit the query:

1. Under Data Model, select the data set you defined for this report.

   The Data Set page opens.

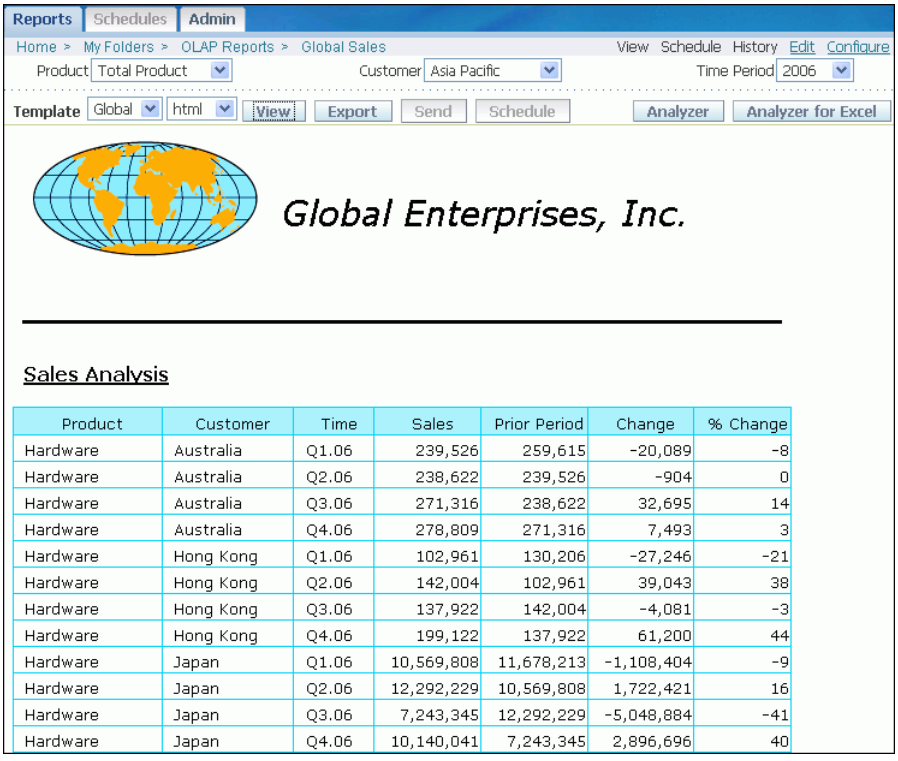
2. In the SQL Query box, edit the WHERE clause to use the bind variables created by the parameter definitions.

3. Click **Save**.

Figure 6–6 shows a report in HTML format displayed in BI Publisher. The choice lists for Product, Customer, and Time appear across the top. The crosstab lists the months in Q3.06, the Hardware products, and the countries in Europe. To see a different selection of data, you choose a Time Period, Product, and Customer from the menus, then click **View**. This report was generated by the same report entry, using the same query, as the one shown in Figure 6–1.

You can continue working on this report, adding charts and other tables.

Figure 6–6 Sales Report With Choice Lists in BI Publisher



Product	Customer	Time	Sales	Prior Period	Change	% Change
Hardware	Australia	Q1.06	239,526	259,615	-20,089	-8
Hardware	Australia	Q2.06	238,622	239,526	-904	0
Hardware	Australia	Q3.06	271,316	238,622	32,695	14
Hardware	Australia	Q4.06	278,809	271,316	7,493	3
Hardware	Hong Kong	Q1.06	102,961	130,206	-27,246	-21
Hardware	Hong Kong	Q2.06	142,004	102,961	39,043	38
Hardware	Hong Kong	Q3.06	137,922	142,004	-4,081	-3
Hardware	Hong Kong	Q4.06	199,122	137,922	61,200	44
Hardware	Japan	Q1.06	10,569,808	11,678,213	-1,108,404	-9
Hardware	Japan	Q2.06	12,292,229	10,569,808	1,722,421	16
Hardware	Japan	Q3.06	7,243,345	12,292,229	-5,048,884	-41
Hardware	Japan	Q4.06	10,140,041	7,243,345	2,896,696	40

Developing a Dashboard Using Application Express

Oracle Application Express is a rapid Web application development tool for Oracle Database. Application Express offers built-in features such as user interface themes, navigational controls, form handlers, and flexible reports, which simplify the development process.

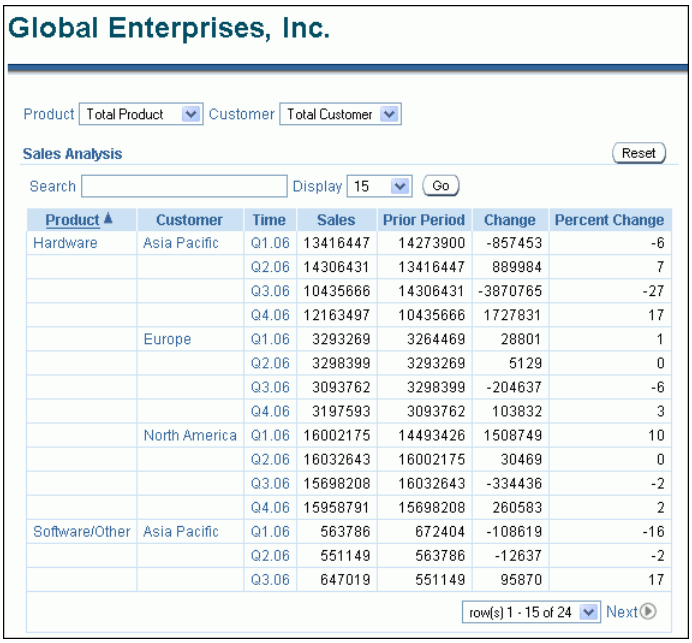
Chapter 1 shows a sophisticated dashboard that extracts analytic data from cubes and presents it in a variety of graphs and reports. You can easily create dashboards from your cubes that display the rich analytical content generated by Oracle OLAP.

If you have not used Application Express, you can download the software, tutorials, and full documentation from the Oracle Technology Network at

`http://www.oracle.com/technology/products/database/application_express`

Figure 6–7 shows a crosstab with display lists for Product and Customer, and links in all three dimension columns. Choosing a new Product or Customer changes the related column to show the children for the selected key. Clicking a dimension key in any column displays its children. The Reset button refreshes the page with the initial selection of data.

Figure 6–7 Drillable Dimensions in Application Express



Creating an OLAP Application in Application Express

In Application Express, the Administrator creates a **workspace** in which you can develop your Web applications. An **application** consists of one or more HTML pages, a **page** consists of regions that identify specific locations on the page, and a **region** contains a report (crosstab), a chart, or some other item.

Application Express runs in Oracle Database. If your dimensional objects are stored in a different database, then you need to use a database link in your queries. The following procedure assumes that you have a workspace and access to at least one cube. It creates an application with one page containing a crosstab.

To create a Web page from a SQL query:

1. Open a browser to the Application Express home page and log in.
2. Click the **Application Builder** icon.

   The Application Builder opens.
3. Click **Create**.

   The Create Application wizard opens.
4. Select **Create Application**, then **Next**.
5. On the Name page, enter a title for the application such as `Global Dashboard` and select **From Scratch**.
6. On the Pages page, select the **Report** page type, then define the page:
   a. For Page Source, select **SQL Query**.
   b. For Title, enter a name such as `Sales Analysis`.

      This title is displayed on the page.
   c. For Query, enter a SQL `SELECT` statement for your cube, like the one shown in Example 6–1. Do not include an `ORDER BY` clause or a semicolon.
   d. Click **Add Page**.

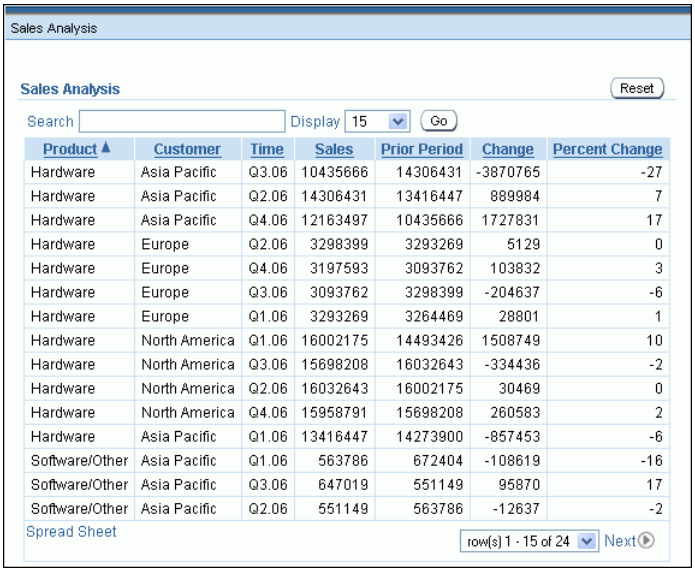
      The page definition appears in the Create Application Box.
7. Click **Next**, then complete the Create Application wizard according to your own preferences.

   This example was created with no tabs, no shared components, no authentication, and Theme 15 (Light Blue).
8. On the Confirm page, click **Create**.
9. On the Application Builder home page, click the **Run Application** icon.

> **Tip:** To continue working on this page, click the **Edit Page 1** link at the bottom of the display.

Figure 6–8 shows the results of the query displayed in Application Express. Several items are automatically added to the page: breadcrumbs, Search box, Display list, Go button, Reset button, and Spread Sheet link. This application only needs the Reset button, so you can delete the other items if you wish.

Figure 6–8 Basic Sales Report in Application Express



Product	Customer	Time	Sales	Prior Period	Change	Percent Change
Hardware	Asia Pacific	Q3.06	10435666	14306431	-3870765	-27
Hardware	Asia Pacific	Q2.06	14306431	13416447	889984	7
Hardware	Asia Pacific	Q4.06	12163497	10435666	1727831	17
Hardware	Europe	Q2.06	3298399	3293269	5129	0
Hardware	Europe	Q4.06	3197593	3093762	103832	3
Hardware	Europe	Q3.06	3093762	3298399	-204637	-6
Hardware	Europe	Q1.06	3293269	3264469	28801	1
Hardware	North America	Q1.06	16002175	14493426	1508749	10
Hardware	North America	Q3.06	15698208	16032643	-334436	-2
Hardware	North America	Q2.06	16032643	16002175	30469	0
Hardware	North America	Q4.06	15958791	15698208	260583	2
Hardware	Asia Pacific	Q1.06	13416447	14273900	-857453	-6
Software/Other	Asia Pacific	Q1.06	563786	672404	-108619	-16
Software/Other	Asia Pacific	Q3.06	647019	551149	95870	17
Software/Other	Asia Pacific	Q2.06	551149	563786	-12637	-2

Adding Dimension Choice Lists

Like BI Publisher, Application Express enables you to drill on the dimensions by adding choice lists of dimension keys. The dashboard user can choose a particular item from the list and dynamically change the selection of data displayed in one or more graphics and crosstabs on the page. To implement a choice list, take these steps:

- Create a new region on the page to display the list.
- Create a list of values (LOV).
- Create a list item with a bind variable to display the LOV.
- Create an unconditional branch for the list.
- Edit the query to use the bind variable.

The Page Definition is where you can create new pages and edit existing ones, including adding new graphical items and modifying existing ones. The items are organized in three columns: Page Rendering, Page Processing, and Shared Components.

To open the Page Definition:

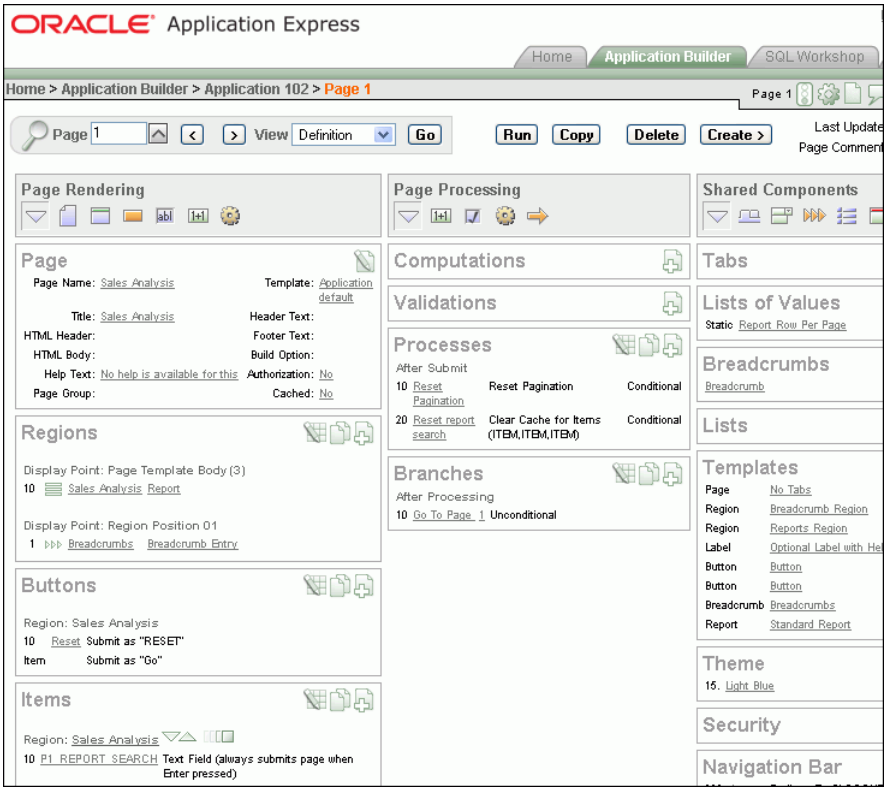
After running the application, click the **Edit Page** link at the bottom of the page.

or

On the Application home page, click the icon for the page where the report is defined.

Figure 6–9 shows an area of the Page Definition.

Figure 6–9 Application Express Page Definition



Creating a Region

You can create the choice list in a plain HTML area at the top of the page.

To create an empty HTML region:

1. On the Page Definition under Regions, click the **Create** icon.

   The Create Region wizard opens.

2. On the Region pages, select **HTML**, click **Next**, then select **HTML** again.

3. On the Display Attributes page, enter a descriptive title and select an appropriate template and location on the page for the lists.

   For this example, the name is `lov_region`, the template is **No Template**, and the location is **Page Template Body (1 items below template content)**. The name can be displayed on the rendered page, but it is hidden in this example.

4. Click **Create Region**.

   The new region appears on the Page Definition under Regions.

Creating a List of Values

For a list of values, use a SQL query like the one shown here. Include the LONG_DESCRIPTION and DIM_KEY columns from the hierarchy view. This query creates a list for the Customer Shipments hierarchy:

```
SELECT long_description, dim_key
     FROM customer_shipments_view
     WHERE parent = 'TOTAL'
     OR dim_key= 'TOTAL'
     ORDER BY level_name, long_description;

LONG_DESCRIPTION     DIM_KEY
-------------------- ----------
Asia Pacific         APAC
Europe               EMEA
North America        AMER
Total Customer       TOTAL
```

To create a List of Values

1. On the Page Definition under List of Values, click the **Create** icon.

   The Create List of Values wizard opens.

2. On the Source page, select **From Scratch.**
3. On the Name and Type page, enter a descriptive name and select **Dynamic**.

   This example uses the name CUSTOMER_LOV.

4. On the Query page, enter a query like the one shown previously. Do not use a semicolon.
5. Click **Create List of Values**.

   The new LOV appears in the Page Definition under List of Values.

For additional LOVs, repeat these steps. This example creates LOVs for the Product and Customer dimensions.

Creating the Choice List

For a choice list, you create a list item that displays the LOV.

To create a list item:

1. On the Page Definition under Items, click the **Create** icon.

   The Create Item wizard opens.

2. On the Item Type page, select **Select List.**
3. For Control Type, select **Select List with Submit.**
4. On the Display Position and Name page:
   - Enter a name that identifies the dimension, such as P1_CUSTOMER for the name of the Customer bind variable. P1 is the page number, and CUSTOMER identifies the Customer dimension.
   - Choose the new HTML region for the location of the list.

5. On the List of Values page, set these values:
   - Named LOV to the List Of Values created for this dimension, such as `CUSTOMER_LOV`.
   - Display Null Option to **No**.
6. Select the Item attributes according to your own preferences.
7. On the Source page, enter the name of the top dimension key for the default value.

   For the Global Customer dimension, the value is `TOTAL`.
8. Click **Create Item**.

Repeat these steps for other lists. This example creates lists for the Product and Customer dimensions.

To activate the list item:

1. On the Page Definition under Branches, click the **Create** icon.

   The Edit Branch wizard opens.
2. On the Point and Type page, accept the default settings.
3. On the Target page:
   - Set Target to **Page in This Application**.
   - Set Page to the page with the list item, which is `1` in this example.
   - Select **Reset Pagination For This Page**.
4. On the Branch Conditions page, accept the default settings to create an unconditional branch.
5. Click **Apply Changes**.

   The Edit Branch page closes, and you return to the Page Definition. The new unconditional branch is listed under Branches.

Editing the Query

This is the format for the dynamic conditions in the `WHERE` clause:

```
parent_column = NVL(:bind_variable, 'top dim_key')
```

The `NVL` function substitutes the name of the top dimension key in the hierarchy for null values. The dimension keys at the top have no parent key.

To edit the query:

1. Open the Page Definition.
2. Under Regions, click the **Edit Region** link. In this example, the region is named Sales Report.

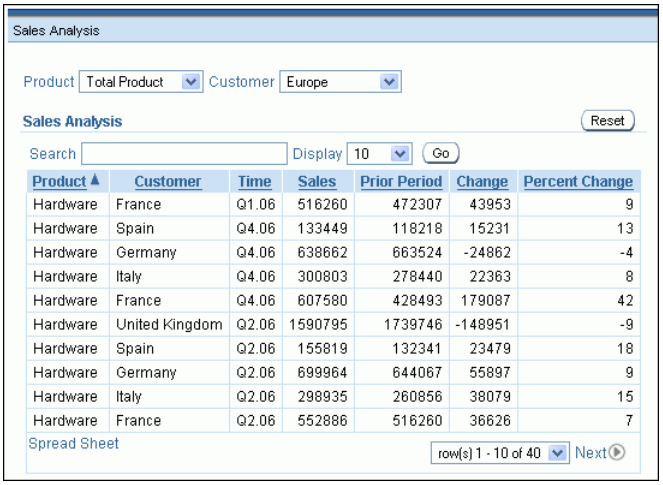
   The Edit Region page opens.
3. Under Source, modify the query:
   - Change the `WHERE` clause to use the bind variables.
   - Delete the outer `SELECT` added by Application Express.
4. Click **Apply Changes**.

For this example, the WHERE clause now looks like this:

```
WHERE p.parent = NVL(:P1_PRODUCT, 'TOTAL')
    AND cu.parent = NVL(:P1_CUSTOMER, 'TOTAL')
    AND t.parent = 'CY2006'
    AND ch.level_name = 'TOTAL'
```

Figure 6–10 shows the modified page with choice lists for Product and Customer.

Figure 6–10 Dashboard With Choice Lists for Drilling



Drilling on DImension Columns

You can enable users to drill down from the top of a hierarchy to the detail level using a single query. To implement drilling in Application Express, take these steps:

- Create hidden items with bind variables.
- Edit the query to use the bind variables.
- Add links to the dimension columns of the crosstab.

This example adds drilling to all displayed dimensions.

Creating Hidden Items

You can create various types of items in Application Express that provide bind variables. They store the session state for a particular element, in this case, the current selection of a parent dimension key.

Each dimension that will support drilling needs a bind variable. In this example, Product and Customer already have bind variables created with the list items. Time is the only displayed dimension in the report that does not have a bind variable. Because links in the Time dimension column will provide the user interface for changing the session state, Time does not need any other graphical user interface. A hidden item serves the purpose.

To create a hidden item:

1. Open the Page Definition.
2. Under Items, click the **Create** icon.

   The Create Item wizard opens.
3. On the Item Type page, select **Hidden**.
4. On the Display Position and Name page:
   - Enter a name that identifies the dimension, such as `P1_TIME` for the name of the Time bind variable.
   - Choose the region where the report is defined.
5. On the Source page, enter the dimension key at the top of the hierarchy.

   `TOTAL` is the top of all hierarchies in the Global schema. For this example, Time is set to `CY2006` to restrict the selection to one year.
6. Click **Create Item**.
7. Repeat these steps for any other dimensions that will support drilling only on the column links.

   For this example, a hidden item is defined for Time.

Editing the Query

To add column links to a report, you must change two areas of the `SELECT` statement:

- Select list: Application Express manages only those columns that appear in the select list. You can choose to display or hide the columns. For defining the column links, add the `DIM_KEY` and `PARENT` columns in the hierarchy views to the query select list.
- `WHERE` clause: Add the bind variables for the hidden items like you did for the choice lists in "Editing the Query" on page 6-10.

Example 6–3 shows the modified sample query.

Example 6–3 Revised Query for Column Links in Application Express

```
SELECT p.long_description "Product",
     cu.long_description "Customer",
     t.long_description "Time",
     ROUND(f.sales) "Sales",
     ROUND(f.sales_pp) "Prior Period",
     ROUND(f.sales_chg_pp) "Change",
     ROUND(f.sales_pctchg_pp * 100) "Percent Change",
/* Add DIM_KEY and PARENT columns for column links */
     p.dim_key product_key,
     p.parent product_parent,
     cu.dim_key customer_key,
     cu.parent customer_parent,
     t.dim_key time_key,
     t.parent time_parent
/* From dimension views and cube view */
FROM product_primary_view p,
     customer_shipments_view cu,
     time_calendar_view t,
     channel_primary_view ch,
     units_cube_view f
```

```
/* Use parent columns and bind variables for drilling */
WHERE p.parent = NVL(:P1_PRODUCT, 'TOTAL')
    AND cu.parent = NVL(:P1_CUSTOMER, 'TOTAL')
    AND t.parent = NVL(:P1_TIME, 'CY2006')
    AND ch.level_name = 'TOTAL'
/* Join dimension views to cube view */
    AND p.dim_key = f.product
    AND cu.dim_key = f.customer
    AND t.dim_key = f.time
    AND ch.dim_key = f.channel
```

Adding Links to the Dimension Columns

When a dashboard user clicks a linked dimension key in the crosstab, the value of the bind variable changes, causing the crosstab to change also. After drilling down a hierarchy, the user can restore the display to its original selection of data by pressing the Reset button. To implement these column links, you must add the column links and activate the Reset button.

To add a link to a dimension column:

1. Open the Page Definition.
2. Under Regions, click the **Report** link.

   The Report Attributes page opens.
3. Under Column Attributes, modify the report display:
   - Clear the Show check boxes for columns that you want to hide, such as the `DIM_KEY` and `PARENT` columns.
   - Set the Sort and Sort Sequence check boxes for appropriate sorting for the report. In this example, the sort order is `Product` (1), `Customer` (2), and `Time` (3).
4. Click the **Edit** icon for a dimension column.

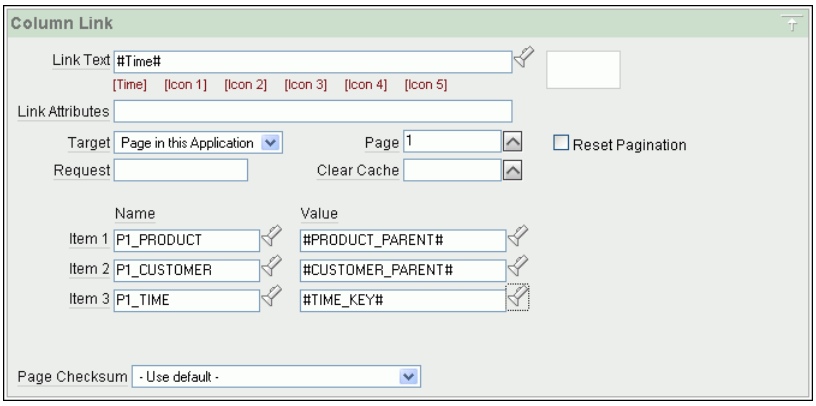
   The Column Attributes page opens.
5. Under Column Link, define the link as follows:
   - Link Text: Choose the dimension name.
   - Page: Enter the page number.
   - Name: List the dimensions in the order they appear in the report. **Item** is the name of the bind variable. **Value** is the `DIM_KEY` column for the dimension being defined or the `PARENT` column for the other dimensions.

   Figure 6–11 shows the link definition for the Time dimension.
6. Click **Apply Changes**.

   The Column Attributes page closes, and you return to the Report Attributes page.
7. Define links on the other dimension columns.
8. Click **Apply Changes**.

   The Report Attributes page closes, and you return to the Page Definition.

Figure 6–11 Definition of the Time Link



To activate the Reset button:

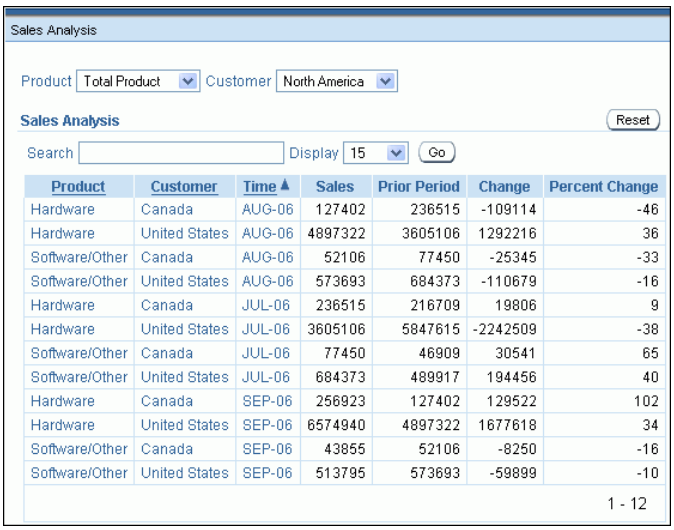
1. Open the Page Definition.
2. Under Branches, click the **Go to Page conditional** link.

   The Reset button was created on the page automatically along with its conditional branch. The Edit Branch page opens.
3. Under Action, set Clear Cache to the page number (in this example, `1`).
4. Under Conditions, set When Button Pressed to **RESET**.
5. Click **Apply Changes**.

   The Edit Branch page closes, and you return to the Page Definition.
6. Click **Run** to display the page.

Figure 6–12 shows the finished page displaying months in Q3.06. You can continue working on this application, adding more reports and charts to the page. For the SQL queries providing data to those reports and charts, you can re-use the same bind variables for the dimensions.

Figure 6–12 Sales Analysis Report With Column Links in Application Express



7

Administering Oracle OLAP

Because Oracle OLAP is contained in the database and its resources are managed using the same tools, the management tasks of Oracle OLAP and the database converge. Nonetheless, you should address tasks such as database tuning in the specific context of data warehousing.

This chapter contains the following topics:

Setting Database Initialization Parameters

Table 7–1 identifies the parameters that affect the performance of Oracle OLAP. Alter your server parameter file or `init.ora` file to these values, then restart your database instance. You can monitor the effectiveness of these settings and adjust them as necessary.

See Also:

- *Oracle Database Performance Tuning Guide* for information about tuning parameter settings
- *Oracle Database Reference* for descriptions of individual parameters

Table 7–1 Initial Settings for Database Parameters

Parameter	Default Value	Recommended Setting	Description
JOB_QUEUE_ PROCESSES	0	Number of CPUs, plus one additional process for every three CPUs; in a multi-core CPU, each core counts as a CPU For example, JOB_QUEUE_ PROCESSES=5 for a four-processor computer	Controls the degree of parallelism in OLAP builds, as described in "Parallelism" on page 7-6
SESSIONS	Derived	2.5 * maximum number of simultaneous OLAP users	Provides sufficient background processes for each user
UNDO_ MANAGEMENT	AUTO (MANUAL in 10*g*)	AUTO	Specifies use of an undo tablespace
UNDO_ TABLESPACE	Derived	Name of the undo tablespace, which must already be defined	Identifies the undo tablespace defined for OLAP use, as shown in "Creating an Undo Tablespace" on page 7-2

To set the system parameters:

1. Open the `init.ora` initialization file in a text editor.
2. Add or change the settings in the file, as described in Table 7–1.
3. Stop and restart the database.

   On Windows, use the Services utility to stop and restart `OracleService`.

   On Linux, use commands like the following. Be sure to identify the initialization file in the `STARTUP` command.

```
SQLPLUS '/ AS SYSDBA'
SHUTDOWN IMMEDIATE
STARTUP pfile=$ORACLE_BASE/admin/orcl/pfile/init.ora.724200516420
```

Storage Management

Analytic workspaces are stored in the owner's default tablespace, unless the owner specifies otherwise. All tablespaces for OLAP use should specify `EXTENT MANAGEMENT LOCAL`. Tablespaces created using default parameters may use resources inefficiently. You should create undo, permanent, and temporary tablespaces that are appropriate for storing analytic workspaces.

Creating an Undo Tablespace

Create an undo tablespace with the `EXTENT MANAGEMENT LOCAL` clause, as shown in this example:

```
CREATE UNDO TABLESPACE olapundo DATAFILE '$ORACLE_BASE/oradata/undo.dbf'
     SIZE 64M REUSE AUTOEXTEND ON NEXT 8M
     MAXSIZE UNLIMITED EXTENT MANAGEMENT LOCAL;
```

After creating the undo tablespace, change your system parameter file to include the following settings, then restart the database as described in "Setting Database Initialization Parameters" on page 7-1.

```
UNDO_TABLESPACE=tablespace
UNDO_MANAGEMENT=AUTO
```

Creating Permanent Tablespaces for OLAP Use

Each dimensional object occupies at least one extent. A fixed extent size may waste most of the allocated space. For example, if an object is 64K and the extents are set to a uniform size of 1M (the default), then only a small portion of the extent is used.

Create permanent tablespaces with the `EXTENT MANAGEMENT LOCAL` and `SEGMENT SPACE MANAGEMENT AUTO` clauses, as shown in this example:

```
CREATE TABLESPACE glo DATAFILE '$ORACLE_BASE/oradata/glo.dbf'
     SIZE 64M REUSE AUTOEXTEND ON NEXT 8M MAXSIZE UNLIMITED
     EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO;
```

Creating Temporary Tablespaces for OLAP Use

Oracle OLAP uses the temporary tablespace to store all changes to the data in a cube, whether the changes are the result of a data load or data analysis. Saving the cube moves the changes into the permanent tablespace and clears the temporary tablespace.

This usage creates numerous extents within the tablespace. A temporary tablespace suitable for use by Oracle OLAP should specify the `EXTENT MANAGEMENT LOCAL` clause and a `UNIFORM SIZE` clause with a small size, as shown in this example:

```
CREATE TEMPORARY TABLESPACE glotmp TEMPFILE '$ORACLE_BASE/oradata/glotmp.tmp'
     SIZE 50M REUSE AUTOEXTEND ON NEXT 5M MAXSIZE UNLIMITED
     EXTENT MANAGEMENT LOCAL UNIFORM SIZE 256K;
```

Spreading Data Across Storage Resources

Oracle Database provides excellent storage management tools to simplify routine tasks. Automatic Storage Management (ASM) provides a simple storage management interface that virtualizes database storage into disk groups. You can manage a small set of disk groups, and ASM automates the placement of the database files within those disk groups.

ASM spreads data evenly across all available storage resources to optimize performance and utilization. After you add or drop disks, ASM automatically rebalances files across the disk group.

Because OLAP is part of Oracle Database, you can use ASM to manage both relational and dimensional data.

ASM is highly recommended for analytic workspaces. A system managed with ASM is faster than a file system and easier to manage than raw devices. ASM optimizes the performance of analytic workspaces both on systems with RAC and those without RAC.

However, you do not need ASM to use Oracle OLAP. You can still spread your data across multiple disks, just by defining the tablespaces like in this example:

```
CREATE TABLESPACE glo DATAFILE
     'disk1/oradata/glo1.dbf' SIZE 64M REUSE AUTOEXTEND ON NEXT 8M MAXSIZE 1024M
     EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO;

ALTER TABLESPACE glo ADD DATAFILE
     'disk2/oradata/glo2.dbf' SIZE 64M REUSE AUTOEXTEND ON NEXT 8M MAXSIZE 1024M,
     'disk3/oradata/glo3.dbf' SIZE 64M REUSE AUTOEXTEND ON NEXT 8M
          MAXSIZE UNLIMITED;
```

Dictionary Views and System Tables

Oracle Database data dictionary views and system tables contain extensive information about analytic workspaces.

Static Data Dictionary Views

Among the static views of the database data dictionary are several that provide information about analytic workspaces. Table 7–2 provides brief descriptions of them. All data dictionary views have corresponding DBA and USER views.

Table 7–2 Static Data Dictionary Views for OLAP

View	Description
ALL_AWS	Describes all analytic workspaces accessible to the current user.
ALL_AW_OBJ	Describes the current objects in all analytic workspaces accessible to the current user.
ALL_AW_PROP	Describes the properties defined in all analytic workspaces accessible to the current user.
ALL_AW_PS	Describes the page spaces currently in use by all analytic workspaces accessible to the current user.

See Also:

- "Querying the Data Dictionary" on page 4-19 for a list of data dictionary views that describe OLAP dimensional objects
- *Oracle Database Reference* for full descriptions of all data dictionary views

System Tables

The SYS user owns several tables associated with analytic workspaces. Table 7–3 provides brief descriptions.

Important: These tables are vital for the operation of Oracle OLAP. Do not delete them or attempt to modify them directly without being fully aware of the consequences.

Table 7–3 OLAP Tables Owned By SYS

Table	Description
AW$	Maintains a record of all analytic workspaces in the database, recording its name, owner, and other information.
AW$AWCREATE	Stores the AWCREATE analytic workspace, which contains programs for using OLAP Catalog metadata in Oracle Database 10*g* Release 10.1.0.2 and earlier releases. It exists only for backward compatibility.
AW$AWCREATE10G	Stores the AWCREATE10G analytic workspace, which contains programs for using OLAP Catalog metadata in Oracle Database 10*g* Release 10.1.0.3. The OLAP Catalog is not used by later releases. It exists only for backward compatibility.
AW$AWMD	Stores the AWMD analytic workspace, which contains programs for creating metadata catalogs.

Table 7–3 (Cont.) OLAP Tables Owned By SYS

Table	Description
AW$AWREPORT	Stores the AWREPORT analytic workspace, which contains a program named AWREPORT for generating a summary space report.
AW$AWXML	Stores the AWXML analytic workspace, which contains programs for creating and managing analytic workspaces for Oracle Database 10*g* Release 10.1.0.4 and later.
AW$EXPRESS	Stores the EXPRESS analytic workspace. It contains objects and programs that support basic operations. EXPRESS is used any time a session is open.
AW_OBJ$	Describes the objects stored in analytic workspaces.
AW_PRG$	Stores program data. Not currently used.
AW_PROP$	Stores analytic workspace object properties.
AW_TRACK$	Stores tracking data about access to aggregate cells. Not currently used.
PS$	Maintains a history of all page spaces. A page space is an ordered series of bytes equivalent to a file. Oracle OLAP manages a cache of workspace pages. Pages are read from storage in a table and written into the cache in response to a query. The same page can be accessed by several sessions. The information stored in PS$ enables Oracle OLAP to discard pages that are no longer in use, and to maintain a consistent view of the data for all users, even when the workspace is being modified during their sessions. When changes to a workspace are saved, unused pages are purged and the corresponding rows are deleted from PS$.

Analytic Workspace Tables

Analytic workspaces are stored in tables in the Oracle database. The names of these tables always begin with AW$.

For example, if the GLOBAL user creates two analytic workspaces, one named FINANCIALS and the other named MARKETING, then these tables will be created in the GLOBAL schema:

```
AW$FINANCIALS
AW$MARKETING
```

The tables store all of the object definitions and data.

Build Logs

The first time you load data into a cube or dimension using Analytic Workspace Manager, a table named CUBE_BUILD_LOG is created in your schema. This table stores information about the build. Information about each subsequent build is added to the table and is identified by its own build identifier. CUBE_BUILD_LOG is populated whenever a cube is refreshed, whether by Analytic Workspace Manager, the database materialized view refresh subsystem, or a PL/SQL procedure. It is updated in real time during the build, so that you can check its status at any time by querying the table in SQL.

The Maintenance Wizard in Analytic Workspace Manager displays the relevant rows from this table at the end of every build on the Log page.

Partitioned Cubes and Parallelism

Cubes are often partitioned to improve build and maintenance times. For information about creating a partitioned cube, refer to "Choosing a Partitioning Strategy" on page 3-17.

Querying Metadata for Cube Partitioning

To discover the current partitioning, query the ALL_CUBES data dictionary view. The PARTITION_DIMENSION_NAME, PARTITION_HIERARCHY_NAME, and PARTITION_LEVEL_NAME columns display partitioning information. For example, the following query shows that the Units Cube is partitioned on the Time dimension, the Calendar hierarchy, and the Calendar Year level.

```
SELECT partition_dimension_name, partition_hierarchy_name,
     partition_level_name FROM all_cubes
     WHERE owner='GLOBAL' AND cube_name='UNITS_CUBE';

PARTITION_DIMENSION_NAME  PARTITION_HIERARCHY_NAME  PARTITION_LEVEL_NAME
------------------------- ------------------------- --------------------
TIME                      CALENDAR                  CALENDAR_YEAR
```

Creating and Dropping Partitions

The OLAP engine automatically creates and drops partitions as part of data maintenance, as members are added and deleted from the partitioning dimension.

For example, assume that in the sample Global analytic workspace, the Units cube is partitioned on the Time dimension, using the Calendar hierarchy, and at the Calendar Quarter level. The OLAP engine creates a partition for each Calendar Quarter and its children. The default top partition contains Calendar Years and all members of the Fiscal hierarchy. If Global has three years of data, then the Units cube has 13 partitions: Four bottom partitions for each Calendar Year, plus the top partition.

A data refresh typically creates new time periods and deletes old ones. Whenever a Calendar Quarter value is loaded into the Time dimension, a corresponding new partition is added to the cube. Whenever a Calendar Quarter value is deleted from the Time dimension, the corresponding empty partition is deleted from the cube.

Parallelism

You can improve the performance of data maintenance by enabling parallel processing. There are two levels of parallelism:

- Parallel job execution: Loading and aggregating the data using multiple processes.
- Parallel update: Moving the data from temporary to permanent tablespaces using multiple processes.

This number of parallel processes is controlled by these factors:

- The number of objects that can be aggregated in parallel. Each cube and each partition (including the top partition) can use a separate process.

  You can control the number of partitions in a cube on the Partitioning tab of the cube property sheet in Analytic Workspace Manager.

- The number of simultaneous database processes the user is authorized to run.

  This number is controlled by the JOB_QUEUE_PROCESSES parameter. The setting for this parameter is based on the number of processors, as described in "Setting

Database Initialization Parameters" on page 7-1. If you have SYS privileges, you can obtain the current parameter setting with the following SQL command:

```
SHOW PARAMETER JOB_QUEUE_PROCESSES
```

- For parallel update, the number of processes you allocate to the job. You can specify the number of processes in the Maintenance Wizard of Analytic Workspace Manager when specifying the task processing options, or on the Materialized View tab of the cube.

Suppose that a cube is partitioned on the Quarter level of Time, and the cube contains three years of data. The cube has 3*4=12 bottom partitions, JOB_QUEUE_PROCESSES is set to 8, and you set the parallelism option to 4 for the build. Oracle Database will process the cube in this way:

1. Load and build the dimensions of the cube serially using a single process.
2. Load and build the 12 bottom partitions in parallel using 4 processes. As soon as one process finishes, another begins until all 12 are complete.

   This cube could use the 8 processes allowed by JOB_QUEUE_PROCESSES, but it is limited to 4 by the build setting.
3. Load and build the top partition.

Example 7–1 shows excerpts from CUBE_BUILD_LOG for a build of the Units cube and its dimensions. Partitioning on the Calendar Year level of the Time dimension created 10 bottom partitions for 1998 to 2007. JOB_QUEUE_PROCESSES is set to 2 and the parallelism option is set to 2 for the build also. The log shows that Oracle Database processed the Global in this way:

1. Processed the four dimensions serially.
2. Processed each partition of the Units cube

Example 7–1 Build Log for the Global Analytic Workspace

```
SELECT slave_number, status, command, build_object, partition FROM cube_build_log
     WHERE build_id='4';

SLAVE_NUMBER STATUS     COMMAND                   BUILD_OBJECT PARTITION
------------ ---------- ------------------------- ------------ ----------
             STARTED    BUILD
             STARTED    FREEZE
             COMPLETED  FREEZE
             STARTED    LOAD NO SYNCH             TIME
             COMPLETED  LOAD NO SYNCH             TIME
             STARTED    COMPILE                   TIME
             COMPLETED  COMPILE                   TIME
             STARTED    UPDATE                    TIME
             COMPLETED  UPDATE                    TIME
             STARTED    LOAD NO SYNCH             CHANNEL
             COMPLETED  LOAD NO SYNCH             CHANNEL
             STARTED    COMPILE                   CHANNEL
             COMPLETED  COMPILE                   CHANNEL
             STARTED    UPDATE                    CHANNEL
             COMPLETED  UPDATE                    CHANNEL
             STARTED    LOAD NO SYNCH             CUSTOMER
             COMPLETED  LOAD NO SYNCH             CUSTOMER
             STARTED    COMPILE                   CUSTOMER
             COMPLETED  COMPILE                   CUSTOMER
             STARTED    UPDATE                    CUSTOMER
```

```
   COMPLETED  UPDATE                    CUSTOMER
   STARTED    LOAD NO SYNCH             PRODUCT
   COMPLETED  LOAD NO SYNCH             PRODUCT
   STARTED    COMPILE                   PRODUCT
   COMPLETED  COMPILE                   PRODUCT
   STARTED    UPDATE                    PRODUCT
   COMPLETED  UPDATE                    PRODUCT
   STARTED    COMPILE AGGMAP            UNITS_CUBE
   COMPLETED  COMPILE AGGMAP            UNITS_CUBE
   STARTED    UPDATE                    UNITS_CUBE
   COMPLETED  UPDATE                    UNITS_CUBE
   STARTED    DBMS_SCHEDULER.CREATE_JOB UNITS_CUBE   P10
   COMPLETED  DBMS_SCHEDULER.CREATE_JOB UNITS_CUBE   P10
   STARTED    DBMS_SCHEDULER.CREATE_JOB UNITS_CUBE   P9
   COMPLETED  DBMS_SCHEDULER.CREATE_JOB UNITS_CUBE   P9
 1 STARTED    BUILD                                  P10
 1 STARTED    LOAD                      UNITS_CUBE   P10
 1 COMPLETED  LOAD                      UNITS_CUBE   P10
 1 STARTED    UPDATE                    UNITS_CUBE   P10
 1 COMPLETED  UPDATE                    UNITS_CUBE   P10
 1 COMPLETED  BUILD                                  P10
 2 STARTED    BUILD                                  P9
              .
              .
              .
   STARTED    DBMS_SCHEDULER.CREATE_JOB UNITS_CUBE   P0
   COMPLETED  DBMS_SCHEDULER.CREATE_JOB UNITS_CUBE   P0
11 STARTED    BUILD                                  P0
11 STARTED    LOAD                      UNITS_CUBE   P0
11 COMPLETED  LOAD                      UNITS_CUBE   P0
11 STARTED    SOLVE                     UNITS_CUBE   P0
11 COMPLETED  SOLVE                     UNITS_CUBE   P0
11 STARTED    UPDATE                    UNITS_CUBE   P0
11 COMPLETED  UPDATE                    UNITS_CUBE   P0
11 COMPLETED  BUILD                                  P0
   STARTED    ANALYZE                   UNITS_CUBE
   COMPLETED  ANALYZE                   UNITS_CUBE
   STARTED    THAW
   COMPLETED  THAW
   COMPLETED  BUILD

144 rows selected.
```

Oracle Database allocates the specified number of processes regardless of whether all of them can be used simultaneously at any point in the job. For example, if your job can use up to three processes, but you specify five, then two of the processes allocated to your job cannot be used by it or by any other job.

If Oracle Database is installed with Real Application Clusters (RAC), then a script submitted to the job queue will be distributed across all nodes in the cluster. The performance gains can be significant. For example, a job running on four nodes in a cluster may run up to four times faster than the same job running on a single computer.

Analyzing Cubes and Dimensions

If your application executes queries directly against a single cube, you do not need to generate optimizer statistics for the cube. These queries are automatically optimized within the analytic workspace.

Optimizer statistics are used to create execution plans for queries that join two cube views or join a cube view to a table or a view of a table. They are also used for cost-based rewrite to cube materialized views. You need to generate the statistics only for these types of queries.

To generate optimizer statistics, use the `DBMS_AW_STATS` PL/SQL package. You can run this package in Analytic Workspace Manager as part of a cube script, in SQL*Plus, or in any other SQL interface. Generating the statistics does not have a significant performance cost.

`DBMS_AW_STATS` has the following syntax:

```
DBMS_AW_STATS.ANALYZE
     (object       IN VARCHAR2);
```

The argument can be either a cube or a dimension. Example 7–2 shows a sample script for generating statistics on the Units cube and its dimensions.

Example 7–2 Generating Statistics for the Units Cube

```
BEGIN
     DBMS_AW_STATS.ANALYZE('units_cube');
     DBMS_AW_STATS.ANALYZE('time');
     DBMS_AW_STATS.ANALYZE('customer');
     DBMS_AW_STATS.ANALYZE('product');
     DBMS_AW_STATS.ANALYZE('channel');
END;
/
```

Although you cannot view the statistics directly, you can examine the execution plans, as described in "Viewing Execution Plans" on page 4-17.

See Also: *Oracle Database Performance Tuning Guide*

Monitoring Analytic Workspaces

Oracle Database provides various tools to help you diagnose performance problems. As an Oracle DBA, you will find these tools useful in tuning the database:

- Oracle Enterprise Manager Database Control (Database Control) is a general database management and administration tool. In addition to facilitating basic tasks like adding users and modifying datafiles, Database Control presents a graphic overview of a database's current status. It also provides an interface to troubleshooting and performance tuning utilities.
- Automatic Workload Repository collects database performance statistics and metrics for analysis and tuning, shows the exact time spent in the database, and saves session information.
- Automatic Database Diagnostic Monitor watches database performance statistics to identify bottlenecks, analyze SQL statements, and offer suggestions to improve performance.

Oracle Database also provides system views to help you diagnose performance problems. The following topics identify views that are either specific to OLAP or provide database information that is pertinent to OLAP.

Dynamic Performance Views

Each Oracle Database instance maintains fixed tables that record current database activity. These tables collect data on internal disk structures and memory structures. Among them are tables that collect data on Oracle OLAP.

These tables are available to users through a set of dynamic performance views. By monitoring these views, you can detect usage trends and diagnose system bottlenecks. Table 7–4 provides a brief description of each view. Global dynamic performance views (GV$) are also provided.

> **See Also:** *Oracle Database Reference* for full descriptions of the OLAP dynamic performance views.

Table 7–4 OLAP Dynamic Performance Views

View	Description
V$AW_AGGREGATE_OP	Lists the aggregation operators available in analytic workspaces.
V$AW_ALLOCATE_OP	Lists the allocation operators available in analytic workspaces.
V$AW_CALC	Collects information about the use of cache space and the status of dynamic aggregation.
V$AW_LONGOPS	Collects status information about SQL fetches.
V$AW_SESSION_INFO	Collects information about each active session.
V$AW_OLAP	Collects information about the status of active analytic workspaces.

Table 7–5 describes some other dynamic performance views that are not specific to OLAP, but which you may want to use when tuning your database for OLAP.

Table 7–5 Selected Database Performance Views

View	Description
V$LOG	Displays log file information from the control file.
V$LOGFILE	Contains information about redo log files.
V$PGASTAT	Provides PGA memory usage statistics as well as statistics about the automatic PGA memory manager when PGA_AGGREGATE_TARGET is set.
V$ROWCACHE	Displays statistics for data dictionary activity. Each row contains statistics for one data dictionary cache.
V$SYSSTAT	Lists system statistics.

Basic Queries for Monitoring the OLAP Option

The following queries extract OLAP information from the data dictionary. You must have

More complex queries are provided in a script that you can download from the Oracle OLAP web site on the Oracle Technology Network. For descriptions of these scripts and download instructions, refer to "OLAP DBA Scripts" on page 7-12.

Is the OLAP Option Installed in the Database?

The OLAP option is provided with Oracle Database Enterprise Edition. To verify that the OLAP components have been installed, issue this SQL command:

```
SELECT comp_name, version, status FROM DBA_REGISTRY
     WHERE comp_name LIKE '%OLAP%';

COMP_NAME                VERSION                        STATUS
------------------------ ------------------------------ -----------
OLAP Analytic Workspace  11.1.0.7.0                     VALID
Oracle OLAP API          11.1.0.7.0                     VALID
OLAP Catalog             11.1.0.7.0                     VALID
```

What Analytic Workspaces are in the Database?

The DBA_AWS view provides information about all analytic workspaces. Use the following SQL command to get a list of names, their owners, and the version:

```
SELECT owner, aw_name, aw_version FROM DBA_AWS;

OWNER      AW_NAME                        AW_VERSION
---------- ------------------------------ ----------
SYS        EXPRESS                        11.1
GLOBAL     GLOBAL                         11.1
SYS        AWCREATE                       11.1
SH         SH                             11.1
SYS        AWMD                           11.1
SYS        AWXML                          11.1
SYS        AWREPORT                       11.1
SYS        AWCREATE10G                    11.1
```

> **See Also:** "System Tables" on page 7-4 for descriptions of the analytic workspaces owned by SYS.

How Big is the Analytic Workspace?

To find out the size in bytes of the tablespace extents for a particular analytic workspace, use the following SQL statements, replacing GLOBAL with the name of your analytic workspace.

```
SELECT extnum, SUM(dbms_lob.getlength(awlob)) bytes FROM global.aw$global
     GROUP BY extnum;

    EXTNUM      BYTES
---------- ----------
         0  191776956
```

To see the size of the LOB table containing an analytic workspace, use a SQL command like the following, replacing GLOBAL.AW$GLOBAL with the qualified name of your analytic workspace.

```
SELECT ROUND(SUM(dbms_lob.getlength(awlob))/1024,0) kb
     FROM global.aw$global;

        KB
----------
    187282
```

When Were the Analytic Workspaces Created?

The DBA_OBJECTS view provides the creation date of the objects in your database. The following SQL command generates an easily readable report for analytic workspaces.

```
SELECT owner, object_name, created, status FROM dba_objects
      WHERE object_name LIKE 'AW$%' AND object_name!='AW$'
      GROUP BY owner, object_name, created, status
      ORDER BY owner, object_name;

OWNER      OBJECT_NAME     CREATED   STATUS
---------- --------------- --------- -------
GLOBAL     AW$GLOBAL       05-JUL-07 VALID
SYS        AW$AWCREATE     30-JUN-07 VALID
SYS        AW$AWCREATE10G  30-JUN-07 VALID
SYS        AW$AWMD         30-JUN-07 VALID
SYS        AW$AWREPORT     30-JUN-07 VALID
SYS        AW$AWXML        30-JUN-07 VALID
SYS        AW$EXPRESS      30-JUN-07 VALID

7 rows selected.
```

OLAP DBA Scripts

You can download a file that contains several SQL scripts from the Oracle OLAP web site on the Oracle Technology Network. These scripts typically extract information from two or more system views and generate a report that may be useful in monitoring and tuning a database. To download the file, use this URL:

http://www.oracle.com/technology/products/bi/olap/olap_dba_scripts.zip

Table 7–6 describes these scripts. For more information, refer to the README file provided with the scripts.

Table 7–6 OLAP DBA Scripts

SQL Script	Description
aw_objects_in_cache	Identifies the objects in the buffer cache that are related to analytic workspaces.
aw_reads_writes	Tallies the reads from temporary and permanent tablespaces, the writes to cache, and the rows processed in analytic workspaces.
aw_size	Displays the amount of disk space used by each analytic workspace.
aw_tablespaces	Provides extensive information about the tablespaces used by analytic workspaces.
aw_users	Identifies the users of analytic workspaces.
aw_wait_events	Describes the wait events experienced by users of analytic workspaces over the previous hour.
buffer_cache_hits	Calculates the buffer cache hit ratio.
cursor_parameters	Indicates whether the database parameters that limit the number of open cursors are set too low.
olap_pga_performance	Determines how much PGA is in use, the size of the OLAP page pool, and the hit/miss ratio for OLAP pages for each user.

Table 7–6 (Cont.) OLAP DBA Scripts

SQL Script	Description
olap_pga_use	Determines how much PGA is consumed by the OLAP page pool to perform operations on analytic workspaces.
session_resources	Identifies the use of cursors, PGA, and UGA for each open session.
shared_pool_hits	Calculates the shared pool hit ratio.

Scripts for Monitoring Performance

Several of the scripts listed in "OLAP DBA Scripts" on page 7-12 provide detailed information about the use of memory and other database resources by OLAP sessions. You can use these scripts as is, or you can use them as the starting point for developing your own scripts.

Example 7–3 shows the information returned by the session_resources script. It lists the use of resources such as cursors, PGA, and UGA.

Example 7–3 Querying Session Resources

```
@session_resources

USERNAME             NAME                                VALUE
-------------------- ------------------------------ ----------
GLOBAL:86            opened cursors cumulative              621
                     opened cursors current                  18
                     session cursor cache count              50
                     session cursor cache hits              432
                     session pga memory                 5356368
                     session pga memory max            10468176
                     session stored procedure space           0
                     session uga memory                 4230692
                     session uga memory max             7049780

9 rows selected.
```

Monitoring Disk Space

Several of the scripts listed in "OLAP DBA Scripts" on page 7-12 provide detailed information about the use of disk space by analytic workspaces. Example 7–4 shows the information returned by the aw_size script. It lists all of the analytic workspaces in the database, the disk space they consume, and the tablespaces in which they are stored.

Example 7–4 Querying the Use of Disk Space By Analytic Workspaces

```
@aw_size

Analytic Workspace                        On Disk MB Tablespace
----------------------------------------- ---------- --------------------
GLOBAL.GLOBAL                                 249.31 GLOBAL
SYS.AWCREATE                                    7.81 SYSAUX
SYS.AWCREATE10G                                 1.63 SYSAUX
SYS.AWMD                                        8.00 SYSAUX
SYS.AWREPORT                                    1.63 SYSAUX
SYS.AWXML                                      18.00 SYSAUX
SYS.EXPRESS                                     2.25 SYSAUX
```

```
                                                ---------------
Total Disk:                                             288.63

7 rows selected.
```

Backup and Recovery

You can backup and recover analytic workspaces using the same tools and procedures as the rest of your database.

Oracle Recovery Manager (RMAN) is a powerful tool that simplifies, automates, and improves the performance of backup and recovery operations. RMAN enables one time backup configuration, automatic management of backups, and archived logs based on a user-specified recovery window, restartable backups and restores, and test restore/recovery.

RMAN implements a recovery window to control when backups expire. This lets you establish a period of time during which it is possible to discover logical errors and fix the affected objects by doing a database or tablespace point-in-time recovery. RMAN also automatically expires backups that are no longer required to restore the database to a point-in-time within the recovery window. Control file auto backup also allows for restoring or recovering a database, even when an RMAN repository is not available.

Export and Import

You can copy analytic workspaces in several different ways, either to replicate them on another computer or to back them up.

- **Data Pump**. Analytic workspaces are copied with the other objects in a schema or database export. Use the `expdp/impdp` database utilities.
- **Transportable Tablespaces**. Analytic workspaces are copied with the other objects to a transportable tablespace. However, you can only transport the tablespace to the same platform (for example, from Linux to Linux, Solaris to Solaris, or Windows to Windows) because the OLAP `DECIMAL` data type is hardware dependent. Use the `expdp/impdp` database utilities. Transportable tablespaces are much faster than dump files.
- **XML Templates**. A template saves the XML definition of objects in an analytic workspace. You can save the entire analytic workspace, or individual cubes, dimensions, and calculated measures. Using a saved template, you can create a new analytic workspace exactly like an existing one. The template does not save any data, nor does it save any customizations to the analytic workspace. You can copy a template to a different platform.

The owner of an analytic workspace can create an XML template, or export the schema to a dump file. Only users with the `EXP_FULL_DATABASE` privilege or a privileged user (such as `SYS` or a user with the `DBA` role) can export the full database or create a transportable tablespace.

See Also:

- "Using Templates to Re-Create Dimensional Objects" on page 3-27 for information about XML templates
- *Oracle Database Utilities* for information about Oracle Data Pump and the `expdp/impdp` commands

Cube Materialized Views

A cube materialized view is an Oracle OLAP cube that has been enhanced with the capabilities of a materialized view at build time.

> **See Also:** "Adding Materialized View Capability to a Cube" on page 3-24

Acquiring Information From the Data Dictionary

The data dictionary contains numerous static views that provide information about materialized views. They list cube materialized views along with all other materialized views. The ALL_MVIEW_DETAIL_SUBPARTITION view is the only exception, with no information about cube materialized views.

> **See Also:** *Oracle Database Reference* for complete descriptions of the data dictionary views

Identifying Cube Materialized Views

USER_MVIEWS contains a row for each materialized view owned by the current user. The following query lists the materialized views owned by the GLOBAL user. The CB$ prefix identifies a cube materialized view.

```
SELECT mview_name, refresh_mode "MODE", refresh_method "METHOD",
     last_refresh_date "DATE", staleness FROM user_mviews;

MVIEW_NAME                     MODE   METHOD   DATE      STALENESS
------------------------------ ------ -------- --------- -------------------
CB$TIME_CALENDAR               DEMAND FORCE    10-JUL-07 FRESH
CB$TIME_FISCAL                 DEMAND FORCE    10-JUL-07 FRESH
CB$PRODUCT_PRIMARY             DEMAND FORCE    10-JUL-07 FRESH
CB$CUSTOMER_SHIPMENTS          DEMAND FORCE    10-JUL-07 FRESH
CB$CUSTOMER_SEGMENT            DEMAND FORCE    10-JUL-07 FRESH
CB$CHANNEL_PRIMARY             DEMAND FORCE    10-JUL-07 FRESH
CB$UNITS_CUBE                  DEMAND FORCE    10-JUL-07 FRESH

7 rows selected.
```

The example shows the cube materialized views defined by Analytic Workspace Manager: One for each dimension hierarchy and one for each cube.

Identifying the Refresh Logs

Oracle Database can maintain a set of logs on the master tables for the cube materialized views. These logs support incremental (fast) refresh of the cube. The script generated by the Relational Schema Advisor creates a log for each fact and dimension table to record any changes to the data. The following query lists the materialized view logs owned by the GLOBAL user:

```
SELECT master, log_table FROM user_mview_logs;

MASTER                         LOG_TABLE
------------------------------ ------------------------------
CHANNEL_DIM                    MLOG$_CHANNEL_DIM
CUSTOMER_DIM                   MLOG$_CUSTOMER_DIM
PRODUCT_DIM                    MLOG$_PRODUCT_DIM
TIME_DIM                       MLOG$_TIME_DIM
UNITS_FACT                     MLOG$_UNITS_FACT
```

Initiating a Data Refresh

You can initiate a data refresh of a cube materialized view in several different ways using Analytic Workspace Manager or a PL/SQL package:

- **Automatic Refresh**: On the Materialized View tab for a cube, you can create a regular schedule for the materialized view refresh subsystem, as described in "Adding Materialized View Capability to a Cube" on page 3-24.
- **Maintenance Wizard**: The Maintenance Wizard is available for refreshing all cubes and dimensions, including cube materialized views.
- **DBMS_CUBE**: The DBMS_CUBE PL/SQL package is available for refreshing all cubes and dimensions.
- **DBMS_MVIEW**: The DBMS_MVIEW PL/SQL package contains several procedures for use with cube materialized views.

Using DBMS_CUBE

DBMS_CUBE can be used to create and populate an analytic workspace. You can use it to maintain any cube, including cube materialized views.

The following command initiates a complete refresh of UNITS_CUBE, which is enabled as a cube materialized view. It automatically refreshes any stale dimensions before refreshing the cube.

```
EXECUTE dbms_cube.build('GLOBAL.UNITS_CUBE');

PL/SQL procedure successfully completed.
```

You can determine the refresh method from USER_MVIEWS, as shown in "Identifying Cube Materialized Views" on page 7-15.

Using DBMS_MVIEW

DBMS_MVIEW can be used to refresh all types of materialized views. These refresh procedures can be used with cube materialized views:

- REFRESH refreshes a list of one or more materialized views.
- REFRESH_ALL_MVIEWS refreshes all materialized views that meet certain criteria.
- REFRESH_DEPENDENT refreshes all materialized views that depend on a particular master table and meet certain criteria.

Dimensions must be refreshed before the cube. An error is raised during refresh of a cube materialized view if any of its associated dimension materialized views are stale. The procedures in DBMS_MVIEW can refresh multiple materialized views in one call, but they do not guarantee the appropriate refresh order. You must be sure to list all the dimension materialized views before the cube materialized views when using this package.

The following command initiates a refresh of the materialized view for the CHANNEL_PRIMARY hierarchy. Only the Complete refresh type is valid for dimensions.

```
EXECUTE dbms_mview.refresh('CB$CHANNEL_PRIMARY', 'C');

PL/SQL procedure successfully completed.
```

Refresh Methods

In Analytic Workspace Manager, you can specify the COMPLETE, FAST, or FORCE methods for refreshing a cube. Two additional methods, FAST_PCT and FAST_SOLVE, are invoked by the materialized view subsystem. They are not separate choices.

Refresh Method Descriptions

Table 7–7 describes the refresh methods that are supported on cube materialized views.

Table 7–7 Refresh Methods For Cube Materialized Views

Refresh Method	Description
COMPLETE	Deletes and recreates the cube. This option supports arbitrarily complex mappings from the source tables to the cube.
FAST	Loads and re-aggregates only changed values, based on the materialized view logs. The source for the refresh is the incremental differences that have been captured in the materialized view logs, rather than the original mapped sources. These differences are used to incrementally rebuild the cube. Only cells that are affected by the changed values are re-aggregated. This option supports only simple mappings for cube materialized views, that is, where no expressions (other than *table.column*), views, or aggregations occur in the query defining the mapping. The materialized view subsystem determines whether to perform a FAST or a FAST_PCT refresh.
FAST_PCT	Loads and re-aggregates data only from changed partitions. The Partition Change Tracking method is not available for cube materialized views enabled for query rewrite. This method works best when the source table and the cube are partitioned on the same dimension. FAST_PCT does not use change logs. It is always available, and it is always faster than a COMPLETE refresh. The materialized view subsystem determines whether to perform a FAST or a FAST_PCT refresh.
FAST_SOLVE	Loads and re-aggregates only changed values, based on the original mapped data source. FAST_SOLVE is a new type of refresh only for cube materialized views. It incrementally re-aggregates the cube even when the refresh source is the original mapped source instead of the materialized view logs. The aggregation subsystem identifies the differences and then incrementally re-aggregates the cube. This option is supported for arbitrarily complex mappings from the source tables to the cube. To discover whether a FAST_SOLVE refresh has occurred, review the CUBE_BUILD_LOG table as shown in "Fast Solve Refreshes".
FORCE	Loads and re-aggregates only changed values, using the best method possible. The materialized view system first attempts a FAST refresh. If it cannot FAST refresh a cube materialized view, it performs a FAST_SOLVE refresh.

Fast Solve Refreshes

The build log lists the CLEAR LEAVES command when the FAST SOLVE method was used. Example 7–5 shows the rows of CUBE_BUILD_LOG concerned with building UNITS_CUBE.

See Also: "Build Logs" on page 7-5

Example 7–5 Identifying a FAST SOLVE Refresh

```
SELECT build_object, status, command FROM cube_build_log
     WHERE build_object='UNITS_CUBE'
     AND build_id=8;

BUILD_OBJECT STATUS     COMMAND
------------ ---------- ------------------------
UNITS_CUBE   STARTED    COMPILE AGGMAP
UNITS_CUBE   COMPLETED  COMPILE AGGMAP
UNITS_CUBE   STARTED    UPDATE
UNITS_CUBE   COMPLETED  UPDATE
UNITS_CUBE   STARTED    CLEAR LEAVES
UNITS_CUBE   COMPLETED  CLEAR LEAVES
UNITS_CUBE   STARTED    LOAD
UNITS_CUBE   COMPLETED  LOAD
UNITS_CUBE   STARTED    SOLVE
UNITS_CUBE   COMPLETED  SOLVE
UNITS_CUBE   STARTED    UPDATE
UNITS_CUBE   COMPLETED  UPDATE
UNITS_CUBE   STARTED    ANALYZE
UNITS_CUBE   COMPLETED  ANALYZE

14 rows selected.
```

Using Query Rewrite

Query rewrite changes a query to select data from the materialized views instead of calculating the result set from the master tables. The transformation is fully transparent to the client, and requires no mention of the materialized views in the SQL statement. In the case of cube materialized views, the query is written against the tables or views of a star or snowflake schema, and it is transformed into a query against a cube materialized view. This transformation can result in significant improvements in runtime performance.

Query rewrite requires optimizer statistics on the cubes and dimensions. You can discover whether a query will be rewritten by generating and examining its execution plan.

Oracle Database uses two initialization parameters to control query rewrite:

- QUERY_REWRITE_ENABLED: Enables or disables query rewrite globally for the database.
- QUERY_REWRITE_INTEGRITY: Determines the degree to which query rewrite monitors the consistency of materialized views with the source data. The trusted or stale tolerated settings are recommended when using rewrite to cube materialized views.

Administration of cube materialized views is the same as any other materialized view except that the cube materialized views must be in the same schema as the analytic workspace. Users require the GLOBAL QUERY REWRITE privilege to have rewrite to

materialized views that are in schemas other than their own. However, the owner can access the materialized views from any schema without additional privileges.

See Also:

- "Analyzing Cubes and Dimensions" on page 7-9 for information about optimizer statistics
- Viewing Execution Plans on page 4-17 for information about execution plans
- *Oracle Database Reference* for complete descriptions of the initialization parameters

Acquiring Additional Information About Cube Materialized Views

Oracle Database has numerous PL/SQL packages for managing materialized views. Cube materialized views are already optimized to provide the best performance, so you have no need to use most of these packages. Few design decisions remain for you to make. For this reason, the `TUNE_MVIEW` procedure of `DBMS_ADVISOR` is disabled for cube materialized views.

However, there are a few packages that you may find useful, as shown in Table 7–8.

Table 7–8 PL/SQL Packages for Cube Materialized Views

Package	Description
`DBMS_METADATA`	Returns the metadata for an object.
`DBMS_MVIEW`	Executes data refreshes. See "Initiating a Data Refresh" on page 7-16. You can use the `EXPLAIN_REWRITE` and `EXPLAIN_MVIEW` procedures to obtain information about cube materialized views.
`DBMS_XPLAN`	Displays an execution plan. See "Viewing Execution Plans" on page 4-17.

8

Security

Oracle OLAP secures your data using the standard security mechanisms of Oracle Database.

This chapter contains the following topics:

Security of Multidimensional Data in Oracle Database

Your company's data is a valuable asset. The information must be secure, private, and protected. Analytic data is particularly vulnerable because it is highly organized, easy to navigate, and summarized into meaningful units of measurement.

When you use Oracle OLAP, your data is stored in the database. It has the security benefits of Oracle Database, which leads the industry in security. You do not need to expose the data by transferring it to a stand-alone database. You do not need to administer security on a separate system. And you do not need to compromise your data by storing it in a less secure environment than Oracle Database.

Security Management

Because you have just one system to administer, you do not have to replicate basic security tasks such as these:

- Creating user accounts
- Creating and administering rules for password protection
- Securing network connections
- Detecting and eliminating security vulnerabilities
- Safeguarding the system from intruders

The cornerstone of data security is the administration of user accounts and roles. Users open a connection with Oracle Database with a user name and password, and they have access to both dimensional and relational objects in the same session.

Types of Security

Users by default have no access rights to an analytic workspace or any other data type in another user's schema. The owner or an administrator must grant them, or a role to which they belong, any access privileges.

Oracle OLAP provides two types of security: Object security and data security.

- **Object security** provides access to dimensional objects. You must set object security before other users can access them. Object security is implemented using SQL `GRANT` and `REVOKE`.
- **Data security** provides fine-grained control of the data on a cellular level. This type of security is optional. You only need to define data security policies when you want to restrict access to specific areas of a cube. Data security is implemented using the XML DB security of Oracle Database.

You can administer both data security and object security in Analytic Workspace Manager. For object security, you also have the option of using SQL `GRANT` and `REVOKE`.

About the Privileges

Using both object security and data security, you can grant and revoke the following privileges:

- **Alter**: Change the definition of a cube or dimension. Users need this privilege to create and modify a dimensional model.
- **Delete**: Remove old dimension members. Users need this privilege to refresh a dimension.
- **Insert**: Add new dimension members. Users need this privilege to refresh a dimension.
- **Select**: Query the cube or dimension. Users need this privilege to query a view of the cube or dimension or to use the `CUBE_TABLE` function. `CUBE_TABLE` is a SQL function that returns the values of a dimensional object.
- **Update**: Change the data values of a cube or the name of a dimension member. Users need this privilege to refresh a dimension or cube.

Users exercise these privileges either using Analytic Workspace Manager to create and administer dimensional objects, or by using SQL to query them. They do not issue commands such as SQL `INSERT` and `UPDATE` directly on the cubes and dimensions.

Layered Security

For dimensional objects, you can manage security at these levels:

- Dimension member
- Dimension
- Cube
- Analytic workspace
- View
- Materialized view

The privileges are layered so that, for example, a user with `SELECT` data security on Software products must also have `SELECT` object security on the `PRODUCT` dimension and the Global analytic workspace. Users also need `SELECT` privileges on the views of the dimensional objects.

You administer security on views and materialized views for dimensional objects the same way as for any other views and materialized views in the database.

Setting Object Security

You can use either SQL or Analytic Workspace Manager to set object security. The results are identical.

Using SQL to Set Object Security

You can set and revoke object privileges on dimensional objects using the SQL `GRANT` and `REVOKE` commands.

Setting Object Security on an Analytic Workspace

Object privileges on an analytic workspace simply open the container. You must grant object privileges on the cubes and dimensions for users to be able to access them. The table name is the same as the analytic workspace name, with the addition of an `AW$` prefix.

The following command enables Scott to attach the Global analytic workspace, `AW$GLOBAL`, to a session:

```
GRANT SELECT ON aw$global TO scott;
```

Setting Object Security on Dimensions

You can grant privileges on individual dimensions to enable users to query the dimension members and attributes. For users to query a cube, they must have privileges on every dimension of the cube.

The privileges apply to the entire dimension. However, you can set fine-grained access on a dimension to restrict the privileges, as described in "Creating Data Security Policies on Dimensions and Cubes" on page 8-6.

Example 8–1 shows the SQL commands that enable Scott to query the Product dimension. They give Scott `SELECT` privileges on the Product dimension, on the Global analytic workspace, and on the Product view.

Example 8–1 Privileges to Query the Product Dimension

```
GRANT SELECT ON product TO scott;
GRANT SELECT ON aw$global TO scott;
GRANT SELECT ON product_view TO scott;
```

Setting Object Security on Cubes

Privileges on cubes enable users to access business measures and perform analysis. You must also grant privileges on each of the dimensions of the cube.

The privileges apply to the entire cube. However, you can create a data security policy on the cube or on its dimensions to restrict the privileges, as described in "Creating Data Security Policies on Dimensions and Cubes" on page 8-6.

Example 8–2 shows the SQL commands that enable Scott to query the Units cube. They give Scott `SELECT` privileges on the Global analytic workspace, the cube, and all of its dimensions. Scott also gets privileges on the dimension views so that he can query the dimension attributes for formatted reports.

Example 8–2 Privileges to Query the Units Cube

```
/* Grant privileges on the analytic workspace */
GRANT SELECT ON global.aw$global TO scott;

/* Grant privileges on the cube */
GRANT SELECT ON global.units_cube TO scott;

/* Grant privileges on the dimensions */
GRANT SELECT ON global.channel TO scott;
GRANT SELECT ON global.customer TO scott;
GRANT SELECT ON global.product TO scott;
GRANT SELECT ON global.time TO scott;

/* Grant privileges on the cube, dimension, and hierarchy views */
GRANT SELECT ON global.units_cube_view TO scott;
GRANT SELECT ON global.channel_view TO scott;
GRANT SELECT ON global.channel_primary_view TO scott;
GRANT SELECT ON global.customer_view TO scott;
GRANT SELECT ON global.customer_shipments_view TO scott;
GRANT SELECT ON global.customer_segments_view TO scott;
GRANT SELECT ON global.product_view TO scott;
GRANT SELECT ON global.product_primary_view TO scott;
GRANT SELECT ON global.time_view TO scott;
GRANT SELECT ON global.time_calendar_view TO scott;
GRANT SELECT ON global.time_fiscal_view TO scott;

/* Grant privileges to materialized views using query rewrite */
GRANT GLOBAL QUERY REWRITE TO scott;
```

Example 8–3 shows the SQL commands that give SCOTT the privileges to modify and update all dimensional objects in GLOBAL using Analytic Workspace Manager.

> **Note:** The GRANT ALL commands encompass more privileges than those discussed in this chapter. Be sure to review the list of privileges before using GRANT ALL.

Example 8–3 Privileges to Modify and Refresh GLOBAL

```
/* Grant privilege to use Analytic Workspace Manager */
GRANT OLAP_USER TO scott;

/* Grant privileges on the analytic workspace */
GRANT ALL ON global.aw$global TO scott;

/* Grant privileges on the cubes */
GRANT ALL ON global.units_cube TO scott;
GRANT ALL ON global.price_cost_cube TO scott;

/* Grant privileges on the dimensions */
GRANT ALL ON global.channel TO scott;
GRANT ALL ON global.customer TO scott;
GRANT ALL ON global.product TO scott;
GRANT ALL ON global.time TO scott;
```

Using Analytic Workspace Manager to Set Object Security

Analytic Workspace Manager provides a graphical interface for setting object security. It also displays the SQL commands, so that you can cut-and-paste them into a script.

Setting Object Security on an Analytic Workspace

Take these steps to set object security on an analytic workspace in Analytic Workspace Manager:

1. In the navigation tree, right-click the analytic workspace and select **Set Analytic Workspace Object Security**.

   The Set Analytic Workspace Object Security dialog box is displayed.

2. Complete the dialog box, then click **OK**.

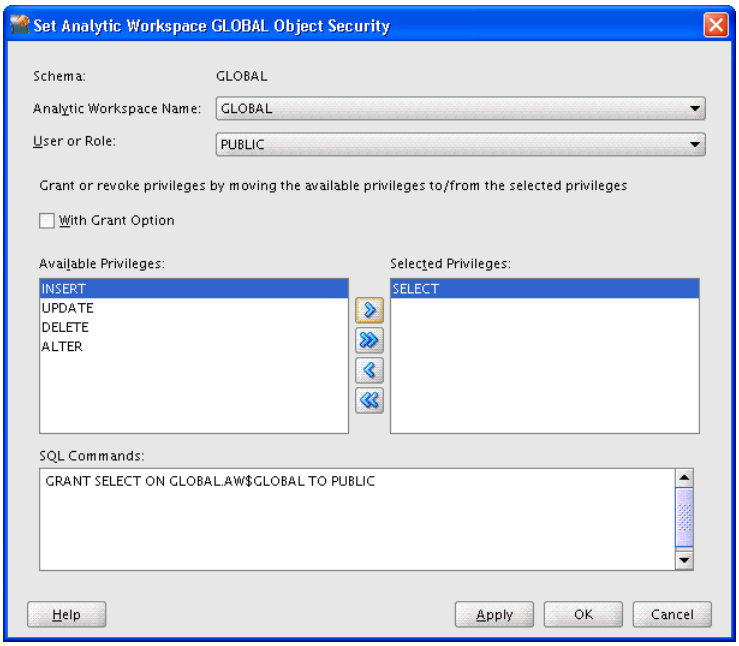
   Click **Help** for specific information about the choices.

3. Grant privileges on one or more cubes and their dimensions.

   Privileges on the analytic workspace do not automatically extend to the cubes and dimensions contained in the analytic workspace.

Figure 8–1 shows the `SELECT` privilege on `GLOBAL` granted to `PUBLIC`.

Figure 8–1 Setting Object Security on GLOBAL



Setting Object Security on Dimensions

Take these steps to set object security on dimensions in Analytic Workspace Manager:

1. In the navigation tree, right-click any dimension and select **Set Dimension Object Security**.

   The Set Dimension Object Security dialog box is displayed.

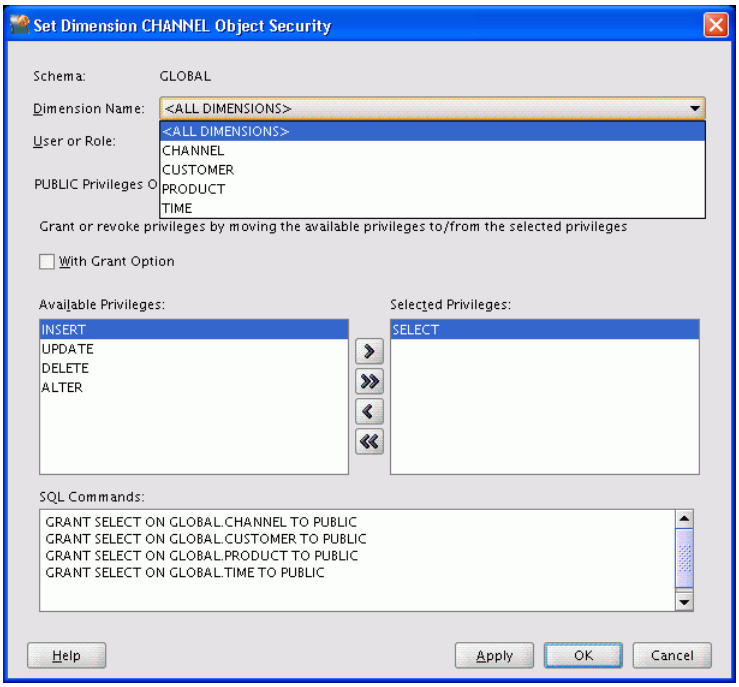
2. Complete the dialog box, then click **OK**.

   You can set privileges on all of the dimensions at one time. Click **Help** for specific information about the choices.

3. Grant privileges on the analytic workspace and one or more cubes. Use SQL to grant privileges on the views.

Figure 8–2 shows the SELECT privilege on all dimensions granted to PUBLIC.

Figure 8–2 Setting Object Security on Dimensions



Setting Object Security on Cubes

Take these steps to set object security on cubes in Analytic Workspace Manager:

1. In the navigation tree, right-click any cube and select **Set Cube Object Security**.

   The Set Cube Object Security dialog box is displayed.

2. Complete the dialog box, then click **OK**.

   You can set privileges on all of the cubes at one time. Click **Help** for specific information about the choices.

3. Grant privileges on the cube's dimensions and the analytic workspace. Use SQL to grant privileges on the views.

Creating Data Security Policies on Dimensions and Cubes

Data security policies enable you to grant users and roles privileges on a selection of dimension members. For example, you might restrict district sales managers to the data for just their own districts instead of all geographic areas. You can create a data security policy on dimensions, cubes, or both:

- When you create a data security policy on a dimension, the policy extends to all cubes with that dimension. You do not need to re-create the policy for each cube.
- When you create a data security policy on a cube, you select the members for each dimension of the cube. The policy only applies to that cube.

- When you create data security policies on both dimensions and cubes, users have privileges on the most narrowly defined portion of the data, where the policies overlap.

Granting Data Privileges

You can apply a policy to one or more users, roles, and data security roles. A data security role is a group of users and database roles that you can manage in Analytic Workspace Manager just for use in security policies. You create data security roles and policies in Analytic Workspace Manager.

As soon as you create a data security policy, all other users are automatically denied access. Analytic Workspace Manager creates a default policy that grants all privileges to the owner. Otherwise, the owner is denied access also.

> **Note:** Do not delete the default policy. It grants you the privileges to access your own data.

Selecting Data By Criteria

When defining a data security policy, you can select specific dimension members or those that meet certain criteria based on the dimension hierarchy. By using criteria instead of hard-coding specific dimension members, the selection remains valid after a data refresh. You do not need to modify the selection after adding new members. For example, a security policy that grants `SELECT` privileges to all Hardware products remains valid when old products are rolled off and new products are added to the `PRODUCT` dimension.

> **Note:** You must have the `OLAP_XS_ADMIN` role to manage data security policies in Analytic Workspace Manager.

To create a data security policy in Analytic Workspace Manager:

1. Expand the folder for a dimension or a cube.
2. Right-click Data Security and choose **Create Data Security Policy**.

   The Create Data Security Policy dialog box is displayed.
3. On the General tab, type a descriptive name in the Data Security Policy Name field.
4. Click **Add Users or Roles**.

   The Add Users or Roles dialog box is displayed.
5. Select the users, roles, and OLAP data security roles that you want to use this policy. Then click **OK** to close the dialog box.

   The selected users and roles are now listed in the table on the General tab.
6. Select the permissions you want to grant to each user or role.
7. On the Member Selection tab, select the dimension members or conditions. For cubes, set the scope for each dimension.
8. Click **OK** to save the data security policy.

   The new data security policy appears in the navigation tree in the Data Security folder for the dimension.

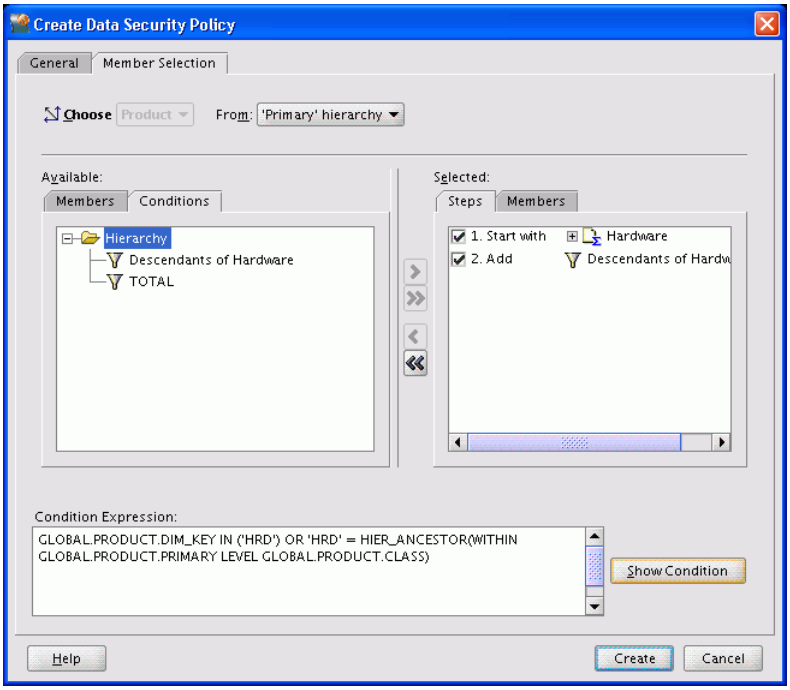
9. Grant these users and roles object privileges on the dimension or cube, and on the analytic workspace.

See Also:

- "Setting Object Security on Dimensions" on page 8-5
- "Setting Object Security on an Analytic Workspace" on page 8-5

Figure 8–3 shows the Member Selection tab of the data security policy for PRODUCT. Users who have privileges on the PRODUCT dimension based on this policy have access to all Hardware products. They do not have access to Software products or Total Product.

Figure 8–3 Restricting Product to Hardware and Descendants



See Also: Analytic Workspace Manager Help for information about creating data security roles.

9

Advanced Aggregations

A cube always returns summary data to a query as needed. While the cube may store data at the day level, for example, it will return a result at the quarter or year level without requiring a calculation in the query. This chapter explains how to optimize the unique aggregation subsystem of Oracle OLAP to provide the best performance for both data maintenance and querying.

This chapter contains the following topics:

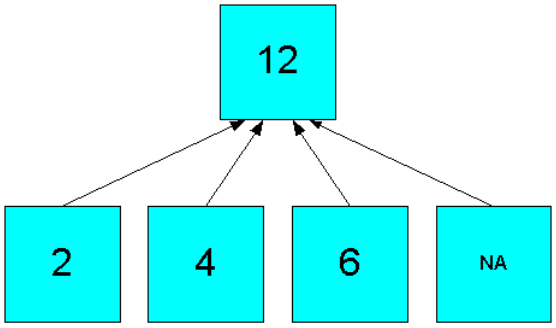
What is Aggregation?

Aggregation is the process of consolidating multiple values into a single value. For example, data can be collected on a daily basis and aggregated into a value for the week, the weekly data can be aggregated into a value for the month, and so on. Aggregation allows patterns in the data to emerge, and these patterns are the basis for analysis and decision making. When you define a data model with hierarchical dimensions, you are providing the framework in which aggregate data can be calculated.

Aggregation is frequently called summarization, and aggregate data is called summary data. While the most frequently used aggregation operator is Sum, there are many other operators, such as Average, First, Last, Minimum, and Maximum. Oracle OLAP also supports weighted and hierarchical methods. Following are some simple diagrams showing how the basic types of operators work. For descriptions of all the operators, refer to "Aggregation Operators" on page 9-3.

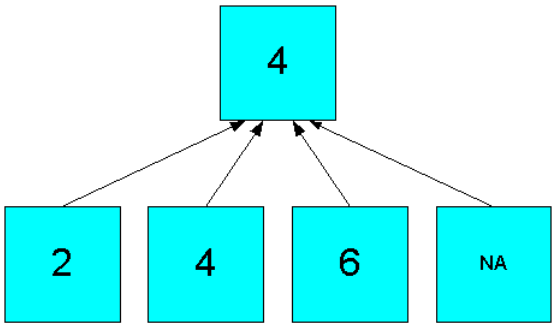
Figure 9–1 shows a simple hierarchy with four children and one parent value. Three of the children have values, while the fourth is empty. This empty cell has a null or NA value. The Sum operator calculates a value of (2 + 4 + 6)=12 for the parent value.

Figure 9–1 Summary Aggregation in a Simple Hierarchy



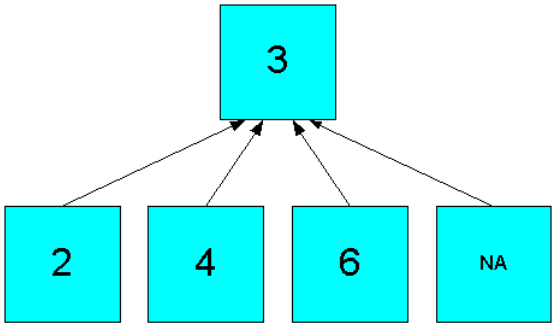
The Average operator calculates the average of all real data, producing an aggregate value of ((2 + 4 + 6)/3)=4, as shown in Figure 9–2.

Figure 9–2 Average Aggregation in a Simple Hierarchy



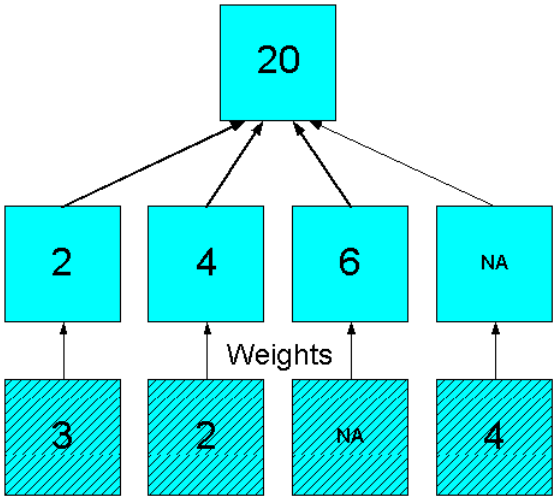
The hierarchical operators include null values in the count of cells. In Figure 9–3, the Hierarchical Average operator produces an aggregate value of ((2 + 4 + 6 +NA)/4)=3.

Figure 9–3 Hierarchical Average Aggregation in a Simple Hierarchy



The weighted operators use the values in another measure to generate weighted values before performing the aggregation. Figure 9–4 shows how the simple sum of 12 in Figure 9–1 changes to 20 by using weights ((3*2) + (2*4) + (NA*6) +(4*NA)).

Figure 9–4 Weighted Sum Aggregation in a Simple Hierarchy



Aggregation Operators

Analytic workspaces provide an extensive list of aggregation methods, including weighted, hierarchical, and weighted hierarchical methods.

Basic Operators

The following are descriptions of the basic aggregation operators:

- **Average**: Adds non-null data values, then divides the sum by the number of data values that were added together.
- **First Non-NA Data Value**: Returns the first real data value.
- **Last Non-NA Data Value**: Returns the last real data value.
- **Maximum**: Returns the largest data value among the children of each parent.
- **Minimum**: Returns the smallest non-null data value among the children of each parent.
- **Nonadditive**: Does not aggregate the data.
- **Sum**: Adds data values.

Scaled and Weighted Operators

These operators require a measure providing the weight or scale values in the same cube. In a weight measure, an NA (null) is calculated like a `1`. In a scale measure, an NA is calculated like a `0`.

The weighted operators use outer joins, as described in "When Does Aggregation Order Matter?" on page 9-4.

These are the scaled and weighted aggregation operators:

- **Scaled Sum**: Adds the value of a weight object to each data value, then adds the data values.
- **Weighted Average**: Multiplies each data value by a weight factor, adds the data values, and then divides that result by the sum of the weight factors.

- **Weighted First**: Multiplies the first non-null data value by its corresponding weight value.
- **Weighted Last**: Multiplies the last non-null data value by its corresponding weight value.
- **Weighted Sum**: Multiplies each data value by a weight factor, then adds the data values.

Hierarchical Operators

The following are descriptions of the hierarchical operators. They include all cells identified by the hierarchy in the calculations, whether or not the cells contain data.

Hierarchical Average and the Hierarchical Weighted operators use outer joins.

- **Hierarchical Average**: Adds data values, then divides the sum by the number of the children in the dimension hierarchy. Unlike Average, which counts only non-null children, hierarchical average counts all of the children of a parent, regardless of whether each child does or does not have a value.
- **Hierarchical First Member**: Returns the first data value in the hierarchy, even when that value is null.
- **Hierarchical Last Member**: Returns the last data value in the hierarchy, even when that value is null.
- **Hierarchical Weighted Average**: Multiplies non-null child data values by their corresponding weight values, then divides the result by the sum of the weight values. Unlike Weighted Average, Hierarchical Weighted Average includes weight values in the denominator sum even when the corresponding child values are null.
- **Hierarchical Weighted First**: Multiplies the first data value in the hierarchy by its corresponding weight value, even when that value is null.
- **Hierarchical Weighted Last**: Multiplies the last data value in the hierarchy by its corresponding weight value, even when that value is null.

When Does Aggregation Order Matter?

The OLAP engine aggregates a cube across one dimension at a time. When the aggregation operators are the same for all dimensions, the order in which they are aggregated may or may not make a difference in the calculated aggregate values, depending on the operator.

You should specify the order of aggregation when a cube uses more than one aggregation method. The only exceptions are that you can combine Sum and Weighted Sum, or Average and Weighted Average, when the weight measure is only aggregated over the same dimension. For example, a weight measure used to calculate weighted averages across Customer is itself only aggregated across Customer.

The weight operators are uncompressible for the specified dimension and all preceding dimensions. For a compressed cube, you should list the weighted operators as early as possible to minimize the number of outer joins. For example, suppose that a cube uses Weighted Sum across Customer, and Sum across all other dimensions. Performance will be best if Customer is aggregated first.

Using the Same Operator for All Dimensions of a Cube

The following information provides guidelines for when you need to specify the order of the dimensions as part of defining the aggregation rules for a cube.

Order Has No Effect

When these operators are used for all dimension of a cube, the order does not affect the results:

- Maximum
- Minimum
- Sum
- Hierarchical First Member
- Hierarchical Last Member
- Hierarchical Average

Order Changes the Aggregation Results

Even when these operators are used for all dimensions of a cube, the order can affect the results:

- Average
- First Non-NA Data Value
- Last Non-NA Data Value
- Weighted First
- Weighted Last
- Hierarchical Weighted First
- Hierarchical Weighted Last
- Scaled Sum

Order May Be Important

When the following weighted operators are used for all dimensions of a cube, the order affects the results only if the weight measure is aggregated over multiple dimensions:

- Weighted Average
- Weighted Sum
- Hierarchical Weighted Average

Example: Mixing Aggregation Operators

Even though you can use the Sum and Maximum operators alone without ordering the dimensions, you cannot use them together without specifying the order. The following figures show how they calculate different results depending on the order of aggregation. Figure 9–5 shows a cube with two dimensions. Sum is calculated first across one dimension of the cube, then Maximum is calculated down the other dimension.

Figure 9–5 Sum Method Followed by Maximum Method

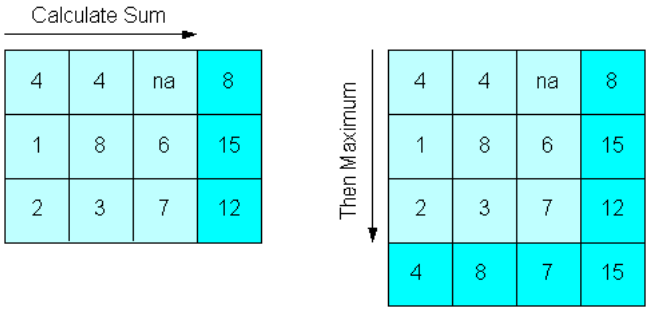
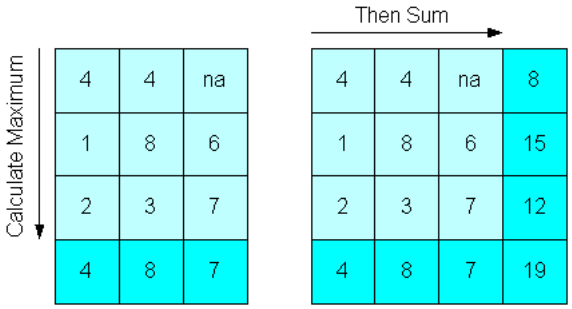


Figure 9–6 shows the same cube, except Maximum is calculated first down one dimension of the cube, then Sum is calculated across the other dimension. The maximum value of the sums in Figure 9–5 is 15, while the sum of the maximum values in Figure 9–6 is 19.

Figure 9–6 Max Method Followed by Sum Method



Example: Aggregating the Units Cube

This example describes changes to the default aggregation of the Units cube in the `GLOBAL` analytic workspace. These changes will take effect in the next data refresh.

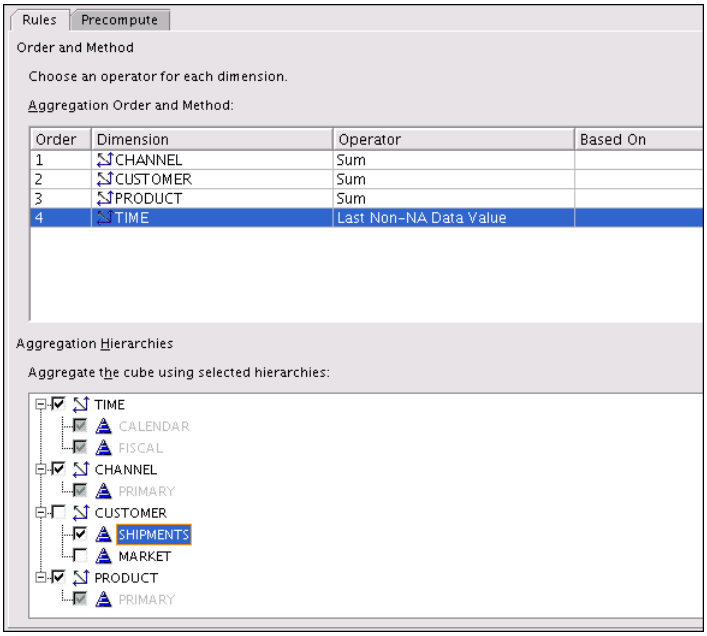
Selecting the Aggregation Operators and Hierarchies

Analytic Workspace Manager initially sets all dimensions to use the Sum operator and aggregates all levels of all dimensions. To change these default settings, use the Rules subtab of the Aggregation tab.

Figure 9–7 shows the operators for the Units Cube. Time is now set to Last Non-NA Data Value, and it will be aggregated after the other dimensions. For operators like First and Last, the order in which the dimensions are aggregated can change the results.

Another change is that only the Shipments hierarchy of the Customer dimension will be aggregated during data maintenance. Because the Segment hierarchy is seldom queried, the Global DBA chose not to calculate these aggregate values in order to save maintenance time and storage space. However, response time will be slower for queries that request Segment aggregations.

Figure 9–7 Selecting the Aggregation Operators



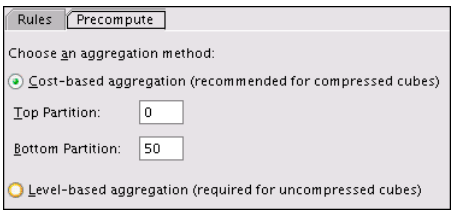
Choosing the Percentage of Precomputed Values

Analytic Workspace Manager initially chooses cost-based aggregation with 20% precomputed values for the bottom partitions and 0% for the top partition. An unpartitioned cube is also set to 20%. This setting means that 20% of the aggregate values will be calculated and stored during data maintenance, and 80% will be calculated in response to a query. These settings optimize data maintenance.

Increasing the materialization of the bottom partitions improves querying of both the bottom and the top partitions. Increasing the materialization of the top partition improves querying of the most aggregate data and any other hierarchies of the partitioned dimension.

Figure 9–8 shows the settings for the Units Cube. In this case, the Global DBA chose to keep the top partition at 0%, and to increase the bottom partitions from 20 to 50%. This change will increase maintenance costs in time and storage space, but will improve runtime performance of all partitions.

Figure 9–8 Setting Cost-Based Presummarization



A

Designing a Dimensional Model

This guide uses the Global schema for its examples. This appendix explores the business requirements of the fictitious Global Computing Company and discusses how the design of a data model emerges from these requirements.

This appendix contains the following topics:

Case Study Scenario

The fictional Global Computing Company was established in 1990. Global Computing distributes computer hardware and software components to customers on a worldwide basis. The Sales and Marketing department has not been meeting its budgeted numbers. As a result, this department has been challenged to develop a successful sales and marketing strategy.

Global Computing operates in an extremely competitive market. Competitors are numerous, customers are especially price-sensitive, and profit margins tend to be narrow. In order to grow profitably, Global Computing must increase sales of its most profitable products.

Various factors in Global Computing's current business point to a decline in sales and profits:

- Traditionally, Global Computing experiences low third-quarter sales (July through September). However, recent sales in other quarters have also been lower than expected. The company has experienced bursts of growth but, for no apparent reason, has had lower first-quarter sales during the last two years as compared with prior years.
- Global has been successful with its newest sales channel, the Internet. Although sales within this channel are growing, overall profits are declining.
- Perhaps the most significant factor is that margins on personal computers - previously the source of most of Global Computing's profits - are declining rapidly.

Global Computing needs to understand how each of these factors is affecting its business.

Current reporting is done by the IT department, which produces certain standard reports on a monthly basis. Any ad hoc reports are handled on an as-needed basis and are subject to the time constraints of the limited IT staff. Complaints have been

widespread within the Sales and Marketing department, with regard to the delay in response to report requests. Complaints have also been numerous in the IT department, with regard to analysts who change their minds frequently or ask for further information.

The Sales and Marketing department has been struggling with a lack of timely information about what it is selling, who is buying, and how they are buying. In a meeting with the CIO, the VP of Sales and Marketing states, "By the time I get the information, it's no longer useful. I'm only able to get information at the end of each month, and it doesn't have the details I need to do my job."

Reporting Requirements

When asked to be more specific about what she needs, the Vice President of Sales and Marketing identifies the following requirements:

- Trended sales data for specific customers, regions, and segments.
- The ability to provide information and some analysis capabilities to the field sales force. A Web interface would be preferred, since the sales force is distributed throughout the world.
- Detail regarding mail-order, phone, and e-mail sales on a monthly and quarterly basis, as well as a comparison to past time periods. Information must identify when, how, and what is being sold by each channel.
- Margin information on products in order to understand the dollar contribution for each sale.
- Knowledge of percent change versus the prior and year-ago period for sales, units, and margin.
- The ability to perform analysis of the data by ad hoc groupings.

The CIO has discussed these requirements with his team and has come to the conclusion that a standard reporting solution against the production order entry system would not be flexible enough to provide the required analysis capabilities. The reporting requirements for business analysis are so diverse that the projected cost of development, along with the expected turnaround time for requests, would make this solution unacceptable.

The CIO's team recommends using an analytic workspace to support analysis. The team suggests that the Sales and Marketing department's IT group work with Corporate IT to build an analytic workspace that meets their needs for information analysis.

Business Goals

The development team identifies the following high-level business goals that the project must meet:

- Global Computing's strategic goal is to increase company profits by increasing sales of higher margin products and by increasing sales volume overall.
- The Sales and Marketing department objectives are to:
  - Analyze industry trends and target specific market segments
  - Analyze sales channels and increase profits
  - Identify product trends and create a strategy for developing the appropriate channels

Information Requirements

Once you have established business goals, you can determine the type of information that will help achieve these goals. To understand how end users will examine the data in the analytic workspace, it is important to conduct extensive interviews. From interviews with key end users, you can determine how they look at the business, and what types of business analysis questions they want to answer

Business Analysis Questions

Interviews with the VP of Sales and Marketing, salespeople, and market analysts at Global Computing reveal the following business analysis questions:

- What products are profitable?
- Who are our customers, and what and how are they buying?
- What accounts are most profitable?
- What is the performance of each distribution channel?
- Is there still a seasonal variance to the business?

We can examine each of these business analysis questions in detail.

What products are profitable?

This business analysis question consists of the following questions:

- What is the percent of total sales for any item, product family, or product class in any month, quarter or year, and in any distribution channel? How does this percent of sales differ from a year ago?
- What is the unit price, unit cost, and margin for each unit for any item in any particular month? What are the price, cost, and margin trends for any item in any month?
- What items were most profitable in any month, quarter, or year, in any distribution channel, and in any geographic area or market segment? How did profitability change from the prior period? What was the percent change in profitability from the prior period?
- What items experienced the greatest change in profitability from the prior period?
- What items contributed the most to total profitability in any month, quarter, or year, in any distribution channel, and in any geographic area or market segment?
- What items have the highest per unit margin for any particular month?
- In summary, what are the trends?

Who are our customers, and what and how are they buying?

This business analysis question consists of the following questions:

- What were sales for any item, product family, or product class in any month, quarter, or year?
- What were sales for any item, product family, or product class in any distribution channel, geographic area, or market segment?
- How did sales change from the prior period? What was the percent change in sales from the prior period?

- How did sales change from a year ago? What was the percent change in sales from a year ago?
- In summary, what are the trends?

Which accounts are most profitable?

This business analysis question consists of the following questions:

- Which accounts are most profitable in any month, quarter, or year, in any distribution channel, by any item, product family, or product class?
- What were sales and extended margin (gross profit) by account for any month, quarter, or year, for any distribution channel, and for any product?
- How does account profitability compare to the prior time period?
- Which accounts experienced the greatest increase in sales as compared to the prior period?
- What is the percent change in sales from the prior period? Did the percent change in profitability increase at the same rate as the percent change in sales?
- In summary, what are the trends?

What is the performance of each distribution channel?

This business analysis question consists of the following questions:

- What is the percent of sales to total sales for each distribution channel for any item, product family, or product class, or for any geographic area or market segment?
- What is the profitability of each distribution channel: direct sales, catalog sales, and the Internet?
- Is the newest distribution channel, the Internet, "cannibalizing" catalog sales? Are customers simply switching ordering methods, or is the Internet distribution channel reaching additional customers?
- In summary, what are the trends?

Is there still a seasonal variance to the business?

This business analysis question consists of the following questions:

- Are there identifiable seasonal sales patterns for particular items or product families?
- How do seasonal sales patterns vary by geographic location?
- How do seasonal sales patterns vary by market segment?
- Are there differences in seasonal sales patterns as compared to last year?

Summary of Information Requirements

By examining the types of analyses that users wish to perform, we can identify the following key requirements for analysis:

- Global Computing has a strong need for profitability analysis. The company must understand profitability by product, account, market segment, and distribution channel. It also needs to understand profitability trends.

- Global Computing needs to understand how sales vary by time of year. The company must understand these seasonal trends by product, geographic area, market segment, and distribution channel.
- Global Computing has a need for ad hoc sales analysis. Analysis must identify what products are sold to whom, when these products are sold, and how customers buy these products.
- The ability to perform trend analysis is important to Global Computing.

Identifying Required Business Facts

The key analysis requirements reveal the business facts that are required to support analysis requirements at Global Computing.

These facts are ordered by time, product, customer shipment or market segment, and distribution channel:

Sales
Units
Change in sales from prior period
Percent change in sales from prior period
Change in sales from prior year
Percent change in sales from prior year
Product share
Channel share
Market share
Extended cost
Extended margin
Extended margin change from prior period
Extended margin percent change from prior period
Units sold, change from prior period
Units sold, percent change from prior period
Units sold, change from prior year
Units sold, percent change from prior year

These facts are ordered by item and month:

Unit price
Unit cost
Margin per unit

Designing a Dimensional Model for Global Computing

"Business Goals" on page A-2 identifies the business facts that will support analysis requirements at Global Computing. Next, we will identify the dimensions, levels, and attributes in a data model. We will also identify the relationships within each dimension. The resulting data model will be used to design the Global schema, the dimensional model, and the analytic workspace.

Identifying Dimensions

Four dimensions will be used to organize the facts in the database.

- Product shows how data varies by product.
- Customer shows how data varies by customer or geographic area.

- Channel shows how data varies according to each distribution channel.
- Time shows how data varies over time.

Identifying Levels

Now that we have identified dimensions, we can identify the levels of summarization within each dimension. Analysis requirements at Global Computing reveal that:

- There are three distribution channels: Sales, Catalog, and Internet. These three values are the lowest level of detail in the data warehouse and will be grouped in the Channel level. From the order of highest level of summarization to the lowest level of detail, levels will be Total and Channel.
- Global performs customer and geographic analysis along the line of shipments to customers and by market segmentation. Shipments and Segment will be two hierarchies in the Customer dimension. In each case, the lowest level of detail in the data model is the Ship To location.
  - When analyzing along the line of customer shipments, the levels of summarization will be (highest to lowest): Total, Region, Warehouse, and Ship To.
  - When analyzing by market segmentation, the levels of summarization will be (highest to lowest): Total, Market Segment, Account, and Ship To.
- The Product dimension will have four levels (highest to lowest): Total, Class, Family, and Item.
- The Time dimension will have four levels (highest to lowest): Total, Year, Quarter, and Month.

All dimensions have a Total level as the highest level of summarization. Adding this highest level provides additional flexibility as application users analyze data.

Identifying Hierarchies

We will identify the hierarchies that organize the levels within each dimension. To identify hierarchies, we will group the levels in the correct order of summarization and in a way that supports the identified types of analysis.

For the Channel and Product dimensions, Global Computing requires only one hierarchy for each dimension. For the Customer dimension, Global Computing requires two hierarchies. Analysis within the Customer dimension tends to be either by geographic area or market segment. Therefore, we will organize levels into two hierarchies, Shipments and Segment. Analysis over time also requires two hierarchies, a Calendar hierarchy and a Fiscal hierarchy.

Identifying Stored Measures

"Identifying Required Business Facts" on page A-5 lists 21 business facts that are required to support the analysis requirements of Global Computing. Of this number, only four facts need to be acquired from the transactional database:

- Units
- Sales
- Unit Price
- Unit Cost

All of the other facts can be derived from these basic facts. The derived facts can be calculated in the analytic workspace on demand. If experience shows that some of these derived facts are being used heavily and the calculations are putting a noticeable load on the system, then some of these facts can be calculated and stored in the analytic workspace as a data maintenance procedure.

Glossary

additive

Describes a measure or fact that can be summarized through addition, such as a `SUM` function. An additive measure is the most common type. Examples include sales, cost, and profit.

Contrast with **nonadditive**.

aggregation

The process of consolidating data values into a single value. For example, sales data could be collected on a daily basis and then be aggregated to the week level, the week data could be aggregated to the month level, and so on. The data can then be referred to as aggregate data.

The term aggregation is often used interchangeably with summarization, and aggregate data is used interchangeably with summary data. However, there are a wide range of aggregation methods available in addition to `SUM`.

allocation

The process of distributing aggregate data down a hierarchy to the detail level, sometimes using an existing set of data as the basis for the allocation. Allocation is often used in forecasting and budgeting systems. An example of a financial allocation is the automated distribution of a bonus pool, based on the current salaries and performance ratings of the employees.

analytic workspace

A container for storing related dimensional objects, such as dimensions and cubes. An analytic workspace is stored in a relational table.

See also **cube**, **cube dimension**.

ancestor

A dimension member at a higher level of aggregation than a particular member. For example, in a Time dimension, the year 2007 is the ancestor of the day 06-July-07. The member immediately above is the parent. In a dimension hierarchy, the data value of the ancestor is the aggregated value of the data values of its descendants.

Contrast with **descendant**. See also **hierarchy**, **level**, **parent**.

attribute

A database object related to an OLAP cube dimension. An attribute stores descriptive characteristics for all dimension members, or members of a particular hierarchy, or only members at a particular level of a hierarchy.

When the values of an attribute are unique, they provide supplementary information that can be used for display (such as a descriptive name) or in analysis (such as the number of days in a time period). When the values of an attribute apply to a group of dimension members, they enable users to select data based on like characteristics. For example, in a database representing footwear, you might use a color attribute to select all boots, sneakers, and slippers of the same color.

See also **cube dimension**.

base level data

See **detail data**.

base measure

See **measure**.

calculated measure

A stored expression that executes in response to a query. For example, a calculated measure might generate the difference in costs from the prior period by using the `LAG_VARIANCE` function on the `COSTS` measure. Another calculated measure might calculate profits by subtracting the `COSTS` measure from the `SALES` measure. The expression resolves only the values requested by the query.

See also **expression**, **measure**.

cell

A single data value of an expression. In a dimensioned expression, a cell is identified by one value from each of the dimensions of the expression. For example, if you have a measure with the dimensions `MONTH` and `CUSTOMER`, then each combination of a month and a customer identifies a separate cell of that measure.

See also **cube dimension**.

child

A dimension member that is part of a more aggregate member in a hierarchy. For example, in a Time dimension, the month Jan-06 might be the child of the quarter Q1-2006. A dimension member can be the child of a different parent in each hierarchy.

Contrast with **parent**. See also **descendant**, **hierarchy**.

composite

A compact format for storing sparse multidimensional data. Oracle OLAP provides two types of composites: a compressed composite for extremely sparse data, and a regular composite for moderately sparse data.

See also **dimension**, **sparsity**.

compressed cube

A cube with very sparse data that is stored in a compressed composite.

See also **composite**.

compression

See **compressed cube**.

consistent solve specification

See **solve specification**.

cube

An organization of measures with identical dimensions and other shared characteristics. The edges of the cube contain the dimension members, and the body of the cube contains the data values. For example, sales data can be organized into a cube whose edges contain values from the Time, Product, and Customer dimensions and whose body contains Volume Sales and Dollar Sales data.

cube dimension

A cube dimension is a dimensional object that stores a list of values. It is an index for identifying the values of a measure. For example, if Sales data has a separate sales figure for each month, then the data has a Time dimension that contains month values, which organize the data by month.

In the context of multidimensional analysis, a cube dimension is called a dimension.

See also **dimension**.

cube materialized view

A cube that has been enhanced with materialized view capabilities. A cube materialized view can be incrementally refreshed through the Oracle Database materialized view subsystem, and it can serve as a target for transparent rewrite of queries against the source tables.

Also called a cube-organized materialized view or a cube-based materialized view.

cube script

A sequence of steps that prepare the data for querying, such as loading and aggregating new data.

cube view

A relational view of the data stored in a cube, which can be queried by SQL. It contains columns for the dimensions, measures, and calculated measures of the cube.

custom measure

See **calculated measure**.

custom member

A dimension member whose data is calculated from the values of other members of the same dimension using the rules defined in a model.

See **model**.

data security role

A group of users and database roles that is defined just for use in managing OLAP security policies.

data source

A relational table, view, synonym, or other database object that provides detail data for cubes and cube dimensions.

data warehouse

A database designed for query and analysis rather than transaction processing. A data warehouse usually contains historical data that is derived from transaction data, but it can include data from other sources. It separates analysis workload from transaction workload and enables a business to consolidate data from several sources.

denormalized

Permit redundancy in a table. Contrast with **normalize**.

derived measure

See **calculated measure**.

descendant

A dimension member at a lower level of aggregation than a particular member. For example, in a Time dimension, the day 06-July-07 is the descendant of year 2007. The member immediately below is the child. In a dimension hierarchy, the data values of the descendants roll up into the data values of the ancestors.

Contrast with **ancestor**. See also **aggregation**, **child**, **hierarchy**, **level**.

detail data

Data at the lowest level, which is acquired from another source.

Contrast with **aggregation**.

dimension

A structure that categorizes data. Among the most common dimensions for sales-oriented data are Time, Geography, and Product. Most dimensions have hierarchies and levels.

In a cube, a dimension is a list of values at all levels of aggregation.

In a relational table, a dimension is a type of object that defines hierarchical (parent/child) relationships between pairs of column sets.

See also **cube dimension**, **hierarchy**.

dimension key

See **dimension member**.

dimension member

One element in the list that composes a cube dimension. For example, a Time dimension might have dimension members for days, months, quarters, and years.

dimension table

A relational table that stores all or part of the values for a dimension in a star or snowflake schema. Dimension tables typically contain columns for the dimension keys, levels, and attributes.

dimension value

See **dimension member**.

dimension view

A relational view of a cube dimension that provides information about all members of all hierarchies. It includes columns for the dimension keys, level, and attributes.

See also **cube dimension**, **hierarchy view**.

drill

To navigate from one item to a set of related items. Drilling typically involves navigating up and down through the levels in a hierarchy.

Drilling down expands the view to include child values that are associated with parent values in the hierarchy.

Drilling up collapses the list of descendant values that are associated with a parent value in the hierarchy.

EIF file

A specially formatted file for transferring data between analytic workspaces, or for storing versions of an analytic workspace (all of it or selected objects) outside the database.

embedded total

A list of dimension members at all levels of a hierarchy, such that the aggregate members (totals and subtotals) are interspersed with the detail members. For example, a Time dimension might contain dimension members for days, months, quarters, and years.

expression

A combination of one or more values (typically provided by a measure or a calculated measure), operators, and functions that evaluates to a value. An expression generally assumes the data type of its components.

The following are examples of expressions, where `SALES` is a measure: `SALES`, `SALES*1.05`, `TRUNC(SALES)`.

fact

See **measure**.

fact table

A table in a star schema that contains factual data. A fact table typically has two types of columns: those that contain facts and those that are foreign keys to dimension tables. The primary key of a fact table is usually a composite key that is made up of all of its foreign keys.

A fact table might contain either detail facts or aggregated facts. Fact tables that contain aggregated facts are typically called summary tables or materialized views. A fact table usually contains facts with the same level of aggregation.

See also **materialized view**.

hierarchy

A way to organize data at different levels of aggregation. Hierarchies are used to define data aggregation; for example, in a Time dimension, a hierarchy might be used to aggregate data from days to months to quarters to years. Hierarchies are also used to define a navigational drill path.

In a relational table, hierarchies can be defined as part of a dimension object.

See also **level-based hierarchy**, **ragged hierarchy**, **skip-level hierarchy**, **value-based hierarchy**.

hierarchy view

A relational view of a cube dimension that provides information about the members that belong to a particular hierarchy. It includes columns for the dimension keys, parents, levels of the hierarchy, and attributes.

See also **cube dimension**, **dimension view**.

key

A column or set of columns included in the definition of certain types of integrity constraints. Keys describe the relationships between the different tables and columns of a relational database.

See also **dimension member**.

leaf data

See **detail data**.

level

A named position in a hierarchy. For example, a Time dimension might have a hierarchy that represents data at the month, quarter, and year levels. The levels might be named Month, Quarter, and Year. The names provide an easy way to reference a group of dimension members at the same distance from the base.

level-based hierarchy

A hierarchy composed of levels. For example, Time is always level based with levels such as Month, Quarter, and Year. Most hierarchies are level based.

See also **value-based hierarchy**.

mapping

The definition of the relationship and data flow between source and target objects. For example, the metadata for a cube includes the mappings between each measure and the columns of a fact table or view.

materialized view

A database object that provides access to aggregate data and can be recognized by the automatic refresh and the query rewrite subsystems.

See also **cube materialized view**.

measure

Data that represents a business measure, such as sales or cost data. You can select, display, and analyze the data in a measure. The terms **measure** and **fact** are synonymous; measure is more commonly used in a multidimensional environment and fact is more commonly used in a relational environment.

Measures are dimensional objects that store data, such as Volume Sales and Dollar Sales. Measures belong to a cube.

See also **calculated measure**, **fact**, **cube**.

measure folder

A database object that organizes and label groups of measures. Users may have access to several schemas with measures named Sales or Costs, and measure folders provide a way to differentiate among them.

model

A set of inter-related equations specified using the members of a particular dimension. Line item dimensions often use models to calculate the values of dimension members.

See also **custom member**. Contrast with **calculated measure**.

NA value

A special data value that indicates that data is "not available" (NA) or null. It is the value of any cell to which a specific data value has not been assigned or for which data cannot be calculated.

See also **cell**, **sparsity**.

nonadditive

Describes a measure or fact that cannot be summarized through addition, such as Unit Price. Maximum is an example of a nonadditive aggregation method.

Contrast with **additive**.

normalize

In a relational database, the process of removing redundancy in data by separating the data into multiple tables. Contrast with **denormalized**.

OLAP

Online Analytical Processing. OLAP functionality is characterized by dynamic, dimensional analysis of historical data, which supports activities such as the following:

- Calculating across dimensions and through hierarchies
- Analyzing trends
- Drilling up and down through hierarchies
- Rotating to change the dimensional orientation

Contrast with **OLTP**.

OLAP DML

The internal data definition and manipulation language for analytic workspaces.

OLTP

Online Transaction Processing. OLTP systems are optimized for fast and reliable transaction handling. Compared to data analysis systems, most OLTP interactions involve a relatively small number of rows, but a larger group of tables.

Contrast with **OLAP**.

on the fly

Calculated at run-time in response to a specific query. In a cube, calculated measures and custom members are typically calculated on the fly. Aggregate data can be precomputed, calculated on the fly, or a combination of the two methods.

Contrast with **precompute**.

override solve specification

See **solve specification**.

page

A unit for swapping data in and out of memory.

Also called a block.

page space

A grouping of related data pages.

parent

A dimension member immediately above a particular member in a hierarchy. In a dimension hierarchy, the data value of the parent is the aggregated total of the data values of its children.

Contrast with **child**. See also **hierarchy**, **level**.

parent-child relation

A one-to-many relationship between one parent and one or more children in a hierarchical dimension. For example, New York (at the state level) might be the parent of Albany, Buffalo, Poughkeepsie, and Rochester (at the city level).

See also **child**, **parent**.

precalculate

See **precompute**.

precompute

Calculate and store as a data maintenance procedure. In a cube, aggregate data can be precomputed, calculated on the fly, or a combination of the two methods.

Contrast with **on the fly**.

ragged hierarchy

A hierarchy that contains at least one member with a different base level, creating a "ragged" base level for the hierarchy. Organization dimensions are frequently ragged.

refresh

Load new and changed values from the source tables and recompute the aggregate values.

security role

See **data security role**.

skip-level hierarchy

A hierarchy that contains at least one member whose parents are more than one level above it, creating a hole in the hierarchy. For example, in a Geography dimension with levels for City, State, and Country, Washington D.C. is a city that does not have a State value; its parent is United States at the Country level.

snowflake schema

A type of star schema in which the dimension tables are partly or fully normalized.

See also **normalize**, **star schema**.

solve specification

The aggregation method for each dimension of the cube.

solved data

A result set in which all derived data has been calculated. Data fetched from an cube is always fully solved, because all of the data in the result set is calculated before it is

returned to the SQL-based application. The result set from the cube is the same whether the data was precomputed or calculated on the fly.

See also **on the fly**, **precompute**.

source

See **data source**.

sparsity

A concept that refers to multidimensional data in which a relatively high percentage of the combinations of dimension values do not contain actual data.

There are two types of sparsity:

- Controlled sparsity occurs when a range of values of one or more dimensions has no data; for example, a new measure dimensioned by Month for which you do not have data for past months. The cells exist because you have past months in the Month dimension, but the cells are empty.
- Random sparsity occurs when nulls are scattered throughout a measure, usually because some combinations of dimension members never have any data. For example, a district might only sell certain products and never have sales data for the other products.

Some dimensions may be sparse while others are dense. For example, every time period may have at least one data value across the other dimensions, making Time a dense dimension. However, some products may not be sold in some cities, and may not be available anywhere for some time periods; both Product and Geography may be sparse dimensions.

See also **composite**.

star query

A join between a fact table and a number of dimension tables. Each dimension table is joined to the fact table using a primary key to foreign key join, but the dimension tables are not joined to each other.

star schema

A relational schema whose design represents a dimensional data model. The star schema consists of one or more fact tables and one or more dimension tables that are related through foreign keys.

See also **snowflake schema**.

status

The list of currently accessible values for a given dimension. The status of a dimension persists within a particular session, and does not change until it is changed deliberately. When an analytic workspace is first attached to a session, all members are in status.

See also **cube dimension**, **dimension member**.

summary

See **aggregation**.

update window

The length of time available for loading new data into a database.

value-based hierarchy

A hierarchy defined only by the parent-child relationships among dimension members. The dimension members at a particular distance from the base level do not form a meaningful group for analysis, so the levels are not named. For example, an employee dimension might have a parent-child relation that identifies each employee's supervisor. However, levels that group together first-, second-, and third-level supervisors and so forth may not be meaningful for analysis.

See also **hierarchy**, **level-based hierarchy**.

Index

A

B

C

D

E

F

G

H

S

T

U

V

W

X