Oracle® OLAP

Application Developer's Guide

10*g* Release 1 (10.1)

Part No. B10333-02

December 2003

ORACLE®

Oracle OLAP Application Developer's Guide, 10*g* Release 1 (10.1)

Part No. B10333-02

Contents

4 Developing Java Applications for OLAP

6 Creating an Analytic Workspace

7 SQL Access to Analytic Workspaces

8 Exploring a Standard Form Analytic Workspace

10 Predicting Future Performance

11 Acquiring Data From Other Sources

13 Materialized Views for the OLAP API

A Database Standard Form for Analytic Workspaces

Send Us Your Comments

Oracle OLAP Application Developer's Guide, 10*g* Release 1 (10.1)

Part No. B10333-02

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the title and part number of the documentation and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: infodev_us@oracle.com
- FAX: 781-238-9850 Attn: Oracle OLAP
- Postal service:
  Oracle Corporation
  Oracle OLAP Documentation
  10 Van de Graaff Drive
  Burlington, MA 01803
  U.S.A.

If you would like a reply, please give your name, address, telephone number, and electronic mail address (optional).

If you have problems with the software, please contact your local Oracle Support Services.

Preface

The *Oracle OLAP Application Developer's Guide* explains how SQL and Java applications can extend their analytic processing capabilities by using the OLAP option in the Enterprise edition of the Oracle Database.

The preface contains these topics:

Intended Audience

This manual is intended for applications developers. To use this manual, you should know SQL and have a general familiarity with the Oracle tools available to SQL developers and database administrators.

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. 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 additional information, visit the Oracle Accessibility Program Web site at

http://www.oracle.com/accessibility/

Accessibility of Code Examples in Documentation JAWS, a Windows screen reader, 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, JAWS 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.

Structure

The following paragraphs describe the chapters that comprise this manual.

Part I, "Fundamentals"

Part I introduces the basic concepts, tools, and capabilities of the OLAP option.

Chapter 1, "Overview"

This chapter introduces the powerful analytic resources available in an Oracle Database installed with the OLAP option.

Chapter 2, "The Multidimensional Data Model"

This chapter describes the multidimensional data model and how it is implemented in relational tables, analytic workspaces, and the OLAP Catalog.

Chapter 3, "The Sample Schema"

This chapter describes the `GLOBAL` sample schema that is used for the examples in this manual.

Chapter 4, "Developing Java Applications for OLAP"

This chapter presents the rich development environment and the powerful tools that you can use to create OLAP applications.

Part II, "Fundamentals of Creating and Using Analytic Workspaces"

Part II provides instructions for creating standard form analytic workspaces from a relational schema using a set of graphical tools.

Chapter 5, "Defining a Logical Multidimensional Model"

This chapter describes the OLAP Catalog and the methods for working with OLAP metadata so that you can describe your data as logical multidimensional objects.

Chapter 6, "Creating an Analytic Workspace"

This chapter explains how to create a standard form analytic workspace by using the wizards in Analytic Workspace Manager.

Chapter 7, "SQL Access to Analytic Workspaces"

This chapter introduces various SQL packages that provide access to the data in an analytic workspace.

Chapter 8, "Exploring a Standard Form Analytic Workspace"

This chapter describes the objects created in a standard form analytic workspace. It serves as a guide to your own analytic workspace, and you can examine the property sheets of the objects described here by opening the Object View in Analytic Workspace Manager.

Part III, "Acquiring Data From Additional Sources"

Part III describes ways that you can create a new analytic workspace or enhance an existing one with data from sources other than a star or snowflake schema.

Chapter 9, "Adding Measures to a Standard Form Analytic Workspace"

This chapter explains how to add calculated measures as a permanent addition to a standard form analytic workspace.

Chapter 10, "Predicting Future Performance"

This chapter explains how to forecast future results based on past performance and make this information available in a new standard form cube.

Chapter 11, "Acquiring Data From Other Sources"

This chapter introduces the data acquisition facilities in the OLAP DML, which you can use to create a standard form analytic workspace, or add data to an existing workspace, from sources other than a star or snowflake schema.

Part IV, "Database Administration for OLAP"

Part IV explains how to perform administrative tasks associated with the OLAP option.

Chapter 12, "Administering Oracle OLAP"

This chapter identifies the administrative tasks associated with the OLAP option and provides performance tips.

Chapter 13, "Materialized Views for the OLAP API"

This chapter explains how to create materialized views that can be used by the Business Intelligence Beans when relational tables (instead of an analytic workspace) are used to store the data.

Appendix A, "Database Standard Form for Analytic Workspaces"

This appendix specifies the rules for a database standard form analytic workspace.

Appendix B, "Upgrading From Express Server"

This appendix provides upgrade instructions and identifies some of the major differences between Oracle Express Server 6.3 and Oracle OLAP.

Appendix C, "Programs Used to Create GLOBALX"

This appendix provides additional source code used to create the example in

Glossary

The glossary contains definitions of terms that are specific to OLAP.

Related Documents

For more information, see the following manuals in the Oracle Database 10*g* documentation set:

- *Oracle OLAP Application Developer's Guide*

  Explains how SQL and Java applications can extend their analytic processing capabilities by using Oracle OLAP in the Enterprise Edition of Oracle Database.

- *Oracle OLAP Reference*

  Explains the syntax of PL/SQL packages and types and the column structure of views related to Oracle OLAP.

- *Oracle OLAP DML Reference*

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

- *Oracle OLAP Developer's Guide to the OLAP API*

  Introduces the Oracle OLAP API, a Java application programming interface for Oracle OLAP, which is used to perform online analytical processing of the data stored in an Oracle database. Describes the API and how to discover metadata, create queries, and retrieve data.

- *Oracle OLAP Java API Reference*

  Describes the classes and methods in the Oracle OLAP Java API for querying analytic workspaces and relational data warehouses.

- *Oracle OLAP Analytic Workspace Java API Reference*

  Describes the classes and methods in the Oracle OLAP Java API for building and maintaining analytic workspaces.

Conventions

The following conventions are also used in this manual:

Convention	Meaning
. . .	Vertical ellipsis points in an example mean that information not directly related to the example has been omitted.
. . .	Horizontal ellipsis points in statements or commands mean that parts of the statement or command not directly related to the example have been omitted
boldface text	Boldface type in text indicates a term defined in the text, the glossary, or in both locations.
< >	Angle brackets enclose user-supplied names.
[ ]	Brackets enclose optional clauses from which you can choose one or none.
$	The dollar sign represents the DIGITAL Command Language prompt in Windows and the Bourne shell prompt in Digital UNIX

Part I

Fundamentals

Part I introduces basic concepts, tools, and capabilities of the OLAP option. By reading the chapters in this part, you will learn how the OLAP option works within the Oracle Database, how the multidimensional data model is implemented, and how applications can use multidimensional data to improve their analytic capabilities.

Part I contains the following chapters:

1

Overview

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

OLAP Technology Within the Oracle Database

Multidimensional technology is now available within the Oracle Database. Organizations no longer need to choose between a multidimensional OLAP database and a relational database. By integrating multidimensional tables and an analytic engine into the database, Oracle provides the power of multidimensional analysis along with the manageability, scalability, and reliability of the Oracle Database.

Problems Maintaining Two Distinct Systems

The integration of multidimensional technology in a relational database is important because maintaining a standalone multidimensional database is costly. It requires additional hardware and DBAs who are skilled at using the specialized administrative tools of the multidimensional database. Moreover, standalone multidimensional databases require applications that use proprietary APIs. This severely limits the number of applications that can be run against them, not only

because fewer applications are available in these APIs, but because all the data that they run on must be transferred from the relational database to the multidimensional database. These requirements often force enterprises into supporting two sets of query and reporting tools, one for the relational database and the other for the multidimensional database.

Full Integration of Multidimensional Technology

In contrast, the OLAP option is fully integrated into the Oracle Database. DBAs use the same tools to administer this option as they use to administer all other components of the database. The DBA can decide the best location for storing and calculating the data as part of optimizing the operations of the database. A single application can access both relational and multidimensional data.

SQL-based applications can now use pure SQL against information-rich relational views of multidimensional data provided by an OLAP-enabled Oracle Database. OLAP calculations can be queried using SQL, enabling application developers to leverage their investment in SQL while expanding the analytic sophistication of their software to include modeling, forecasting, and what-if analysis. Standard reporting applications can present the results of complex multidimensional calculations, while ad-hoc querying tools such as custom aggregate members and custom measures can expand the analyst's range of calculation functions.

Using OLAP to Answer Business Questions

Relational databases provide the online transactional processing (OLTP) that is essential for businesses to keep track of their affairs. Designed for efficient selection, storage, and retrieval of data, relational databases are ideal for housing gigabytes of detailed data.

The success of relational databases is apparent in their use to store information about an increasingly wide scope of activities. As a result, they contain a wealth of data that can yield critical information about a business. This information can provide a significant edge in an increasingly competitive marketplace.

The challenge is in deriving answers to business questions from the available data, so that decision makers at all levels can respond quickly to changes in the business climate.

A standard transactional query might ask, "When did order 84305 ship?" This query reflects the basic mechanics of doing business. It involves simple data selection and retrieval of one record (or, at most, several related records) identified by a unique order number. Any follow-up questions, such as which postal carrier was used and

where was the order shipped to, can probably be answered by the same record. This record has a useful life span in the transactional world: it begins when a customer places the order and ends when the order is shipped and paid for. At this point, the record can be rolled off to an archive.

In contrast, a typical series of analytical queries might ask, "How do sales in the Pacific Rim for this quarter compare with sales a year ago? What can we predict for sales next quarter? What factors can we alter to improve the sales forecast? What happens if I change this number?"

These are not questions about doing business transactions, but about analyzing past performance and making decisions that will improve future performance, provide a more competitive edge, and thus enhance profitability. The analytic database is a "crystal ball" for decision makers whose ability to make sound decisions today is dependent on how well they can predict the future. Getting the answers to these questions involves single-row calculations, time series analysis, and access to aggregated historical and current data. This requires OLAP -- online analytical processing.

Common Analytical Applications

Here are a few examples of common applications that can use the OLAP option to realize valuable gains in functionality and performance:

- Planning applications enable organizations to predict outcomes. They generate new data using predictive analytical tools such as models, forecasts, aggregation, allocation, and scenario management. Some examples of this type of application are corporate budgeting and financial analyses, and demand planning systems.
- Budgeting and financial analysis systems enable organizations to analyze past performance, build revenue and spending plans, manage to attain profit goals, and model the effects of change on the financial plan. Management can determine spending and investment levels that are appropriate for the anticipated revenue and profit levels. Financial analysts can prepare alternative budgets and investment plans contingent on factors such as fluctuations in currency values.
- Demand planning systems enable organizations to predict market demand based on factors such as sales history, promotional plans, and pricing models. They can model different scenarios that forecast product demand and then determine appropriate manufacturing goals.

As this discussion highlights, the data processing required to answer analytical questions is fundamentally different from the data processing required to answer transactional questions. The users are different, their goals are different, their queries are different, and the type of data that they need is different. A relational data warehouse enhanced with the OLAP option provides the best environment for data analysis.

Deciding When to Use Analytic Workspaces

The types of analyses performed by applications that run against your data warehouse will help you decide whether to store the data entirely in analytic workspaces or distributed between analytic workspaces and relational tables.

Analytic workspaces provide an alternative to materialized views for generating and storing aggregate data. They provide complex aggregation methods that are not available in materialized views, such as weighted calculations, non-additive methods, and models. You might also choose analytic workspaces when you have storage issues concerning aggregate data. Analytic workspaces always present fully solved data to the application, regardless of whether the data is entirely pre-aggregated, partially pre-aggregated, or entirely aggregated on demand. The flexibility of the OLAP aggregation system enables you to pre-aggregate within the limitations of your data refresh window without compromising run-time response time. Moreover, analytic workspaces can store pre-aggregated data very efficiently.

You may also prefer to use analytic workspaces for applications that support predictive analysis functions, such as models, forecasts, and what-if scenarios. Moreover, analytic workspaces are highly optimized for performing single-row calculations, which they can compute at run-time to support custom measures.

A distributed solution may be optimal for query and reporting applications that use the advanced calculation capabilities of analytic workspaces less frequently. For these types of applications, you can create and populate analytic workspaces at run-time for more intensive analysis; the results can be sent directly to the analyst or written to relational tables. The implementation of a distributed model can, of course, vary widely since it encompasses solutions that range from storing all data in relational tables to storing all data in analytic workspaces.

The BI Beans can run against analytic workspaces or relational tables. If you do not plan to use analytic workspaces, then refer to the information in "Managing Data Sources for the BI Beans and OLAP API" on page 4-9.

Working With Oracle OLAP

There are several levels at which you can work with analytic workspaces:

- Graphical user interfaces (GUIs) provide wizards and property sheets for performing the basic tasks for creating and managing analytic workspaces. This topmost level formulates calls to the underlying SQL packages. Your introduction to developing and maintaining analytic workspaces is learning to use these GUIs.
- SQL packages perform all the tasks needed to create, maintain, and expose analytic workspaces for use by applications. Some SQL packages work directly with workspaces and execute the underlying OLAP DML. Other SQL packages work with relational tables and views, and execute SQL.
- Java packages can build and query analytic workspaces.
- OLAP DML is the native language of analytic workspaces and implements all operations initiated at the other levels.

Installation of the OLAP option with the Oracle Database includes the following components:

OLAP Analytic Engine
Analytic Workspaces
Analytic Workspace Manager
OLAP Worksheet
SQL Interface to OLAP
OLAP DML
OLAP Catalog
Analytic Workspace Java APIs
OLAP API

The following applications can provide important functionality when working in OLAP, and are available online at the Oracle Web site:

Oracle Warehouse Builder
Oracle Enterprise Manager

All of these components and applications are described in the following paragraphs. The relationships among them are described throughout this guide.

OLAP Analytic Engine

The OLAP analytic engine supports the selection and rapid calculation of multidimensional data. The status of an individual session persists to support a series of queries, which is typical of analytical applications; the output from one query is easily used as input to the next query. A comprehensive set of data manipulation tools supports modeling, aggregation, allocation, forecasting, and what-if analysis. The OLAP engine runs within the Oracle kernel.

Analytic Workspaces

Analytic workspaces store data in a multidimensional format where it can be manipulated by the OLAP engine. An analytic workspace is stored as a LOB table in a relational schema. Within a single database, many analytic workspaces can be created and shared among users. Like a relational schema, an analytic workspace is owned by a particular user ID, and other users can be granted access to it. Because individual users can save a personal copy of their alterations to a workspace, the workspace environment is particularly conducive to planning applications.

Analytic Workspace Manager

Analytic Workspace Manager provides a user interface for creating an analytic workspace in database standard form. This form enables the analytic workspace to be used with various tools that aggregate, refresh, and enable the data so that it is accessible to OLAP applications. These tools are also provided by Analytic Workspace Manager.

For more information about Analytic Workspace Manager, refer to Chapter 6, "Creating an Analytic Workspace".

OLAP Worksheet

OLAP Worksheet is an interactive environment for working with analytic workspaces, similar to SQL*Plus Worksheet. It provides easy access to the OLAP DML, and enables you to perform sophisticated business analysis, such as modeling, forecasting, and allocation. You can switch between two different modes, one for working with analytic workspaces in the OLAP DML, and the other for working with relational tables and views in SQL. It is available through Analytic Workspace Manager or as a separate executable.

For more information about OLAP Worksheet, refer to Chapter 9.

SQL Interface to OLAP

The SQL interface to OLAP provides access to analytic workspaces from SQL. The SQL interface is implemented in PL/SQL packages. These are the primary ones:

- `CWM2` is a large collection of packages for defining OLAP Catalog metadata. These packages support the BI Beans enabler in Analytic Workspace Manager.
- `DBMS_AW` contains procedures for executing OLAP DML commands. This package supports OLAP Worksheet, and the property sheets and dialogs in Analytic Workspace Manager. Using the procedures and functions in the `DBMS_AW` package, SQL programmers can issue OLAP DML commands directly against analytic workspace data. They can move data from relational tables into an analytic workspace, perform advanced analysis of the data (for example, forecasting), and copy the results of that analysis into relational tables.
- `DBMS_AWM` contains procedures for creating analytic workspaces. It supports the Create Analytic Workspace wizard in Analytic Workspace Manager.
- `DBMS_AW_UTILITIES` contains procedures for creating and managing custom measures in a standard form analytic workspace. Custom measures are defined at run-time, and are calculated from stored measures.

For more information about these PL/SQL packages, refer to Chapter 7, "SQL Access to Analytic Workspaces" and the *Oracle OLAP Reference*.

OLAP DML

OLAP DML is a mature low-level language that is native to analytic workspaces. It is the data definition and manipulation language for creating analytic workspaces, defining data containers, and manipulating the data stored in these containers. All other levels of operation (GUIs, Java, and SQL) resolve to the OLAP DML. It offers the maximum power and flexibility in acquiring, manipulating, and analyzing data.

If you are upgrading from Oracle Express, or if your data is stored in formats not supported by the higher level tools, then you may work directly in the OLAP DML at an early stage. Otherwise, you may use the OLAP DML directly only to enhance the functionality of your workspaces.

OLAP Catalog

OLAP Catalog is the metadata repository provided for the OLAP option. It consists of write APIs, which are a set of PL/SQL procedures, and read APIs, which are relational views within the Oracle Database. The metadata describes data, which is presented as a star schema, in multidimensional terms such as cubes, measures,

dimensions, and attributes. The OLAP Catalog is used to perform two distinct functions:

- To create an analytic workspace from a star or snowflake schema.
- To provide a Java application, which uses the BI Beans, with access to data stored in either an analytic workspace or relational tables. The BI Beans requires OLAP Catalog metadata. If data is not defined in the OLAP Catalog, then it is not available to applications that use the BI Beans.

The OLAP Catalog read APIs make the metadata that you have defined available to applications. They are useful to any application that uses SQL `SELECT` statements to run against views of analytic workspace data.

SQL applications do not require the use of the OLAP Catalog, but may benefit from using it. They can run against the logical objects that are defined in the OLAP catalog, without an awareness of where the underlying data resides.

Analytic Workspace Java APIs

The Analytic Workspace Java APIs provide a Java interface for the creation and maintenance of analytic workspaces. These APIs are an alternative to using the OLAP Catalog for defining an analytic workspace build.

See Also: *Oracle OLAP Analytic Workspace Java API Reference*

OLAP API

The OLAP API is the Java-based programming interface for OLAP applications, and supports the BI Beans. The BI Beans are building blocks for developing analytic applications in Java, and are available for use with JDeveloper. If you are a Java developer, then you should consider using the BI Beans for your analytic applications. Note that the BI Beans are not included with the OLAP option, but they require an OLAP-enabled Oracle Database.

Oracle Enterprise Manager

Oracle Enterprise Manager is a system management tool that provides you with an integrated solution for managing Oracle products without formulating complex SQL commands. You can use Enterprise Manager to set up user accounts, define tablespaces, monitor performance, and do other administrative tasks associated with your database, including the OLAP option.

The OLAP Management tool is part of the Enterprise Manager support for data warehouses. Using a graphical user interface, you can define logical metadata dimensions, measures, and cubes in the OLAP Catalog for the dimension tables and fact tables of a star or snowflake schema that complies with the database requirements for creating a dimension.

For more information about the OLAP Management tool, refer to Chapter 5, "Defining a Logical Multidimensional Model".

Oracle Warehouse Builder

Oracle Warehouse Builder can extract data from many different sources, transform it into a star schema in the relational database, generate OLAP Catalog metadata, and create an analytic workspace. Warehouse Builder provides an alternative to using the OLAP Management tool in Enterprise Manager, and the Create Analytic Workspace wizard in Analytic Workspace Manager. The resulting analytic workspace is in database standard form, so you can then use Analytic Workspace Manager to aggregate, enhance, and enable your data.

If your data requires transformation, then Oracle Warehouse Builder provides the best method for generating an analytic workspace. Once you have created a logical model for your data warehouse, Oracle Warehouse Builder requires only a few extra steps to generate an analytic workspace in addition to a star schema.

See Also: *Oracle Warehouse Builder User's Guide*

Process Overview: Creating and Maintaining Analytic Workspaces

Analytic workspaces can be created in a variety of ways, depending on the characteristics of the data source and your own personal preference. However, the basic process is the same for all of them.

These are the basic stages:

1. Define a logical multidimensional model in the metadata, and map the logical objects to physical data sources. See Chapter 5.
2. Create and populate an analytic workspace. See Chapter 6.
3. Generate information-rich data using aggregation, allocation, modeling, forecasting, and other analytic methods. See Chapter 6, Chapter 9, and Chapter 10.
4. Generate relational views of the analytic workspace. See Chapter 6.

5. Define metadata specifically for use by particular applications. See Chapter 6.
6. Periodically refresh the analytic workspace with new data. See Chapter 6.
7. Calculate custom measures and dimension members. See Chapter 7 and Chapter 9.

Table 1–1 identifies the tools available for performing each stage. Using these tools to perform the various stages of creating and managing analytic workspaces is the topic of this guide.

Table 1–1 Tools for Working With Analytic Workspaces

Stage	Tools
Design a logical model and map it to data sources	Oracle Enterprise Manager CWM2 Write APIs Oracle Warehouse Builder Analytic Workspace Java APIs
Create and populate an analytic workspace	Analytic Workspace Manager wizards DBMS_AWM PL/SQL package Oracle Warehouse Builder Analytic Workspace Java APIs
Generate information	Analytic Workspace Manager wizards DBMS_AWM PL/SQL package Analytic Workspace Java APIs OLAP DML
Create views	Analytic Workspace Manager enablers OLAP_TABLE function
Generate metadata for views	Analytic Workspace Manager enablers CWM2 Write APIs
Generate custom measures	DBMS_AW_UTILITIES PL/SQL package DBMS_AW PL/SQL package OLAP_TABLE function Analytic Workspace Java APIs OLAP DML

Table 1–1 (Cont.) Tools for Working With Analytic Workspaces

Stage	Tools
Refresh the data	Analytic Workspace Manager wizards
	`DBMS_AWM` PL/SQL package
	Analytic Workspace Java APIs
	OLAP DML

2

The Multidimensional Data Model

This chapter describes the multidimensional data model and how it is implemented in relational tables and standard form analytic workspaces. It consists of the following topics:

The Logical Multidimensional Data Model

The multidimensional data model is an integral part of On-Line Analytical Processing, or 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 multidimensional data model is designed to solve complex queries in real time.

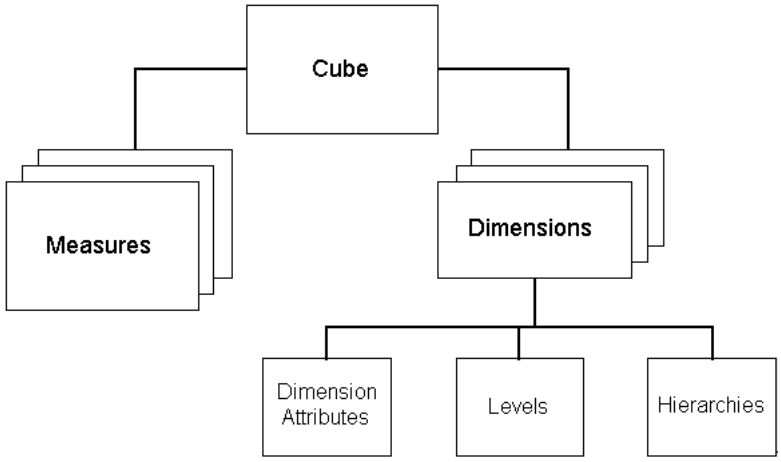
The multidimensional data model is important because it enforces simplicity. As Ralph Kimball states in his landmark book, *The Data Warehouse Toolkit*:

"The central attraction of the dimensional model of a business is its simplicity.... that simplicity is the fundamental key that allows users to understand databases, and allows software to navigate databases efficiently."

The multidimensional data model is composed of logical cubes, measures, dimensions, hierarchies, levels, and attributes. 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 2–1 shows the relationships among the logical objects.

Figure 2–1 Diagram of the Logical Multidimensional Model



Logical Cubes

Logical 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 have the same relationships to other logical objects and can easily be analyzed and displayed together.

Logical Measures

Measures populate the cells of a logical 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 legacy system, transactional database, 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. Many applications 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 applications 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 (sometimes called the grain). Users may never view this **base level 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, 2002, at 2:34 p.m. But they might want to find out which color of dress was most popular in the summer of 2002 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 pre-aggregated before being loaded into a data warehouse.

Logical Dimensions

Dimensions contain a set of unique values that identify and categorize data. They form the edges of a logical 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-01), a customer (Warren Systems), a product (Portable PCs), and a channel (Catalog).

Logical 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.

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, `Q1-02` and `Q2-02` are the children of `2002`, thus `2002` is the parent of `Q1-02` and `Q2-02`.

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.

Logical 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 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 2002? 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.

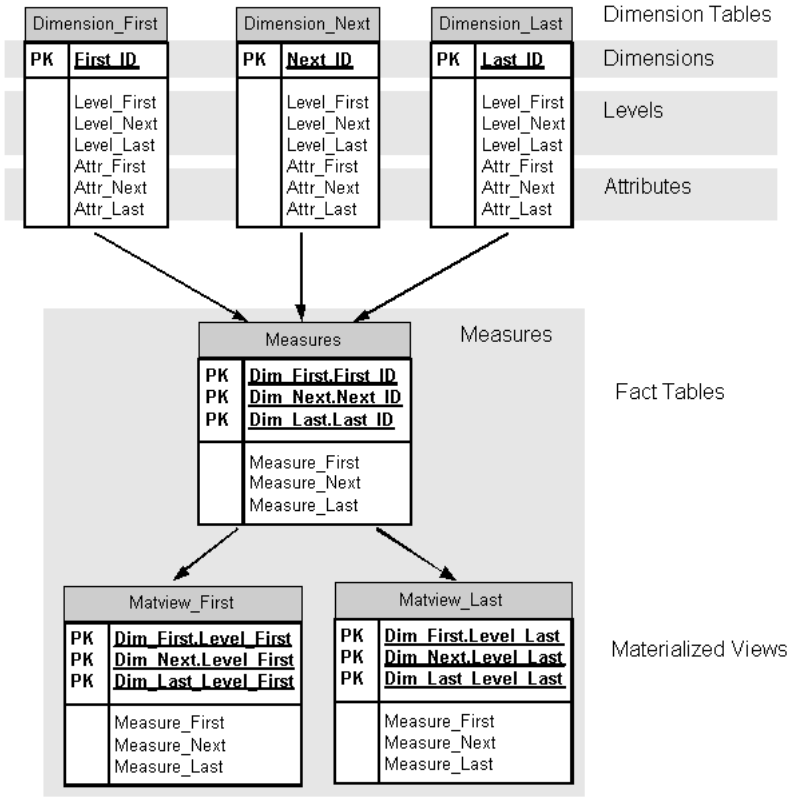
The Relational Implementation of the Model

The relational implementation of the multidimensional data model is typically a **star schema**, as shown in Figure 2–2, or a **snowflake schema**. A star schema is a convention for organizing the data into dimension tables, fact tables, and materialized views. Ultimately, all of the data is stored in columns, and metadata is required to identify the columns that function as multidimensional objects.

In Oracle Database, you can define a logical multidimensional model for relational tables using the OLAP Catalog or `AWXML`, as described in Chapter 5. The metadata distinguishes level columns from attribute columns in the dimension tables and

specifies the hierarchical relationships among the levels. It identifies the various measures that are stored in columns of the fact tables and aggregation methods for the measures. And it provides display names for all of these logical objects.

Figure 2–2 Diagram of a Star Schema



Dimension Tables

A star schema stores all of the information about a dimension in a single table. Each level of a hierarchy is represented by a column or column set in the dimension table. A dimension object can be used to define the hierarchical relationship between two columns (or column sets) that represent two levels of a hierarchy; without a dimension object, the hierarchical relationships are defined only in metadata. Attributes are stored in columns of the dimension tables.

A snowflake schema normalizes the dimension members by storing each level in a separate table.

Fact Tables

Measures are stored in fact tables. Fact tables contain a composite primary key, which is composed of several foreign keys (one for each dimension table) and a column for each measure that uses these dimensions.

Materialized Views

Aggregate data is calculated on the basis of the hierarchical relationships defined in the dimension tables. These aggregates are stored in separate tables, called summary tables or materialized views. Oracle provides extensive support for materialized views, including automatic refresh and query rewrite.

Queries can be written either against a fact table or against a materialized view. If a query is written against the fact table that requires aggregate data for its result set, the query is either redirected by query rewrite to an existing materialized view, or the data is aggregated on the fly.

Each materialized view is specific to a particular combination of levels; in Figure 2–2, only two materialized views are shown of a possible 27 (3 dimensions with 3 levels have 3**3 possible level combinations).

The Analytic Workspace Implementation of the Model

Analytic workspaces have several different types of data containers, such as dimensions, variables, and relations. Each type of container can be used in a variety of ways to store different types of information. For example, a **dimension** can define an edge of a measure, or store the names of all the languages supported by the analytic workspace, or all of the acceptable values of a **relation**. Dimension objects are themselves one dimensional lists of values, while variables and relations are designed specifically to support the efficient storage, retrieval, and manipulation of multidimensional data.

Note: Analytic workspaces are registered in the database data dictionary as tables. However, objects stored in analytic workspaces are not registered in the database data dictionary.

Like relational tables, analytic workspaces have no specific content requirements. You can create an empty analytic workspace, populate it only with OLAP DML programs, or define a single dimension to hold a list of values. This guide, however, describes analytic workspaces that comply with **database standard form**. Database standard form (or simply, standard form) is a convention for instantiating the logical multidimensional model in a particular way so that it can be managed by the current set of Oracle OLAP utilities. It defines a set of metadata that can be queried by any application. Standard form is discussed extensively in this guide, and is described in Appendix A.

Multidimensional Data Storage in Analytic Workspaces

In the logical multidimensional model, a cube represents all measures with the same shape, that is, the exact same dimensions. In a cube shape, each edge represents a dimension. The dimension members are aligned on the edges and divide the cube shape into cells in which data values are stored.

In an analytic workspace, the cube shape also represents the physical storage of multidimensional measures, in contrast with two-dimensional relational tables. An advantage of the cube shape is that it can be rotated: there is no one right way to manipulate or view the data. This is an important part of multidimensional data storage, calculation, and display, because different analysts need to view the data in different ways. For example, if you are the Sales Manager for the Pacific Rim, then you need to look at the data differently from a product manager or a financial analyst.

Assume that a company collects data on sales. The company maintains records that quantify how many of each product was sold in a particular sales region during a specific time period. You can visualize the sales measure as the cube shown in Figure 2–3.

Figure 2–3 Comparison of Product Sales By City

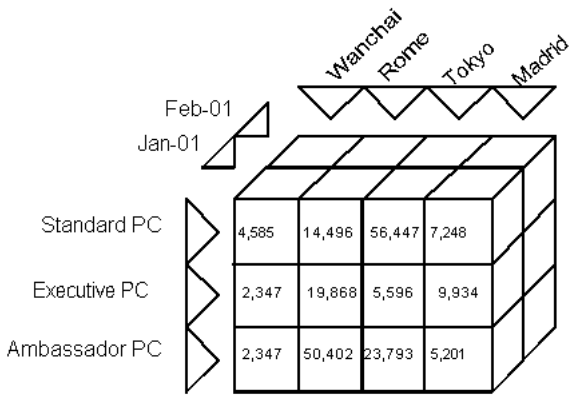
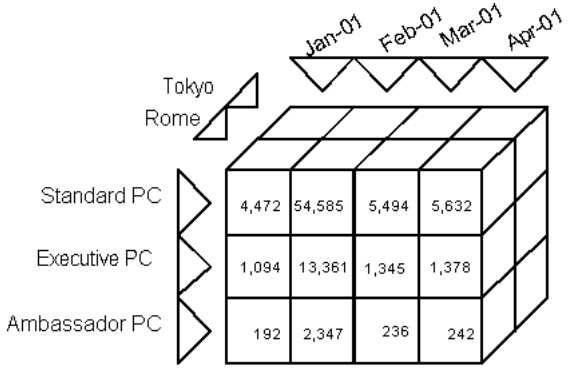


Figure 2–3 compares the sales of various products in different cities for January 2001 (shown) and February 2001 (not shown). This view of the data might be used to identify products that are performing poorly in certain markets. Figure 2–4 shows sales of various products during a four-month period in Rome (shown) and Tokyo (not shown). This view of the data is the basis for trend analysis.

Figure 2–4 Comparison of Product Sales By Month

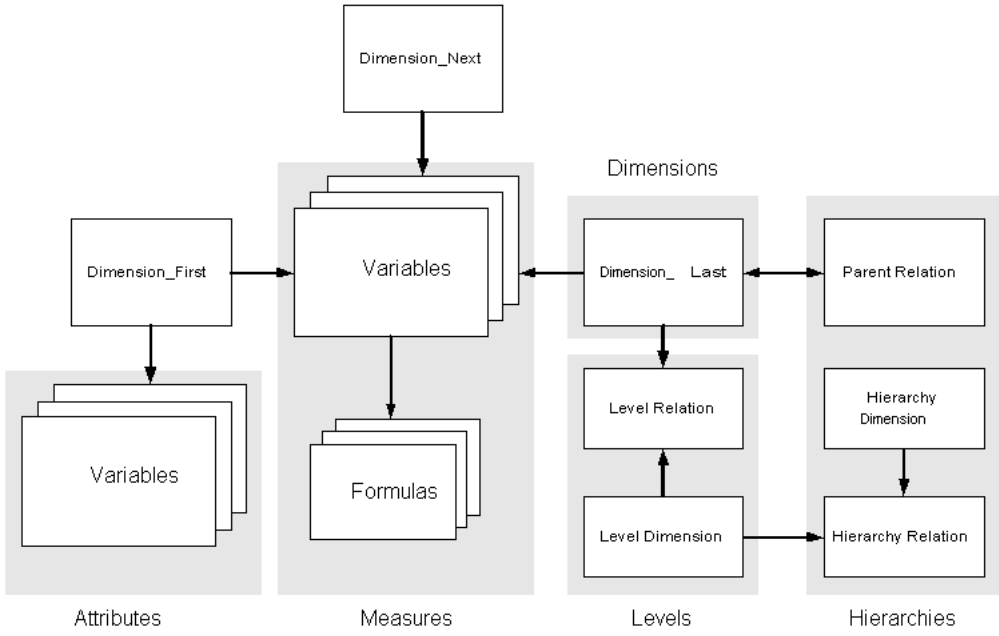


A cube shape is three dimensional. Of course, measures can have many more than three dimensions, but three dimensions are the maximum number that can be represented pictorially. Additional dimensions are pictured with additional cube shapes.

Database Standard Form Analytic Workspaces

Figure 2–5 shows how dimension, variable, formula, and relation objects in a standard form analytic workspace are used to implement the multidimensional model. Measures with identical dimensions compose a logical cube. All dimensions have attributes, and all hierarchical dimensions have level relations and self-relations; for clarity, these objects are shown only once in the diagram. Variables and formulas can have any number of dimensions; three are shown here.

Figure 2–5 Diagram of a Standard Form Analytic Workspace



Analytic Workspace Dimensions

A dimension in an analytic workspace is a highly optimized, one-dimensional index of values that serves as a key table. Variables, relations, formulas (which are stored equations) are among the objects that can have dimensions.

Dimensions have several intrinsic characteristics that are important for data analysis:

- Referential integrity. Each dimension member is unique and cannot be `NA` (that is, null). If a measure has three dimensions, then each data value of that measure must be qualified by a member of each dimension. Likewise, each combination of dimension members has a value, even if it is `NA`.
- Consistency. Dimensions are maintained as separate containers and are shared by measures. Measures with the same dimensionality can be manipulated together easily. For example, if the sales and expense measures are dimensioned by time and line, then you can create equations such as profit = sales - expense.
- Preserved order of members. Each dimension has a default **status**, which is a list of all of its members in the order they are stored. The default status list is always the same unless it is purposefully altered by adding, deleting, or moving members. Within a session, a user can change the selection and order of the status list; this is called the current status list. The current status list remains the same until the user purposefully alters it by adding, removing, or changing the order of its members.

  Because the order of dimension members is consistent and known, the selection of members can be relative. For example, this function call compares the sales values of all currently selected time periods in the current status list against sales from the prior period.

  ```
  lagdif(sales, 1, time)
  ```

- Highly denormalized. A dimension typically contains members at all levels of all hierarchies. This type of dimension is sometimes called an **embedded total** dimension.

In addition to simple dimensions, there are several special types of dimensions used in a standard form analytic workspace, such as composites and concat dimensions. These dimensions are discussed later in this guide.

Use of Dimensions in Standard Form Analytic Workspaces

In an analytic workspace, data dimensions are structured hierarchically so that data at different levels can be manipulated for aggregation, allocation, and navigation.

However, all dimension members at all levels for all hierarchies are stored in a single data dimension container. For example, months, quarters, and years are all stored in a single dimension for Time. The hierarchical relationships among dimension members are defined by a parent relation, described in "Analytic Workspace Relations" on page 2-13.

Not all data is hierarchical in nature, however, and you can create data dimensions that do not have levels. A line item dimension is one such dimension, and the relationships among its members require a model rather than a multilevel hierarchy. The extensive data modeling subsystem available in analytic workspaces enables you to create both simple and complex models, which can be solved alone or in conjunction with aggregation methods.

As a one-dimensional index, a dimension container has many uses in an analytic workspace in addition to dimensioning measures. A standard form analytic workspace uses dimensions to store various types of metadata, such as lists of hierarchies, levels, and the dimensions composing a logical cube.

Analytic Workspace Variables

A **variable** is a data value table, that is, an array with a particular data type and indexed by a specific list of dimensions. The dimensions themselves are not stored with the variable.

Each combination of dimension members defines a data cell, regardless of whether a value exists for that cell or not. Thus, the absence of data can be purposefully included or excluded from the analysis. For example, if a particular product was not available before a certain date, then the analysis may exclude null values (called `NA`s) in the prior periods. However, if the product was available but did not sell in some markets, then the analysis may include the `NA`s.

No special physical relationship exists among variables that share the same dimensions. However, a logical relationship exists because, even though they store different data that may be a different data type, they are identical containers. Variables that have identical dimensions compose a logical cube.

If you change a dimension, such as adding new time periods to the Time dimension, then all variables dimensioned by Time are automatically changed to include these new time periods, even if the other variables have no data for them. Variables that share dimensions (and thus are contained by the same logical cube) can also be manipulated together in a variety of ways, such as aggregation, allocation, modeling, and numeric calculations. This type of calculation is easy and fast in an analytic workspace, while the equivalent single-row calculation in a relational schema can be quite difficult.

Use of Variables to Store Measures

In an analytic workspace, facts are stored in variables, typically with a numeric data type. Each type of data is stored in its own variable, so that while sales data and expenses data might have the same dimensions and the same data type, they are stored in two distinct variables. The containers are identical, but the contents are unique.

An analytic workspace provides a valuable alternative to materialized views for creating, storing, and maintaining summary data. A very sophisticated aggregation system supports modeling in addition to an extensive number of aggregation methods. Moreover, finely grained aggregation rules enable you to decide precisely which data within a single measure is pre-aggregated, and which data within the same measure will be calculated at run-time.

Pre-aggregated data is stored in a compact format in the same container as the base-level data, and the performance impact of aggregating data on the fly is negligible when the aggregation rules have been defined according to known good methods. If aggregate data needed for the result set is stored in the variable, then it is simply retrieved. If the aggregate data does not exist, then it is calculated on the fly.

Use of Variables to Store Attributes

Like measures, attributes are stored in variables. However, there are significant differences between attributes and measures. While attributes are often multidimensional, only one dimension is a data dimension. A hierarchy dimension, which lists the data dimension hierarchies, and a language dimension, which provides support for multiple languages, are typical of the other dimensions.

Attributes provide supplementary information about each dimension member, regardless of its level in a dimension hierarchy. For example, a Time dimension might have three attribute variables, one for descriptive names, another for the period end dates, and a third for period time spans. These attributes provide Time member `OCT-02` with a descriptive name of `October 2002`, an end date of `31-OCT-02`, and a time span of `31`. All of the other days, months, quarters, and years in the Time dimension have similar information stored in these three attribute variables.

Analytic Workspace Formulas

A formula is a stored equation. A call to any function in the OLAP DML or to any custom program can be stored in a formula. In this way, a formula in an analytic workspace is like a relational view.

In a standard form analytic workspace, one of the uses of a formula object is to provide the interface between aggregation rules and a variable that holds the data. The name of a measure is always the name of a formula, not the underlying variable. While the variable only contains stored data (the base-level data and precalculated aggregates), the formula returns a fully solved measure containing data that is both stored and calculated on the fly. This method enables all queries against a particular measure to be written against the same column of the same relational view for the analytic workspace, regardless of whether the data is calculated or simply retrieved. That column presents data acquired from the formula.

Formulas can also be used to calculated other results like ratios, differences, moving totals, and averages on the fly.

Analytic Workspace Relations

Relations are identical to variables except in their data type. A variable has a general data type, such as `DECIMAL` or `TEXT`, while a relation has a dimension as its data type. For example, a relation with a data type of `PRODUCT` only accepts values that are members of the `PRODUCT` dimension; an attempt to store any other value causes an error. The dimension members provide a finite list of acceptable values for a relation. A relation container is like a foreign key column, except that it can be multidimensional.

In a standard form analytic workspace, two relations support the hierarchical content of the data dimensions: a parent relation and a level relation.

A **parent relation** is a self-relation that identifies the parent of each member of a dimension hierarchy. This relation defines the hierarchical structure of the dimension.

A **level relation** identifies the level of each member of a dimension hierarchy. It is used to select and sort dimension members based on level.

3

The Sample Schema

This guide uses the Global schema for its examples. This chapter describes this schema and explains how it will be mapped to multidimensional objects. It consists of the following topics:

- Case Study Scenario
- The Global Star Schema
- Mapping the Global Schema to an Analytic Workspace

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 weekly and monthly 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?*

What accounts are most profitable?

This business analysis question consists of the following questions:

- What 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
Margin, percent of total product sales
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 Logical Data Model for Global Computing

"Business Goals" on page 3-3 identifies the business facts that will support analysis requirements at Global Computing. Next, we will identify the dimensions, levels, and attributes in a logical data model. We will also identify the relationships within each dimension. The resulting data model will be used to design the Global star schema, the OLAP Catalog metadata, and the analytic workspace.

Identifying Dimensions

Four dimensions that 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 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 All Channels and Channel.
- Global performs customer and geographic analysis along the line of shipments to customers and by market segmentation. 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): All Customers, Region, Warehouse, and Ship To.
  - When analyzing by market segmentation, the levels of summarization will be (highest to lowest): Total Market, 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 three levels (highest to lowest): Year, Quarter, and Month.

Within the Channel, Customer, and Product dimensions, we added a Total or All level as the highest level of summarization. Adding this highest level will provide 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, Product, and Time dimensions, Global Computing requires only one hierarchy for each dimension. For the Customer dimension, however, 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 Market Segment.

Identifying Stored Measures

"Identifying Required Business Facts" on page 3-6 lists 21 business facts that are required to support the analysis requirements of Global Computing. Of this number, only three facts need to be acquired from the transactional database:

- Units
- Unit Price

- Unit Cost

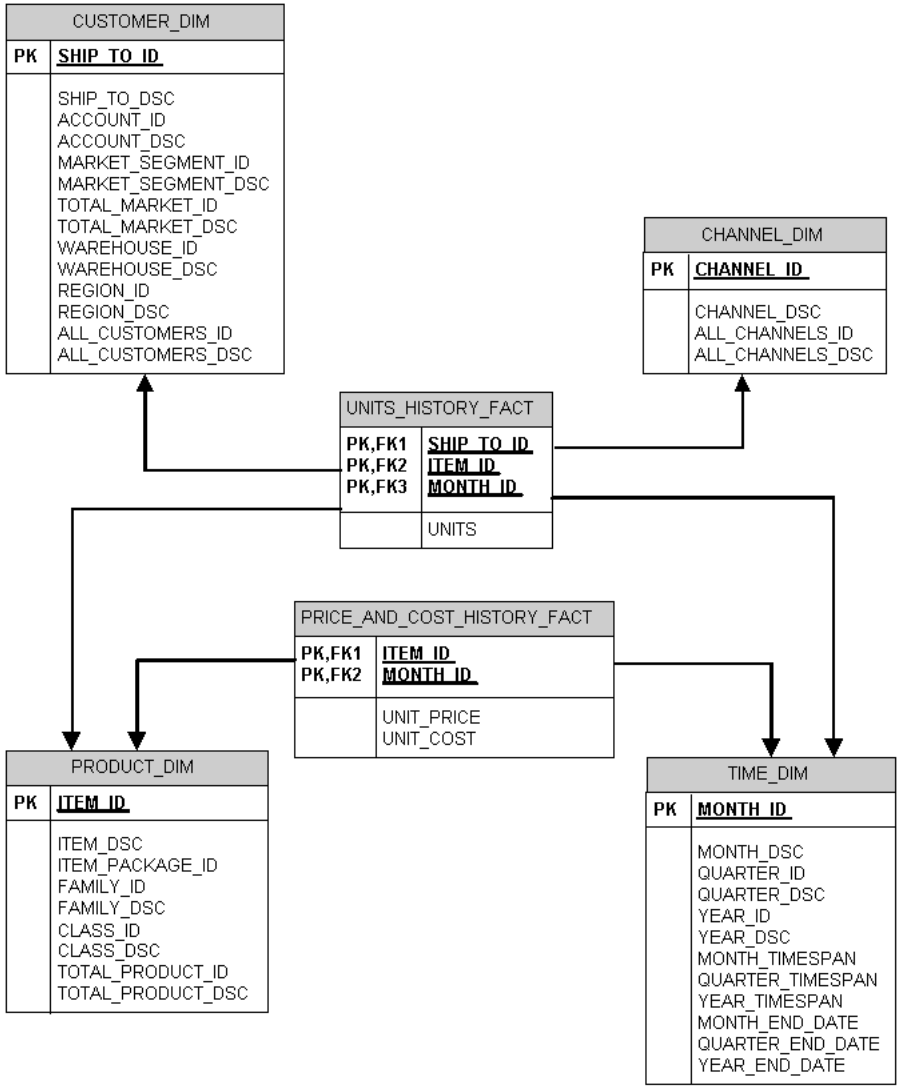
All of the other facts can be derived from these basic facts. The derived facts can be calculated in the analytic workspace on demand. If experience shows that some of these derived facts are being used heavily and the calculations are putting a noticeable load on the system, then some of these facts can be calculated and stored in the analytic workspace as a data maintenance procedure.

The Global Star Schema

The Global schema consists of two fact tables and four dimension tables. The dimension tables use numeric surrogate keys for each level column to assure that dimension members are unique across levels. For example, a geography dimension can easily have identical values at different levels, for example, New York at the City level and New York at the state level. In an analytic workspace, dimension members at all levels are fetched into a single dimension, and duplicate values overwrite each other unless additional steps are taken to assure uniqueness.

Figure 3–1 shows the relationships among the tables. In addition, the Global schema contains update fact tables, which are omitted from the diagram but occupy the same logical position as the history fact tables.

Figure 3–1 Global Schema Diagram



Dimension Table: TIME_DIM

The TIME_DIM table defines a time dimension with three levels. Each level is supported by four columns: a numeric surrogate key, a textual description, an end date (last day in time period), and a time span (number of days in time period). This is the most basic information required to define a time dimension.

The surrogate keys are artificial values with no meaning outside the context of the table. They assure that the same values are not repeated at different levels, and they provide the fastest processing speeds both in the relational tables and in the analytic workspace. The descriptive columns provide meaning to these numeric identifiers.

The end date and time span columns support time-series analysis, such as:

- Change from a prior period
- Change from a year ago
- Year-to-date
- Range of time

There are seven years defined, from 1998 to 2004, with data provided for 1998 to early 2003. The last year is available for forecasting.

In the standard hierarchy, the rollup sequence from the base to the top level is:

```
MONTH -> QUARTER -> YEAR
```

Table 3–1 describes the columns of the TIME_DIM table.

Table 3–1 TIME_DIM Column Descriptions

Column	Datatype	Role	Unique Values	Sample Value
MONTH_ID	NUMBER	Primary key Surrogate key	78	34
MONTH_DSC	VARCHAR2	Attribute	-	Apr-99
QUARTER_ID	NUMBER	Surrogate key	26	10
QUARTER_DSC	VARCHAR2	Attribute	-	Q2-99
YEAR_ID	NUMBER	Surrogate key	7	2
YEAR_DSC	VARCHAR2	Attribute	-	1999
MONTH_TIME_SPAN	NUMBER	Attribute	-	30
QUARTER_TIMESPAN	NUMBER	Attribute	-	91

Table 3–1 (Cont.) TIME_DIM Column Descriptions

Column	Datatype	Role	Unique Values	Sample Value
YEAR_TIMESPAN	NUMBER	Attribute	-	365
MONTH_END_DATE	DATE	Attribute	-	30-Apr-1999
QUARTER_END_DATE	DATE	Attribute	-	30-Jun-1999
YEAR_END_DATE	DATE	Attribute	-	31-Dec-1999

Dimension Table: CUSTOMER_DIM

The CUSTOMER_DIM table defines seven levels that will be used to define two hierarchies. Each level has a numeric surrogate key and a textual description, which is the most basic information to define a "normal" dimension, that is, a dimension that is not time. SHIP_TO is the primary key, and its values will become the base-level members for both Customer hierarchies.

In the Market hierarchy, the rollup sequence from the base to the top level is:

```
SHIP_TO -> ACCOUNT -> MARKET_SEGMENT -> TOTAL_MARKET
```

In the Customer hierarchy, the rollup sequence is:

```
SHIP_TO -> WAREHOUSE -> REGION-> ALL_CUSTOMERS
```

Table 3–2 describes the columns of the CUSTOMER_DIM table.

Table 3–2 CUSTOMER_DIM Column Descriptions

Column	Datatype	Role	Hierarchy	Unique Values	Sample Value
SHIP_TO_ID	NUMBER	Primary key Surrogate key	Both	61	89
SHIP_TO_DSC	VARCHAR2	Attribute	-	-	Monolith Motor Co. Knoxville
ACCOUNT_ID	NUMBER	Surrogate key	Market	24	36
ACCOUNT_DSC	VARCHAR2	Attribute	-	-	Monolith Motor Company
MARKET_SEGMENT_ID	NUMBER	Surrogate key	Market	5	5
MARKET_SEGMENT_DSC	VARCHAR2	Attribute	-	-	Manufacturing

Table 3–2 (Cont.) CUSTOMER_DIM Column Descriptions

Column	Datatype	Role	Hierarchy	Unique Values	Sample Value
TOTAL_MARKET_ID	NUMBER	Surrogate key	Market	1	7
TOTAL_MARKET_DSC	VARCHAR2	Attribute	-	-	Total Market
WAREHOUSE_ID	NUMBER	Surrogate key	Customers	11	21
WAREHOUSE_DSC	VARCHAR2	Attribute	-	-	United States
REGION_ID	NUMBER	Surrogate key	Customers	3	10
REGION_DSC	VARCHAR2	Attribute	-	-	North America
ALL_CUSTOMERS_ID	NUMBER	Surrogate key	Customers	1	1
ALL_CUSTOMERS_DSC	VARCHAR2	Attribute	Customers	-	All Customers

Dimension Table: PRODUCT_DIM

The PRODUCT_DIM table defines a product dimension with four levels. Each level has a numeric surrogate key and descriptive text. ITEM_ID is the primary key, so its values will become the base-level members of the Product dimension.

In the Product hierarchy, the rollup sequence from the base level to the top level is:

```
ITEM -> FAMILY -> CLASS -> TOTAL_PRODUCT
```

Table 3–3 describes the columns of the PRODUCT_DIM table.

Table 3–3 PRODUCT_DIM Column Descriptions

Column	Datatype	Role	Unique Values	Sample Value
ITEM_ID	NUMBER	Primary key Surrogate key	36	48
ITEM_DSC	VARCHAR2	Attribute	-	Keyboard Wrist Rest
ITEM_PACKAGE_ID	NUMBER	Attribute	4	Laptop Value Pack
FAMILY_ID	NUMBER	Surrogate key	9	7
FAMILY_DSC	VARCHAR2	Attribute	-	Accessories
CLASS_ID	NUMBER	Surrogate key	2	3

Table 3–3 (Cont.) PRODUCT_DIM Column Descriptions

Column	Datatype	Role	Unique Values	Sample Value
CLASS_DSC	VARCHAR2	Attribute	-	Software/Other
TOTAL_PRODUCT_ID	NUMBER	Surrogate key	1	1
TOTAL_PRODUCT_DSC	VARCHAR2	Attribute	-	Total Product

Dimension Table: CHANNEL_DIM

The CHANNEL_DIM table contains four columns. CHANNEL_ID is the primary key, and its values will become the base-level members of the Channel dimension. ALL_CHANNELS_ID defines a single value that represents all of the channels. In the OLAP Catalog, these two columns will define the two levels of a single Channel hierarchy. The rollup sequence from the base level to the top level is simply:

```
CHANNEL -> ALL_CHANNELS
```

The remaining columns, CHANNEL_DSC and ALL_CHANNELS_DSC, provide textual descriptions that give the surrogate keys meaning.

Table 3–4 describes the columns of the CHANNEL_DIM table.

Table 3–4 CHANNEL_DIM Column Descriptions

Column	Datatype	Role	Unique Values	Sample Value
CHANNEL_ID	NUMBER	Primary key Surrogate key	3	4
CHANNEL_DSC	VARCHAR2	Attribute	-	Internet
ALL_CHANNELS_ID	NUMBER	Surrogate key	1	1
ALL_CHANNELS_DSC	VARCHAR2	Attribute	-	All Channels

Fact Tables: UNITS_HISTORY_FACT and _UPDATE_FACT

The UNITS_HISTORY_FACT and UNITS_UPDATE_FACT tables contain four foreign key columns, which together comprise a multi-column primary key. The foreign keys are related to the primary keys of the four dimension tables.

In UNITS_HISTORY_FACT, every foreign key value for Product, Customer, and Channel is used at least once, and 65 time periods are used. The table contains 169,487 rows of a possible 513,864 unique key combinations.

UNITS_UPDATE_FACT adds data for month 91 (Jun-03). Every foreign key value for Product, Customer, and Channel is used at least once. The table contains 3,459 rows of a possible unique 6,588 key combinations.

Table 3–5 describes the columns in both tables.

Table 3–5 UNITS_HISTORY_FACT and UNITS_UPDATE_FACT Column Descriptions

Column	Datatype	Role	Description
CHANNEL_ID	NUMBER	Key	Related to CHANNEL_DIM
ITEM_ID	NUMBER	Key	Related to PRODUCT_DIM
SHIP_TO_ID	NUMBER	Key	Related to CUSTOMER_DIM
MONTH_ID	NUMBER	Key	Related to TIME_DIM
UNITS	NUMBER	Facts	Number of units sold

Fact Tables: PRICE_AND_COST_HISTORY_FACT and _UPDATE_FACT

The PRICE_AND_COST_HISTORY_FACT and PRICE_AND_COST_UPDATE_FACT tables contain two foreign key columns, which together comprise a multi-column primary key, and two fact columns.

In PRICE_AND_COST_HISTORY_FACT, data is provided for all products for 65 months.

PRICE_AND_COST_UPDATE_FACT adds data for month 91 (Jun-03) for all products.

Table 3–6 describes the columns of both tables.

Table 3–6 PRICE_AND_COST_HISTORY_FACT and PRICE_AND_COST_UPDATE_FACT Column Descriptions

Column	Datatype	Role	Description
ITEM_ID	NUMBER	Key	Related to PRODUCT_DIM
MONTH_ID	NUMBER	Key	Related to TIME_DIM
UNIT_PRICE	NUMBER	Facts	List price of a unit
UNIT_COST	NUMBER	Facts	Cost to produce a unit

Mapping the Global Schema to an Analytic Workspace

The OLAP Catalog provides an interface for mapping the columns of relational tables to the multidimensional objects of an analytic workspace. The following tables identify the mapping for the PRICE_AND_COST_HISTORY_FACT fact table and its related dimension tables, PRODUCT_DIM and TIME_DIM. These tables are the source for the PRICE_CUBE cube.

The analytic workspace objects listed in these tables will be created in Chapter 6, "Creating an Analytic Workspace", and are described in more detail in Chapter 8, "Exploring a Standard Form Analytic Workspace".

Global Product Dimension Mapping

Table 3–7 shows how the columns of the PRODUCT_DIM dimension table are mapped to workspace objects to provide an **embedded total** PRODUCT dimension.

PRODUCT_DIM supports a single dimension hierarchy, PRODUCT_ROLLUP, with four levels: ITEM, FAMILY, CLASS, and TOTAL_PRODUCT. The descriptive columns are mapped to both the long and short descriptions, but this redundancy in the analytic workspace is not required.

Table 3–7 Mapping the Global Product Dimension

PRODUCT_DIM Table Columns	OLAP Catalog Logical Objects	GLOBAL Analytic Workspace Objects
ITEM_ID	PRODUCT_ROLLUP hierarchy, ITEM level	PRODUCT dimension PRODUCT_PARENTREL parent relation PRODUCT_LEVELLIST level dimension (ITEM, FAMILY, CLASS, TOTAL_PRODUCT) PRODUCT_LEVELREL level relation PRODUCT_HIERLIST hierarchy dimension (PRODUCT_ROLLUP)
FAMILY_ID	PRODUCT_ROLLUP hierarchy, FAMILY level	
CLASS_ID	PRODUCT_ROLLUP hierarchy, CLASS level	
TOTAL_PRODUCT_ID	PRODUCT_ROLLUP hierarchy, TOTAL_PRODUCT level	
ITEM_PACKAGE	PACKAGE attribute	PRODUCT_PACKAGE variable

Table 3–7 (Cont.) Mapping the Global Product Dimension

PRODUCT_DIM Table Columns	OLAP Catalog Logical Objects	GLOBAL Analytic Workspace Objects
ITEM_DSC	ITEM Long Description attribute ITEM Short Description attribute	PRODUCT_LONG_DESCRIPTION variable PRODUCT_SHORT_DESCRIPTION variable
FAMILY_DSC	FAMILY Long Description attribute FAMILY Short Description attribute	
CLASS_DSC	CLASS Long Description attribute CLASS Short Description attribute	
TOTAL_PRODUCT_DSC	CLASS Long Description attribute CLASS Short Description attribute	

Global Time Dimension Mapping

Table 3–8 shows how the columns of the TIME_DIM dimension table are mapped to workspace objects to provide an **embedded total** TIME dimension.

TIME_DIM supports a single dimension hierarchy, Calendar, with three levels: Month, Quarter, and Year. For time-based analysis in the analytic workspace, a Time dimension must have End_Date and Time_Span attributes, as it does here. The descriptive columns are mapped to both the long and short descriptions, but this redundancy in the analytic workspace is not required.

Table 3–8 Mapping the Global Time Dimension

TIME_DIM Table Columns	OLAP Catalog Logical Objects	GLOBAL Analytic Workspace Objects
MONTH_ID	Calendar hierarchy, Month level	TIME dimension TIME_PARENTREL parent relation TIME_LEVELLIST level dimension (Month, Quarter, Year)) TIME_LEVELREL level relation TIME_HIERLIST hierarchy dimension (Calendar)
QUARTER_ID	Calendar hierarchy, Quarter level	
YEAR_ID	Calendar hierarchy, Year level	
MONTH_DSC	Month Long Description attribute Month Short Description attribute	TIME_LONG_DESCRIPTION variable TIME_SHORT_DESCRIPTION variable
QUARTER_DSC	Quarter Long attribute Attribute Quarter Short attribute Attribute	
YEAR_DSC	Year Long Description attribute Year Short Description attribute	
MONTH_TIMESPAN	Month Time_Span attribute	TIME_TIME_SPAN variable
QUARTER_TIMESPAN	Quarter Time_Span attribute	
YEAR_TIMESPAN	Year Time_Span attribute	
MONTH_END_DATE	Month End_Date attribute	TIME_END_DATE variable
QUARTER_END_DATE	Quarter End_Date attribute	
YEAR_END_DATE	Year End_Date attribute	

Global Price Cube Mapping

Table 3–9 shows how the columns of the PRICE_AND_COST_HISTORY_FACT fact table are mapped to workspace objects to provide a PRICE_CUBE cube with two measures, UNIT_COST and UNIT_PRICE.

An aggregation operator is defined in the OLAP Catalog and is the basis for an initial aggmap for the cube. The aggmap provides the rules of aggregation. The measure formulas use the aggmap to aggregate the base-level data loaded into the measure variables.

Most variables are sparse and require a composite dimension, which is associated with the cube.

Table 3–9 Mapping the Global Price Cube

PRICE_AND_COST_HISTORY_FACT Table Columns	OLAP Catalog Logical Objects	GLOBAL Analytic Workspace Objects
ITEM_ID	PRICE_CUBE cube	PRICE_CUBE dimension (TIME, PRODUCT)
MONTH_ID	SUM aggregation operator	PRICE_CUBE_COMPOSITE dimension PRICE_CUBE_AGGMAP_AWCREATEDEFAULT_1 aggmap
UNIT_PRICE	UNIT_PRICE measure	UNIT_PRICE formula UNIT_PRICE_VARIABLE variable
UNIT_COST	UNIT_COST measure	UNIT_COST formula UNIT_COST_VARIABLE variable

4

Developing Java Applications for OLAP

This chapter presents the rich development environment and the powerful tools that you can use to create OLAP applications. It includes the following topics:

- Building Analytical Java Applications
- Introducing the BI Beans
- Understanding the OLAP API
- Managing Data Sources for the BI Beans and OLAP API

For information about SQL access to analytic workspaces, refer to Chapter 7, "SQL Access to Analytic Workspaces".

Building Analytical Java Applications

Java is the language of the Internet. Using Java, an application developer can write standalone Java applications (which can be launched from a browser with Java's WebStart technology), or they can create HTML applications through servlets, JavaServer Pages (JSP), and Oracle User Interface XML (UIX), which access live data from an Oracle Database.

About Java

Java is the preferred programming language for an ever-increasing number of professional software developers. For those who have been programming in C or C++, the move to Java is easy because it provides a familiar environment while avoiding many of the shortcomings of the C language. Developed by Sun Microsystems, Java is fast superseding C++ and Visual Basic as the language of choice for application developers, for the following reasons:

- Object oriented. Java enables application developers to focus on the data and methods of manipulating that data, rather than on abstract procedures; the programmer defines the desired object rather than the steps needed to create that object. Almost everything in Java is defined as an object.
- Platform independent. The Java compiler creates byte code that is interpreted at runtime by the Java Virtual Machine (JVM). As the result, the same software can run on all Windows, Unix, and Macintosh platforms where the JVM has been installed. All major browsers have the JVM built in.
- Network based. Java was designed to work over a network, which enables Java programs to handle remote resources as easily as local resources.
- Secure. Java code is either trusted or untrusted, and access to system resources is determined by this characteristic. Local code is trusted to have full access to system resources, but downloaded remote code (that is, an applet) is not trusted. The Java "sandbox" security model provides a very restricted environment for untrusted code.

The Java Solution for OLAP

To develop an OLAP application, you can use the Java programming language. Java enables you to write applications that are platform-independent and easily deployed over the Internet.

The OLAP API is a Java-based application programming interface that provides access to multidimensional data for analytical business applications. The OLAP API uses OLAP Catalog metadata to access data that is stored either in the relational tables of a star or snowflake schema, or in views of an analytic workspace that has been enabled for its use.

Java classes in the OLAP API provide all of the functions required of an OLAP application: Connection to an OLAP instance; authentication of user credentials; access to data in the RDBMS controlled by the permissions granted to those credentials; and selection and manipulation of that data for business analysis.

The BI Beans simplify application development by providing these functions as JavaBeans. Moreover, the BI Beans include JavaBeans for presenting the data in graphs, crosstabs, and tables.

Note: Oracle JDeveloper and the BI Beans are not packaged with the Oracle RDBMS.

Oracle Java Development Environment

Oracle JDeveloper provides an integrated development environment (IDE) for developing Java applications. Although third-party Java IDEs can also be used effectively, only JDeveloper achieves full integration with the Oracle Database and BI Beans wizards. The following are a few JDeveloper features:

- Remote graphical debugger with break points, watches, and an inspector.
- Multiple document interface (MDI)
- *Codecoach* feature that helps you to optimize your code
- Generation of 100% Pure Java applications, applets, servlets, Java beans, and so forth with no proprietary code or markers
- Oracle Database browser

> **Note:** Oracle JDeveloper is an application and is not packaged with the Oracle RDBMS.

Introducing the BI Beans

The BI Beans provide reusable components that are the basic building blocks for OLAP decision support applications. Using the BI Beans, developers can rapidly develop and deploy new applications, because these large functional units have already been developed and tested — not only for their robustness, but also for their ease of use. And because the BI Beans provide a common look and feel to OLAP applications, the learning curve for end users is greatly reduced.

The BI Beans contain the following:

- **Presentation Beans** display the data in a rich variety of formats so that trends and variations can easily be detected. Among the Presentation Beans currently available are Graph, Table, and Crosstabs.
- **Data Beans** acquire and manipulate the data. The Data Beans use the OLAP API to connect to a data source, define a query, manipulate the resultant data set, and return the results to the Presentation Beans for display. Data Beans include a Query Builder and a Calculation Builder.
- **Persistence Services** is a set of packages that support the storage and retrieval of objects in the BI Beans Catalog, not only so that you can save your work, but also so that you can share the work with others who have access to the Catalog.

The BI Beans can be implemented as a Java client or a thin client. Java clients best support users who do immersed analyses, that is, use the system for extensive periods of time with a lot of interaction. For example, users who create reports benefit from a Java client. Thin clients best support remote users who use a low bandwidth connection and have basic analytical needs. Thin clients can be embedded in a portal or other Web site for these users.

Metadata

The OLAP API and the BI Beans use the OLAP Catalog to provide the information they need about multidimensional objects defined in an Oracle data warehouse, such as measures and dimensions. The BI Beans generates additional metadata to support its additional functionality. This additional metadata is called the BI Beans Catalog.

Navigation

The Presentation Beans support navigation techniques such as drilling, pivoting, and paging.

- *Drilling* displays lower-level values that contribute to a higher-level aggregate, such as the cities that contribute to a state total.
- *Pivoting* rotates the data cube so that the dimension members that labeled a series now label groups, or the dimension members that labeled columns in a crosstab now label rows instead. For example, if products label the rows and regions label the columns, then you can pivot the data cube so that products label the columns and regions label the rows.
- *Paging* handles additional dimensions by showing each member in a separate graph, crosstab, or table rather than nesting them in the columns or rows. For example, you might want to see each time period in a separate graph rather than all time periods on the same graph.

Formatting

The Presentation Beans enable you to change the appearance of a particular display. In addition, the values of the data itself can affect the format.

- Number formatting. Numerical displays can be modified by changing their scale, number of decimal digits and leading zeros, currency symbol, negative notation, and so forth. Currency symbols and scaling factors can be displayed in the column or row headers rather than in the cells.

- Stoplight formatting. The formatting of the cell background color, border, font, and so forth can be data driven so that outstanding or problematic results stand out visually from the other data values.
- Ranking. In ranking reports, the numerical rank of each dimension value, based on the value of the measure, is displayed.

Graphs

The Graph bean presents data in a large selection of two- and three-dimensional business chart types, such as bar, area, line, pie, ring, scatter, bubble, pyramid, and stock market. Many of the 2D graphs can be displayed as clustered, stacked, dual-Y, percentage, horizontal, vertical, or 3D effect.

Bar, line, and area graphs can be combined so that individual rows in the data cube can be specified as one of these graph types. You can also assign marker shape and type, data line type, color, and width, and fill colors on a row-by-row basis.

The graph image can be copied to the system clipboard and exported in GIF and other image formats.

Users can zoom in and out of selected areas of a graph. They can also scroll across the axes.

Crosstabs

The Crosstab bean presents data in a two-dimensional grid similar to a spreadsheet. Multiple dimensions can be nested along the rows or columns, and additional dimensions can appear as separate pages. Among the available customizations are: Font style, size, color and underlining; individual cell background colors; border formats; and text alignment.

Users can navigate through the data using either a mouse or the keyboard. They can insert rows and columns to display totals, and edit cells for what-if analysis.

Tables

The Table bean presents data in record format like a relational table or view. In contrast to the crosstab, the table display handles measures individually rather than as members of a measure dimension. Thus, each measure can be manipulated individually.

Data Beans

The Data Beans use the OLAP API to provide the basic services needed by an application. They enable clients to identify a database, present credentials for accessing that database, and make a connection. The application can then access the metadata and identify the available data. Users can select the measures they want to see and the specific slice of data that is of interest to them. That data can then be modified and manipulated.

Wizards

The BI Beans offer wizards that can be used both by application developers in creating an initial environment and by end users in customizing applications to suit their particular needs. The wizards lead you step-by-step so that you provide all of the information needed by an application. The following are some of the tasks that can be done using wizards.

- Building a query. Fact tables and materialized views often contain much more data than users are interested in viewing. Fetching vast quantities of data can also degrade performance unnecessarily. In addition to selecting measures, you can limit the amount of data fetched in a query by selecting dimension members from a list or using a set of conditions. A selection can be saved and used again just by picking its name from a list.

  The BI Beans take advantage of all of the new OLAP functions in the database, including ranking, lag, lead, and windowing. End users can create powerful queries that ask sophisticated analytical questions, without knowing SQL at all.

- Generating custom measures. You can define new "custom" measures whose values are calculated from data stored within the database. For example, a user might create a custom measure that shows the percent of change in sales from a year ago. The data in the custom measure would be calculated using the lag method on data in the Sales measure. Because a DBA cannot anticipate and create all of the calculations required by all users, the BI Beans enable users to create their own.

Understanding the OLAP API

OLAP applications typically have object-oriented user interfaces where users manipulate objects that represent organized groupings of their data. Thus, there is a natural relationship between an object-oriented user interface and an object-oriented API such as the Oracle OLAP API. The OLAP API exploits this

natural relationship by providing objects that match the end-user behavior that an application needs.

Object-oriented languages such as Java manipulate data by applying methods on objects. This approach enables the objects to maintain a current state and support incremental modifications to that state. This approach provides excellent support for common OLAP actions such as drill and rotate.

For example, a central activity for users of OLAP applications is refining queries. A user has a question in mind and devises a query to answer that question. In most cases, the initial results of the query prompt the user to want to dig deeper for a solution, perhaps by drilling to see more detailed data or by rotating the report to highlight correlations in the data. The OLAP API is able to use the result of one query as the input to the next query.

How the OLAP API Accesses Multidimensional Data

The OLAP API accesses the data through the OLAP Catalog, that is, the relational tables that contain OLAP metadata. The application does not need to be aware of whether the data is located in relational tables or in an analytic workspace, nor does it need to know the mechanism for accessing that data.

Oracle OLAP translates all queries from the OLAP API into SQL; when a query is issued through the OLAP API, the SQL generator in Oracle OLAP issues a `SELECT` statement against a relational table or view. This has several advantages for application developers:

- The difficult task of writing the complex SQL needed to resolve multidimensional queries, and even more difficult task of optimizing that complex SQL, is left for Oracle OLAP to do. Application developers can be more productive writing in the OLAP API, which is designed for OLAP.
- Updates to SQL and the OLAP DML will be incorporated into new versions of the OLAP API. Applications can make use of new analytic and performance features without recoding.

As an alternative access method, the OLAP API provides a way for a Java application to directly manipulate workspace data, without the need for any metadata and without the use of the OLAP API data manipulation classes. The Java application uses the `SPLExecutor` class in the OLAP API to send DML commands directly to Oracle OLAP for execution in the workspace.

Whichever access method is used, the application establishes a connection, opens the workspace, accesses the data (either through MDM metadata or through `SPLExecutor`), closes the workspace, and closes the connection.

See Also:

- *Oracle OLAP Developer's Guide to the OLAP API*
- OLAP API Javadoc

Calculation Capabilities

The OLAP API generates SQL commands to select and manipulate data stored in the relational tables or views. When the data is stored in an analytic workspace, the computational power of the OLAP engine can be used to manipulate the data, including:

- Modeling
- Forecasting
- What-if scenarios

When the data is stored in a star or snowflake schema, the SQL commands generated by the OLAP API can include the "N-pass" functions, such as `RANK`, `PERCENTILE`, `TOPN`, `BOTTOMN`, `LAG`, `LEAD`, `SUM`, `AVG`, `MIN`, `MAX`, `COUNT`, and `STDDEV`. Data fetches use many database innovations, including concatenated rollup, scrollable cursors, and query rewrite.

The OLAP API provides expanded calculation capabilities beyond those that can be handled efficiently in other OLAP solutions, such as:

- Totals broken out by multiple attributes
- Suppression of `NA` and zero rows, columns, and pages
- Union dimensions
- Measures as dimensions
- Inter-row calculations such as the following book-to-bill ratio:

```
Balance(Account "BOOKED", Period "PRIOR")/ Balance(Account "BILLED", Period
"LAST")
```

- Asymmetric queries

Intelligent Caching

Analytical queries are by nature iterative. An analyst formulates a query, sees the results, and then formulates other queries based on those results. Since the likelihood is very high in business analysis of needing the same data to answer

subsequent queries, the OLAP API caches the metadata so that it is available throughout the session without fetching it again. Moreover, the OLAP API defines the result set of a query geometrically. Using multidimensional cursors, the OLAP API can randomly access disparate regions of the result set. This enables an application to retrieve just the data currently of interest instead of all of the data in the result set. For example, you might scroll to the end of a page without having to fetch all of the data on the page.

Managing Data Sources for the BI Beans and OLAP API

Applications built using the BI Beans and the OLAP API can have as a data source either an analytic workspace or a relational schema (star or snowflake). This guide is written primarily to describe the creation and management of analytic workspaces. However, the information for creating a relational data warehouse for use by the BI Beans and the OLAP API is also contained here.

Take these steps if you plan to use a star or snowflake schema as the data source for OLAP applications, and you do not plan to create an analytic workspace:

1. Create `CWM1` or `CWM2` metadata as described in "Overview of the OLAP Catalog" on page 5-6.
2. Using a SQL command processor such as SQL*Plus, issue this command to make the metadata accessible to the BI Beans.

   ```
   EXECUTE CWM2_OLAP_METADATA_REFRESH.MR_REFRESH
   ```

3. Create materialized views as described in Chapter 13.

Part II

Fundamentals of Creating and Using Analytic Workspaces

Part II contains information about creating analytic workspaces. It contains the following chapters:

5

Defining a Logical Multidimensional Model

This chapter describes methods of creating a logical multidimensional model. It includes the following sections:

Introduction to OLAP Metadata

Metadata is used throughout Oracle OLAP to define a logical multidimensional model:

- To describe the source data as multidimensional objects for use by the analytic workspace build tools.

  There are several methods of creating this type of metadata, as described in this chapter.

- To identify the components of logical objects in an analytic workspace for use by the refresh, aggregation, and enablement tools.

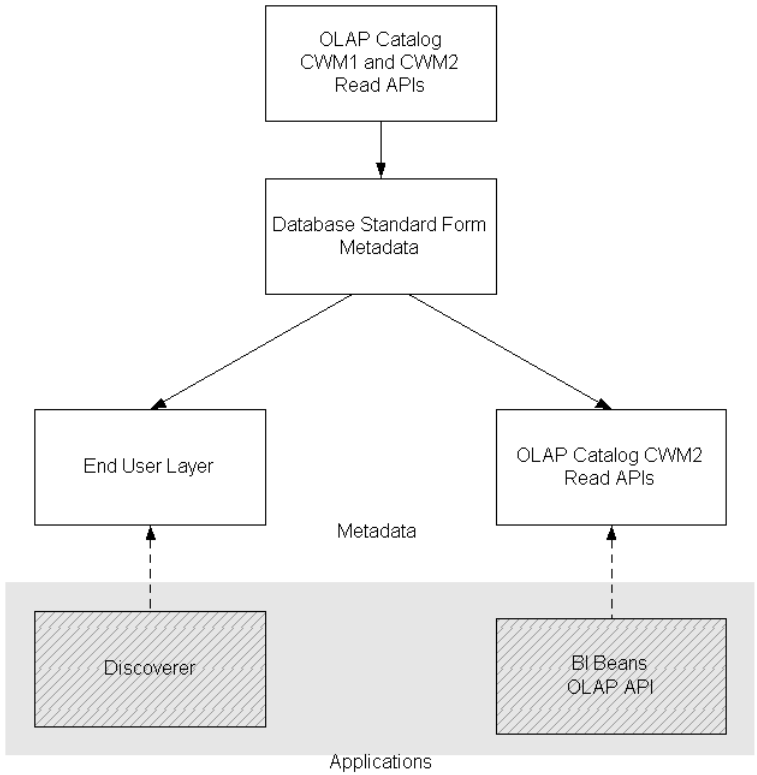
  Database standard form describes this metadata, which is generated by the workspace creation tools. Refer to Appendix A for a description of standard form.

- To describe relational views of analytic workspaces as multidimensional objects for use by OLAP applications.

  The application determines the type of metadata that is needed. The BI Beans require OLAP Catalog metadata, which is described in this chapter.

You only need to describe your source data; the OLAP tools can generate the equivalent metadata for the analytic workspace and the workspace views. The logical model is transformed along with the data. Figure 5–1 shows the metadata transformations performed by the OLAP tools. These metadata types are discussed in this chapter.

Figure 5–1 Transformation of the Logical Model



Creating Metadata for Your Source Data

Defining the logical model is the first stage of metadata creation; the second stage is mapping the logical objects to physical data sources. Different types of metadata have different requirements for the storage format of the source data; you must choose the method that is appropriate for your data source. Moreover, there are multiple methods of creating metadata, including graphical user interfaces and PL/SQL APIs.

For Source Data in a Basic Star or Snowflake Schema

The `CWM1` write APIs, which are used by the OLAP Management tool, create a database dimension object for each logical OLAP dimension. The database dimension object imposes the following restrictions on dimension tables and the related fact tables of a star or snowflake schema:

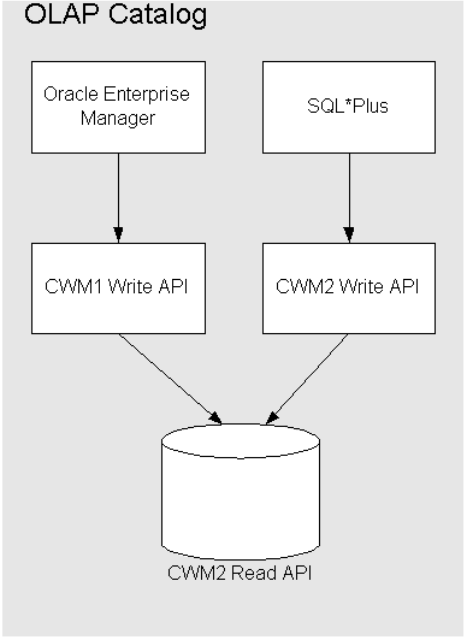
- All hierarchies must be level-based; the schema cannot use parent-child dimension tables.
- Multiple hierarchies defined for a dimension must have the same base level.
- Level columns cannot contain `NULL`s.
- Fact data must be unsolved, that is, it is stored only at the lowest level of the hierarchy, and all the data for a cube must be stored in a single fact table.

If your source data is a star or snowflake schema and conforms to these additional requirements, then you can use either Oracle Enterprise Manager or the `CWM2` APIs, depending on your personal preference. The OLAP Management tool in Oracle Enterprise Manager provides a graphical user interface. The `CWM2` APIs enable you to generate a SQL program that you can easily modify and port to other databases.

If your source data is a star or snowflake schema that does not conform with these requirements, then use the `CWM2` APIs.

Figure 5–2 shows the tools for creating OLAP Catalog metadata.

Figure 5–2 Tools for Creating OLAP Catalog Metadata for Source Data



This chapter introduces the OLAP Management tool in Oracle Enterprise Manager and the CWM2 APIs.

> **See Also:** *Oracle OLAP Reference* for complete syntax and descriptions of the CWM2 APIs

For Dimension Tables with Complex Hierarchies

If your source data is a star or snowflake schema, but the dimension tables include any of the following variations, then use the CWM2 APIs:

- Level columns containing NULLs, such as skip-level hierarchies
- Multiple hierarchies with different base levels (sometimes called **ragged hierarchies**)
- Multiple hierarchies with values mapped to different levels
- Embedded total dimensions
- Parent-child dimensions

If your schema contains parent-child dimension tables, then you must convert them to level-based dimension tables. The `CWM2` write APIs include a package for this transformation.

For Other Schema Configurations

If you are using Oracle Warehouse Builder already to transform your data, then generating an analytic workspace takes only a few additional steps. Warehouse Builder provides a graphical interface for designing a logical model, and deploys the model as metadata. When you use the OLAP Bridge in Warehouse Builder, it generates CWM1 metadata from its Design Repository. Warehouse Builder also creates and populates an analytic workspace, and enables it for use by the BI Beans.

If your data is stored in flat files or SQL tables, then you can use a manual method described in this guide. This method enables you to use the OLAP Catalog, but requires you to write data loading programs in the OLAP DML.

If you are upgrading from Oracle Express, then you may be able to automate the conversion process.

See Also:

- *Oracle Warehouse Builder User's Guide* if your data requires transformation
- Chapter 11 if your data is in flat files, or in a purely relational schema and you prefer to use the OLAP DML
- Appendix B if your data is in an Express database

Creating Metadata for Your Analytic Workspace

The tools for creating analytic workspaces comply with the requirements of database standard form, and transform the source metadata into standard form metadata. You do not need to perform any extra steps to maintain the standard form metadata when you use the OLAP tools to maintain the analytic workspace. You can make changes to the logical model in the metadata for the data source, and the refresh tool makes the appropriate changes to the standard form metadata.

However, if you make manual changes to your analytic workspace, such as adding a measure, then you are responsible for making the appropriate changes to the standard form metadata. Standard form is described in Appendix A.

Creating Metadata for Your Applications

Applications that use the BI Beans require OLAP Catalog metadata, and those that use Discoverer require an End User Layer. Both types of metadata require the data source to be in relational tables or views for SQL access. Thus, the enablers in Analytic Workspace Manager for these types of applications generate views of analytic workspace objects in the format required by the metadata, and then generate the metadata itself. The enablers transform the standard form metadata provided in the analytic workspace; you do not need to redefine the logical model. Instructions for enabling an analytic workspace are provided in Chapter 6.

Overview of the OLAP Catalog

The OLAP Catalog defines logical multidimensional objects and maps them to physical data sources. The logical objects are cubes, measures, dimensions, and so forth as described in "The Logical Multidimensional Data Model" on page 2-1. The physical data sources are the columns of a relational table or view. A number of different warehouse configurations can be represented by OLAP Catalog metadata.

The OLAP Catalog serves these distinct functions for analytic workspaces:

- Describes the relational tables of a star or snowflake schema so that the data can be fetched into an analytic workspace. This metadata is used only when building or refreshing the analytic workspace.
- Describes the relational views of an analytic workspace so that the data can be queried by the BI Beans. This metadata is used only at runtime so that applications have access to the workspace data.

Thus, when you are developing an analytic workspace, you may create two sets of OLAP Catalog metadata: one for the source schema, and the other for the analytic workspace. If your analytic workspace is used by another application, such as Oracle Discoverer, then you only define OLAP Catalog metadata for your source schema. For your analytic workspace, you create an End User Layer (EUL), which is the type of metadata required by Discoverer.

The OLAP Catalog is also used to describe the relational tables of a star schema so that the data can be queried by the BI Beans. In this type of scenario, no analytic workspace is used; aggregate data is stored in materialized views, as described in Chapter 13.

The BI Beans query metadata stored in the OLAP Catalog. Your data, whether it is stored in relational tables or in an analytic workspace, is inaccessible to applications

based in these technologies unless the data is identified in the OLAP Catalog. The OLAP Catalog is also available to any other applications that want to use it.

OLAP Catalog Components

The OLAP Catalog includes the following:

- **Metadata model tables**: A set of relational tables within the database that instantiate the OLAP metadata model. These tables define all the OLAP metadata objects: dimensions, measures, cubes, measure folders, and so on. Within the metadata definitions are references to the actual data sources.
- **Write API**: A set of PL/SQL packages for creating and editing OLAP metadata. These packages contain procedures for inserting, updating, and deleting rows in the model tables.
- **Read API**: A set of relational views within the database that provide information about the metadata registered in the model tables.

Two versions of the OLAP Catalog are currently in use, `CWM1` (also called CWM-Lite) and `CWM2`. Each version has its own metadata model tables, write API, and read API. However, applications can query a set of union views that contains all of the OLAP Catalog metadata, regardless of the write API used to generate it.

About CWM1

`CWM1` is available through the OLAP Management tool of Oracle Enterprise Manager. You can use `CWM1` only to describe a schema that complies with the requirements listed in "Choosing a Tool for Creating OLAP Catalog Metadata" on page 5-8. You can then use the OLAP Catalog to create an analytic workspace or to access the relational schema directly through the BI Beans.

You can view `CWM1` metadata in the OLAP Management tool of Enterprise Manager, or in the OLAP Catalog View of Analytic Workspace Manager.

About CWM2

`CWM2` is available through the BI Beans enabler in Analytic Workspace Manager and as a set of PL/SQL packages. You can use `CWM2` to describe a star or snowflake schema that does not comply with the requirements for `CWM1`. You can use only `CWM2` to define the metadata for an analytic workspace; you cannot use `CWM1` for this purpose.

You can view `CWM2` metadata in the OLAP Catalog View of Analytic Workspace Manager.

Steps for Creating OLAP Metadata

Whether you create OLAP metadata programmatically or by using a graphic interface, you follow the same basic steps.

To create OLAP metadata:

1. Create logical dimensions. Specify the levels, attributes, and hierarchies associated with each one. ("Procedure: Defining a Logical Dimension in the OLAP Catalog" on page 5-13)
2. Create logical cubes and specify their edges (dimensions). ("Procedure: Defining a Logical Cube in the OLAP Catalog" on page 5-14)
3. Create logical measures that represent the fact data. Associate each measure with a cube. ("Procedure: Defining a Logical Cube in the OLAP Catalog")
4. Map the logical entities to the source data. ("Procedure: Defining a Logical Cube in the OLAP Catalog" on page 5-14)

Choosing a Tool for Creating OLAP Catalog Metadata

The tools for creating OLAP Catalog metadata depend on whether you are creating the metadata for a relational schema or for an analytic workspace. Some tools have specific prerequisites.

Creating Metadata for an Analytic Workspace

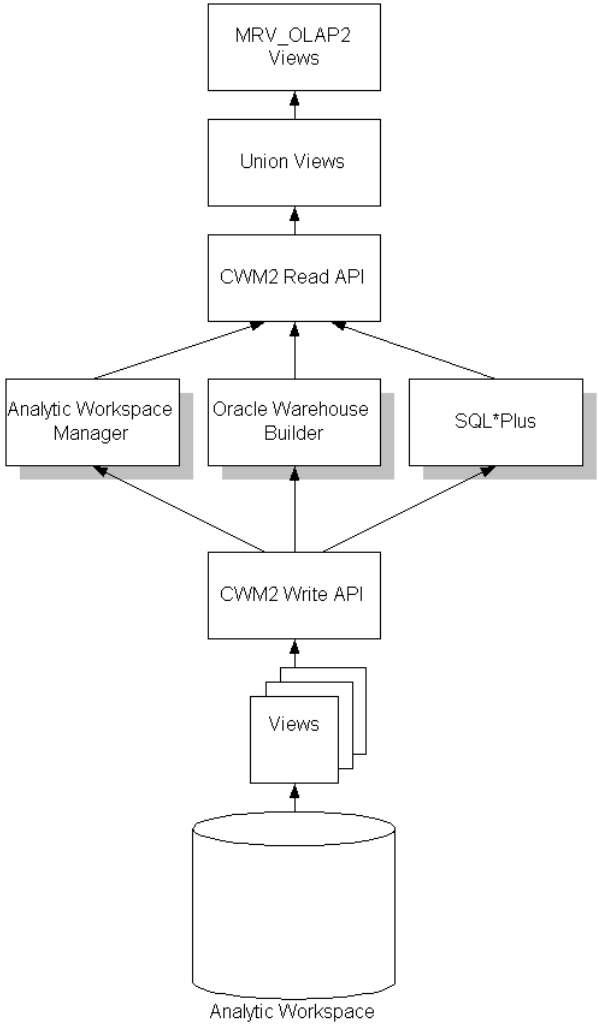
When you create OLAP Catalog metadata for the data stored in an analytic workspace, you define it against relational views of the multidimensional objects in the workspace. These views emulate a star schema, but are different in the way they expose dimensions. Instead of dedicating a separate column to each level of a dimension hierarchy, these views list all dimension members at all levels in a single column. For this reason, views of this type are called **embedded total** views. The views display the dimensions in the same format in which they are stored in the analytic workspace. You can create `CWM2` metadata for embedded total views; you cannot create `CWM1` metadata for them.

You can choose among three tools for creating OLAP Catalog metadata for an analytic workspace:

- Analytic Workspace Manager
- Oracle Warehouse Builder
- `CWM2` APIs

Figure 5–3 shows the relationships among these tools and an analytic workspace. When an analytic workspace is enabled for use by the BI Beans, relational views are created that can access workspace objects in response to a query. The `CWM2` write APIs store metadata about the logical model represented by the views in the `CWM2` read APIs. This metadata is automatically available through the Union views. You must run a PL/SQL procedure to make the metadata available to the Metadata Refresh Views, which are the views that provide the best performance when queried by the OLAP API.

Figure 5–3 Tools for Creating Metadata for Analytic Workspaces



If you have a standard form analytic workspace, then use the BI Beans enabler in Analytic Workspace Manager to generate the relational views and the `CWM2` metadata in a single step.

If you use Oracle Warehouse Builder to generate your analytic workspace, then it also creates the views and the `CWM2` metadata for access by the BI Beans.

If your analytic workspace includes objects that do not comply with database standard form, or you wish to generate the relational views manually, then write your own CWM2 script. You may wish to start by modifying a script generated by Analytic Workspace Manager.

Chapter 6, explains how to use the BI Beans enabler in Analytic Workspace Manager.

> **See Also:** *Oracle OLAP Reference* for complete syntax and descriptions of the CWM2 APIs

Creating Metadata Using Oracle Enterprise Manager

If your data warehouse complies with the requirements listed in "For Source Data in a Basic Star or Snowflake Schema" on page 5-3, you can create OLAP metadata using the OLAP Management tool in Oracle Enterprise Manager.

You generate the SQL statements that create the metadata primarily by following the steps presented by a wizard or by completing a property sheet. If you wish, you can display the SQL statements before executing them.

Procedure: Accessing OLAP Management

Follow these steps to start Oracle Enterprise Manager and access OLAP Management:

1. Open Oracle Enterprise Manager 10*g* Grid Control in your browser.

   The login page is displayed.

2. Enter a user name and password for Enterprise Manager.

   The Grid Control home page is displayed.

3. Click the Targets tab.

   The Hosts page is displayed.

4. Click the Database tab.

   The Databases page is displayed.

5. Click the link for the database you want to manage.

   The Oracle Database home page is displayed.

6. Click the Administration link.

The Database Administration page is displayed.

7. Look for the Warehouse heading. Links in the left column are used for Oracle OLAP. (The other Warehouse links are used only for relational warehouses that do not use the OLAP option. Do not use those links.)

   You see the types of objects that you can create: Cubes, OLAP dimensions, and measure folders. These links are for OLAP Management.

Defining Metadata for Dimension Tables

When creating OLAP metadata, you must first define the metadata objects for the dimension tables. These metadata objects are logical dimensions based on database dimension objects. You can use the Dimension Creation wizard or supply information directly in the Create Dimension dialog box.

Information That You Supply for Dimensions

To define a dimension, you provide all the information that will be needed to label and aggregate the measures dimensioned by it, including:

- The name of the dimension
- The tables that contain the data for the dimension
- The name of each level, and the columns that contain the data for each level
- The number and order of levels in each hierarchy
- Join keys for levels that are stored in separate tables
- The columns that contain attributes for the levels
- A display name and description for the dimension and each of its hierarchies, levels, and attributes

Time Dimension

Business analysis is performed on historical data, so fully defined time periods are vital. Your time dimension table must have columns for period end dates and time span. This information supports time-series analysis, such as comparisons with earlier time periods. If your schema does not have these columns, then you can define time as a normal dimension, but it will not support time-based analysis.

Typical levels and hierarchies for Time dimensions are suggested by the Dimension wizard, but you do not have to use them.

Procedure: Defining a Logical Dimension in the OLAP Catalog

Follow these steps to create a dimension and its associated levels, hierarchies, and attributes:

1. Start Oracle Enterprise Manager and access OLAP Management, as described in "Procedure: Accessing OLAP Management" on page 5-11.
2. Click the OLAP Dimensions link under Warehouse.

   If you have not already logged into the database, the Database Login page is displayed. Enter your login name and password for the database.

   The Search Objects page is displayed.
3. Click **Create**.

   The Create Dimension page is displayed.
4. Choose **Help** if you need further information.

Defining Metadata for Fact Tables

After you have defined the metadata objects for the dimension tables, you can create metadata objects for the fact tables. These metadata objects are measures and cubes. A cube is a collection of identically dimensioned measures. Cubes and measures are defined entirely in the OLAP metadata; there are no corresponding database objects.

> **Tip:** If you plan to calculate and store custom measures such as forecasts as a permanent part of your analytic workspace, you can add empty columns to the fact tables and define logical measures from those columns. Then the analytic workspace creation process will create and register all of the objects associated with the custom measures, so you only need to populate them. Refer to Chapter 11 for more information about this method of creating custom measures.

Information That You Supply for Cubes

When you define a cube, you identify information such as the following:

- The name of the cube and the fact table associated with it. All measures in a cube must be from a single fact table.

- The names of the dimensions and the levels in the dimension hierarchies that will be used in the cube.
- The names of the measures and the columns in the fact table where the values for each measure is stored.
- Default aggregation operators for each dimension of each measure (such as sum or average).
- Any calculation dependencies.

Procedure: Defining a Logical Cube in the OLAP Catalog

Follow these steps to create a cube:

1. If you have not done so already, start Oracle Enterprise Manager and access OLAP Management, as described in "Procedure: Accessing OLAP Management" on page 5-11.
2. Click the Cubes link under Warehouse.

   If you have not already logged into the database, the Database Login page is displayed. Enter your login name and password for the database.

   The Search Objects page is displayed.
3. Click **Create**.

   The Create Cube page is displayed.
4. Choose **Help** if you need further information.

Note: If you are creating OLAP Catalog metadata for use by the BI Beans running directly against a relational schema (that is, with no analytic workspace, then your last step is to open SQL*Plus Worksheet and issue this command:

```
EXECUTE CWM2_OLAP_METADATA_REFRESH.MR_REFRESH
```

For relational source data, be sure to create materialized views as described in Chapter 13.

Case Study: Creating Metadata for the GLOBAL Star Schema

The Global star schema conforms to all of the requirements of `CWM1`, so you can use the OLAP Management tool in Oracle Enterprise Manager.

If you have installed the Global schema, the OLAP Catalog metadata may be already defined. However, you can follow this example by creating the metadata in a different schema. All of the mappings between logical objects and source columns are described in Chapter 3. The following procedures explain how to define just one dimension and one cube.

Defining a Logical Time Dimension for the Global Schema

The TIMES_DIM table supports a single Calendar hierarchy with three levels (Month, Quarter, and Year) as described in "Dimension Table: TIME_DIM" on page 3-11. These are the steps to define a logical Time dimension using the Create Dimension wizard.

1. On the Add Dimension page, do the following:
   - For Name, type TIME.
   - For Schema, choose GLOBAL.
   - For Type, select **Time**.
2. On the Add Level page, do the following
   a. For Name, type YEAR.
   b. For Type, choose **Year**.
   c. For Table, choose **TIME_DIM**.
   d. Click **Populate Columns**.
   e. Move YEAR_ID from Available Columns to Selected Columns.
   f. Click **OK**.
   g. Repeat these steps for the Quarter and Month levels. Map Quarter to QUARTER_ID and MONTH to MONTH_ID.
3. On the Add Hierarchy page, do the following:
   a. For Name, type Calendar.
   b. Choose **Move All**.
   c. Use the arrow keys to order the levels like this:

```
Year
Quarter
Month
```

4. On the Create Dimension page, choose **Attributes**. Edit the Long_Description and Short_Description attributes and create the Time_Span and End_Date attributes. Map the attributes to the columns shown in "Global Time Dimension Mapping" on page 3-17.
5. On the Create Dimension page, choose **OLAP Options**. Type whatever descriptions you want to add.

When you have successfully created a dimension, it appears on the Dimensions page.

Defining a Logical Units Cube for the Global Schema

The `UNITS_HISTORY_FACT` table has a multi-column primary key, composed of four surrogate keys from the four dimension tables, and one measure (`UNITS`). These are the steps to define a logical Units cube. If you have installed the Global schema, this cube may be defined already. However, you can follow these steps by creating the cube under a different name or in a different schema.

1. On the Create Cube page, do the following:
   a. For Name, type `UNITS_CUBE`.
   b. For Display Name, type `Units Cube`.
   c. For Schema, choose **GLOBAL**.
   d. For Description, type your own description.
   e. For Fact Type, choose **Table**.
   f. For Fact Schema, choose **GLOBAL**.
   g. For Fact Table, choose **UNITS_HISTORY_FACT**.
2. Use the Add Dimension page to add each dimension (`CHANNEL`, `CUSTOMER`, `PRODUCT`, and `TIME`). Use the default properties and identify the appropriate foreign key columns in the fact tables.

Creating Metadata Using PL/SQL

The `CWM2` PL/SQL packages contain stored procedures that can create OLAP metadata for a variety of schema designs, as described in "Choosing a Tool for Creating OLAP Catalog Metadata" on page 5-8.

Before using these packages, make sure that you have performed any required preprocessing steps.

See Also:

- "SQL Scripts for OLAP Catalog Metadata" on page C-4 for a full example of using the `CWM2` PL/SQL packages.
- *Oracle OLAP Reference* for the comprehensive syntax of the `CWM2` packages.

CWM2 Packages for Creating OLAP Dimensions

The following packages contain procedures that create metadata for dimension tables:

- `CWM2_OLAP_DIMENSION` contains procedures for creating dimensions.
- `CWM2_OLAP_HIERARCHY` contains procedures for creating hierarchies for dimensions.
- `CWM2_OLAP_LEVEL` contains procedures for creating levels for dimensions and for associating levels with hierarchies.
- `CWM2_OLAP_LEVEL_ATTRIBUTE` contains procedures for creating level attributes and associating them with levels.
- `CWM2_OLAP_DIMENSION_ATTRIBUTE` contains procedures for creating dimension attributes and associating them with dimensions.

CWM2 Packages for Creating Cubes

The following packages contain procedures that create metadata for fact tables:

- `CWM2_OLAP_CUBE` contains procedures for creating the multidimensional structure of cubes.
- `CWM2_OLAP_MEASURE` contains procedures for creating measures and associating them with cubes.

CWM2 Package for Mapping Metadata

The `CWM2_OLAP_TABLE_MAP` package contains procedures that map logical metadata entities to their physical data source. The data may be stored in relational tables, or it may be represented by relational views. When the dimension tables and fact tables are defined as views, the actual data may reside in analytic workspaces.

CWM2 Package for Creating Level-Based Dimension Tables

The CWM2_OLAP_PC_TRANSFORM package contains a procedure for transforming parent-child dimension tables to level-based dimension tables. This conversion is necessary if the dimension will be accessed by the BI Beans.

CWM2 Packages for Classification and Validation

The following packages contain procedures for creating measure folders and validating OLAP metadata:

- CWM2_OLAP_CATALOG provides procedures for creating and maintaining measure folders.
- CWM2_OLAP_VALIDATE provides procedures for validating OLAP Catalog metadata.
- CWM2_OLAP_METADATA_REFRESH provides procedures for refreshing metadata tables that support queries by the BI Beans.

6

Creating an Analytic Workspace

This chapter explains how to create a standard form analytic workspace that performs optimally. It explores the various design decisions that affect how the data is stored in an analytic workspace, and thus how quickly it can be retrieved. These decisions are discussed in the context of using the wizards in Analytic Workspace Manager.

This chapter contains the following topics:

Methods of Creating a Workspace

With OLAP Catalog metadata defined for your data source, you can choose from these methods for creating an analytic workspace:

- **Analytic Workspace Manager**. The Create Analytic Workspace wizard is the easiest method of creating an analytic workspace, and the one described in this

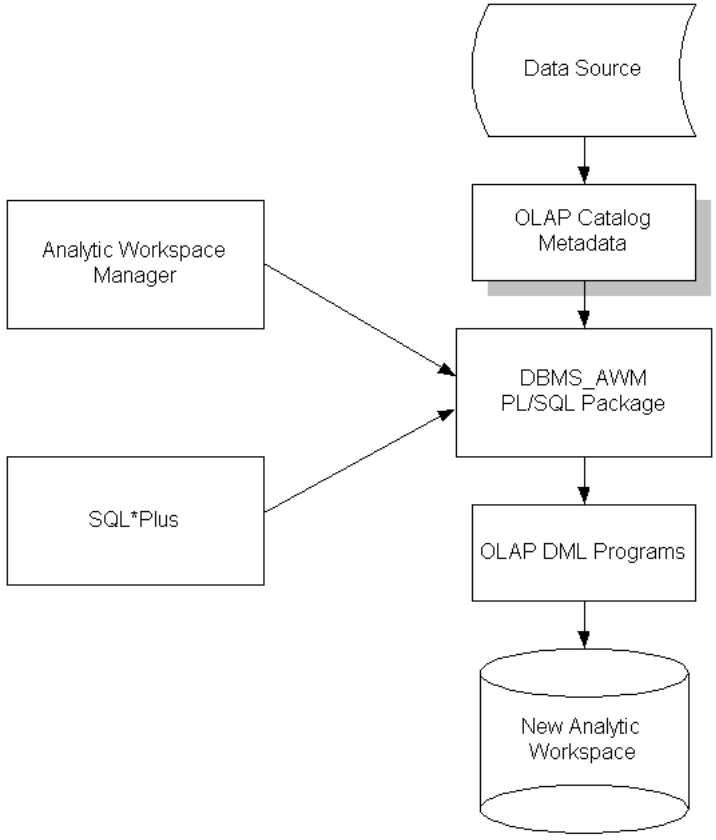
chapter. The wizard generates a SQL script of DBMS_AWM statements, which you can choose to execute immediately or save for execution later.

- **DBMS_AWM PL/SQL package**. The DBMS_AWM package enables you to automate the build process in a script and schedule it to run overnight in a batch window. For most production systems, this is the preferred practice.

  You can use the Create Analytic Workspace wizard to generate a basic script, then edit the script to make any changes. This is a much less tedious and time-consuming method than coding the scripts entirely by hand. The DBMS_AWM package provides greater flexibility in designing your analytic workspace than the graphical methods.

Figure 6–1 shows the relationships among the tools used in the build process.

Figure 6–1 Components Used With the OLAP Catalog to Build an Analytic Workspace



Introduction to Analytic Workspace Manager

Analytic Workspace Manager is a primary tool for creating, developing, and managing analytic workspaces. It is a Java application that provides a graphical user interface to several PL/SQL packages and to the OLAP DML.

The console, or main window, provides two views: the OLAP Catalog View, and the Object View. You can switch between these two views using the View menu. In addition, there are menus, a toolbar, a navigation pane, and a display pane. When you select an object in the navigator pane, the display pane 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.

You can also conduct an interactive session using the OLAP DML, by opening OLAP Worksheet. You can switch between the console and OLAP Worksheet, and have an up-to-date view of your workspace in each one, because they share the same session.

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

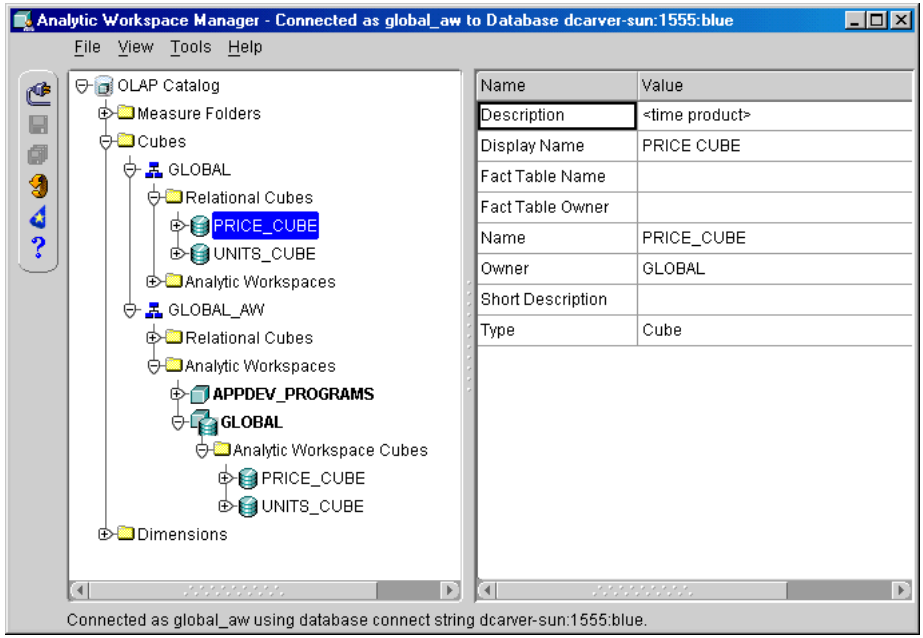
OLAP Catalog View

OLAP Catalog View displays all OLAP Catalog metadata that you own or that you have been given access rights to read, both `CWM1` and `CWM2`. The view lists the primary logical objects: measure folders, cubes, and dimensions. You cannot create or change the metadata; to do that, you must use Oracle Enterprise Manager, Oracle Warehouse Builder, or the `CWM2` PL/SQL procedures.

After you create an analytic workspace, the OLAP Catalog View shows the instantiated cubes in the analytic workspace in addition to the logical cubes in the relational model. You can augment the metadata by creating and deploying an aggregation plan on a workspace cube. An aggregation plan is associated with a workspace cube and is stored in the analytic workspace. It provides the rules for solving the data in a cube so that values are calculated for all levels.

Figure 6–2 shows the OLAP Catalog View. A relational cube named `PRICE_CUBE` in the `GLOBAL` schema is selected in the view, so the pane to the right displays information about the cube. Notice that an analytic workspace was created in the `GLOBAL_AW` schema from the relational cubes defined in `GLOBAL`.

Figure 6–2 OLAP Catalog View in Analytic Workspace Manager



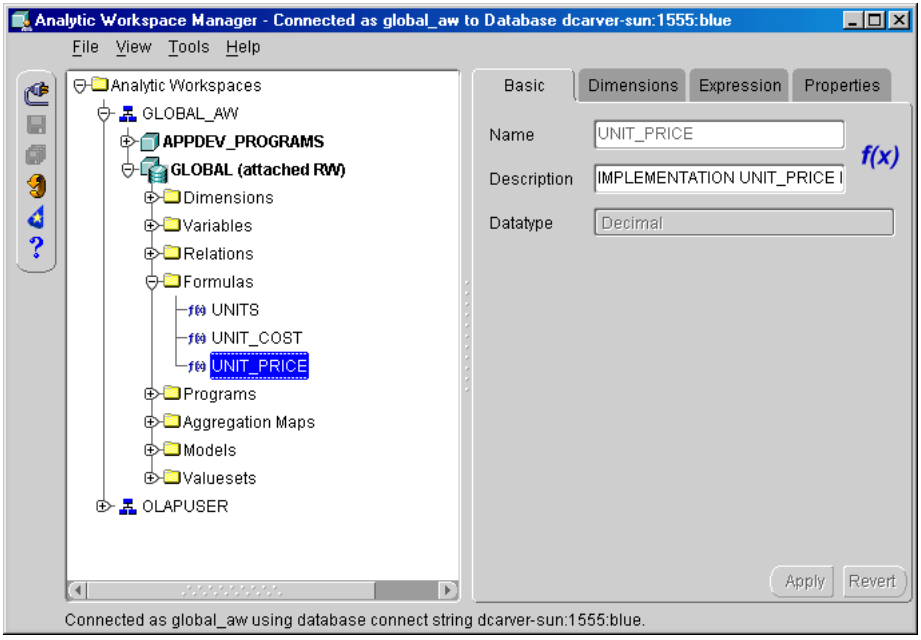
Object View

The Object View provides a graphical user interface to the OLAP DML. You can create, modify, and delete individual workspace objects. Using the tools available through the Object View, you can modify a workspace made by the Create Analytic Workspace wizard.

The Object View is useful primarily for managing the definitions of workspace objects. If you need to manage the contents of a workspace object or execute a program, or if you are adept at using the OLAP DML, then you can open OLAP Worksheet and have full use of the OLAP DML.

Figure 6–3 shows the Object View. A formula named `UNIT_PRICE` is currently selected, so the right pane shows information about the formula.

Figure 6–3 Object View in Analytic Workspace Manager

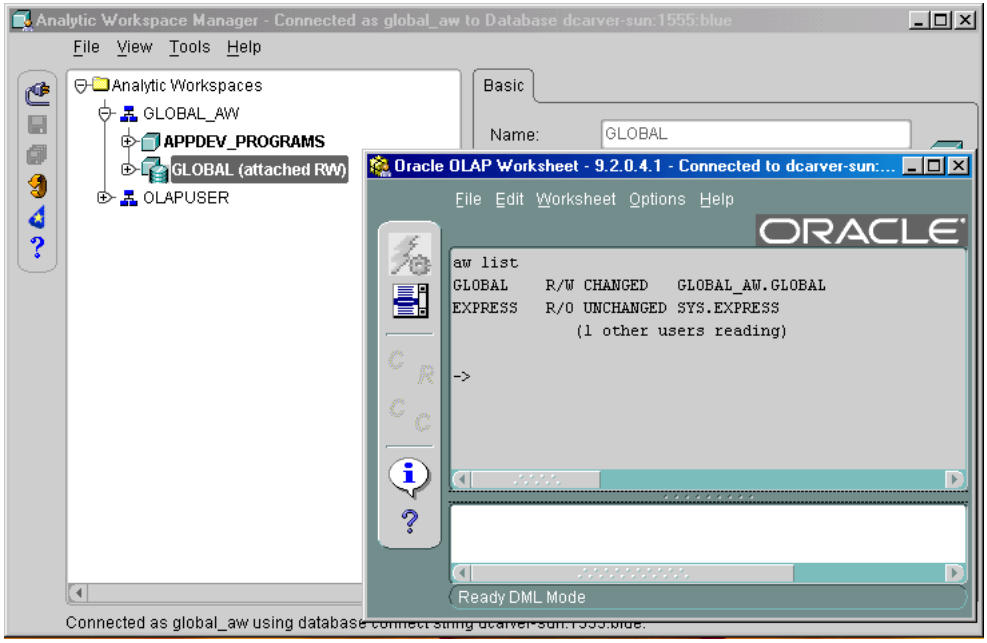


OLAP Worksheet

OLAP Worksheet opens in a separate window from the Analytic Workspace Manager console. This window provides menus, a toolbar, an input pane for OLAP DML commands on the bottom, and an output pane on the top.

Figure 6–4 shows OLAP Worksheet opened from Analytic Workspace Manager. Notice that the `GLOBAL` workspace is attached with read/write access in both OLAP Worksheet (as shown by the `AW LIST` command) and Analytic Workspace Manager (as shown by the Object View). The two applications share the same session.

Figure 6–4 OLAP Worksheet Opened From Analytic Workspace Manager



Opening a Database Connection With Analytic Workspace Manager

To connect to a database:

1. From the File menu, choose **Connect**.

   The Connect to Database dialog appears.

2. Enter your database user name, password, and service. Then select **OK**.

   Specify the service in the form *host:port:sid* (for example, `myhost-sun:1521:rel10g`).

Creating a Standard Form Workspace Using Analytic Workspace Manager

Using the Create Analytic Workspace wizard, you can create a workspace from one or more cubes in the OLAP Catalog. The resulting workspace is in **database standard form**, which is described in Appendix A.

The Create Analytic Workspace wizard provides reasonable defaults so that you can create an analytic workspace with the minimum number of decisions. By accepting the default settings, you create a physical storage model for your data that is appropriate for many types of data. If you are new to OLAP-type analysis, you may want to start by creating an analytic workspace with the default settings, and find out from using it whether its performance characteristics are acceptable.

However, good performance is critical in a production system. To create a workspace with the best performance, explore the characteristics of your data and set the advanced storage options appropriately for its requirements.

Choosing a Schema for the Analytic Workspace

Always create an analytic workspace in a separate schema from the relational tables. This practice prevents conflicts in defining unique names within a single namespace.

Setting Advanced Storage Options

Storage settings can have a huge impact on performance. When they are set incorrectly, you will experience poor performance for both data loads and run-time analysis. The default settings are appropriate when your data has the expected characteristics: a dense time dimension, and random sparsity (less than 30% non-null values) across the other dimensions.

Defining a Composite Dimension

Composite dimensions are a means of reducing the amount of space needed to store sparse data. Without composites, an analytic workspace stores a value for each combination of dimension members, even if that value is null (`NA`). With composites, only the combinations of dimension members with real data are stored. A combination of dimension members for which there is a data value is called a **tuple**. The addition of a data value to a variable automatically triggers the creation of a new composite tuple.

The order in which the dimensions are listed in a composite controls the order in which the composite tuples are stored. The dimensions are typically ordered by size (that is, number of members) so that the largest is the fastest varying and the smallest is the slowest varying. The composite tuples are stored so that the fastest varying dimension members are clustered together, and the slowest varying dimension members are the most widely separated. In the composite definition, the first dimension is the fastest varying, and the last dimension is the slowest varying.

For example, in a composite for <CUSTOMER PRODUCT CHANNEL>, CUSTOMER is the fastest varying and CHANNEL is the slowest varying.

Analytic Workspace Manager assumes that your data is sparse (that is, few data values compared to the number of cells in the cube), and will create one composite for each cube. The composite includes all dimensions except Time. Measures are often defined with the Time dimension first, regardless of its size, to facilitate time-based analysis and data refreshes.

If your Time dimension is sparse, then define a composite that includes Time. If another dimension is dense, then define a composite that excludes that dimension. (If you have a very small, dense dimension, you can include it as the last dimension in the composite.) Remember to order the dimensions in the composite from the one with the most members to the one with the fewest members.

Ordering the Dimensions in a Cube

The order in which the dimensions are listed in a cube affects performance because it determines the way the data is stored on disk. The first dimension in a cube is the fastest-varying dimension, and the last dimension is the slowest-varying dimension. The data for each measure in a cube is stored as a linear stream, in which the values of the fastest-varying dimension are clustered together.

Data storage is typically optimized for loads. List the dense dimension (such as Time) before the composite. If there is more than one dense dimension, then list the largest one first.

Setting the Segment Size

A segment is the amount of contiguous disk space reserved for storing the data in a measure. Performance is best when all of the data for a measure is stored in one segment.

Analytic Workspace Manager creates a segment large enough for the initial load. This setting defines a segment exactly large enough for the Time dimension (which is outside the composite) and plentifully large enough to store composite tuples as they are created. When you update your data, a second segment is created that is the same size as the first segment. Subsequent updates will be stored in this segment until it reaches its capacity in one of the dimensions, at which time a third, equal sized segment will be created, and so forth.

When you manually set the segment size, you can create a segment that permits future growth. Be sure to specify ample space, particularly for the fastest-varying dense dimension (that is, the first one in the list). Keep in mind, however, that you

are reserving disk space that will not be available for other uses. Do not create a segment far in excess of realistic growth.

You can only specify the segment size for multidimensional variables. You cannot specify the segment size for variables dimensioned only by a composite or by a single dimension.

Choosing Build Options

The Create Analytic Workspace wizard enables you to decide how much data you want to load and when you want to initiate the load.

You can create an analytic workspace with all of the object definitions and several choices of data:

- No data
- All data from the dimension tables but none from the fact tables
- All data from the dimension tables and the fact tables

Partial builds enable you to make manual changes to the object definitions before loading the data. To load the data later, use the Refresh Analytic Workspace wizard.

Generating Scripts

The option of generating SQL scripts is particularly useful if you are working with a large amount of data. Scripts enable you to:

- Make changes to the build process.
- Schedule the build to run overnight in a batch window.

Basic Steps for Creating a Standard Form Workspace

To create a workspace in database standard form:

1. Configure your database instance for OLAP use. Define permanent, temporary, and undo tablespaces, and set the database parameters to values appropriate for data loads. Refer to Chapter 12 for details.
2. Define a user who will own the analytic workspace. Grant the user the `OLAP_USER` role and `SELECT` privileges on the source data tables.

   While you can create the workspace in the same schema as the relational tables, doing so causes problems in defining unique names within a single namespace.

3. Examine the sparsity characteristics of your data. The default data storage settings are appropriate for data that is dense across the Time dimension and sparse across all other dimensions. If your data has these sparsity characteristics, or you cannot determine them, then use the default data storage settings. Otherwise, when you run the Create Analytic Workspace wizard, define a **composite** that is appropriate for your data cube.
4. Open Analytic Workspace Manager and connect to your database instance as the user you defined earlier for this purpose.
5. If you want to generate log files, from the Tools menu choose **Configuration**. Click **Help** for further information.

   Log files store the messages that are generated by the various wizards. They do not provide additional information, but they are useful if the wizard fails and you want to explore the reasons for the failure.
6. In the OLAP Catalog View, verify that you have defined dimensions, hierarchies, measures, and cubes for the source data, and that you have access to these logical objects from your current session.
7. From the Tools menu, choose **Create Analytic Workspace Using Wizard**. Complete the steps of the wizard. If you need to define a composite, be sure to select the advanced storage options.

   Click the **Help** button to get specific information about each step.
8. You can enable the workspace for the BI Beans either now or later. You may want to postpone enabling until after you enhance your analytic workspace with aggregate data and custom measures.

   If the workspace will support other types of applications, then you can enable it later using the appropriate wizard.
9. From the File menu, choose **Save**. This option commits all changes to the database made during this session.
10. If you chose a build option that defines objects without loading all of the data, then run the Refresh Analytic Workspace wizard when you are ready to complete the build.

When you have finished, you will have an analytic workspace populated with the detail data fetched from your relational star or snowflake schema.

Case Study: Creating the Global Analytic Workspace

The following case study explains the choices made in creating an analytic workspace from the GLOBAL star schema. Chapter 3 describes the tables.

Defining the GLOBAL_AW Workspace User

This example creates the GLOBAL analytic workspace in a different schema from the source tables. Example 6–1 lists the SQL commands to define the GLOBAL_AW user with sufficient access rights to use Analytic Workspace Manager and to access the GLOBAL star schema. Alternatively, you can define users through Oracle Enterprise Manager.

Example 6–1 SQL Script for Defining the GLOBAL_AW User

```
CREATE USER "GLOBAL_AW" PROFILE "DEFAULT"
    IDENTIFIED BY "global_aw" DEFAULT TABLESPACE "GLOBAL"
    TEMPORARY TABLESPACE "OLAPTEMP"
    QUOTA UNLIMITED ON "GLOBAL"
    QUOTA UNLIMITED ON "OLAPTEMP"
    ACCOUNT UNLOCK;

GRANT OLAP_USER TO GLOBAL_AW;

GRANT SELECT ON global.channel_dim TO global_aw;
GRANT SELECT ON global.product_dim TO global_aw;
GRANT SELECT ON global.customer_dim TO global_aw;
GRANT SELECT ON global.time_dim TO global_aw;
GRANT SELECT ON global.units_history_fact TO global_aw;
GRANT SELECT ON global.price_and_cost_history_fact TO global_aw;
```

Examining Sparsity Characteristics for GLOBAL

By using SQL SELECT commands with the COUNT and COUNT(DISTINCT) functions, you can estimate how dense the resulting multidimensional cubes will be in the analytic workspace.

The PRICE_AND_COST_HISTORY_FACT table has 1407 rows with values for both UNIT_PRICE and UNIT_COST out of a possible 1728 dimension value combinations (48 months * 36 products). The Price cube (which is mapped to the PRICE_AND_COST_HISTORY_FACT table) is 80% dense, and so a composite will actually slow performance rather than improve it. A dense cube is fairly unusual, and the Create Analytic Workspace wizard does not support it, nor does the

DBMS_AWM PL/SQL package. Thus, you will need to make some modifications to the workspace object definitions before loading the data.

The UNITS_HISTORY_FACT table has 110,166 rows with values for UNITS out of a possible 316,224 dimension value combinations (48 months * 36 products * 61 customers * 3 channels). This cube is 30% dense, which is sufficiently sparse for a composite.

Because Time is aggregated to the month level, the Time dimension is probably dense. This cube can use the default composite.

Running the Create Analytic Workspace Wizard

Make these choices when running the Create Analytic Workspace wizard:

- On the Choose Data Loading Options page, do the following:
  - Choose the second build option, **Build analytic workspace and load dimensions**. This choice enables you to make modifications to the variable definitions before loading the data.
  - Clear the **Generate unique keys** box. The dimension tables use surrogate keys for all levels to assure unique dimension values.
- On the Choose Advanced Storage and Naming Options page, no prefix is needed when naming the workspace objects because the analytic workspace is being created in a different schema from the relational tables. The cube name prefix is optional, but may be useful when an analytic workspace contains multiple cubes.

The selected build option causes the Create Analytic Workspace wizard to define all of the workspace objects and fetch all of the data from the dimension tables. The measures are defined but do not have data. You can make changes to the object definitions before loading the data.

(An alternative approach to the one taken in this example is to generate a SQL script and modify the calls to DBMS_AWM, so that the analytic workspace is generated correctly and does not require modification.)

Manually Changing Object Definitions

Measure names such as UNIT_PRICE and UNIT_COST are given to the formulas in the analytic workspace that are used to aggregate the variables. The data itself is stored in variables with names like UNIT_PRICE_VARIABLE and UNIT_CUBE_VARIABLE.

Using OLAP Worksheet and the OLAP DML, you can make two changes to UNIT_PRICE_VARIABLE and UNIT_COST_VARIABLE:

- **Remove the composite.** The Price cube is 80% dense, as explained in "Examining Sparsity Characteristics for GLOBAL" on page 6-12. UNIT_PRICE_VARIABLE and UNIT_CUBE_VARIABLE store the data for the Price cube. Because these variables are dense, defining them with a composite would actually slow performance.
- **Change the data type.** The data for the Price cube is easily handled by the SHORTDECIMAL data type. Use DECIMAL (8 bytes) and SHORTDECIMAL (4 bytes) whenever possible to save storage space and maximize performance. DECIMAL is the default numeric data type. For a full list of data types, search Analytic Workspace Manager Help.

These are significant changes that require the variables to be redefined, not simply modified.

Take these steps to redefine the variables:

1. In the Object View, attach the GLOBAL analytic workspace with read/write access.
2. Choose **OLAP Worksheet** from the Tools menu.
3. In the lower pane (the query window) enter this command to display the object definitions for the two variables:

   ```
   FULLDSC price_cube_unit_price_variable price_cube_unit_cost_variable
   ```

4. Select the full object definitions from the top pane (the response window) and paste them into a text editor.
5. Edit the definitions so that they look like the following code example. Note the DELETE command at the beginning, the changes to the data type and dimensionality in the DEFINE commands, the addition of a CONSIDER command after each DEFINE, and the standard form PROPERTY commands listed with the full command on a single line.

   ```
   DELETE PRICE_CUBE_UNIT_PRICE_VARIABLE PRICE_CUBE_UNIT_COST_VARIABLE

   DEFINE PRICE_CUBE_UNIT_PRICE_VARIABLE VARIABLE SHORTDECIMAL <TIME PRODUCT>
   CONSIDER PRICE_CUBE_UNIT_PRICE_VARIABLE
   PROPERTY 'AW$CLASS''EXTENSION'
   PROPERTY 'AW$CREATEDBY' 'AW$CREATE'
   PROPERTY 'AW$LASTMODIFIED' '20MAY03_10:37:08'
   PROPERTY 'AW$PARENT_NAME' 'PRICE_CUBE_UNIT_PRICE'
   ```

```
PROPERTY 'AW$ROLE' 'VARIABLE'
PROPERTY 'AW$SEGWDTH_CMD' 'chgdfn
GLOBAL_AW.GLOBAL!PRICE_CUBE_UNIT_PRICE_VARIABLE segwidth 120 72'
PROPERTY 'AW$STATE' 'CREATED'

DEFINE PRICE_CUBE_UNIT_COST_VARIABLE VARIABLE SHORTDECIMAL <TIME PRODUCT>
CONSIDER PRICE_CUBE_UNIT_COST_VARIABLE
PROPERTY 'AW$CLASS' 'EXTENSION'
PROPERTY 'AW$CREATEDBY' 'AW$CREATE'
PROPERTY 'AW$LASTMODIFIED' '20MAY03_10:37:08'
PROPERTY 'AW$PARENT_NAME' 'PRICE_CUBE_UNIT_COST'
PROPERTY 'AW$ROLE' 'VARIABLE'
PROPERTY 'AW$SEGWDTH_CMD' 'chgdfn
GLOBAL_AW.GLOBAL!PRICE_CUBE_UNIT_COST_VARIABLE segwidth 85 72'
PROPERTY 'AW$STATE' 'CREATED'
```

6. Save the file in, or move it to, a disk directory that has been defined as a directory object in the database. Then use the OLAP DML `INFILE` command to execute the file. `INFILE` is equivalent to the SQL `@` command.

   ```
   INFILE 'directory_object/filename'
   ```

   Check the response window for errors. To fix them, edit the file and execute it again.

7. To save these changes, type these commands into the query window and execute them:

   ```
   UPDATE
   COMMIT
   ```

Completing the Build

After all of the changes are made to the workspace object definitions, run the Refresh wizard to load the data, as described in "Refreshing the Data in an Analytic Workspace" on page 6-31 The dimensions do not need to be refreshed; only the cubes.

The workspace can be enabled now or after deploying an aggregation plan.

Case Study: Creating the Sales History Analytic Workspace

Although Global is used for most of the examples in this manual, Sales History has a very different set of data characteristics and demonstrates a correspondingly different set of build choices.

Sales History (SH) is a sample star schema that is delivered with your Oracle Database, along with a fully defined logical model stored in the OLAP Catalog. The SH schema has two cubes, SALES and COSTS. The SALES cube has five dimensions, and the COSTS cube uses two of these dimensions. This case study uses only the SALES cube.

> **See Also:** *Oracle Database Sample Schemas* for a full description of Sales History.

Defining Startup Parameters for the SH Build

When building a large analytic workspace, the startup parameters for the Oracle Database affect how quickly the build proceeds. Example 6–2 shows a few of the settings in the init.ora file for building Sales History. For more information about these settings, refer to Chapter 12.

Example 6–2 Startup Parameters for Building Sales History

```
UNDO_TABLESPACE=OLAPUNDO
UNDO_MANAGEMENT=AUTO
PGA_AGGREGATE_TARGET=128M
```

Defining Tablespaces for SH

While the GLOBAL analytic workspace has less than a million cells for base-level data in its largest cube, the Sales History COST cube has over 235 trillion. This makes Sales History quite large for a sample schema, yet it is small to average for a real application. It is sufficiently large for the build to fail unless resources have been allocated specifically for its use. The build needs adequate temporary and permanent tablespaces:

- Define a tablespace just for use by the Sales History analytic workspace, which is sufficiently large to hold the base-level data, stored aggregates, forecast data, and so forth. If possible, define extension files on separate physical disks. For the best performance, do not use the same tablespace as the star schema.
- Define a temporary tablespace that is sufficiently large to hold the data for the SALES cube. Use a small EXTENT MANAGEMENT SIZE value, such as 256K.

Example 6–3 shows how the tablespaces might be defined for Sales History.

Example 6–3 SQL Script for Defining Tablespaces for the Sales History Analytic Workspace

```
/* Create a permanent tablespace on four disks */
CREATE TABLESPACE sh_aw DATAFILE '/disk1/oradata/sh_aw1.dbf' SIZE 64M AUTOEXTEND
ON NEXT 64M MAXSIZE 1024M EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO;

ALTER TABLESPACE sh_aw ADD DATAFILE '/disk2/oradata/sh_aw2.dbf' SIZE 64M REUSE
AUTOEXTEND ON NEXT 64M MAXSIZE 1024M,'/disk3/oradata/sh_aw3.dbf' SIZE 64M REUSE
AUTOEXTEND ON NEXT 64M MAXSIZE 1024M,'/disk4/oradata/sh_aw4.dbf' SIZE 64M REUSE
AUTOEXTEND ON NEXT 64M MAXSIZE UNLIMITED;

/* Create a temporary tablespace on four disks */
CREATE TEMPORARY TABLESPACE sh_temp TEMPFILE '/disk1/oradata/sh_aw.tmp' SIZE 64M
REUSE AUTOEXTEND ON NEXT 64M MAXSIZE 1024M EXTENT MANAGEMENT LOCAL UNIFORM SIZE
256K;

ALTER TABLESPACE sh_temp ADD TEMPFILE '/disk2/oradata/sh_aw2.tmp' SIZE 64M REUSE
AUTOEXTEND ON NEXT 64M MAXSIZE 1024M,'/disk3/oradata/sh_aw3.tmp' SIZE 64M REUSE
AUTOEXTEND ON NEXT 64M MAXSIZE 1024M,'/disk4/oradata/sh_aw4.tmp' SIZE 64M REUSE
AUTOEXTEND ON NEXT 64M MAXSIZE UNLIMITED;
```

Examining the Sparsity Characteristics of SH Data

The data in the SH relational schema is extremely sparse. Many dimension keys are never used as foreign keys in the SALES fact table, much less used in all possible combinations with the other four dimensions. For example, CUSTOMERS.CUST_ID has 5100 values, of which only 2557 are used in the SALES.CUST_ID column.

Time is also a sparse dimension, with only 1075 of 1826 dimension members used. Thus, TIMES_DIM must be included in the composite. You can define a composite with all five dimensions by choosing Advanced Storage Options. List TIMES as the first dimension (the fastest varying) in the composite, to facilitate time-based analysis and data maintenance, even though it is smaller than PRODUCTS and CUSTOMERS. List the other dimensions from largest to smallest. This information is easily obtained by issuing a SELECT COUNT(*) on the dimension tables.

Managing the SH Build

Because SH is large, you may want to manage these aspects of the build:

- Time: Execute the build during off-peak hours. To do this, generate a SQL script for the build instead of creating it immediately.
- Progress Monitor: Add comments to the SQL script so that you can monitor its progress. If the build fails for any reason, or if you need to interrupt it, you can restart the script from where the build stopped.
- Permanent Tablespace Size: If possible, define some measures with smaller data types, as described in "Manually Changing Object Definitions" on page 6-13. This type of change requires you to choose one of the partial build options.

  For a description of data types, search Help in Analytic Workspace Manager.

Running the Create Analytic Workspace Wizard

Make these choices in the Create Analytic Workspace wizard for building Sales History, based on the previous discussion:

- On the Choose Data Loading Options page, do the following:
  - Choose the second build option, **Build analytic workspace and load dimensions**. This choice enables you to make modifications to the variable definitions before loading the data.
  - Clear the **Generate unique keys** box. The dimension tables use surrogate keys for all levels to assure unique dimension values.
- On the Choose Advanced Storage and Naming Options page, select **Advanced Storage Options**. This choice enables you to define a composite that includes the `TIMES` dimension.

  No prefix is needed because the analytic workspace is being created in a different schema from the relational tables. The cube name prefix is optional, but may be useful when an analytic workspace contains multiple cubes.
- When creating the composite, include all of the dimensions and put them in this order. Do *not* specify a segment size, because a very large segment is allocated automatically for composites.

  ```
  TIMES
  PRODUCTS
  CUSTOMERS
  PROMOTIONS
  CHANNELS
  ```

- On the Choose Create and Enablement Options page, select **Save Script to File**. You do not need to enable the workspace at this time.

Building the Sales History Analytic Workspace

Take these steps to build the Sales History analytic workspace:

1. Run the build script. After it has completed successfully, you have an analytic workspace with all of the dimensions, hierarchies, levels, and attributes from the dimension tables.
2. Make any changes or additions to the object definitions.
3. Run the Refresh wizard to load the data, as described in "Refreshing the Data in an Analytic Workspace" on page 6-31. The dimensions do not need to be refreshed; only the cubes.

The workspace can be enabled now or after deploying an aggregation plan.

Generating Aggregate Data

An analytic workspace initially contains only the detail data from the relational schema. However, it also contains the hierarchies, levels, and parent relations that are needed to aggregate the data. The aggregate data in an analytic workspace replaces the use of materialized views; all of the aggregate data is created in the analytic workspace. To optimize run-time performance, you must generate and store some aggregate data.

Strategies for Calculating Aggregates

A data cube in an analytic workspace can be solved at two distinct times:

- **At run-time when needed.** The cells for the aggregate values are `NA` (null) until a query requests the aggregate values. The aggregates are then calculated in response to the query. This type of aggregation is referred to as **on-the-fly** or **run-time** aggregation. Run-time aggregation slows querying time since the data must be calculated instead of just retrieved, but it does not require storage in a permanent tablespace for the aggregate values.
- **As a data maintenance procedure.** The DBA calculates the aggregate values and stores them in the analytic workspace for all users to share. This type of aggregate data is sometimes call **precomputed** or **stored** aggregates. Stored aggregates support the fastest querying time, but increase the size of the analytic workspace and therefore the size of the relational database. The amount of stored data may also be limited by the amount of time available for data maintenance, which is typically limited to a batch window. Fully

materializing a cube may simply take more time than the batch window permits.

When dimensions have multiple hierarchies or the hierarchies have many levels, fully aggregating the measures increases the size of your analytic workspace (and thus your database) geometrically. At the same time, much of the intermediate level data may be accessed infrequently or not at all.

A typical strategy is to aggregate some of the data as a data maintenance procedure and the rest of the data on demand. This strategy is called **skip-level aggregation**. The data cube is presented to the application fully solved, with no detectable difference between the values that were retrieved from storage and the values that were calculated for the query. When skip-level aggregation is done correctly, the time to calculate the unsolved levels is negligible.

How to Select Levels to Pre-Aggregate and Store

A good strategy for identifying levels for pre-aggregation is to determine the ratio of dimension members at each level, and to keep the ratio of members to be rolled up on the fly at approximately 10:1. This ratio assures that all answer sets can be returned quickly. Either the data is stored in the analytic workspace, or it can be calculated by rolling up 10 values at a time into each aggregate value.

This 10:1 rule is best applied with some judgment. You might want to permit a higher ratio for levels that you know are seldom accessed. Or you might want to pre-calculate levels at a lower ratio if you know they have heavy use.

About Aggregation Plans

An analytic workspace that was created by the Create Analytic Workspace wizard has a default set of rules, called an **aggregation plan**, for each cube. A default plan specifies that:

- No aggregate levels are stored. All aggregate data is calculated at run-time as necessary to return an answer set to a query.
- All measures in the cube use the default plan.

The default aggregation plans assure that the measures in a cube are always presented to an application with fully solved data; that is, all levels of all hierarchies in an answer set are populated. However, because the default plans specify that all levels must be calculated during the user's session, their use typically causes unacceptably slow performance.

You can define and deploy aggregation plans that precalculate some of the data. Each measure can have its own aggregation plan, or any number of measures in a cube can share the same one. For each aggregation plan that you create, you must specify:

- Which aggregate levels are stored.
- Which measures in the cube use the plan.

An aggregation plan does not take effect until it is deployed. Deployment creates and modifies objects in the analytic workspace to support the aggregation plan, and then calculates all stored aggregate levels. While you can create an aggregation plan in a few minutes, deployment can take much longer, depending on the amount of data that needs to be calculated and the available resources. You may want to schedule aggregation for off-peak hours.

How to Create and Deploy an Aggregation Plan

An aggregation plan can be used for all the measures in a cube, or just for selected measures.

Creating an Aggregation Plan

To create an aggregation plan:

1. Expand the Cubes folder of OLAP Catalog View so that you can see the names of the analytic workspaces in your schema.
2. Right-click the name of your workspace.
3. Choose **Create Aggregation Plan Using Wizard**. Complete the steps of the wizard.

   Click the **Help** button to get specific information about each step.

The aggregation plan is a permanent part of your analytic workspace until you explicitly delete it.

Changing the Aggregation Operator

The Aggregation Plan wizard always specifies `SUM` as the method of aggregation. However, you can change the method to one of these operators:

```
AVERAGE
FIRST
LAST
```

```
MAX
MIN
```

A procedure in the DBMS_AWM package changes the operator in an existing aggregation plan. The syntax of the procedure call is this:

```
EXECUTE DBMS_AWM.SET_AWCUBEAGG_SPEC_AGGOP('aggplan', 'aw_owner', 'aw_name',
'cube', 'measure', 'dimension', 'operator')
```

For the BI Beans to be aware of this change, you must either re-enable your analytic workspace or manually execute the CWM2_OLAP_METADATA_REFRESH.MR_AC_REFRESH procedure.

See the example in "Aggregating the Global Price Cube" on page 6-24. For a full discussion of the syntax, refer to the *Oracle OLAP Reference*.

Deploying an Aggregation Plan

Deployment first deletes any previously aggregated data, then solves the data for the specified levels.

To deploy an aggregation plan:

1. Expand the OLAP Catalog View sufficiently to see the names of the aggregation plans for your workspace.
2. Right-click the name of an aggregation plan, and choose **Deploy Aggregation Plan Using Wizard**. Complete the steps of the wizard.

You can edit an aggregation plan, but no change is made to your data until you redeploy the modified plan. Similarly, you can delete an aggregation plan, but no change is made until you deploy another plan for the same measures.

You must redeploy your aggregation plans whenever you refresh the data. Refer to "Refreshing the Data in an Analytic Workspace" on page 6-31 for further information.

Case Study: Aggregating Data in the GLOBAL Analytic Workspace

GLOBAL contains two cubes. Aggregates for the Units cube are summed, which is the default, but aggregates for the Price cube are averaged. All measures within each cube are aggregated in the same way.

Identifying Levels for Precalculation

To identify the levels to be precalculated, you must know the number of dimension members at each level. You can easily acquire this information using either SQL statements or OLAP DML commands.

For example, this SQL statement:

```
SELECT COUNT(DISTINCT year_id) FROM global.time_dim;
```

and this OLAP DML command in the GLOBAL analytic workspace:

```
SHOW NUMLINES(LIMIT(time TO time_levelrel EQ 'Year'))
```

both return the number of TIME dimension members at the Year level.

Global is a very small data set, so few adjacent levels have a 10:1 ratio of dimension members. Table 6–1 identifies the levels to be calculated and stored in the analytic workspace.

Table 6–1 Precalculated Levels in the Global Workspace

Dimension	Level	Members	Precalculate
TIME	Month	60	X
TIME	Quarter	20	--
TIME	Year	5	X
CUSTOMER	Ship_To	61	X
CUSTOMER	Account	24	--
CUSTOMER	Market_Segment	5	X
CUSTOMER	Total_Market	1	--
CUSTOMER	Warehouse	11	--
CUSTOMER	Region	3	X
CUSTOMER	All_Customers	1	--

Table 6–1 (Cont.) Precalculated Levels in the Global Workspace

Dimension	Level	Members	Precalculate
PRODUCT	Item	36	X
PRODUCT	Family	9	--
PRODUCT	Class	2	X
PRODUCT	Total_Product	1	--
CHANNEL	Channel	3	X
CHANNEL	All_Channels	1	--

Aggregating the Global Price Cube

Take these steps to aggregate the Price cube.

1. Run the Create Aggregation Plan wizard and create a plan for PRICE_CUBE.
2. In the Object View, attach the GLOBAL analytic workspace in read/write mode.
3. From the Tools menu, choose **OLAP Worksheet**.
4. From the Options menu of OLAP Worksheet, select **SQL Mode**.
5. Type these SQL statements in the query window.

```
EXECUTE DBMS_AWM.SET_AWCUBEAGG_SPEC_AGGOP('price_aggplan', 'global_aw',
'global', 'price_cube', 'unit_price', 'time', 'AVERAGE')

EXECUTE DBMS_AWM.SET_AWCUBEAGG_SPEC_AGGOP('price_aggplan', 'global_aw',
'global', 'price_cube', 'unit_cost', 'time', 'AVERAGE')
```

6. In the OLAP Catalog View, run the Deploy Aggregation Plan wizard to generate the aggregate data.
7. From the File menu, choose **Save**.
8. Enable the GLOBAL analytic workspace for the BI Beans.

Enabling an Analytic Workspace for an Application

Oracle applications are typically designed to run against relational tables in the Oracle Database. The relational tables must conform to certain standards set by the application, and some form of metadata is used to identify the data to the application. For example, the BI Beans requires a star or snowflake schema with

embedded total dimension views for solved data, and OLAP Catalog metadata to describe the schema.

The same applications, without modification, can run against analytic workspaces which have been enabled for their use. Enabling an analytic workspace means that you have:

- Created views of analytic workspace data that conform to the same criteria as the relational tables typically used by the application. These views use the `OLAP_TABLE` function to extract workspace data.
- Created any metadata required by the application in order to access the views.

How to Enable an Analytic Workspace

To enable an analytic workspace, complete these steps:

1. Expand the OLAP Catalog View sufficiently to see the workspaces for your schema.
2. Right-click the name of the analytic workspace you want to enable.
3. Choose **Enable Workspace for OLAP API & BI Beans**.

   or

   Enable Workspace for Oracle Discoverer Using Wizard.

   Complete the steps of the wizard. Click the **Help** button to get specific information about each step.

About Enabling for the BI Beans

When you enable an analytic workspace for the BI Beans, you create several views that form a star schema. In addition, you create `CWM2` metadata, which makes these views accessible to BI Beans applications.

> **See Also:** *Oracle OLAP Reference* for full descriptions of the views and `CWM2` read API.

Star Schema of Views

The star schema for a BI Beans-enabled analytic workspace includes:

- A dimension view for each hierarchy
- A fact view for each combination of dimension hierarchies

These views are sometimes called **embedded total** views because dimension members at all levels are listed in a single column. Information about which level a particular member belongs to and the parent-child relationships among members is stored in separate columns. Within the fact tables, summary data is interspersed with (or embedded in) the base-level data. This differs markedly from the source star schema, in which there is no summary data in the fact tables and each level is represented by its own column in the dimension tables.

Table 6–2 describes the views created in the GLOBAL_AW schema when the GLOBAL analytic workspace is enabled.

Table 6–2 Views of the GLOBAL Analytic Workspace for the BI Beans

Name of View	Description
GLOB_GLOBA_CHANN_CHANN5VIEW	CHANNEL dimension view, CHANNEL_ROLLUP hierarchy
GLOB_GLOBA_CUSTO_MARKE6VIEW	CUSTOMER dimension view, MARKET_SEGMENT hierarchy
GLOB_GLOBA_CUSTO_SHIPM7VIEW	CUSTOMER dimension view, SHIPMENTS hierarchy
GLOB_GLOBA_PRODU_PRODU1VIEW	PRODUCT dimension view, PRODUCT_ROLLUP hierarchy
GLOB_GLOBA_TIME_CALEN2VIEW	TIME dimension view, CALENDAR hierarchy
GLOB_GLOBA_PRICE_CU4VIEW	PRICE_CUBE measure view
GLOB_GLOBA_SALES_CU10VIEW	SALES_CUBE measure view, CUSTOMER MARKET_SEGMENT hierarchy
GLOB_GLOBA_SALES_CU9VIEW	SALES_CUBE measure view, CUSTOMER SHIPMENTS hierarchy
GLOB_GLOBA_UNITS_CU12VIEW	UNITS_CUBE measure view, CUSTOMER MARKET_SEGMENT hierarchy
GLOB_GLOBA_UNITS_CU13VIEW	UNITS_CUBE measure view, CUSTOMER SHIPMENTS hierarchy

OLAP Catalog Metadata for Analytic Workspaces

When an analytic workspace is enabled for the BI Beans, OLAP Catalog metadata is created for the relational views, as described previously.

OLAP Catalog metadata for analytic workspaces is stored in a set of tables, which you can access through a set of views owned by OLAPSYS. The public synonyms for these views have a prefix of ALL_OLAP2.

The BI Beans query a special set of views, also owned by OLAPSYS, with the public synonym MRV_OLAP2. These views are created from the CWM2 metadata tables with a special structure that improves performance.

You can browse the metadata in the OLAP Catalog View of Analytic Workspace Manager, or you can use SQL commands to query the views. Example 6–4 shows the results of a query.

Example 6–4 Querying the CWM2 Read API

```
SELECT * FROM OLAPSYS.MRV_OLAP2_CATALOGS;

CATALOG_ID  CATALOG_NAME          PARENT_CATALOG_ID   DESCRIPTION
1849        GLOBAL_CAT                                Global Business Areas
475         XADEMO_MULTIKEY_CAT                       XADEMO MULTIKEY Measures
669         XADEMO_CAT XADEMO                         CWM Business Area
```

How to Enable an Analytic Workspace for Oracle Discoverer

To enable an analytic workspace, complete these steps:

1. Expand the OLAP Catalog View to see the workspaces for your schema.
2. Right-click the name of the analytic workspace you want to enable.
3. Choose **Enable Workspace for Oracle Discoverer Using Wizard**. Complete the steps of the wizard.

   Click the **Help** button to get specific information about each step.
4. Detach the analytic workspace, saving your changes first if necessary.
5. To create the views, execute the SQL script generated by the wizard.
6. To create an End User Layer (EUL), use Oracle Discoverer Administrator to import the EEX file generated by the wizard.

About Enabling for Oracle Discoverer

The Enable for Discoverer wizard generates two files on your local computer, where you are running Analytic Workspace Manager:

- A SQL script that creates views of workspace data in the format required by Discoverer.
- An EEX file that contains XML for creating an End User Layer.

Your analytic workspace is not enabled until you run the script and import the EEX file.

This release supports only one hierarchy for each dimension. If a dimension has multiple hierarchies, you must select one of them for access through Discoverer.

Views Created for Discoverer

Enabling an analytic workspace for Discoverer generates two sets of views.

The first set of views contains the OLAP_TABLE function calls that actually extract the data. This set exists to simplify creation of views in the format used by Discoverer (the second set of views), requiring that only a few object types and table types be defined. This set contains two types of views. Both sets use a letter and digit identifier, instead of the names, to identify the schema, the dimensions, and the levels.

- A view of the complete analytic workspace, with all dimensions and all measures. The name in this format:

  *workspace_schema*_FULL_VIEW

  where *schema* is an S followed by a digit, such as S1.

- A view for each dimension. The names are in this format:

  TFDV_*dimension*_VIEW

  where *dimension* is a D followed by a digit, such as D5.

The second set of views selects data from the first set, and presents the analytic workspace data in the format required by Discoverer. This set also has two types of views.

- A measure view for each combination of dimension levels. This view runs against the FULL_VIEW described earlier. The names are in this format:

  FACTVIEW_*schema_level_level_level...*

  where *schema* is an S followed by a digit, such as S1, and *level* is an L followed by a digit, such as L1.

- A view for each dimension. This view runs against the TFDV view described earlier. The names are in this format:

  DV_dimension

  where *dimension* is a D followed by a number, such as D1.

Figure 6–5 shows the relationships among the views.

Figure 6–5 Relationships Among Views for Discoverer

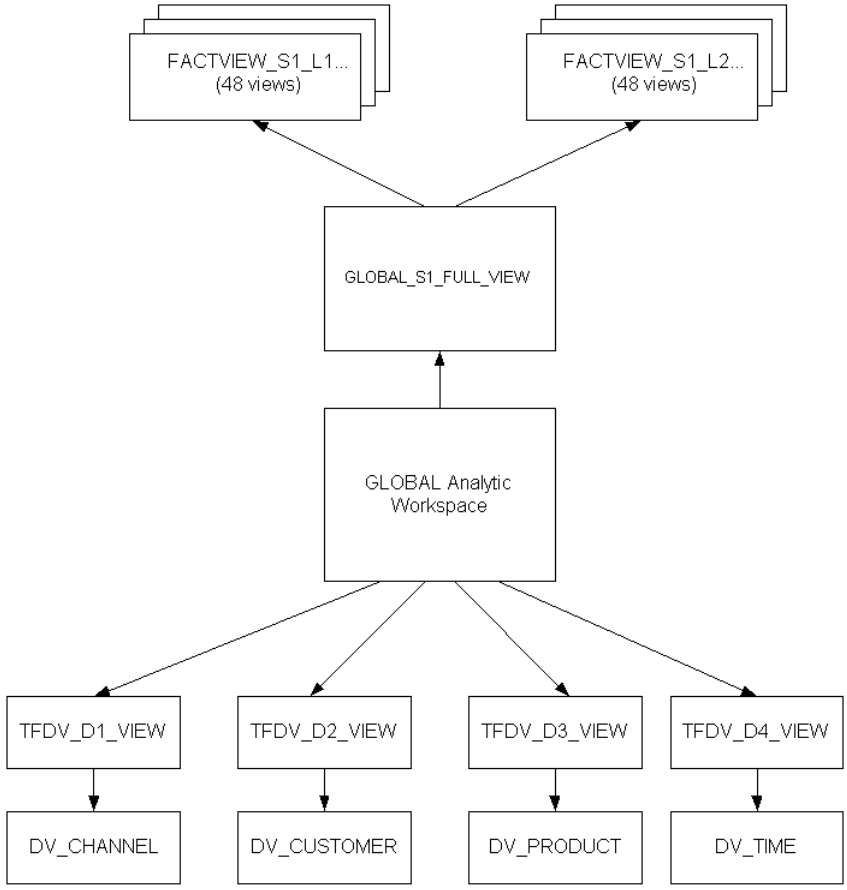


Table 6–3 describes the views of the GLOBAL analytic workspace.

Table 6–3 Views of the GLOBAL Analytic Workspace for Oracle Discoverer

View	Description
GLOBAL_S1_FULL_VIEW	Uses the OLAP_TABLE function to create a complete view of all dimensions and measures in the GLOBAL analytic workspace

Table 6–3 (Cont.) Views of the GLOBAL Analytic Workspace for Oracle Discoverer

View	Description
TFVD_D1_VIEW	Uses the OLAP_TABLE function to create a PRODUCT dimension view
TFVD_D2_VIEW	Uses the OLAP_TABLE function to create a TIME dimension view
TFVD_D3_VIEW	Uses the OLAP_TABLE function to create a CHANNEL dimension view
TFVD_D4_VIEW	Uses the OLAP_TABLE function to create a CUSTOMER dimension view
FACTVIEW_S1_L1_L6_L10_L14	Selects all measures at the ALL_CHANNELS, ALL_CUSTOMERS, TOTAL_PRODUCT, and YEAR levels from GLOBAL_S1_FULL_VIEW.
FACTVIEW_S1_L1_L6_L10_L15	Selects all measures at the ALL_CHANNELS, ALL_CUSTOMERS, TOTAL_PRODUCT, and QUARTER levels from GLOBAL_S1_FULL_VIEW.
.	.
.	.
.	.
FACTVIEW_S1_L2_L9_L13_L15	Selects all measures at the CHANNEL, SHIP_TO, ITEM, and QUARTER levels from GLOBAL_S1_FULL_VIEW.
FACTVIEW_S1_L2_L9_L13_L16	Selects all measures at the CHANNEL, SHIP_TO, ITEM, and MONTH levels from GLOBAL_S1_FULL_VIEW.
DV_CHANNEL	Selects the CHANNEL dimension view from TFVD_D3_VIEW.
DV_CUSTOMER	Selects the CUSTOMER dimension view from TFVD_D4_VIEW.
DV_PRODUCT	Selects the PRODUCT dimension view from TFVD_D1_VIEW.
DV_TIME	Selects the TIME dimension view from TFVD_D2_VIEW.

Refreshing the Data in an Analytic Workspace

Some build options do not load data, so you must perform an initial refresh before your analytic workspace can be used. Over time, all analytic workspaces need to be refreshed with new data. The data source will have new time periods as well as other new dimension members.

Using the Refresh Wizard

The Refresh Analytic Workspace wizard adds new members for selected dimensions and reloads all of the data for selected measures. The wizard requires the new data to be in the same tables as the original data.

To refresh your data, complete these steps:

1. Expand the OLAP Catalog View to see the workspaces for your schema.
2. Right-click the name of the analytic workspace you want to enable.
3. Choose **Refresh Analytic Workspace Using Wizard**. Complete the steps of the wizard. You can refresh individual dimensions, or measures, or both.

   Click the **Help** button to get specific information about each step.
4. Re-enable the cube if necessary.
5. Re-deploy the aggregation plans.

Refreshing From Different Relational Tables

The Refresh wizard accesses the same tables that the Create Analytic Workspace wizard used originally. However, you may bring new data into your relational schema in separate tables. You can either write a load program in SQL using the `DBMS_AWM` package, or follow these alternative steps.

These are the basic steps for refreshing a cube:

1. In the Object View of Analytic Workspace Manager, attach the analytic workspace in Read/Write mode.
2. In the Object View, expand the Programs folder and select the load program that was generated by the Create Analytic Workspace wizard to load data for the original build.

   If you are not sure which program to choose, then expand the Dimension folder and select the *cubedef* dimension. On the Properties page, note the value of the `AW$LOADPRGS` property.

3. On the Program page of the property viewer, edit the load program and change the name of the source table.
4. Choose **Apply**, then **Compile**.

   Correct any errors before continuing.
5. Right-click the name of the load program, then choose **Copy to Clipboard**.
6. Open OLAP Worksheet, and execute the program. Paste the name of the program into the query window by typing `Ctrl+V`.

   ```
   CALL program
   ```
7. Check the new data using the `LIMIT` and `REPORT` commands.
8. Issue `UPDATE` and `COMMIT` commands to save the new data.

Case Study: Refreshing the Units Cube

The Global star schema provides an additional month of data in separate update tables.

1. In the Object View, expand the Programs folder and select `GLOBAL_AW.GLOBAL!___GET.CUBE.DATA_UNITS_CUBE_1`.
2. Display the Program page of the load program, and locate the name of the source file, `UNITS_HISTORY_FACT`.

   ```
   SQL DECLARE C1 CURSOR FOR SELECT CHANNEL_ID,SHIP_TO_ID,ITEM_ID, -
        MONTH_ID,UNITS FROM GLOBAL.UNITS_HISTORY_FACT
   ```
3. Edit this statement by replacing `UNITS_HISTORY_FACT` with `UNITS_UPDATE_FACT`.
4. Choose **Apply**, then **Compile**.
5. Open OLAP Worksheet and run the revised load program with a command like this:

   ```
   CALL ___get.cube.data_units_cube_1
   ```
6. Display a sample of the new data with commands like these:

   ```
   LIMIT channel TO '1'       "Select any channel
   LIMIT product TO '1'        "Select any product
   LIMIT time TO '91'          "Select the new time period
   LIMIT customer TO '76'      "Select any customer
   " Add the parent values of customer 76
   ```

```
LIMIT customer ADD ANCESTORS USING customer_parentrel
REPORT DOWN customer units  "View the new data

CHANNEL: 1
PRODUCT: 1
                --UNITS---
                ---TIME---
CUSTOMER            91
--------------- ----------
76                1,220.00
17                7,378.00
8                12,985.00
1                50,632.00
```

7. Save the changes with these commands:

   ```
   UPDATE
   COMMIT
   ```

   or

   From the File menu of Analytic Workspace Manager, choose **Save**.

When a Data Refresh Requires Re-Enabling

Routine refreshes of the data do not require you to re-enable the workspace for a particular application, because the views created by the enablers do not need to be redefined for new dimension members. However, you do need to re-enable your workspace if you make changes to the logical model, such as:

- Change the OLAP Catalog metadata for the source cubes
- Add or delete a cube in the analytic workspace
- Add or delete a measure in the analytic workspace
- Add or delete a hierarchy in the analytic workspace
- Add or delete a level in the analytic workspace
- Change the OLAP Catalog metadata for the analytic workspace.

Because the enabling step takes only a short time to complete, you may prefer to re-enable your analytic workspace each time you refresh it.

If you want to delete old dimension members (for example, roll off the same number of old time periods as you added new ones) or load only new data values, then you can generate build scripts and modify the calls to the DBMS_AWM package.

> **See Also:** *Oracle OLAP Application Developer's Guide* for reference information about the DBMS_AWM package.

7

SQL Access to Analytic Workspaces

This chapter introduces methods of accessing the data in an analytic workspace using SQL. Most of these methods can be used at runtime as part of an application.

This chapter contains the following topics:

Overview of SQL Access

Using SQL, you can manipulate analytic workspace data and extract that data into your application. There are various methods that you can use, and the best one depends on the type of analytic workspace you have, the particular task you want to accomplish, and your personal preferences.

Manipulating Analytic Workspace Data

To manipulate analytic workspace data using SQL, you must use PL/SQL procedures that execute OLAP DML commands. The OLAP DML is the language for working in an analytic workspace. Using it, you can create, modify, delete, and

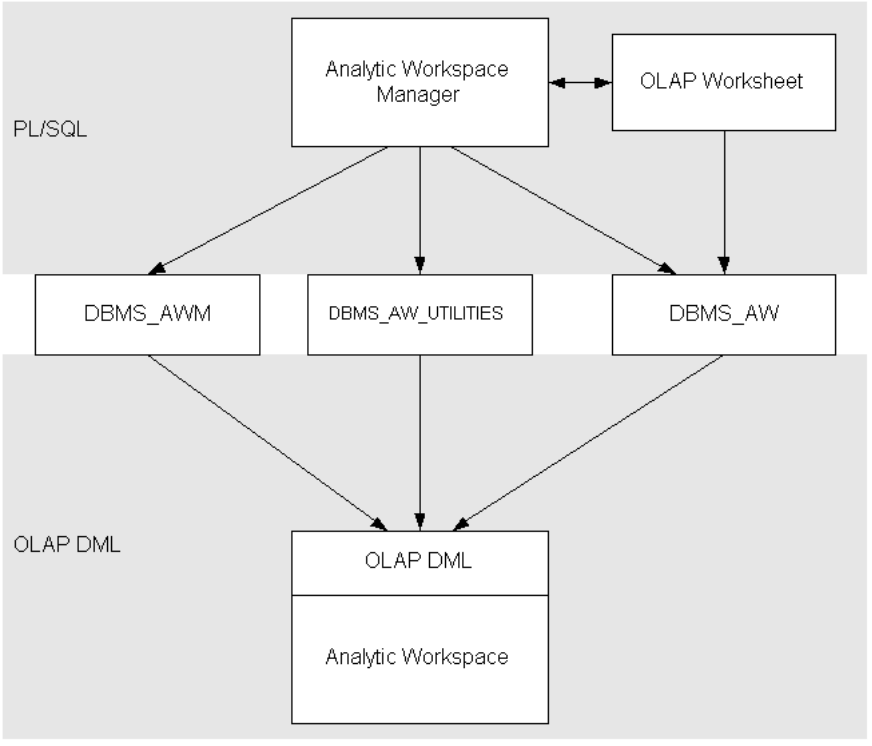
populate workspace objects. Any method that you use for performing these tasks uses the OLAP DML.

Several PL/SQL packages are available that execute OLAP DML commands. A call to a single procedure can execute a single OLAP DML command, or dozens of commands to perform a specific task. Among these packages are:

- `DBMS_AW` contains procedures for executing individual OLAP DML commands.
- `DBMS_AW_UTILITIES` contains procedures for managing custom measures in standard form analytic workspaces that have been enabled for the BI Beans.
- `DBMS_AWM` contains procedures for creating standard form analytic workspaces.

You can use any of these packages directly in a SQL interface such as SQL*PLus. Analytic Workspace Manager and OLAP Worksheet are applications that use these SQL packages. Figure 7–1 shows the relationships among them.

Figure 7–1 Analytic Workspace Manager's Use of PL/SQL Packages



Querying an Analytic Workspace

The OLAP_TABLE function provides the basic technology for querying an analytic workspace, as described in "Using OLAP_TABLE for Direct Access to Workspace Data" on page 7-12. It operates outside of the conventions of standard form, and can access data from any analytic workspace. However, tools that use OLAP_TABLE, such as the enablers, require standard form to construct the appropriate syntax.

About the Active Catalogs

Oracle OLAP provides catalogs of information about standard form analytic workspaces. These **active catalogs** are generated and maintained automatically without requiring any action by the DBA.

The active catalogs are implemented as public views with names that begin ALL_OLAP2_AW. For example, ALL_OLAP2_AW_CUBES lists cubes in all analytic workspaces, and ALL_OLAP2_AW_DIMENSIONS lists all of the dimensions. You can query the active catalogs directly from SQL.

For descriptions of the active catalogs, refer to the *Oracle OLAP Reference*.

Support for Custom Measures

A custom measure is calculated from one or more measures stored in the analytic workspace. Often, it is created by an analyst just for the duration of a session. However, a custom measure can also be saved as a permanent part of the analytic workspace.

These saved custom measures can either be solved at run-time or stored in variables. Run-time calculations do not require disk storage space and do not extend the processing time required for data maintenance. However, they may slow performance. You need to decide which measures to calculate on demand and which, if any, to store. The custom measures described in this chapter are calculated for a query. For instructions on creating stored custom measures, refer to Chapter 9.

Methods of Defining Custom Measures

Two PL/SQL packages support custom measures in an analytic workspace:

- DBMS_AW_UTILITIES contains procedures for creating, updating, and deleting custom measures. This package operates only on the views created by the enabler for the BI Beans. The custom measures are stored in the predefined

columns provided in these views for custom measures. You can define a custom measure to persist either for the duration of the session or permanently.

- DBMS_AW contains various procedures to execute OLAP DML commands. Several of them can be used in SELECT statements to execute a calculation or data manipulation in the analytic workspace. The calculations are returned along with the rest of the result set. This type of custom measure exists only for the duration of the SELECT statement.

In addition, you can use the OLAP_TABLE function to define and access custom measures outside of the framework of standard form, as described in "Using OLAP_TABLE for Direct Access to Workspace Data" on page 7-12.

Analytic Support for Custom Measures

Regardless of the method that you use to define a custom measure, you will express the calculation itself using the OLAP DML. Following are descriptions of the many functions and commands available for manipulating your data. In addition, you can perform inter-row calculations using operators for multiplication (*), division (/), addition (+), subtraction (-), and so forth.

Forecasts and Regressions

The OLAP DML offers the most sophisticated and up-to-date forecasting and regression tools, including simple linear regressions, non-linear regression methods, single exponential smoothing, double exponential smoothing, and the Holt-Winters method.

Time Series Manipulation

The time series functions perform operations such as lead, lag, and moving average. Table 7–1 describes the time series functions, which can easily be incorporated into custom measures.

Table 7–1 OLAP DML Time Series Functions

Function	Returns
CUMSUM	Cumulative totals
LAG	Value for a previous time period at a specified offset
LAGABSPCT	Percentage difference between a value and the absolute value for a previous time period at a specified offset

Table 7–1 (Cont.) OLAP DML Time Series Functions

Function	Returns
LAGDIF	Difference between a value and the value for a previous time period at a specified offset
LAGPCT	Percentage difference between a value and the value for a previous time period at a specified offset
LEAD	Value for a subsequent time period at a specified offset
MOVINGAVERAGE	A series of averages over a specified range
MOVINGMAX	A series of maximum values over a specified range
MOVINGMIN	A series of minimum values over a specified range
MOVINGTOTAL	A series of totals over a specified range

Financial Operations

The financial functions include interest rate calculations, depreciation, and payment schedules, similar to those provided in spreadsheets.

For example, the FPMTSCHED function calculates a payment schedule (principal plus interest) for paying off a series of fixed-rate installment loans over a specified number of time periods. The following call to FPMTSCHED calculates 36 payments based on the amounts listed in the LOANS variable, at the interest rates listed in the RATES variable, for the MONTH dimension of these variables.

```
FPMTSCHED(loans, rates, 36, month)
```

Statistical Operations

Statistical operations include standard deviation, rank, and correlation. For example, the STDDEV function calculates the standard deviation. The function call

```
STDDEV(units month)
```

returns the standard deviation of values in the UNITS measure for all months that are currently selected.

Numeric Computations

Functions are available to perform a wide variety of computations (such as sine, cosine, square root, minimum, and maximum) and data type conversions.

For example, the MAX function compares two expressions and returns the larger value. This function call

```
MAX(actual, forecast)
```

compares the ACTUAL and FORECAST measures and returns the larger values for all dimension members currently selected.

Text Manipulation

The OLAP DML provides support for manipulating both single- and multibyte character sets, with functions for concatenating strings, locating a string within a larger body of text, inserting a string, and so forth.

For example, the EXTCHARS function extracts a portion of text. The function call

```
EXTCHARS('lastname,firstname', 1,8)
```

extracts the first 8 characters, which contains the characters

```
lastname
```

Allocation

Allocations are a critical part of planning applications. Given a target for the organization, whether for sales quota, product growth, salary, or equipment, managers must allocate that target among its contributors. The supported allocation methods include:

- Copy methods (hierarchical copy, minimum, maximum, first, last)
- Even distribution (even, hierarchical even)
- Proportional distribution (including weighted distributions and user-defined multidimensional functions)

Aggregation

Aggregation is a basic feature of analytic workspaces. When you create a standard form analytic workspace, it contains a default aggregation plan for each cube. Wizards in Analytic Workspace Manager enable you to identify stored aggregate levels quickly and easily.

The OLAP DML offers a broader range of aggregation methods than are currently available through Analytic Workspace Manager or PL/SQL procedures. You can choose whatever method seems appropriate: by level, individual member, member attribute, time range, data value, or other criteria.

Models

A model is a set of interrelated equations. These are some of the modeling features supported by the OLAP DML:

- You can perform calculations for individual dimension members following unique calculation rules.
- Oracle OLAP determines the order of the calculations, so you can list them in any order without concern for dependencies.
- Oracle OLAP solves simultaneous equations.

You can assign results either to a variable or to a dimension member. Dimension-based equations provide flexibility; since you do not need to specify the modeling variable until you solve a model, you can run the same model with any other measure with the same dimension. For example, you could run the same model on Budget and Actual, which both have a Line dimension.

Creating Custom Measures Using DBMS_AW_UTILITIES

The enabler for the BI Beans creates fact views with columns specifically for custom measures defined by the DBMS_AW_UTILITIES package. There are 100 columns for numeric data named CUST_MEAS_NUM1 to CUST_MEAS_NUM100, and 100 columns for text data named CUST_MEAS_TEXT1 to CUST_MEAS_TEXT100.

This is the basic syntax for creating a custom measure:

```
CALL DBMS_AW_UTILITIES.CREATE_CUSTOM_MEASURE(
     schema.aw_name, aw_formula_name, aw_formula_expression,
    'PERMANENT'|'TEMPORARY', schema.view_name;
```

The BI Beans enabler creates CWM2 metadata for the views of analytic workspaces, and DBMS_AW_UTILITIES creates CWM2 metadata for the custom measures added to these views. This metadata is stored in tables that identify the mapping between the custom measures and the generic column names of the view:

- CWM2$_AW_TEMP_CUST_MEAS_MAP lists temporary custom measures for the current user.
- CWM2$_AW_PERM_CUST_MEAS_MAP lists permanent custom measures for users with the DBA role.

Case Study: Adding Sales to Global Using DBMS_AW_UTILITIES

"Identifying Required Business Facts" on page 3-6 identifies the data requirements of the Global Corporation. Only three facts are stored in the star schema; the others must be calculated in the analytic workspace. Because GLOBAL is a standard form analytic workspace that has been enabled for the BI Beans, the DBMS_AW_UTILITIES package is available for the DBA to define these measures.

Acquiring Information About the Analytic Workspace

Before you can define custom measures, you must know the names of measures that are already defined in the analytic workspace. You can query the ALL_OLAP2_AW_CUBE_MEASURES view in the Active Catalog for the names of measures defined in the GLOBAL analytic workspace. Example 7–1 shows how to obtain the names of the measures.

Example 7–1 Querying the Active Catalog for Measure Names

```
SELECT aw_cube_name, aw_measure_name
   FROM all_olap2_aw_cube_measures
   WHERE aw_owner = 'GLOBAL_AW' AND
   aw_name = 'GLOBAL';

AW_CUBE_NAME  AW_MEASURE_NAME
------------- -----------------
PRICE_CUBE    UNIT_COST
PRICE_CUBE    UNIT_PRICE
UNITS_CUBE    UNITS
```

The ALL_AW_CUBE_ENABLED_VIEWS view identifies the cubes that are enabled for the BI Beans, the names of the views created by the enabler to access those cubes, and the dimensions and dimension hierarchies for each view.

Example 7–2 shows that the Price cube is dimensioned by PRODUCT and TIME, and can be queried through a view named GLOB_GLOBA_PRICE_CU4VIEW. The Units cube is dimensioned by CHANNEL, CUSTOMER, PRODUCT, and TIME. The CUSTOMER dimension has two hierarchies: MARKET_SEGMENT is shown in GLOB_GLOBA_UNITS_CU9VIEW and SHIPMENTS is shown in GLOB_GLOBA_UNITS_CU10VIEW.

Example 7–2 SELECT Statement for Querying the Active Catalog

```
SELECT cube_name, system_viewname, hiercombo_str
   FROM all_aw_cube_enabled_views WHERE
```

```
   aw_name =   'GLOBAL' AND
   cube_name = 'PRICE_CUBE' OR
   cube_name = 'UNITS_CUBE';

CUBE_NAME    SYSTEM_VIEWNAME           HIERCOMBO_STR
------------ ------------------------- ------------------------------------------------------
PRICE_CUBE  GLOB_GLOBA_PRICE_CU4VIEW   DIM:PRODUCT/HIER:PRODUCT_ROLLUP;DIM:TIME/HIER:Calendar
UNITS_CUBE  GLOB_GLOBA_UNITS_CU9VIEW   DIM:CHANNEL/HIER:CHANNEL_ROLLUP;
                                       DIM:CUSTOMER/HIER:MARKET_SEGMENT;
                                       DIM:PRODUCT/HIER:PRODUCT_ROLLUP;
                                       DIM:TIME/HIER:Calendar
UNITS_CUBE  GLOB_GLOBA_UNITS_CU10VIEW  DIM:CHANNEL/HIER:CHANNEL_ROLLUP;
                                       DIM:CUSTOMER/HIER:SHIPMENTS;
                                       DIM:PRODUCT/HIER:PRODUCT_ROLLUP;
                                       DIM:TIME/HIER:Calendar
```

Using DBMS_AW_UTILITIES to Define Sales as a Custom Measure

After getting the information you need to define a custom measure, you can define your custom measures using DBMS_AW_UTILITIES. This example defines SALES, which calculates the product of two other measures, UNITS and UNIT_PRICE, for each combination of dimension members.

UNITS is a measure in the Units cube, and UNIT_PRICE is a measure in the Price cube. The Units cube has four dimensions: TIME, PRODUCT, CUSTOMER, and CHANNEL. The Price cube has only two dimensions, TIME and PRODUCT. The product of these two measures will have four dimensions, so SALES must be added to a view of the Units cube.

Example 7–3 adds the SALES measure to both views for the Units cube. Notice that only the first call specifies the equation for the SALES formula. The second call just identifies the existing SALES formula.

Note: Whenever you use DBMS_AW_UTILITIES in a SQL environment such as SQL*Plus, be sure to begin with these settings:

```
SET SERVEROUT ON
EXECUTE CWM2_OLAP_MANAGER.SET_ECHO_ON
```

Otherwise, you will not see any diagnostic messages.

Example 7–3 Defining SALES Using DBMS_AW_UTILITIES

```
SET SERVEROUT ON
EXECUTE CWM2_OLAP_MANAGER.SET_ECHO_ON
EXECUTE DBMS_AW_UTILITIES.CREATE_CUSTOM_MEASURE(
     'global_aw.global', 'sales', 'units * unit_price',
     'PERMANENT', 'global_aw.glob_globa_units_cu9view');

EXECUTE DBMS_AW_UTILITIES.CREATE_CUSTOM_MEASURE(
     'global_aw.global', 'sales', '',
     'PERMANENT', 'global_aw.glob_globa_units_cu10view');
```

Viewing the Workspace Formula

Use this command to see the formula created in the analytic workspace:

```
EXECUTE DBMS_AW.EXECUTE('DESCRIBE sales');

DEFINE SALES FORMULA DECIMAL <TIME CUSTOMER PRODUCT CHANNEL>
EQ units * unit_price
```

You can also view the property sheet for `SALES` in Analytic Workspace Manager.

Querying the Sales Custom Measure

`OLAPSYS.CWM2$_AW_PERM_CUST_MEAS_MAP` identifies the mapping between the `SALES` custom measure and a column in the views.

```
SELECT aw_access_view_name, cust_adt_column, aw_measure_name
   FROM olapsys.cwm2$_aw_perm_cust_meas_map
   WHERE workspace_name = 'global_aw.global';

AW_ACCESS_VIEW_NAME                 CUST_ADT_COLUMN  AW_MEASURE_NAME
----------------------------------- ---------------- ---------------
global_aw.glob_globa_units_cu9view   CUST_MEAS_NUM1   sales
global_aw.glob_globa_units_cu10view  CUST_MEAS_NUM1   sales
```

Queries for the `SALES` measure must select values from the `CUST_MEAS_NUM1` columns of the two tables.

Creating Custom Measures Using OLAP_EXPRESSION

The DBMS_AW package contains several procedures for specifying run-time calculations.

- OLAP_EXPRESSION performs numeric calculations
- OLAP_EXPRESSION_BOOL performs Boolean calculations
- OLAP_EXPRESSION_DATE performs date calculations
- OLAP_EXPRESSION_TEXT performs text manipulations

You can use these procedures to specify inter-row calculations using SELECT statements on a view of analytic workspace data. The calculations are performed by the OLAP engine. The only requirement for using these functions is that the SELECT statement for the view must contain a call to the OLAP_TABLE function with a ROW2CELL clause. The enabler for the BI Beans generates views of this type, and you can also generate custom views with ROW2CELL columns as described in "Using OLAP_TABLE for Direct Access to Workspace Data" on page 7-12.

The syntax of the four functions is identical. The difference between them is only in data type. This is the basic syntax for OLAP_EXPRESSION:

```
OLAP_EXPRESSION(r2c, expression)
```

For example: `OLAP_EXPRESSION('R2C', 'units * unit_price')`

Case Study: Adding Sales to Global Using OLAP_EXPRESSION

Enablement for the BI Beans created two views of the Units cube, one for each of the two hierarchies for the CUSTOMER dimension. The following SELECT statement queries one of the views and generates a new column for Sales. The SALES column is calculated in the analytic workspace.

```
SELECT time_et, units, OLAP_EXPRESSION(r2c, 'units * unit_price') sales
   FROM global_aw.glob_globa_units_cu9view WHERE
      channel_et ='1' AND
      product_et = '4' AND
      customer_et = '24' AND
      time_et > '66' AND
      units IS NOT NULL
   ORDER BY OLAP_EXPRESSION(r2c, 'units * unit_price') DESC;
```

The result set of this SELECT statement is sorted so that the sales figures are listed in descending order.

```
TIME_ET   UNITS      SALES
--------- -------- ----------
 8         6       170017.38
68         5       123300.25
 9         3        93293.85
 7         3        64931.7
67         2        50932.26
```

Using OLAP_TABLE for Direct Access to Workspace Data

The OLAP_TABLE function provides the basic technology for extracting data from an analytic workspace. All of the views of analytic workspaces that are generated by the enablers use the OLAP_TABLE function. By using OLAP_TABLE directly, you have full control over data access. You can develop your own views to support applications for which there are no enablers, and you can extract workspace data directly into your application. This capability can provide your application with tremendous flexibility, since user queries can be formulated into calls to OLAP_TABLE at runtime.

While the OLAP tools that use the OLAP_TABLE function require a standard form analytic workspace, the OLAP_TABLE function itself does not use standard form metadata.

Designing Views of an Analytic Workspace

The number of views that you create, and the number and characteristics of the columns in these views, depends largely on the requirements of the applications that these views are designed to support.

Because analytic workspaces contain aggregate data, the views must include the aggregates. There are several formats for presenting aggregate data:

- Create a star schema with dimension views and measure views. The dimension views list dimension members at all levels in a single column.
- Create a view that includes columns for all of the dimensions, attributes, and measures.
- Create a view in rollup form that shows the full parentage of each dimension member in multiple columns.
- Create a separate table for each aggregation level.

Choose a format that is appropriate for your application and its metadata.

Process Overview

These are the basic steps you must follow to generate views of data stored in an analytic workspace.

1. Explore the analytic workspace and identify the variables, formulas, relations, and dimensions that you want to expose to your application.
2. Decide how you want to present these objects in relational tables or views, based on the requirements of the application that will use them.
3. For each table or view that you plan to create, issue a `SELECT` statement using the `OLAP_TABLE` function. The `SELECT` statement can be an argument to a `CREATE VIEW` statement.
4. Commit these changes to the database if you are creating views for general use.
5. Create whatever metadata is required by your application to query the views.

Using OLAP_TABLE

You use the `OLAP_TABLE` function in a SQL `SELECT` statement to query the multidimensional data stored in an analytic workspace. `OLAP_TABLE` can be used wherever you would use the name of a table or view. You can use `SELECT` statements to create views, or to fetch data directly from an analytic workspace into an application.

`OLAP_TABLE` returns a table of objects that can be joined to relational tables and views, or to other tables of objects populated by `OLAP_TABLE`. It can also return stored workspace data, or it can perform calculations on stored data and return the results of the calculations.

Example 7–4 is a template that you can use as the starting point for the SQL scripts that you will develop for extracting data from your analytic workspace. You can then execute the script with the `@` command in SQL*Plus.

Note: Be sure to verify that you have created the views correctly by issuing `SELECT` statements against them. Only at that time will any errors in the call to `OLAP_TABLE` appear.

Example 7–4 Template for Using OLAP_TABLE

```
SET ECHO ON
SET SERVEROUT ON

--CREATE OR REPLACE VIEW view_name AS
SELECT column1, column2, columnn
   FROM TABLE(OLAP_TABLE(
      'connection',
      'table_obj',
      'datamap',
      'limit_map'))
    MODEL (
      DIMENSION BY(et_dims, gids)
      MEASURES(measures, attributes, row2cell)
      RULES UPDATE SEQUENTIAL ORDER();
/
COMMIT
/
GRANT SELECT ON view_name TO PUBLIC;
```

> **See Also:** *Oracle OLAP Reference* for a description of OLAP_TABLE syntax.

Using the SELECT MODEL Clause

When used in a SELECT statement that queries OLAP_TABLE, the MODEL clause is an optimization that results in significantly faster response time. It can be used only when creating a table type with embedded total dimensions, such as the views used by the BI Beans and the OLAP API.

Note that while the MODEL clause is used in relational queries for inter-row calculations, you should not use it for this purpose with OLAP_TABLE. For OLAP_TABLE, the MODEL clause is used only to optimize the query.

> **See Also:** The SELECT entry in the *Oracle Database SQL Reference* for general information about the MODEL clause.

When used in a SELECT statement that queries an analytic workspace, MODEL has the following arguments.

DIMENSION BY

The names of the embedded total dimension columns, as defined in the limit map. For BI Beans applications, include the GID columns in this list.

Any other columns in the DIMENSION BY list disables this optimization. A properly constructed SELECT statement still executes, but more slowly.

MEASURES

The measures, attributes, R2Cs, and any other columns excluded from the DIMENSION BY list.

RULES UPDATE SEQUENTIAL ORDER

The RULES clause is required, but it should not include complex or inter-row calculations since they will slow the query. Any calculations specified in the RULES clause are performed by SQL. If you want to perform inter-row calculations, you can create a custom measure in the analytic workspace using any of the alternative methods discussed in this chapter, including the limit map of OLAP_TABLE.

UPDATE indicates that you are not adding any custom members in the DIMENSION BY clause. Be sure to include this keyword, because otherwise the SQL WHERE clauses for measures are discarded, which can significantly degrade performance.

SEQUENTIAL ORDER prevents Oracle from evaluating the rules to ascertain their dependencies.

Case Study: Using OLAP_TABLE to Create Global Custom Measures

The Global Corporation requires numerous custom measures in addition to the three stored measures fetched from a star schema into the GLOBAL analytic workspace. The OLAP_TABLE function offers a method of creating these derived measures, although other methods (described previously in this chapter) are also available to GLOBAL.

UNITS is one of the stored measures, and the units for the prior period is a required derived measures. Although they are not required, other derived measures such as the difference from the prior period or the percent change may also be desirable.

Derived measures can be defined permanently in the analytic workspace or specified in the syntax of the OLAP_TABLE function. This example adds these two measures:

- UNITS_PP calculates the units sold in the prior period.
- UNITS_PCTCHG_PP is the percent change from the prior period.

This example creates a new OLAP Catalog cube for these measures.

Defining Formulas in the Analytic Workspace

If it does not already exist, add UNITS_PP, which returns the value of the prior time period, to the GLOBAL analytic workspace with these commands:

```
DEFINE units_pp FORMULA LAG(units, 1, time, LEVELREL time_levelrel)
UPDATE;COMMIT
```

This syntax for defining a formula gives it the same data type and dimensionality as the source object. The new formula has this definition:

```
DEFINE UNITS_PP FORMULA DECIMAL <TIME CUSTOMER PRODUCT CHANNEL>
EQ lag(units, 1, time, levelrel time_levelrel)
```

Alternatively, you can define UNITS_PP using the property sheets in Analytic Workspace Manager.

Example 7–7 defines UNITS_PCTCHG_PP in the OLAP_TABLE function, using the OLAP DML LAGPCT function. UNITS_PCTCHG_PP calculates the percent change from the prior period.

Neither UNITS_PP nor UNITS_PCTCHG_PP are defined as standard form measures. To comply with standard form, they need several OLAP DML properties, and they must be registered as measures in the standard form catalogs. However, OLAP_TABLE and the OLAP Catalog do not require standard form; only the tools that simplify their use require standard form.

Querying an Analytic Workspace Using OLAP_TABLE

Example 7–5 shows a script that fetches data directly into a SQL application using a SELECT statement with the OLAP_TABLE function. This selection is separate from any application enablement process.

To query the Units measures in the GLOBAL analytic workspace, take these steps:

1. Open a file with any text editor, and enter the body of the SQL script shown in Example 7–5. Save it with a name such as units_query.sql.
2. Open a SQL*Plus session with a user name that has access rights to the GLOBAL analytic workspace.
3. Execute the SQL script with a command like this one:

   ```
   @units_query
   ```

There is neither standard form metadata nor application metadata for UNITS_PP. An explanation of the example follows the code.

Example 7–5 UNITS_QUERY Script for Querying with OLAP_TABLE

```
SELECT time_name, units, units_pp FROM TABLE(OLAP_TABLE(
      'global DURATION SESSION',
      '',
      'LIMIT customer_hierlist TO 2',
      'MEASURE units AS NUMBER(16) FROM units
       MEASURE units_pp AS NUMBER(16) FROM units_pp
       DIMENSION channel_dim FROM channel WITH
         HIERARCHY channel_parentrel
       DIMENSION product_dim FROM product WITH
         HIERARCHY product_parentrel
       DIMENSION customer_dim FROM customer WITH
         HIERARCHY customer_parentrel
       DIMENSION time_dim FROM time WITH
         HIERARCHY time_parentrel
         ATTRIBUTE time_name AS VARCHAR2(8) FROM time_long_description'))
WHERE units IS NOT NULL and
      channel_dim = '1' and
      product_dim = '1' and
      customer_dim = '21';
MODEL
DIMENSION BY(channel_dim, product_dim, customer_dim, time_dim)
MEASURES(units, units_pp, time_name)
RULES UPDATE SEQUENTIAL ORDER ();
/
```

Example 7–6 shows the results of running the script in Example 7–5.

Example 7–6 Results of Running the UNITS_QUERY Script

```
@units_query

TIME_NAM      UNITS   UNITS_PP
-------- ---------- ----------
Jan-98        11357
Feb-98        11336      11357
Mar-98        11184      11336
                .
                .
                .
2001         230913     202580
2002         201590     230913
2003         109711     201590

93 rows selected.
```

OLAP_TABLE Function

In Example 7–5, the arguments to OLAP_TABLE provide the most basic information: the measures you want to see, their dimensions, and the descriptive names for time periods that make this data meaningful. In addition, the OLAP_TABLE function needs the names of the parent relations, which define the hierarchical structure of the dimensions. Since these dimensions were created by the Create Analytic Workspace wizard in Analytic Workspace Manager, the parent relations are named *dimension*_PARENTREL.

The CUSTOMER dimension has two hierarchies, and a LIMIT command selects the second hierarchy, MARKET_SEGMENTS; SHIPMENTS is the first hierarchy in the CUSTOMER_HIERLIST hierarchy dimension, and so it is the default. The other dimensions have only one hierarchy, so there is no need to limit their *hierlist* dimensions.

The limit map identifies two measures (UNITS and UNITS_PP), both of which are formulas in the analytic workspace. UNITS calculates aggregates from a stored measure, and UNITS_PP returns the value of the prior period, as defined in "Defining Formulas in the Analytic Workspace" on page 7-16. Data types are specified only for the selected columns: TIME_NAME, UNITS, and UNITS_PP.

SELECT Statement

In Example 7–5, the SELECT statement identifies the columns and rows of interest, just as it does for physical tables in the database. In this particular selection, the WHERE clause limits all dimensions except TIME to a single value, then labels the result set only with the long descriptions for TIME.

Using OLAP_TABLE to Create a Measure View for the BI Beans

Example 7–7 shows how you can make the data in an analytic workspace available to the BI Beans using OLAP_TABLE. The process involves these steps:

1. Create views that conform with the requirements of the BI Beans.
2. Define OLAP Catalog metadata so that the views can be queried by the BI Beans.

This example creates a measure view of UNITS, UNITS_PP, and UNITS_PCTCHG_PP for the CUSTOMER MARKET_ROLLUP hierarchy. A second view is required for the SHIPMENTS_ROLLUP hierarchy. The example does not show the dimension views either, although the OLAP Catalog and the BI Beans require views of each dimension.

UNITS_PCTCHG_PP is a custom measure defined in the limit map using the AW_EXPR keyword. It uses the OLAP DML LAGPCT function to calculate the percent difference from the prior period.

Creating and Executing the SQL Script

To create the views for the OLAP API, take these steps:

1. Open a file with any text editor, and enter the body of the SQL script shown in Example 7–7. Save it with a name such as ts_views.sql.
2. Open a SQL*Plus session with a user name that has access rights to the GLOBAL analytic workspace.
3. Execute the SQL script with a command like this one:

   ```
   @ts_view
   ```

4. Commit these changes to the database.
5. Issue SELECT commands against the views to verify that they were defined correctly; if not, an error will be generated.

Example 7–7 Creating Views for the OLAP API

```
CREATE OR REPLACE VIEW ts_view_1 AS SELECT * FROM TABLE(OLAP_TABLE(
      'global DURATION SESSION',
      '',
      'LIMIT customer_hierlist to 1',
      'MEASURE units AS NUMBER(16) FROM units
       MEASURE units_pp AS NUMBER(16) FROM units_pp
       MEASURE units_pctchg_pp AS NUMBER(8,2)
          FROM AW_EXPR LAGPCT(units, 1, time LEVELREL time_levelrel
       ROW2CELL r2c
       DIMENSION channel_et AS VARCHAR2(4) FROM channel WITH
         HIERARCHY channel_parentrel
         INHIERARCHY channel_inhier
         GID channel_gid AS NUMBER(2) FROM channel_gid
       DIMENSION product_et AS VARCHAR2(4) FROM product WITH
         HIERARCHY product_parentrel
         INHIERARCHY product_inhier
         GID product_gid AS NUMBER(2) FROM product_gid
       DIMENSION customer_et AS VARCHAR2(4) FROM customer WITH
         HIERARCHY customer_parentrel
         INHIERARCHY customer_inhier
         GID customer_gid AS NUMBER(2) from customer_gid
```

```
        DIMENSION time_et AS VARCHAR2(8) FROM time WITH
          HIERARCHY time_parentrel
          INHIERARCHY time_inhier
          GID time_gid AS NUMBER(2) FROM time_gid'))
  WHERE units IS NOT NULL
MODEL
          DIMENSION BY(channel_et, channel_gid, product_et, product_gid,
            customer_et, customer_gid, time_et, time_gid)
          MEASURES(units, units_pp, units_pctchg_pp,r2c)
          RULES UPDATE SEQUENTIAL ORDER ();
```

About the Sample Script

Example 7–7 defines a view that conforms to the requirements of the OLAP API for a fact table:

- Each dimension has one embedded total column for its members at all hierarchical levels. The columns are named *dimension*_ET to match the views generated by the OLAP API enabler.
- Each dimension has a column for its grouping IDs. The columns are named *dimension*_GID to match the views generated by the OLAP API enabler.
- A ROW2CELL column is defined for use by the OLAP_EXPRESSION function.

For each dimension, the view identifies these analytic workspace objects:

- The HIERARCHY relation, which defines the hierarchical relationship among dimension members by identifying the parent of each member.
- The INHIERARCHY variable, which identifies whether a dimension member is in the selected hierarchy.
- The GID variable, as described previously.

These objects were created by the Create Analytic Workspace wizard. Notice that the GID variables are the only ones that are mapped to columns in the view.

Defining OLAP Catalog Metadata for Workspace Views

To define OLAP Catalog metadata for views of an analytic workspace, you must use the CWM2 write APIs. You can then view CWM2 metadata in the OLAP Catalog view of Analytic Workspace Manager, or by querying the OLAP Catalog views directly in SQL. You can neither define nor view CWM2 metadata using Oracle Enterprise Manager.

The new measures (UNITS_PP and UNITS_PCTCHG_PP) could be added to the existing Units cube. However, Example 7–8 shows how you can create a new cube for them using predefined dimensions. The example also creates a new measure folder.

To create the OLAP Catalog metadata for the new measures, follow these steps:

1. Open a file with any text editor, and enter the body of the SQL script shown in Example 7–8. Save it with a name such as ts_cwm.sql.

   Refer to the *Oracle OLAP Reference* for the complete syntax and usage notes for the CWM2 APIs.

2. Open a SQL*Plus session with a user name that has access rights to the GLOBAL analytic workspace and issue these commands:

   ```
   SET ECHO ON
   SET LINESIZE 135
   SET PAGESIZE 50
   SET SERVEROUTPUT ON FORMAT WRAPPED SIZE 1000000
   EXECUTE CWM2_OLAP_MANAGER.SET_ECHO_ON;
   ```

   These settings enable you to see any error messages and view the full report from the validation programs that are run by the script. It is important to validate the metadata before committing it to your database.

3. Execute the SQL script with a command like this one:

   ```
   @ts_cwm
   ```

   Note: If the validation messages exceed the maximum buffer size for SQL*Plus, you can redirect them to a log file by using CWM2_OLAP_MANAGER.BEGIN_LOG.

4. If there are errors, then take these steps:

   a. Issue a ROLLBACK command,

   b. Fix the errors in the script.

   c. Rerun the script.

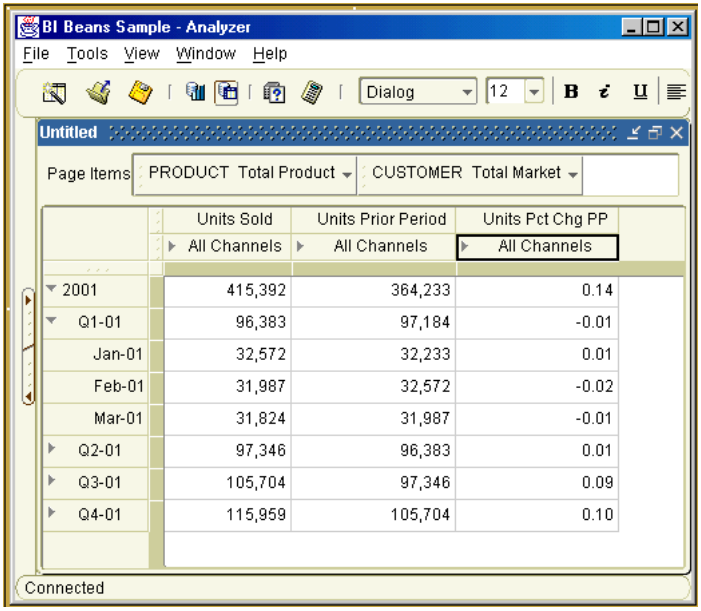
5. Copy the metadata to special views for the BI Beans:

   ```
   EXECUTE CWM2_OLAP_METADATA_REFRESH.MR_REFRESH();
   ```

   This procedure issues a COMMIT.

Once these measures are defined in the OLAP Catalog, they are available to BI Beans applications the same as the standard form measures. Figure 7–2 shows the result set of a query issued through a BI Beans application.

Figure 7–2 New Measures Queried Using a BI Beans Sample Application



Example 7–8 Script for Creating OLAP Catalog Metadata for GLOBAL Measures

```
BEGIN
-- Define TS_CUBE cube with predefined dimensions
CWM2_OLAP_CUBE.CREATE_CUBE('GLOBAL_AW', 'TS_CUBE', 'TS Cube', 'TS Cube', 'Units Time Series
Cube');
CWM2_OLAP_CUBE.ADD_DIMENSION_TO_CUBE('GLOBAL_AW', 'TS_CUBE', 'GLOBAL_AW', 'CHANNEL');
CWM2_OLAP_CUBE.ADD_DIMENSION_TO_CUBE('GLOBAL_AW', 'TS_CUBE', 'GLOBAL_AW', 'PRODUCT');
CWM2_OLAP_CUBE.ADD_DIMENSION_TO_CUBE('GLOBAL_AW', 'TS_CUBE', 'GLOBAL_AW', 'CUSTOMER');
CWM2_OLAP_CUBE.ADD_DIMENSION_TO_CUBE('GLOBAL_AW', 'TS_CUBE', 'GLOBAL_AW', 'Time');
CWM2_OLAP_MEASURE.CREATE_MEASURE('GLOBAL_AW', 'TS_CUBE', 'UNITS_PP', 'Units PP',
   'Units Prior Period', 'Units Sold in Prior Period');
CWM2_OLAP_MEASURE.CREATE_MEASURE('GLOBAL_AW', 'TS_CUBE', 'UNITS_PCTCHG_PP', 'Units PctChgPP',
   'Units Pct Chg PP', 'Percent Difference in Units Sold From Prior Period');

-- Map TS_VIEW_1 view to metadata cube TS_CUBE
CWM2_OLAP_TABLE_MAP.MAP_FACTTBL_LEVELKEY('GLOBAL_AW', 'TS_CUBE', 'GLOBAL_AW', 'TS_VIEW_1', 'ET',
```

```
   'DIM:GLOBAL_AW.CHANNEL/HIER:CHANNEL_ROLLUP/GID:CHANNEL_GID/LVL:CHANNEL/COL:CHANNEL_ET;
   DIM:GLOBAL_AW.CUSTOMER/HIER:SHIPMENTS/GID:CUSTOMER_GID/LVL:SHIP_TO/COL:CUSTOMER_ET;
   DIM:GLOBAL_AW.PRODUCT/HIER:PRODUCT_ROLLUP/GID:PRODUCT_GID/LVL:ITEM/COL:PRODUCT_ET;
   DIM:GLOBAL_AW.Time/HIER:Calendar/GID:TIME_GID/LVL:Month/COL:TIME_ET');

CWM2_OLAP_TABLE_MAP.ADD_AWVIEW('GLOBAL_AW', 'TS_VIEW_1', 'r2c');

CWM2_OLAP_TABLE_MAP.MAP_FACTTBL_MEASURE('GLOBAL_AW', 'TS_CUBE', 'UNITS_PP', 'GLOBAL_AW', 'TS_VIEW_1',
'UNITS_PP',
   'DIM:GLOBAL_AW.CHANNEL/HIER:CHANNEL_ROLLUP/GID:CHANNEL_GID/LVL:CHANNEL/COL:CHANNEL_ET;
   DIM:GLOBAL_AW.CUSTOMER/HIER:SHIPMENTS/GID:CUSTOMER_GID/LVL:SHIP_TO/COL:CUSTOMER_ET;
   DIM:GLOBAL_AW.PRODUCT/HIER:PRODUCT_ROLLUP/GID:PRODUCT_GID/LVL:ITEM/COL:PRODUCT_ET;
   DIM:GLOBAL_AW.TIME/HIER:Calendar/GID:TIME_GID/LVL:Month/COL:TIME_ET;');

CWM2_OLAP_TABLE_MAP.MAP_FACTTBL_MEASURE('GLOBAL_AW', 'TS_CUBE', 'UNITS_PCTCHG_PP', 'GLOBAL_AW',
'TS_VIEW_1', 'UNITS_PCTCHG_PP',
   'DIM:GLOBAL_AW.CHANNEL/HIER:CHANNEL_ROLLUP/GID:CHANNEL_GID/LVL:CHANNEL/COL:CHANNEL_ET;
   DIM:GLOBAL_AW.CUSTOMER/HIER:SHIPMENTS/GID:CUSTOMER_GID/LVL:SHIP_TO/COL:CUSTOMER_ET;
   DIM:GLOBAL_AW.PRODUCT/HIER:PRODUCT_ROLLUP/GID:PRODUCT_GID/LVL:ITEM/COL:PRODUCT_ET;
   DIM:GLOBAL_AW.TIME/HIER:Calendar/GID:TIME_GID/LVL:Month/COL:TIME_ET;');

CWM2_OLAP_TABLE_MAP.MAP_FACTTBL_LEVELKEY('GLOBAL_AW', 'TS_CUBE', 'GLOBAL_AW', 'TS_VIEW_2', 'ET',
   'DIM:GLOBAL_AW.CHANNEL/HIER:CHANNEL_ROLLUP/GID:CHANNEL_GID/LVL:CHANNEL/COL:CHANNEL_ET;
   DIM:GLOBAL_AW.CUSTOMER/HIER:MARKET_SEGMENT/GID:CUSTOMER_GID/LVL:SHIP_TO/COL:CUSTOMER_ET;
   DIM:GLOBAL_AW.PRODUCT/HIER:PRODUCT_ROLLUP/GID:PRODUCT_GID/LVL:ITEM/COL:PRODUCT_ET;
   DIM:GLOBAL_AW.TIME/HIER:Calendar/GID:TIME_GID/LVL:Month/COL:TIME_ET;');

CWM2_OLAP_TABLE_MAP.ADD_AWVIEW('GLOBAL_AW', 'TS_VIEW_2', 'r2c');

-- Map TS_VIEW_2 view to metadata cube TS_CUBE
CWM2_OLAP_TABLE_MAP.MAP_FACTTBL_MEASURE('GLOBAL_AW', 'TS_CUBE', 'UNITS_PP', 'GLOBAL_AW',
'TS_VIEW_2', 'UNITS_PP',
   'DIM:GLOBAL_AW.CHANNEL/HIER:CHANNEL_ROLLUP/GID:CHANNEL_GID/LVL:CHANNEL/COL:CHANNEL_ET;
   DIM:GLOBAL_AW.CUSTOMER/HIER:MARKET_SEGMENT/GID:CUSTOMER_GID/LVL:SHIP_TO/COL:CUSTOMER_ET;
   DIM:GLOBAL_AW.PRODUCT/HIER:PRODUCT_ROLLUP/GID:PRODUCT_GID/LVL:ITEM/COL:PRODUCT_ET;
   DIM:GLOBAL_AW.TIME/HIER:Calendar/GID:TIME_GID/LVL:Month/COL:TIME_ET;');

CWM2_OLAP_TABLE_MAP.MAP_FACTTBL_MEASURE('GLOBAL_AW', 'TS_CUBE', 'UNITS_PCTCHG_PP', 'GLOBAL_AW',
'TS_VIEW_2', 'UNITS_PCTCHG_PP',
   'DIM:GLOBAL_AW.CHANNEL/HIER:CHANNEL_ROLLUP/GID:CHANNEL_GID/LVL:CHANNEL/COL:CHANNEL_ET;
   DIM:GLOBAL_AW.CUSTOMER/HIER:MARKET_SEGMENT/GID:CUSTOMER_GID/LVL:SHIP_TO/COL:CUSTOMER_ET;
   DIM:GLOBAL_AW.PRODUCT/HIER:PRODUCT_ROLLUP/GID:PRODUCT_GID/LVL:ITEM/COL:PRODUCT_ET;
   DIM:GLOBAL_AW.TIME/HIER:Calendar/GID:TIME_GID/LVL:Month/COL:TIME_ET;');

-- Validate the cube metadata
```

```
CWM2_OLAP_VALIDATE.VALIDATE_CUBE('GLOBAL_AW', 'TS_CUBE', 'OLAP API');

-- Create a measure folder
CWM2_OLAP_CATALOG.CREATE_CATALOG('GLOBAL_ANALYTIC_CAT', 'Global Analytic Measures');
CWM2_OLAP_CATALOG.ADD_CATALOG_ENTITY('GLOBAL_ANALYTIC_CAT', 'GLOBAL_AW', 'TS_CUBE', 'UNITS');
CWM2_OLAP_CATALOG.ADD_CATALOG_ENTITY('GLOBAL_ANALYTIC_CAT', 'GLOBAL_AW', 'TS_CUBE', 'UNITS_PP');
CWM2_OLAP_CATALOG.ADD_CATALOG_ENTITY('GLOBAL_ANALYTIC_CAT', 'GLOBAL_AW', 'TS_CUBE',
'UNITS_PCTCHG_PP');
--COMMIT;
end;
/
```

8

Exploring a Standard Form Analytic Workspace

This chapter describes the objects created in a standard form analytic workspace. It serves as a guide to your own analytic workspace, and you can examine the property sheets of the objects described here by opening the Object View in Analytic Workspace Manager.

This chapter contains the following topics:

> **See Also:** Appendix A for the complete database standard form specification.

About Workspaces Created Using OLAP Tools

As described in Chapter 6, there are several methods for creating analytic workspaces. All of these methods create analytic workspaces with the same basic characteristics. These characteristics include compliance with the **database standard form** conventions.

About Database Standard Form

Just as a relational schema can be set up in countless ways, the design of an analytic workspace can be structured in as many ways as there are application developers. However, when an application is created to run against analytic workspaces, it requires one particular design so that it can locate particular objects and identify their role within the workspace. The design for the tools available through Analytic Workspace Manager is called **database standard form**.

Analytic Workspace Manager and the current generation of tools can only be used with database standard form analytic workspaces. Database standard form (or simply, standard form) stipulates:

- Certain objects must exist in the analytic workspace. These objects and properties are used by tools in Analytic Workspace Manager that perform tasks such as aggregation, data refresh, and applications enablement. The active catalogs, described in "Overview of SQL Access" on page 7-1, also rely on database standard form, as do some PL/SQL packages, such as `DBMS_AW_UTILITIES`.
- OLAP DML properties (which begin with `AW$`) must be defined on these objects. The property values are metadata for the object, and identify its relationships with other objects in the analytic workspace.
- Objects must be registered in workspace catalogs. OLAP tools query these metadata catalogs to get information about how the logical cubes, measures, and dimensions are instantiated in the analytic workspace. When you define objects using the tools in Analytic Workspace Manager, the tools also maintain the catalogs. However, when you define objects manually, as described in some chapters of this guide, you must also maintain the properties and the catalogs for the tools to be aware of the new objects.

The Create Analytic Workspace wizard in Analytic Workspace Manager creates analytic workspaces in standard form. By using the Object View to browse the workspace objects, you can gain familiarity with standard form.

Not all of the objects required by standard form are currently used by the analytic workspace tools. These objects are not described in this chapter, and you can ignore them at this time.

> **See Also:** Appendix A for a full description of the database standard form convention.

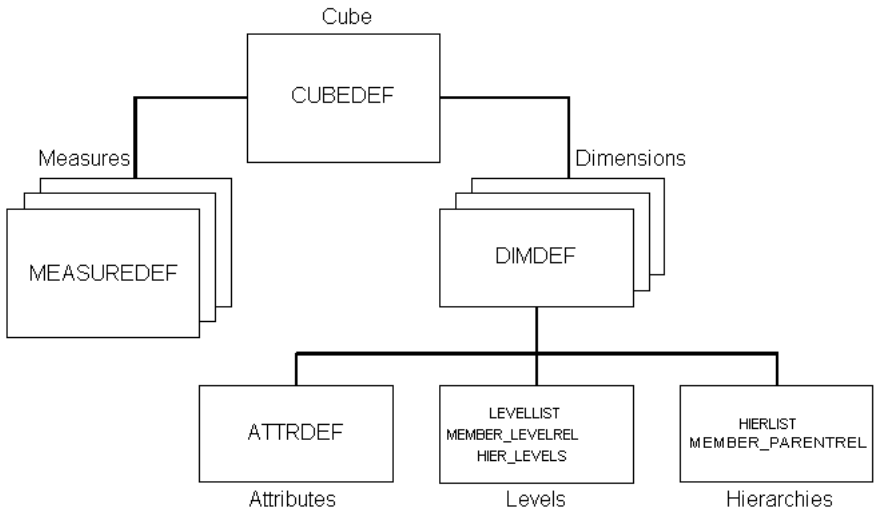
Standard Form Implementation of the Logical Model

The standard form logical model includes cubes, measures, and dimensions, as well as the hierarchies, levels, and attributes that are associated with dimensions. A cube is considered to be the parent of the measures that it contains, and a dimension is considered to be the parent of its hierarchies, levels, and attributes. A cube has dimensionality; that is, it is associated with its list of dimensions.

It is important to remember that standard form is a logical metadata model that is imposed on an analytic workspace. It does not describe the inherent relationships among workspace objects, such as the relationship between variables and formulas and their dimensions, or among dimensions in a workspace relation.

Figure 8–1 shows the basic objects that implement this model in a standard form analytic workspace.

Figure 8–1 Standard Form Implementation of the Basic Logical Model



Additional Requirements for OLAP Tools

Some of the tools in Analytic Workspace Manager require additional properties and objects, which are defined in the analytic workspace.

Some property values identify build parameters in DBMS_AWM procedures. If you created your analytic workspace by running DBMS_AWM directly, then you will immediately recognize these values. If you used Analytic Workspace Manager or Oracle Warehouse Builder, which generated the calls to DBMS_AWM, then you can see the choices made for you.

Standard form does not specify a naming convention for workspace objects. However, DBMS_AWM creates objects with standardized names that typically identify the role of the object within the analytic workspace. This chapter identifies objects by the value of their AW$ROLE property and identifies the standardized names given by DBMS_AWM. When creating an analytic workspace, you may choose to add prefixes to these names.

The term "standard form" is thus used loosely in this chapter to refer to both the convention and its implementation by DBMS_AWM.

Querying a Standard Form Analytic Workspace

Standard form enables you to discover the names of logical objects and the names of the physical workspace objects that implement the logical model.

Querying the Standard Form Catalogs

You can acquire information about an analytic workspace by querying its standard form catalogs. These catalogs are implemented as dimensions, variables, relations, and valuesets in the analytic workspace. Some of these objects are in the CATALOGS class, and others are in the EXTENSIONS class.

The ALL_OBJECTS dimension is a catalog that contains the names of all logical objects. ALL_OBJECTS is a concat dimension, that is, it is a concatenated list of the members of other simple dimensions. Separate dimensions for each logical object type contain the names of logical objects, for example, the ALL_HIERARCHIES dimension contains the names of all hierarchies, and the ALL_LEVELS dimension contains the names of all levels. You can query these dimensions to discover the logical model implemented by an analytic workspace.

For example, the following command displays the names of all measures in the analytic workspace.

```
REPORT W 40 all_measures

ALL_MEASURES
----------------------------------------
GLOBAL_AW.UNITS_CUBE.UNITS.MEASURE
GLOBAL_AW.PRICE_CUBE.UNIT_COST.MEASURE
GLOBAL_AW.PRICE_CUBE.UNIT_PRICE.MEASURE
```

ALL_OBJECTS and its simple dimensions (such as ALL_LEVELS) are used in dimensional catalogs that are implemented as variables, relations, and valuesets.

Refer to "Catalogs Class Objects" on page A-25 for more information about standard form catalogs.

Querying Properties

By querying the standard form properties attached to workspace objects, you can discover the relationship between the logical model and the physical objects that implement the model.

You can query the properties on a particular object, or limit the NAME dimension to objects with particular properties or property values. The NAME dimension contains the names of all objects in an analytic workspace. By limiting the status of the NAME dimension, you can limit the scope of commands that otherwise act on all objects.

All objects have the following properties, which are described in Table A–2 on page A-8:

```
AW$CLASS
AW$CREATEDBY
AW$LASTMODIFIED
AW$ROLE
```

The following commands show how you can use the AW$ROLE property to discover the names of *measuredef* objects:

```
LIMIT name TO OBJ(PROPERTY 'AW$ROLE') EQ 'MEASUREDEF'
REPORT name

NAME
------------
UNITS
UNIT_COST
UNIT_PRICE
```

The FULLDSC command lists all the properties and their values:

```
FULLDSC units

DEFINE UNITS FORMULA DECIMAL <TIME PRODUCT CUSTOMER CHANNEL>
EQ
aggregate(GLOBAL_AW!UNITS_STORED using GLOBAL_AW!-
GLOBAL.DEFAULTAGGMAP1.AGGREGATIONDEFINITION COUNTVAR GLOBAL_AW!-
UNITS_COUNTVAR)
PROPERTY 'AW$CLASS' 'IMPLEMENTATION'
PROPERTY 'AW$CREATEDBY' 'AW$XML'
PROPERTY 'AW$LASTMODIFIED' '25SEP03_14:01:54'
PROPERTY 'AW$LOGICAL_NAME' 'UNITS'
PROPERTY 'AW$PARENT_NAME' 'UNITS_CUBE'
PROPERTY 'AW$ROLE' 'MEASUREDEF'
PROPERTY 'AW$STATE' 'VALID_MEMBER'
```

Or you can use the OBJ function to get the value of a specific property:

```
SHOW OBJ(PROPERTY 'AW$PARENT_NAME', 'UNITS')

UNITS_CUBE
```

Standard Form Dimensions

The dimensions of a cube are typically hierarchical in nature and thus have levels and hierarchies. Dimensions in an analytic workspace are frequently called **embedded total** dimensions because they contain members at all levels, and thus are used to define measures with aggregate data. Dimension members are acquired from multiple level columns of a relational dimension table.

An embedded total dimension has, in addition to the dimension object, at least one level and one hierarchy. A flat dimension does not require them.

All dimensions have a default order attribute, as described in "Standard Form Attributes" on page 8-20. Time attributes must have end date and time span attributes.

For additional information about dimensions, refer to "Implementation Class Objects" on page A-14.

Dimdef Dimension

A *dimdef* dimension (that is, a dimension used in a cube) in an analytic workspace has the name defined in the metadata, such as TIME or PRODUCT, and may have a

prefix specified in the build. The dimension has a TEXT data type unless you redefine it before loading the dimension members. Dimension members may have a level prefix added to the source values.

Contents of an Analytic Workspace Dimension

The analytic workspace dimension members may be exactly the same as those in the relational dimension table, or they may have a level prefix. The prefix is an option in the build. Example 8–1 shows how the Global PRODUCT dimension members would appear if a prefix were specified in the build. (The Global star schema provides surrogate keys, so no prefix is actually needed to assure unique dimension members across levels.)

All dimension members are sorted during the load process. For the Time dimension, the members are sorted by level and by end-date within the levels. This order is required to support time-series analysis, which is based on the relative position of time periods within the dimension. Other dimensions are sorted by level and alphanumerically by dimension member within the levels. A default order attribute identifies the original order in which the dimension members were loaded into the analytic workspace.

Example 8–1 Global Products with Level Prefixes

```
LIMIT product TO product_levelrel EQ 'ITEM'
LIMIT product KEEP FIRST 3
LIMIT product ADD ANCESTORS USING product_parentrel
REPORT W 20 product

PRODUCT
--------------------
ITEM.13
ITEM.14
ITEM.15
FAMILY.4
CLASS.2
TOTAL_PRODUCT.1
```

Properties of an Analytic Workspace Dimdef Dimension

Table 8–1 describes the OLAP DML properties of a *dimdef* dimension. For descriptions of the properties independent of the object type, refer to Appendix A.

Table 8–1 Dimdef Dimension Properties

Property	Value
AW$CLASS	IMPLEMENTATION
AW$CREATEDBY	Creator of the dimension; the Refresh wizard requires a value of AW$CREATE, which indicates that the dimension was created by AWCREATE programs
AW$LASTMODIFIED	Date and time the dimension was last accessed by an analytic workspace tool
AW$LOGICAL_NAME	OLAP Catalog source name
AW$PARENT_NAME	NA
AW$ROLE	DIMDEF
AW$STATE	ACTIVE
AW$TYPE	Time for a Time dimension, otherwise NA
DESCRIPTION	Optional description of the dimension
LOAD_TYPE	Type of load performed for last refresh, either FULL_LOAD_ADDITIONS_ONLY or FULL_LOAD, as specified by DBMS_AWM.CREATE_AWDIMLOAD_SPEC; used by AWCREATE programs
SOURCE_NAME	OLAP Catalog source dimension; used by AWCREATE programs and the active catalog
SOURCE_OWNER	Owner of OLAP Catalog source metadata; used by AWCREATE programs and the active catalog
UNIQUE_RDBMS_KEY	YES if source dimension tables provided unique keys across levels, or NO if level names were prefixed to the keys to assure uniqueness, as specified by DBMS_AWM.SET_AWDIMLOAD_SPEC_PARAMETER; used by AWCREATE programs
DISPLAY_NAME	OLAP Catalog source display name or DBMS_AWM.SET_AWDIMLOAD_SPEC_PARAMETER setting; used by AWCREATE programs
P_DISPLAY_NAME	OLAP Catalog plural display name or DBMS_AWM.SET_AWDIMLOAD_SPEC_PARAMETER setting.

Standard Form Metadata for Dimensions

Standard form metadata for dimensions is stored in these objects:

- ALL_DIMENSIONS dimension
- ALL_DESCRIPTIONS variable
- AW_NAMES variable
- DIM_LEVELS valueset

ALL_DIMENSIONS Dimension

The ALL_DIMENSIONS dimension contains the names of all dimensions in this format:

workspace.dimension.DIMENSION

For example: GLOBAL_AW.PRODUCT.DIMENSION

ALL_DIMENSIONS is a base dimension of the ALL_OBJECTS concat dimension. ALL_OBJECTS dimensions ALL_DESCRIPTIONS and AW_NAMES, so these catalogs have an entry for each measure.

ALL_DESCRIPTIONS Variable for Dimensions

The ALL_DESCRIPTIONS variable contains short, long, and plural names for the dimensions. All objects have a short name acquired from the metadata, but may or may not have long and plural names.

AW_NAMES Variable for Dimensions

The AW_NAMES measure provides the fully qualified name of the workspace dimension object in this format:

```
schema.workspace!dimension
```

For example: GLOBAL_AW.GLOBAL!PRODUCT

DIM_LEVELS Valueset

The DIM_LEVELS valueset identifies the levels defined for each dimension.

Standard Form Hierarchies

The following objects support dimension hierarchies:

- *Hierlist* dimension
- *Member_parentrel* relation
- *Member_gid* variable
- *Member_inhier* variable

The values of the *member_parentrel* relation, *member_gid* variable, and *member_inhier* variable can be different for different hierarchies, so the *hierlist* dimension is used to define these objects.

For additional information about hierarchies, refer to "Implementation Class Objects" on page A-14 and "Features Class Objects" on page A-35.

Hierlist Dimension

A *hierlist* dimension stores the names of the hierarchies defined for a particular dimension. The names of the hierarchies are acquired from the OLAP Catalog. This text dimension typically has a name of *dimdef*_HIERLIST.

Contents of a Hierlist Dimension

Example 8–2 shows the contents of CUSTOMER_HIERLIST in the GLOBAL analytic workspace.

Example 8–2 GLOBAL Hierlist Dimension for CUSTOMER

```
REPORT W 20 customer_hierlist

CUSTOMER_HIERLIST
--------------------
SHIPMENTS
MARKET_SEGMENT
```

Properties of a Hierlist Dimension

Table 8–2 describes the OLAP DML properties of a *hierlist* dimension. For descriptions of the properties independent of the object type, refer to Appendix A.

Table 8–2 Hierlist Dimension Properties

Property	Value
AW$CLASS	IMPLEMENTATION
AW$CREATEDBY	Creator of the dimension; the Refresh wizard requires a value of AW$CREATE, which indicates that the dimension was created by AWCREATE programs
AW$LASTMODIFIED	Date and time the dimension was last accessed by an analytic workspace tool
AW$PARENT_NAME	*Dimdef* dimension
AW$ROLE	HIERLIST
AW$STATE	CREATED

Member_Parentrel Relation

A *member_parentrel* relation defines the hierarchical relationship among dimension members by identifying the parent of each member. This relation provides the essential hierarchical support for the dimension. This information is acquired from the relational dimension table. The parent relation is named *dimdef*_PARENTREL.

Contents of a Member_Parentrel Relation

A *member_parentrel* relation is a type of self-relation, in which the only valid values are dimension members. Example 8–3 shows the *member_parentrel* relation for the CHANNEL dimension in the GLOBAL analytic workspace. The relation defines a two-level hierarchy in which 1 is the parent of 2, 3, and 4.

Example 8–3 CHANNEL Member_Parentrel Relation in GLOBAL

```
REPORT DOWN channel W 20 channel_parentrel

               -CHANNEL_PARENTREL--
               --CHANNEL_HIERLIST--
CHANNEL           CHANNEL_ROLLUP
-------------- --------------------
1              NA
2              1
3              1
4              1
```

Properties of a Member_Parentrel Relation

Table 8–3 describes the OLAP DML properties of a *member_parentrel* relation. For descriptions of the properties independent of the object type, refer to Appendix A.

Table 8–3 Member_Parentrel Relation Properties

Property	Value
AW$CLASS	IMPLEMENTATION
AW$CREATEDBY	Creator of the relation; the Refresh tool requires a value of AW$CREATE, which indicates that the relation was created by AWCREATE programs
AW$LASTMODIFIED	Date and time the relation was last accessed by an analytic workspace tool
AW$PARENT_NAME	*Dimdef* dimension
AW$ROLE	MEMBER_PARENTREL
AW$STATE	CREATED

Member_Gid Variable

Member_gid variables improve the performance of views for the OLAP API. This integer variable identifies the depth in the hierarchy of each dimension member. This information is generated by the GROUPINGID command in the OLAP DML; refer to its entry in the *Oracle OLAP DML Reference* for information about its contents. The standard name for a *member_gid* variable is *dimdef*_GID.

Contents of a Member_GID Variable

Example 8–4 shows the *member_gid* variable for the CHANNEL dimension in the GLOBAL analytic workspace. It shows that channels 2, 3, and 4 are at the base level (0) and channel 1 is one level deep (1).

Example 8–4 CHANNEL Member_Gid in Global

```
REPORT DOWN channel W 20 channel_gid

               ----CHANNEL_GID-----
               --CHANNEL_HIERLIST--
CHANNEL           CHANNEL_ROLLUP
------------- --------------------
1                                1
2                                0
3                                0
4                                0
```

Properties of a Member_Gid Variable

Table 8–4 describes the OLAP DML properties of a *member_gid* variable. For descriptions of the properties independent of the object type, refer to Appendix A.

Table 8–4 Member_GID Variable Properties

Property	Value
AW$CLASS	FEATURES
AW$CREATEDBY	Creator of the variable; the Refresh tool requires a value of AW$CREATE, which indicates that the variable was created by AWCREATE programs
AW$LASTMODIFIED	Date and time the variable was last accessed by an analytic workspace tool
AW$PARENT_NAME	*Dimdef* dimension
AW$ROLE	MEMBER_GID
AW$STATE	CREATED

Member_Inhier Variable

Member_inhier variables are used to improve the performance of views for the OLAP API. This Boolean variable identifies whether a dimension member belongs to a level that is included in a particular hierarchy. The information is acquired from the OLAP Catalog metadata, and typically is useful only for dimensions with multiple hierarchies. The standard name for a *member_inhier* variable is *dimension*_INHIER.

Contents of a Member_Inhier Variable

Example 8–5 shows the contents of the *member_inhier* variable for the CUSTOMER dimension of the GLOBAL analytic workspace. YES indicates that the dimension member is in the hierarchy; NA indicates that it is not in the hierarchy.

Example 8–5 CUSTOMER Member_Inhier Variable in GLOBAL

```
LIMIT customer TO customer_levelrel EQ 'SHIP_TO' "Select base-level members
LIMIT customer KEEP FIRST 1                      "Keep just the first one
LIMIT customer ADD ANCESTORS USING customer_parentrel "Add its ancestors
REPORT DOWN customer W 15 customer_inhier

               --------CUSTOMER_INHIER--------
               -------CUSTOMER_HIERLIST-------
CUSTOMER          SHIPMENTS    MARKET_SEGMENT
--------------- --------------- ---------------
46                          yes             yes
21                          yes              NA
22                           NA             yes
10                          yes              NA
5                            NA             yes
1                           yes              NA
7                            NA             yes
```

Properties of a Member_Inhier Variable

Table 8–5 describes the OLAP DML properties of a *member_inhier* variable. For descriptions of the properties independent of the object type, refer to Appendix A.

Table 8–5 Member_Inhier Variable Properties

Property	Value
AW$CLASS	FEATURES
AW$CREATEDBY	Creator of the variable; the Refresh tool requires a value of AW$CREATE, which indicates that the variable was created by AWCREATE programs
AW$LASTMODIFIED	Date and time the variable was last accessed by an analytic workspace tool
AW$PARENT_NAME	*Dimdef* dimension
AW$ROLE	MEMBER_INHIER
AW$STATE	CREATED

Standard Form Metadata for Hierarchies

Standard form metadata for hierarchies is stored in these objects:

- ALL_HIERARCHIES dimension
- ALL_DESCRIPTIONS variable
- DIM_HIERARCHIES valueset
- DEFAULT_HIER relation

ALL_HIERARCHIES Dimension

The ALL_HIERARCHIES dimension contains the names of all hierarchies in this format:

workspace.dimension.hierarchy.HIERARCHY

For example: GLOBAL_AW.CUSTOMER.SHIPMENTS.HIERARCHY

ALL_HIERARCHIES is a base dimension of the ALL_OBJECTS concat dimension. ALL_OBJECTS dimensions ALL_DESCRIPTIONS and AW_NAMES. ALL_DESCRIPTIONS provides values for the hierarchies, but AW_NAMES does not.

ALL_DESCRIPTIONS Variable for Hierarchies

The ALL_DESCRIPTIONS variable contains short, long, and plural names for the hierarchies. All objects have a short name acquired from the metadata, but may or may not have long and plural names.

DIM_HIERARCHIES Valueset

The DIM_HIERARCHIES valueset identifies the hierarchies defined for each dimension.

DEFAULT_HIER Relation

The DEFAULT_HIER relation identifies the default hierarchy for each dimension.

Standard Form Levels

Levels are the basis of dimension hierarchies. A level belongs to one or more hierarchies. These objects support level definitions:

- *Levellist* dimension
- *Member_levelrel* relation
- *Member_familyrel* relation

For additional information about levels, refer to "Implementation Class Objects" on page A-14 and "Features Class Objects" on page A-35.

Levellist Dimension

A *levellist* dimension stores the names of all levels for all hierarchies defined for a particular dimension. The information is acquired from the OLAP Catalog. This text dimension typically has the name *dimdef*_LEVELLIST.

Contents of a Levellist Dimension

Example 8–6 shows the CUSTOMER levellist dimension in GLOBAL, which contains the levels for both the SHIPMENTS and MARKET_SEGMENT hierarchies.

Example 8–6 CUSTOMER Levellist Dimension in GLOBAL

```
REPORT W 20 customer_levellist

CUSTOMER_LEVELLIST
--------------------
TOTAL_MARKET
MARKET_SEGMENT
ACCOUNT
ALL_CUSTOMERS
REGION
WAREHOUSE
SHIP_TO
```

Properties of a Levellist Dimension

Table 8–6 describes the properties of a *levellist* dimension. For descriptions of the properties independent of the object type, refer to Appendix A.

Table 8–6 Levellist Dimension Properties

Property	Value
AW$CLASS	IMPLEMENTATION
AW$CREATEDBY	Creator of the level; the Refresh tool requires a value of AW$CREATE, which indicates that the level was created by AWCREATE programs
AW$LASTMODIFIED	Date and time the level was last accessed by an analytic workspace tool
AW$LOGICAL_NAME	NA
AW$PARENT_NAME	*Dimdef* dimension
AW$ROLE	LEVELLIST
AW$STATE	CREATED

Member_Levelrel Relation

A *member_levelrel* relation identifies the level of each dimension member. It facilitates the selection of dimension members by level. The information is acquired from the relational fact tables. This text dimension typically has the name *dimdef*_LEVELREL.

Contents of a Level Relation

Example 8–7 shows the CUSTOMER *member_levelrel* relation in GLOBAL.

Example 8–7 CUSTOMER Member_Levelrel Relation in GLOBAL

```
LIMIT customer TO '62'                                  "Select customer 62
LIMIT customer ADD ANCESTORS USING customer_parentrel "Add ancestors
REPORT DOWN customer W 20 customer_levelrel

CUSTOMER        CUSTOMER_LEVELREL
--------------- --------------------
62              SHIP_TO
21              WAREHOUSE
27              ACCOUNT
10              REGION
6               MARKET_SEGMENT
1               ALL_CUSTOMERS
7               TOTAL_MARKET
```

Properties of a Member_Levelrel Relation

Table 8–7 describes the OLAP DML properties of a *member_levelrel* relation. For descriptions of the properties independent of the object type, refer to Appendix A.

Table 8–7 Member_Levelrel Relation Properties

Property	Value
AW$CLASS	IMPLEMENTATION
AW$CREATEDBY	Creator of the level; the Refresh tool requires a value of AW$CREATE, which indicates that the level was created by AWCREATE programs
AW$LASTMODIFIED	Date and time the level was last accessed by an analytic workspace tool
AW$PARENT_NAME	*Dimdef* dimension
AW$ROLE	MEMBER_LEVELREL
AW$STATE	CREATED

Member_Familyrel Relation

Member_familyrel relations improve the performance of views for Oracle Discoverer. It provides a crosstab with the full parentage of each dimension member within a single row. The standard name for a family relation is *dimdef*_FAMILYREL.

Contents of a Family Relation

Example 8–8 shows the CUSTOMER family relation in GLOBAL.

Example 8–8 CUSTOMER Family Relation in GLOBAL

```
LIMIT customer TO '78'                                "Select customer 78
LIMIT customer ADD ANCESTORS USING customer_parentrel  "Add the ancestors
LIMIT customer_hierlist TO 'SHIPMENTS'                "Select the SHIPMENTS hierarchy

REPORT customer_familyrel
```

```
CUSTOMER_HIERLIST: SHIPMENTS
               ----------------------------CUSTOMER_FAMILYREL----------------------------
               ---------------------------------CUSTOMER---------------------------------
CUSTOMER_LEVEL
LIST               78         21         31         10         2          1          7
-------------- ---------- ---------- ---------- ---------- ---------- ---------- ----------
TOTAL_MARKET   NA         NA         NA         NA         NA         NA         NA
MARKET_SEGMENT NA         NA         NA         NA         NA         NA         NA
ACCOUNT        NA         NA         NA         NA         NA         NA         NA
ALL_CUSTOMERS  1          1          NA         1          NA         1          NA
REGION         10         10         NA         10         NA         NA         NA
WAREHOUSE      21         21         NA         NA         NA         NA         NA
SHIP_TO        78         NA         NA         NA         NA         NA         NA
```

Properties of a Member_Familyrel Relation

Table 8–8 describes the OLAP DML properties of a *member_familyrel* relation. For descriptions of the properties independent of the object type, refer to Appendix A.

Table 8–8 Member_Familyrel Relation Properties

Property	Value
AW$CLASS	FEATURES
AW$CREATEDBY	Creator of the relation; the Refresh tool requires a value of AW$CREATE, which indicates that the relation was created by AWCREATE programs
AW$LASTMODIFIED	Date and time the relation was last accessed by an analytic workspace tool
AW$PARENT_NAME	*Dimdef* dimension
AW$ROLE	MEMBER_FAMILYREL
AW$STATE	CREATED

Standard Form Metadata for Levels

Standard form metadata for levels is stored in these objects:

- ALL_LEVELS dimension
- ALL_DESCRIPTIONS variable
- DIM_LEVELS valueset

ALL_LEVELS Dimension

The ALL_LEVELS dimension contains the names of all levels in this format:

workspace.dimension.level.LEVEL

For example: GLOBAL_AW.TIME.Quarter.LEVEL

ALL_LEVELS is a base dimension of the ALL_OBJECTS concat dimension. ALL_OBJECTS dimensions ALL_DESCRIPTIONS and AW_NAMES. ALL_DESCRIPTIONS provides values for the levels, but AW_NAMES does not.

ALL_DESCRIPTIONS Variable for Levels

The ALL_DESCRIPTIONS variable contains short, long, and plural names for the levels. All levels have a short name acquired from the metadata, but may or may not have long and plural names.

DIM_LEVELS Valueset

The DIM_LEVELS valueset identifies the levels defined for each dimension.

Standard Form Attributes

Attributes are defined as variables, usually with a text data type. They provide information about the dimension members, and are typically acquired from relational dimension tables. An attribute is dimensioned by a *dimdef* dimension, a *hierlist* dimension, and the ALL_LANGUAGES dimension.

Dimension members are sorted during a load, and an attribute named *dimension*_ORDER identifies the original order in which they were fetched, row by row, into the analytic workspace.

Table 8–9 describes the OLAP DML properties for attributes. For descriptions of the properties independent of the object type, refer to Appendix A.

Table 8–9 Attribute Properties

Property	Value
AW$CLASS	IMPLEMENTATION
AW$CREATEDBY	Creator of the attribute; the Refresh tool requires a value of AW$CREATE, which indicates that the attribute was created by AWCREATE programs

Table 8–9 (Cont.) Attribute Properties

Property	Value
AW$LASTMODIFIED	Date and time the attribute was last accessed by an analytic workspace tool
AW$LOGICAL_NAME	OLAP Catalog source attribute name
AW$PARENT_NAME	*Dimdef* dimension
AW$ROLE	ATTRDEF
AW$STATE	CREATED
AW$TYPE	Long Description, Short Description, Time Span, End Date, and DEFAULT_ORDER are currently used as special attribute types
SOURCE_DATATYPE	The basic data type of the source column, such as VARCHAR2 or DATE; used by AWCREATE programs
SOURCE_DIMNAME	OLAP Catalog source dimension name; used by AWCREATE programs
SOURCE_NAME	OLAP Catalog source attribute name; used by AWCREATE programs and the active catalog
SOURCE_OWNER	Owner of OLAP Catalog source metadata; used by AWCREATE programs and the active catalog

For additional information about standard form attributes, refer to the "Implementation Class Objects" on page A-14 and "Catalogs Class Objects" on page A-25.

ALL_LANGUAGES Dimension

The ALL_LANGUAGES dimension enables an analytic workspace to support multiple languages. It initially has one member, which identifies the database (and thus the analytic workspace) territory and language, for example, AMERICAN_AMERICA.

Table 8–10 describes the OLAP DML properties of the ALL_LANGUAGES dimension. For descriptions of the properties independent of the object type, refer to Appendix A.

Table 8–10 ALL_LANGUAGES Dimension Properties

Property	Value
AW$CLASS	CATALOG
AW$CREATEDBY	AW$CREATE
AW$LASTMODIFIED	Date and time
AW$ROLE	ALL_LANGUAGES
AW$STATE	CREATED

Standard Form Metadata for Attributes

Standard form metadata for attributes is stored in these objects:

- ALL_ATTRIBUTES dimension
- ALL_DESCRIPTIONS variable
- AW_NAMES variable
- DIM_ATTRIBUTES valueset

ALL_ATTRIBUTES Dimension

The ALL_ATTRIBUTES dimension contains the names of all attributes in this format:

workspace.dimension.attribute.ATTRIBUTE

For example: GLOBAL_AW.TIME.End_Date.ATTRIBUTE

ALL_ATTRIBUTES is a base dimension of the ALL_OBJECTS concat dimension. ALL_OBJECTS dimensions ALL_DESCRIPTIONS and AW_NAMES, so these catalogs have an entry for each attribute.

ALL_DESCRIPTIONS Variable for Attributes

The ALL_DESCRIPTIONS variable contains short, long, and plural names for the attributes. All objects have a short name acquired from the metadata, but may or may not have long and plural names.

AW_NAMES Variable for Attributes

The AW_NAMES measure provides a name for each attribute in this format:

```
schema.workspace!attribute
```

For example: GLOBAL_AW.GLOBAL!TIME_END_DATE

Standard Form Measures

Each measure is defined by two workspace objects: a variable and a formula.

For additional information about standard form measures, refer to "Implementation Class Objects" on page A-14 and "Extensions Class Objects" on page A-40.

Measure Variable

A measure variable initially contains only base-level data, which is typically acquired from a relational fact table. If you deploy an aggregation plan, then the variable also contains precalculated aggregate levels.

A measure variable has a DECIMAL data type unless you redefined it before loading data during the initial build. The standard name for a measure variable is *measuredef*_VARIABLE.

Table 8–11 describes its properties. For descriptions of the properties independent of the object type, refer to Appendix A.

Table 8–11 Measuredef_VARIABLE Properties

Property	Value
AW$CLASS	EXTENSION
AW$CREATEDBY	Creator of the cube; the Refresh tool requires a value of AW$CREATE, which indicates that the object was created by AWCREATE programs
AW$LASTMODIFIED	Date and time the cube was last accessed by an analytic workspace tool
AW$PARENT_NAME	Name of the measure
AW$ROLE	VARIABLE
AW$SEGWDTH_CMD	CHGDFN command for defining the segment size
AW$STATE	CREATED

Measuredef Formula

A *measuredef* formula calculates the aggregate data using a set of aggregation rules stored in an aggmap. Its standard name is the name of the measure. Table 8–12 describes its properties. For descriptions of the properties independent of the object type, refer to Appendix A.

Table 8–12 Measuredef Formula Properties

Property	Value
AW$CLASS	IMPLEMENTATION
AW$COMPSPEC	Name of the current deployed aggmap
AW$CREATEDBY	Creator of the measure; some tools may require a value of AW$CREATE, which indicates that the measure was created by AWCREATE programs
AW$LASTMODIFIED	Date and time the measure was last accessed by an analytic workspace tool
AW$LOGICAL_NAME	OLAP Catalog measure name
AW$PARENT_NAME	Cube name
AW$ROLE	MEASUREDEF
AW$STATE	CREATED
MEASCOLS	Name of the analytic workspace catalog (named *cube*.MSCL) that identifies the source column in the fact table; used by AWCREATE programs
SOURCE_CUBENAME	Name of the OLAP Catalog source cube; used by AWCREATE programs
SOURCE_NAME	OLAP Catalog source measure; used by AWCREATE programs and the active catalog
SOURCE_OWNER	Owner of OLAP Catalog source metadata; used by AWCREATE programs and the active catalog

Standard Form Metadata for Measures

Standard form metadata for measures is stored in these objects:

- ALL_MEASURES dimension
- ALL_DESCRIPTIONS variable

- AW_NAMES variable
- CUBE_MEASURES valueset

ALL_MEASURES Dimension

The ALL_MEASURES dimension contains the names of all measures in this format:

workspace.cube.measure.MEASURE

For example: GLOBAL_AW.UNITS_CUBE.UNITS.MEASURE

ALL_MEASURES is a base dimension of the ALL_OBJECTS concat dimension. ALL_OBJECTS dimensions ALL_DESCRIPTIONS and AW_NAMES, so these catalogs have an entry for each measure.

ALL_DESCRIPTIONS Variable for Measures

The ALL_DESCRIPTIONS variable contains short, long, and plural names for the measures. All objects have a short name acquired from the metadata, but may or may not have long and plural names.

AW_NAMES Variable for Measures

The AW_NAMES measure provides a name for each measure in this format:

schema.workspace!measure

For example: GLOBAL_AW.GLOBAL!UNITS

CUBE_MEASURES Valueset

The CUBE_MEASURES valueset identifies the measures for each cube.

Standard Form Cubes

Cubes are implemented as text dimensions that list the names of the dimensions (sometimes called the edges) of the cube. A default aggregation map and composite dimension are also defined for all measures in the cube.

For additional information about standard form cubes, refer to "Implementation Class Objects" on page A-14.

Cubedef Dimension

The *cubedef* dimension lists the names of the *dimdef* dimensions, such as TIME and PRODUCT, for measures in the cube. The standard name for this dimension is the name of the logical cube, such as UNITS_CUBE. The name has a prefix if you specified one in the build options.

Contents of a Cubedef Dimension

Example 8–9 shows the *cubedef* dimension for the UNITS_CUBE in GLOBAL.

Example 8–9 Units Cube Dimension in GLOBAL

```
REPORT units_cube

UNITS_CUBE
--------------
CHANNEL
CUSTOMER
PRODUCT
TIME
```

Properties of a Cubedef Dimension

Table 8–13 describes the OLAP DML properties of a *cubedef* dimension. For descriptions of the properties independent of the object type, refer to Appendix A.

Table 8–13 Cubedef Properties

Property	Value
AGGMAPLIST	Single- or multiline text string with the names of all aggmaps defined for this cube, used by AWCREATE programs
AW$CLASS	IMPLEMENTATION
AW$CREATEDBY	Creator of the cube; the Refresh tool and the Aggregation Plan tool requires a value of AW$CREATE, which indicates that the object was created by AWCREATE programs
AW$LASTMODIFIED	Date and time the cube was last accessed by an analytic workspace tool
AW$LOADPRGS	Name of the OLAP DML program used to fetch data from the relational schema into the analytic workspace
AW$LOGICAL_NAME	Logical name of the source cube, such as a cube defined in the OLAP Catalog

Table 8–13 (Cont.) Cubedef Properties

Property	Value
AW$LOOPSPEC	The workspace composite used to define variables for this cube
AW$PARENT_NAME	NA
AW$ROLE	CUBEDEF
DISPLAY_NAME	OLAP Catalog source display name
FORMDIMS	Ordered list of dimensions for measure formulas; used by AWCREATE programs
LOADNAME	The name of the load program used to populate the cube; used by AWCREATE programs
LOADTYPE	LOAD_DATA when data is loaded as part of the build, or LOAD_PROGRAM if the DML load program is created but not run; these are keywords for DBMS_AWM.CREATE_AWCUBELOAD_SPEC and are used by AWCREATE programs
SOURCE_NAME	OLAP Catalog source cube; used by AWCREATE programs and the active catalog
SOURCE_OWNER	Owner of OLAP Catalog source metadata; used by AWCREATE programs and the active catalog
SYS_DIMS	Ordered list of dimensions for measure variables, usually Time followed by a composite of all other dimensions; used by AWCREATE programs
SYS_DIMSML	Alphabetized list of dimensions for the cube; used by AWCREATE programs

Comspec Aggregation Map

A default aggmap is created for each cube, which specifies runtime aggregation across all dimensions. This aggmap is initially referenced by the formulas for all measures associated with the cube. When you create and deploy aggregation plans using the wizards in Analytic Workspace Manager, you create new aggmaps and change the formulas for specified measures.

The standard name for default aggmaps is *cubedef*_AGGMAP_AWCREATEDDEFAULT_1, for example, UNITS_CUBE_AGGMAP_AWCREATEDDEFAULT_1.

Table 8–14 describes the properties of a *comspec* aggmap. For descriptions of the properties independent of the object type, refer to Appendix A.

Table 8–14 Comspec Aggmap Properties

Property	Value
AW$CLASS	IMPLEMENTATION
AW$COUNTVARCMD	Name of the integer variable used when the aggmap calculates an average; otherwise NA
AW$CREATEDBY	Creator of the aggmap; some tools may require a value of AW$CREATE, which indicates that the aggmap was created by AWCREATE programs
AW$LASTMODIFIED	Date and time the aggmap was last accessed by an analytic workspace tool
AW$PARENT_NAME	Workspace cube name
AW$ROLE	COMSPEC
AW$STATE	CREATED
ISDFLTAGGMAP	YES for the default aggregation map, or NO for aggmaps created after the initial build; used during a data refresh

Loopspec Composite Dimension

Using the Create Analytic Workspace wizard, you can accept the default composite or define a composite yourself.

A default composite is named *cubedef*_COMPOSITE and consists of all dimensions of the cube except Time. The dimensions are ordered from the one with the most members to the one with the least members.

A custom composite has the name and characteristics that you assigned to it.

Table 8–15 describes the properties of a *loopspec* composite dimension. For descriptions of the properties independent of the object type, refer to Appendix A.

Table 8–15 Loopspec Composite Properties

Property	Value
AW$CLASS	IMPLEMENTATION
AW$CREATEDBY	Creator of the composite; some tools may require a value of AW$CREATE, which indicates that the composite was created by AWCREATE programs
AW$LASTMODIFIED	Date and time the composite was last accessed by an analytic workspace tool

Table 8–15 (Cont.) Loopspec Composite Properties

Property	Value
AW$PARENT_NAME	*Cubedef* dimension
AW$ROLE	LOOPSPEC
AW$STATE	CREATED

For additional information about *loopspec* composites, refer to "Implementation Class Objects" on page A-14.

Standard Form Metadata for Cubes

Standard form metadata for cubes is stored in these objects:

- ALL_CUBES dimension
- ALL_DESCRIPTIONS variable
- AW_NAMES variable
- CUBE_MEASURES valueset

ALL_CUBES Dimension

The ALL_CUBES dimension contains the names of all cubes in this format:

workspace.cube.CUBE

For example: GLOBAL_AW.UNITS_CUBE.CUBE

ALL_CUBES is a base dimension of the ALL_OBJECTS concat dimension. ALL_OBJECTS dimensions ALL_DESCRIPTIONS and AW_NAMES, so these catalogs have an entry for each cube.

ALL_DESCRIPTIONS Variable for Cubes

The ALL_DESCRIPTIONS variable contains short, long, and plural names for the cubes. All objects have a short name acquired from the metadata, but may or may not have long and plural names.

AW_NAMES Variable for Cubes

The AW_NAMES measure provides a name for each cube in this format:

```
schema.workspace!cube
```

For example: GLOBAL_AW.GLOBAL!UNITS_CUBE

CUBE_MEASURES Valueset

The CUBE_MEASURES valueset identifies the measures for each cube.

Standard Form Catalogs

Database standard form requires a Catalogs class of objects. These objects hold information about the objects in the analytic workspace that implement the logical model. Table 8–16 describes these objects. For additional information about the catalogs, refer to "Catalogs Class Objects" on page A-25.

Table 8–16 Standard Form Catalogs

Catalog	Object Type	Contents
ALL_ATTRIBUTES	Dimension	The full name of each attribute of each workspace dimension in the form *schema.dimension.attribute*.ATTRIBUTE
ALL_CUBES	Dimension	The full name of each workspace cube (that is, a dimension whose values are the dimensions of a cube) in the form *schema.cube*.CUBE
ALL_DESCRIPTIONS	Variable	Contains the short, long, and plural descriptions of the logical objects
ALL_DIMENSIONS	Dimension	The full name of each workspace data dimension (that is, dimensions used in data cubes) in the form *schema.dimension*.DIMENSION
ALL_HIERARCHIES	Dimension	The full name of each hierarchy of each workspace dimension in the form *schema.dimension.hierarchy*.HIERARCHY
ALL_LEVELS	Dimension	The full name of each level of each workspace dimension in the form *schema.dimension.level*.LEVEL
ALL_MEASURES	Dimension	The full name of each workspace measure (that is, a formula that returns a fully solved measure) in the form *schema.measure*.MEASURE.
ALL_OBJECTS	Concat dimension	ALL_DIMENSIONS, ALL_CUBES, ALL_MEASURES, ALL_HIERARCHIES, ALL_LEVELS, and ALL_ATTRIBUTES

Table 8–16 (Cont.) Standard Form Catalogs

Catalog	Object Type	Contents
AW_NAMES	Variable	The names of the analytic workspace objects that implement each logical object defined by the source metadata
CUBE_MEASURES	Valueset	A list of measures that belong to each cube
DEFAULT_HIER	Relation	The full name of the default hierarchy for each dimension
DIM_ATTRIBUTES	Valueset	A list of attributes that belong to each dimension
DIM_HIERARCHIES	Valueset	A list of hierarchies that belong to each dimension
DIM_LEVELS	Valueset	A list of levels that belong to each dimension

OLAP API Enabler Catalogs

The build process creates numerous objects within an analytic workspace to support the enabler for the OLAP API and BI Beans, and the active catalog. The enabler also creates some objects. Table 8–17 describes the catalogs used by the OLAP API.

Note: The OLAP API Enabler catalogs may change or disappear in future software releases.

Table 8–17 OLAP API Enabler Catalogs

Catalog	Object Type	Contents
__SYS_HIERCJT	Conjoint	The combinations of dimension hierarchies for which a fact view is required; created for transient use during enablement
__SYS_HIERCJT_BUILD	Conjoint	The combinations of dimension hierarchies for which a fact view is required; created for transient use during cube refresh
*cube*_NEWSNAPSHOT_DIM	Dimension	An integer dimension for *cube*_NEWSNAPSHOT_VAR
*cube*_SNAPSHOT_DIM	Dimension	An integer dimension for *cube*_SNAPSHOT_VAR
OLAP_SYS_ADTDIM	Dimension	The names of the object types used by the OLAP_TABLE function to generate the views
OLAP_SYS_ADTTBLDIM	Dimension	The names of the table types used by the OLAP_TABLE function to generate the views

Table 8–17 (Cont.) OLAP API Enabler Catalogs

Catalog	Object Type	Contents
OLAP_SYS_CUBENAME_DIM	Dimension	The names of the cubes in the analytic workspace
OLAP_SYS_CUBEVIEW_DIM	Dimension	The names of the fact views defined for the analytic workspace; used by the ALL_OLAP2_CUBE_ENABLED_VIEW active catalog
OLAP_SYS_DIMNAME_DIM	Dimension	The names of the dimensions in the analytic workspace
OLAP_SYS_DIMVIEW_DIM	Dimension	The name of the relational dimension view for each hierarchy; used by the ALL_OLAP2_DIM_ENABLED_VIEW active catalog
OLAP_SYS_VIEWDIM	Dimension	The names of the relational views defined for the analytic workspace
OLAP_SYS_ADTREL	Relation	The name of the object type used by the OLAP_TABLE function for each relational view
OLAP_SYS_ADTTBLREL	Relation	The name of the table type used by the OLAP_TABLE function for each relational view
OLAP_SYS_CUBENAME_REL	Relation	The analytic workspace cube represented by each fact view
OLAP_SYS_DIMNAME_REL	Relation	The analytic workspace dimension represented by each dimension view.
OLAP_SYS_CUBEVALSET	Valueset	The names of fact views during cube refresh; otherwise, NA
OLAP_SYS_DIMVALSET	Valueset	The names of dimension views during dimension refresh; otherwise, NA
*cube*_NEWSNAPSHOT_VAR	Variable	Identifies the dimensions and hierarchies associated with a cube at the time that the cube is being enabled for the OLAP API. If this variable is identical to *cube*_SNAPSHOT_VAR, then the views are still current and do not need to be regenerated.
*cube*_SNAPSHOT_VAR	Variable	Identifies dimensions and hierarchies associated with a cube for the views currently generated for the OLAP API.
*cube*_SYS_ENABLE	Variable	A multiline text string with the names of the object type, table type, and views generated by the enabler for an analytic workspace cube
*dimension*_SYS_ENABLE	Variable	A multiline text string with the names of the object type, table type, and views generated by the enabler for an analytic workspace dimension

Table 8–17 (Cont.) OLAP API Enabler Catalogs

Catalog	Object Type	Contents
OLAP_SYS_CUBEADTNAME_VAR	Variable	The object type used by the OLAP_TABLE function to generate each fact view
OLAP_SYS_CUBEAWOWNER_VAR	Variable	The schema owner of each fact view; used by the ALL_OLAP2_CUBE_ENABLED_VIEW active catalog
OLAP_SYS_CUBEHIERCOMBO_VAR	Variable	An integer value for each combination of dimension hierarchies represented by a fact view; used by the ALL_OLAP2_CUBE_ENABLED_VIEW active catalog
OLAP_SYS_CUBEHIERCOMBOSTR_VAR	Variable	Text strings that identify the dimensions and hierarchies represented by each fact view; used by the ALL_OLAP2_CUBE_ENABLED_VIEWS active catalog
OLAP_SYS_CUBENAME_VAR	Variable	The analytic workspace cube represented by each fact view; used by the ALL_OLAP2_CUBE_ENABLED_VIEW active catalog
OLAP_SYS_CUBETBLNAME_VAR	Variable	The table type used by the OLAP_TABLE function to generate each fact view
OLAP_SYS_CUBEUSERVIEW_VAR	Variable	New names assigned to workspace cubes using the CWM2_OLAP_CUBE.SET_CUBE_NAME procedure, or NA when new names have not been defined; used by the ALL_OLAP2_CUBE_ENABLED_VIEW active catalog
OLAP_SYS_DIMADTNAME_VAR	Variable	The name of the object type used by the OLAP_TABLE function for each dimension view
OLAP_SYS_DIMAWOWNER_VAR	Variable	The schema owner of each dimension view; used by the ALL_OLAP2_DIM_ENABLED_VIEW active catalog
OLAP_SYS_DIMHIERNAME_VAR	Variable	The name of the hierarchy represented by each dimension view; used by the ALL_OLAP2_DIM_ENABLED_VIEW active catalog
OLAP_SYS_DIMHIERPOS_VAR	Variable	The numerical position of each hierarchy in the *dimension*_HIERLIST hierarchy dimension.
OLAP_SYS_DIMNAME_VAR	Variable	The analytic workspace name of the dimension represented by each dimension view; used by the ALL_OLAP2_DIM_ENABLED_VIEW active catalog

Table 8–17 (Cont.) OLAP API Enabler Catalogs

Catalog	Object Type	Contents
OLAP_SYS_DIMTBLNAME_VAR	Variable	The name of the table type used by the OLAP_TABLE function for each dimension view.
OLAP_SYS_DIMUSERVIEW_VAR	Variable	New names assigned to workspace cubes using the CWM2_OLAP_DIMENSION.SET_DIMENSION_NAME procedure, or NA when new names have not been defined; used by the ALL_OLAP2_DIM_ENABLED_VIEW active catalog
OLAP_SYS_LIMITMAP	Variable	The limit map used by the OLAP_TABLE function for each relational view

AWCREATE Catalogs

Several catalogs are used during the build and refresh process, and currently persist in the analytic workspace. Some of them are also used transiently during the enablement process for Oracle Discoverer. Table 8–18 describes the AWCREATE catalogs.

Note: The AWCREATE catalogs may change or disappear in future software releases.

Table 8–18 AWCREATE Catalogs

Catalog	Object Type	Contents
*dimension*_SRCCOMPOSITE	Composite	A composite dimension composed of the *dimension*_HIERLIST, *dimension*_LEVELLIST, and *dimension*_LEVELCOLLIST dimensions
*cube*_HIERCJT	Conjoint	The names of the hierarchies for the dimensions of the cube
*dimension*_LEVELCOLLIST	Dimension	Integer values
*cube*_HIERCJT.DMKY	Variable	The name of the key column of source dimension table for the dimensions of the cube
*cube*_HIERCJT.DMLV	Variable	The name of the dimension level at which data is stored
*cube*_HIERCJT.FT	Variable	The name of the fact table that is the source for the cube
*cube*_HIERCJT.HC	Variable	Integer values

Table 8–18 (Cont.) AWCREATE Catalogs

Catalog	Object Type	Contents
cube_measure.MSCL	Variable	The name of the source column for the measure
*dimension_attribute*_SRCATTRCOL	Variable	The names of the source columns for the attributes by hierarchy and level
*dimension_attribute*_SRCATTROWNER	Variable	The name of the schema owner of the source dimension table for the attributes by hierarchy and level
*dimension_attribute*_SRCATTRTBL	Variable	The name of the source dimension table for the attributes by hierarchy and level
*dimension*_LEVELCOLMAP	Variable	The source dimension value corresponding to each dimension member in the analytic workspace; the values can acquire a prefix during the build
*dimension*_SRCLVLCOL	Variable	The name of the source column for the level
*dimension*_SRCLVLOWNER	Variable	The name of the schema owner of the source dimension table
*dimension*_SRCLVLPNTCOL	Variable	The name of the source column for the parent level
*dimension*_SRCLVLTBL	Variable	The name of the source dimension table

Part III

Acquiring Data From Additional Sources

Part III describes ways that you can create a new analytic workspace or enhance an existing one with data from sources other than a star or snowflake schema. The data can be generated by analytic functions available in an analytic workspace, or from external sources such as flat files.

Part III contains the following chapters:

9

Adding Measures to a Standard Form Analytic Workspace

In this chapter, you will learn how to create new measures as a permanent addition to a standard form analytic workspace. Using the method described in this chapter, you can define custom measures that can store data (instead of calculating it on demand) and are indistinguishable from any other measures in the analytic workspace. However, the process is more complex than using `DBMS_AW_UTILITIES` or `OLAP_EXPRESSION`, as described in Chapter 7.

You can also populate the new workspace objects by using advanced calculation methods (such as forecasting or allocation) or by loading the data from external sources.

You will also learn various methods of executing OLAP DML commands.

This chapter contains the following topics:

Working in a Standard Form Analytic Workspace

Analytic Workspace Manager and the current generation of tools can only be used with database standard form analytic workspaces. As described in Chapter 8, database standard form (or simply, standard form) identifies the types of objects that must exist, the OLAP DML properties that must be assigned to them, and a variety of catalogs within the analytic workspace for registering workspace objects.

Conformity with this standard enables the tools to perform their jobs, such as aggregating and refreshing the data, and generating views and metadata. Otherwise, the tools have no means of identifying the function of workspace objects within a logical multidimensional model.

Over the life span of an analytic workspace, you may want to add measures from a new data source, or define a permanent custom measure using some of the more advanced calculation techniques, such as forecasting or allocation.

While you would need to refresh these measures manually, you would still want to use the aggregation and enablement tools with the new measures.

For the new measures to be accessible to the tools, you must take these steps:

1. Define the appropriate workspace objects (formulas, or measures, or both).
2. Attach OLAP DML properties to the objects with the appropriate values.
3. Register the objects in the standard form workspace catalogs.

This chapter explains how to perform these steps.

See Also:

- Chapter 7 for alternative methods of adding custom measures to an analytic workspace.
- Chapter 8 for descriptions of catalogs, objects, and properties in a standard form analytic workspace.
- Chapter 11 for methods of populating new measures from external sources such as flat files.

Methods of Executing OLAP DML Commands

When working with an analytic workspace, you can use any of these methods for issuing OLAP DML commands to the OLAP engine for execution:

- Using dialogs in Analytic Workspace Manager, you can create analytic workspaces, and define and modify workspace objects such as dimensions, variables, models, and aggmaps.
- Within OLAP Worksheet, you can open an interactive OLAP session in which to issue OLAP DML commands. You can run OLAP Worksheet from Analytic Workspace Manager. Note that when you attach a workspace in Analytic Workspace Manager, you can modify it using either the dialogs or OLAP Worksheet; they share the same session.

- Within a SQL session (such as in SQL*Plus), you can embed OLAP DML commands in calls to the `DBMS_AW.EXECUTE` PL/SQL procedure.

This chapter identifies how to use the Analytic Workspace Manager dialogs as much as possible. As you become more familiar with the OLAP DML, you may find that the other methods are very useful.

When developing SQL- or Java-based applications, you can also embed OLAP DML in these ways:

- In SQL programs, you can embed OLAP DML commands using the procedures in the `DBMS_AW` package.
- In Java programs, you can embed OLAP DML commands using the `SPLExecutor` class in the OLAP API.

Both the `DBMS_AW` package and OLAP Worksheet enable you to intersperse SQL and OLAP DML commands within a single working environment.

See Also:

- OLAP API Javadoc for a description of the SPLExecutor class.
- *Oracle OLAP Reference* for descriptions of the procedures in the `DBMS_AW` package.

Using Analytic Workspace Manager to Execute OLAP DML

In Chapter 6, you learned to use the various wizards in Analytic Workspace Manager for creating and managing analytic workspaces. These wizards are available in the OLAP Catalog view. The Object View in Analytic Workspace Manager has property sheets and menus for defining all of the object types available in analytic workspaces. The wizards, property sheets, and menu choices send OLAP DML commands to the OLAP engine for execution.

For example, to create a dimension, you open the Create Dimension dialog and define the dimension using a property sheet. When you click the **Create** button, a `DEFINE DIMENSION` command in the OLAP DML is formulated and executed in your analytic workspace.

This method is particularly good for enhancing an analytic workspace that was generated by a wizard. You may want to create a formula or modify an aggmap. By selecting an object in the Object View, you can modify many of its characteristics in the property pages. Note that some characteristics cannot be changed after an object is created (such as the dimensions or data type of a variable), so those characteristics are dimmed.

If you need to populate any new objects, you can do so by executing the appropriate DML commands in OLAP Worksheet. When you run OLAP Worksheet from within Analytic Workspace Manager, you are accessing the same session. Any changes that you make in OLAP Worksheet are immediately reflected in Analytic Workspace Manager, and vice versa. In this environment, you can alternate between the graphical and command line interfaces within the same session.

Using OLAP Worksheet to Execute OLAP DML

For anyone who is already familiar with the OLAP DML or is doing extensive development work in an analytic workspace, OLAP Worksheet offers the most suitable environment. You can open an OLAP session and work interactively in the OLAP DML, using all facets of this feature-rich language. OLAP Worksheet provides an editor for writing programs, models, and aggmaps. You can also switch to a SQL mode and issue SQL commands against relational tables and views.

You can run OLAP Worksheet from Analytic Workspace Manager.

Procedure: Opening OLAP Worksheet from Analytic Workspace Manager

1. Open Analytic Workspace Manager.
2. Connect to your database.
3. From the Tools menu, select **OLAP Worksheet**.

   The OLAP Worksheet window opens. If you have attached an analytic workspace in Analytic Workspace Manager, then that workspace is attached to your session in OLAP Worksheet.
4. To execute an OLAP DML command, type it in the input pane at the bottom of the window.

   For example, to view the list of attached analytic workspaces, issue this command:

   ```
   AW LIST
   ```

   Note that the `EXPRESS` workspace must always be attached.

Note: Because OLAP Worksheet and Analytic Workspace Manager share the same session, you must be careful when moving between the two applications. Your actions in the Object View may have consequences on commands that you issue in OLAP Worksheet. Use the `AW LIST` command to check the order in which analytic workspaces are attached, since some commands, like `LISTNAMES` and `DEFINE`, operate only on the first workspace. Also issue a `LIMIT NAME TO ALL` command before using commands like `EXPORT`, which use the status of the `NAME` dimension.

Procedure: Using the Editor in OLAP Worksheet

Use the Edit window to change the content of a program, model, or aggmap. Alternatively, you can use the property pages in Analytic Workspace Manager to edit these objects, but you cannot execute them there.

You cannot change the contents of a dimension, variable, relation, valueset, or other data container using the editor.

1. To add contents to a program object, issue this command to open the edit window:

   `EDIT` *`program_name`*

   For example, `EDIT CREATE_MEASURE`.

   `PROGRAM` is the default object type; you must specify the other types. For example, you would issue a command such as `EDIT AGGMAP units_cube_aggmap` to edit an aggregation map.

2. Type the OLAP commands that you want in the program.

3. When you are done editing the program, from the editor's File menu, choose **Save**, then **Close**.

4. To compile and execute the program, issue these commands:

   `CALL` *`program_name`*

   Note: The `COMPILE` command is optional because the program will compile automatically. However, a separate `COMPILE` command is useful for quickly identifying syntax errors in the OLAP DML commands.

5. To issue SQL commands, from the Options menu, select **SQL Mode**. To resume issuing OLAP DML commands, clear **SQL Mode**.

Example 9–1 shows a sample session in which a program named CREATE_MEASURE is created, compiled, and executed within OLAP Worksheet

Example 9–1 Creating an OLAP DML Program in OLAP Worksheet

```
DEFINE create_measure PROGRAM

LD Define a database standard form measure

EDIT create_measure
     .
     .           " Enter program code
     .           " Choose Save

CALL create_measure
```

Using DBMS_AW.EXECUTE to Execute OLAP DML

The DBMS_AW.EXECUTE procedure enables you to issue OLAP DML commands at any time within a SQL session. To see the output of the OLAP DML commands, issue this SQL command once during your session:

```
SET SERVEROUT ON FORMAT WRAPPED
```

DBMS_AW.EXECUTE Command Format

Following is the basic format of DBMW_AW.EXECUTE, in which you can substitute one or more OLAP DML commands, separated by semicolons, between the single quotes.

```
EXECUTE DBMS_AW.EXECUTE('dml_command_1; dml_command_2; dml_command_n');
```

In the following example, the first command opens the GLOBAL analytic workspace. The second command sets the focus on a measure named SALES_PP and assigns an equation to it.

```
EXECUTE DBMS_AW.EXECUTE('AW ATTACH global RW');
EXECUTE DBMS_AW.EXECUTE('-
     CONSIDER sales_pp; EQ LAG(sales, 1, time, LEVELREL time_levelrel)');
```

Adding Contents to a DML Program From SQL

You can define the contents of a program in a text file, which you can easily modify. Follow these steps:

1. Define a database directory object if you have not done so already.

   ```
   CREATE DIRECTORY directory AS 'path_name';
   GRANT permission ON DIRECTORY directory TO users;
   ```

2. Open an analytic workspace if you have not done so already.

   ```
   EXECUTE DBMS_AW.EXECUTE('AW CREATE aw_name ');
   ```

   or

   ```
   EXECUTE DBMS_AW.EXECUTE('AW ATTACH aw_name RW');
   ```

   It is a good practice to develop your OLAP DML programs in a separate analytic workspace from your data.

3. Create a program object and, optionally, document it by attaching a description. The following syntax defines a new program.

   ```
   EXECUTE DBMS_AW.EXECUTE('DEFINE object PROGRAM; LD object description');
   ```

4. Open a text editor and create a file with the following contents:

   ```
   CONSIDER program_name
   PROGRAM
        .
        .                  " OLAP DML commands
        .
   END
   ```

   Tip: If your operating system permits you to open multiple windows, you can use one window for your SQL session and another for editing the text file.

5. Execute the text file.

   ```
   EXECUTE DBMS_AW.EXECUTE('INFILE directory/filename');
   ```

6. Compile and execute the program.

   ```
   EXECUTE DBMS_AW.EXECUTE('CALL program_name');
   ```

Example 9–2 shows a sample session in which a program named `CREATE_MEASURE` is created, compiled, and executed within a SQL session.

Example 9–2 Creating an OLAP DML Program in SQL*Plus

```
% sqlplus
      .
      .
      .
SQL> CREATE DIRECTORY olapfiles AS '/users/oracle/olapfiles';
SQL> GRANT all ON DIRECTORY olapfiles TO ALL;
SQL> EXECUTE DBMS_AW.EXECUTE('AW ATTACH global_programs RW');
SQL> EXECUTE DBMS_AW.EXECUTE('DEFINE create_measure PROGRAM; LD Get measures');

-- Create a file named getmeas.inf in directory olapfiles with the
-- contents of the program. Start with the template and edit it for
-- the sample data.

SQL> EXECUTE DBMS_AW.EXECUTE('INFILE olapfiles/getmeas.inf');
SQL> EXECUTE DBMS_AW.EXECUTE('CALL create_measure');
```

Adding Custom Measures to a Cube

Most of the variables in your analytic workspace are created from the base-level data in your source star or snowflake schema. However, you may want to store the results of your analysis in a variable. For example, if you generate a forecast, you must identify a target variable in which to store the forecast.

In a standard form workspace, a logical measure is implemented with a variable and a formula, as described in Chapter 8. To add a custom measure in database standard form, you need to create these objects manually. Afterward, the enablers will include the custom measures in the views exactly the same was as the other measures. For applications, the original measures and the custom measures will be indistinguishable.

Defining a Standard Form Measure Variable

If you just want to define a formula for calculating a custom measure on the fly from existing measures in your analytic workspace, you can bypass this step. You can define as many custom measures you wish from the same measure variables.

However, if you want to store the results of a forecast or other analysis, or load data from other sources, then take these steps to define a measure variable in Analytic Workspace Manager.

Tip: To define a variable just like another one in a cube (including data type), right-click that variable from the Object View and choose **Create Like**. Then you can just modify the new variables properties as shown in Table 9–1. For an example of this method, refer to "Creating New Variables in GLOBAL" on page 9-18.

1. Open the Object View and expand the folder for your analytic workspace.
2. Right-click the Variable folder and choose **Create Variable**.

   The Create Variable dialog is displayed.
3. On the Basic page, specify a name, description, and data type for the variable. To conform with the other variable names in the workspace, the name should end with `_VARIABLE`, and may begin with the cube name (such as `SALES_CUBE_`). Click **Help** for more information about these choices.
4. On the Dimensions page, select the dimensions for the variable and list them in the appropriate order.

   Note: The correct order is very important for performance.

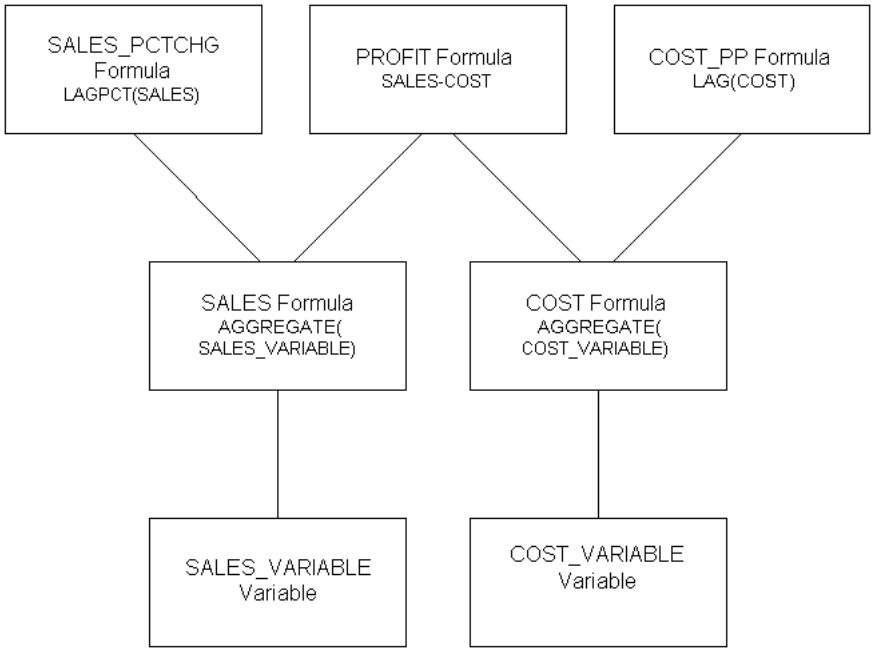
   If you will add this measure to an existing cube, then dimension it the same as the other variables in the cube. In most cases, the Time dimension is first, followed by a composite of all the other dimensions. Otherwise, click **Help** for information about ordering the dimensions.
5. On the Properties page, define the properties listed in Table 8–11 on page 8-23. If you are adding the measure to an existing cube, then you can duplicate many of the property settings of the other variables. Otherwise, search Help for information about segment size.
6. From the File menu, choose **Save** to update the analytic workspace and all objects in the current schema.
7. Populate the variable, either by performing calculations on existing measures in your analytic workspace, or from external data sources, as described in Chapter 11.

Defining a Formula

A single variable can be the source of data for numerous formulas. Applications run their queries against formulas (or relational views of the formulas), not against variables. Thus the names of the formulas are the names of the measures.

Every data variable has a corresponding formula that aggregates data at runtime by using an AGGREGATE function in its equation. You can create additional formulas for manipulating the data, and thus add information-rich data to your analytic workspace, by using the wealth of functions and operators available in the OLAP DML. Figure 9–1 shows the relationships among these workspace objects.

Figure 9–1 Relationships Among Formulas and Variables



Note: If the source variable already has a formula defined, then the easiest way to add another formula is to right-click the existing formula in the Object View and choose **Create Like**. Then you can replace the expression and modify the new formula's properties. For an example of this method, refer to "Creating Measure Formulas" on page 9-19.

Take these steps to define a formula:

1. Open the Object View and expand the folder for your analytic workspace.
2. Right-click the Formulas folder and choose **Create Formula**.

   The Create Formula dialog is displayed.
3. On the Basic page, specify a name, description, and data type for the formula. Choose the same data type for the formula as the source variable. Click **Help** for more information about these choices.
4. On the Dimensions page, select the dimensions for the formula and list them in the appropriate order.

   List the base dimensions of the source variable. Do not specify the composite, but the dimensions that compose the composite in the order in which they are listed. Reflecting the dimension order of the source variable in the formula is important for good performance.

   For example, if the source variable is dimensioned by `TIME` and `UNITS_CUBE_COMPOSITE`, then dimension the formula by `TIME`, `CUSTOMER`, `PRODUCT`, and `CHANNEL` (in that order), because `UNITS_CUBE_COMPOSITE` is dimensioned by `CUSTOMER`, `PRODUCT`, and `CHANNEL`.

   You can see the dimensions of a composite by selecting the composite from the Dimensions folder and looking at its Dimensions property sheet.
5. On the Expression page, type the equation for the formula.

   For aggregation, call the `AGGREGATE` function using this basic syntax:

   ```
   AGGREGATE(variable USING aggmap)
   ```

   Refer to the *Oracle OLAP DML Reference* for the full syntax of `AGGREGATE` and other data manipulation functions.
6. On the Properties page, define the properties listed in Table 8–12 on page 8-24.
7. Delete these properties: `SOURCE_CUBENAME`, `SOURCE_NAME`, `SOURCE_OWNER`.

   These properties identify a relational data source for a measure, and so are not relevant in this context.

Registering a New Measure

Tools such as the enablers and refresh wizards in Analytic Workspace Manager use metadata that is stored within the workspace to identify objects. The metadata is

stored in standard form catalogs, which are implemented as dimensions, variables, and valuesets. After you create a new measure, you must register it in several catalogs. Since registration involves adding data to workspace objects, you must use OLAP Worksheet and the OLAP DML to register a measure.

Registration of a new measure involves four catalogs. You can examine their property sheets in Analytic Workspace Manager, or you can issue this command in OLAP Worksheet to see their definitions:

```
DESCRIBE all_measures all_descriptions aw_names cube_measures
```

ALL_MEASURES Dimension

The ALL_MEASURES **dimension** is a list of all measures in the analytic workspace. The REPORT command shows the contents of data containers, such as dimensions and measures. It has this basic syntax:

```
REPORT object
```

To see the contents of the ALL_MEASURES dimension, issue this OLAP DML command:

```
REPORT W 60 all_measures
```

The W parameter specifies the column width of the report.

Adding a Dimension Member The MAINTAIN command enables you to add, remove, and reorder the members of a dimension. It has this basic syntax:

```
MAINTAIN dimension ADD member
```

The names of the measures have this detailed format:

```
schema.cube.formula.MEASURE
```

To add the name of a measure to the ALL_MEASURES dimension, use this command syntax:

```
MAINTAIN ALL_MEASURES ADD 'detailed_measure_name'
```

For example:

```
MAINTAIN ALL_MEASURES ADD 'GLOBAL.UNITS_CUBE.PROFIT.MEASURE'
```

> **Note:** The OLAP DML interprets upper- and lower-case letters the same for commands and workspace object names, so that `MAINTAIN ALL_MEASURES`, `maintain all_measures`, and `mAiNtAiN all_MEASures` are interpreted identically. However, text strings (including dimension members) are case-sensitive, so that `'global.units_cube.profit.measure'` and `'Global.Units_Cube.Profit.Measure'` are not the same value. Text strings are always enclosed in single quotes.

Saving Changes to an Analytic Workspace Issue another `REPORT` command to make sure that the change was made correctly, then issue these commands to save it:

```
UPDATE; COMMIT
```

The `UPDATE` command copies your changes to the LOB table where the analytic workspace is stored. The `COMMIT` command issues a SQL `COMMIT` and saves all changes to the database for the session. You must issue both these commands, in this order, for changes to an analytic workspace to be saved for future sessions.

ALL_DESCRIPTIONS Variable

The `ALL_DESCRIPTIONS` **variable** stores the short, long, and plural descriptions of each dimension member, which can be used for display. `ALL_DESCRIPTIONS` is dimensioned by `ALL_OBJECTS`, `ALL_DESCTYPES`, and `ALL_LANGUAGES`.

- `ALL_OBJECTS` is a concat dimension, which means that it is composed of two or more other dimensions in a concatenated list of dimension members. `ALL_OBJECTS` is composed of `ALL_DIMENSIONS`, `ALL_CUBES`, `ALL_MEASURES`, `ALL_HIERARCHIES`, `ALL_LEVELS`, and `ALL_ATTRIBUTES`. Maintenance and selection of dimension members is performed on these base dimensions rather than directly on the `ALL_OBJECTS` concat dimension.
- `ALL_DESCTYPES` lists `LONG`, `SHORT`, and `PLURAL` as its members. Objects dimensioned by `ALL_DESCTYPES` can provide multiple descriptors of these types.
- `ALL_LANGUAGES` lists the languages supported in the analytic workspace. It initially has the database language, such as `AMERICAN_AMERICA`. If additional languages are added to `ALL_LANGUAGES`, objects dimensioned by it can provide text in multiple languages.

Limiting the Number of Active Dimension Members In analytic workspaces, all of the data for an object is initially selected, or in **status**. To view or manipulate a subset of the data, you must use the LIMIT command to restrict the number of active values. The LIMIT command is similar to a WHERE clause in a SQL SELECT statement. However, in an analytic workspace, the selection persists for subsequent commands until you explicitly change the selection.

LIMIT operates on dimensions, and has many options for selecting dimension members. The most basic form of the LIMIT command is:

```
LIMIT dimension TO values|position
```

where *values* is one or more dimension members, and *position* is a member's numeric order (1, 2, and so forth) or a keyword (such as FIRST or LAST).

> **Note:** If your dimension members are integers, then be sure to specify them correctly in the LIMIT syntax. In a dimension with a TEXT data type, '1' (with quotes) identifies the member whose value is 1, while 1 (without quotes) identifies the first member in the dimension list.

By limiting the dimensions of a variable, you restrict the number of its cells for use by subsequent commands.

To see the contents of ALL_DESCRIPTIONS, issue OLAP DML commands such as these:

```
LIMIT all_languages TO 1
REPORT W 65 DOWN all_objects w 20 ACROSS all_desctypes: all_descriptions
```

One method of limiting a concat dimension is to limit a base dimension, then limit the concat dimension to the status of the base dimension. For example, to view only the descriptions of the first two dimensions, issue commands like these:

```
LIMIT all_dimensions TO FIRST 2
LIMIT all_objects TO all_dimensions
LIMIT all_languages TO 1
REPORT W 65 DOWN all_objects w 20 ACROSS all_desctypes: all_descriptions
```

Targeting a Specific Cell Although you can use the LIMIT command to restrict a variable to a single active cell, qualified data references (QDRs) are used more often for this purpose. QDRs operate independent of the current status of the dimensions, and are in effect only for the duration of a command. If a dimension is omitted from

a QDR, the first value in status is used, so LIMIT can be used to simplify the syntax of a QDR for a multidimensional variable.

A QDR has this syntax:

```
variable(dimension 'member', dimension 'member'...)
```

For example, the following command shows the short description of UNITS_CUBE:

```
LIMIT all_measures TO 'GLOBAL_AW.UNITS_CUBE.UNITS.MEASURE'
LIMIT all_objects TO all_measures
report all_descriptions(all_desctypes 'SHORT', all_languages 'AMERICAN_AMERICA')
```

Assigning Values to a Variable Limiting dimension members is particularly important when manually setting the values of a variable. You use the assignment operator (=) to assign the value of an expression to the current selection of cells.

Use commands such as these to add descriptions of your new measure:

```
LIMIT all_languages TO 1
LIMIT all_measures TO 'detailed_measure_name'
LIMIT all_objects TO all_measures
all_descriptions(all_desctypes, 'SHORT')= 'short description'
all_descriptions(all_desctypes, 'LONG')= 'long description'
all_descriptions(all_desctypes, 'PLURAL')= 'plural description'
```

Issue another REPORT command to make sure that the changes were made correctly, then issue these commands to save them:

```
UPDATE; COMMIT
```

AW_NAMES Variable

The AW_NAMES variable identifies the full name of objects in the analytic workspace, which correspond to the detailed names of the ALL_OBJECTS dimension. Full workspace object names have this format:

```
schema.workspace!object
```

For example, GLOBAL_AW.GLOBAL!SALES.

To see the contents of AW_NAMES, issue this OLAP DML command:

```
REPORT W 60 DOWN all_objects W 35 aw_names
```

Use commands such as these to add the workspace name of a new measure:

```
LIMIT all_measures TO 'detailed_measure_name'
LIMIT all_objects TO all_measures
aw_names = 'full workspace object name'
```

Issue another REPORT command to make sure that the changes were made correctly, then issue these commands to save them:

```
UPDATE; COMMIT
```

CUBE_MEASURES Valueset

The CUBE_MEASURES **valueset** identifies the measures in each cube. It is dimensioned by ALL_CUBES and contains values of the ALL_MEASURES dimension.

The VALUES function returns the contents of a valueset. To see the contents of CUBE_MEASURES, issue this OLAP DML command:

```
REPORT W 35 DOWN all_cubes W 55 VALUES(cube_measures)
```

Use commands such as these to add a measure to an existing cube:

```
LIMIT all_cubes TO cube
LIMIT cube_measures ADD 'detailed measure name'
```

For example:

```
LIMIT all_cubes TO 'UNITS_CUBE'
LIMIT cube_measures ADD 'GLOBAL.UNITS_CUBE.PROFIT.MEASURE'
```

Issue another REPORT command to make sure that the changes were made correctly, then issue these commands to save them:

```
UPDATE; COMMIT
```

Case Study: Adding Measures to the Global Analytic Workspace

"Identifying Required Business Facts" on page 3-6 identifies the business measures required by the Global Corporation. Only three measures were acquired from the star schema: Units, Unit Price, and Unit Cost. The remaining business measures can be calculated from those three.

Custom measures can either be solved at run-time or stored in variables. Run-time calculations do not require disk storage space and do not extend the processing time

required for data maintenance. However, they may slow performance. You need to decide which measures to calculate on the fly and which, if any, to store.

Many of the required business measures are based on sales, extended cost, and margin, as shown in Table 9–1. Because these three calculated measures are used so heavily, the example stores them in variables. The other measures can be implemented as formulas and calculated on demand.

Table 9–1 Custom Measures for the GLOBAL Analytic Workspace

Required Business Measures	Object Name in GLOBAL Analytic Workspace	Expression
Sales	SALES	UNITS * UNIT_PRICE
Extended Cost	EXTENDED_COST	UNITS * UNIT_COST
Margin	MARGIN	SALES - EXTENDED_COST
Change in sales from prior period (month, quarter, or year)	SALES_PP	LAG(sales, 1, time, LEVELREL time_levelrel)
Percent change in sales from prior period	SALES_PCTCHG_PP	LAGPCT(sales, 1, time, LEVELREL time_levelrel) * 100
Product share	SHARE_SALES_PROD	(sales/sales(product '1')) * 100
Channel share	SHARE_SALES_CHAN	(sales/sales(channel '1')) * 100
Market share	SHARE_SALES_CUST	(sales/sales(customer '1')) * 100
Extended margin change from prior period	MARGIN_PP	LAG(margin, 1, time, LEVELREL time_levelrel)
Extended margin percent change from prior period	MARGIN_PCTCHG_PP	LAGPCT(margin, 1, time, LEVELREL time_levelrel) * 100
Extended margin, percent of total product sales	MARGIN_PCT_SALES	(margin/sales(product '1')) * 100
Units sold, change from prior period	UNITS_PP	LAG(units, 1, time, LEVELREL time_levelrel)
Margin per unit	UNIT_MARGIN	margin/units

Creating Measures for SALES, EXTENDED_COST, and MARGIN

The variables for Sales, Extended Cost, and Margin will have the same dimensions as Units, and will be added to the Units cube.

Creating New Variables in GLOBAL

Follow these steps to create SALES_VARIABLE.

1. In the Object View of Analytic Workspace Manager, expand the Variables folder for the GLOBAL analytic workspace.
2. Right-click UNITS_VARIABLE, and choose **Create Like** from the menu.

   The Create Like dialog is displayed.
3. Type SALES_VARIABLE in the Destination Name box, and click **OK**.

   SALES_VARIABLE is added to the list in the Variables folder.
4. Click SALES_VARIABLE to display it in the property viewer. On the Properties page, make the following changes to the settings:

   AW$PARENT_NAME: Change to SALES.

   AW$SEGWDTH_CMD: Change the variable name to SALES_VARIABLE.

   Click **Apply** to save changes to the property pages.
5. Repeat these steps for EXTENDED_COST_VARIABLE and MARGIN_VARIABLE.
6. To save the new definitions, choose **Save** from the File menu.

Calculating and Storing Values in Variables

The following commands calculate data just at the base level so that the new variables can be aggregated separately. The ACROSS command loops over the dimension members currently in status.

A hyphen at the end of a line continues a command to the next line.

> **Note:** The data in SALES_VARIABLE, EXTENDED_COST_VARIABLE, and MARGIN_VARIABLE must be refreshed manually each time the source variables are refreshed. Commands like the following can be copied into an OLAP DML program and executed as part of the refresh process.

```
" Select base level dimension members
LIMIT time TO time_levelrel 'Month'
LIMIT channel TO channel_levelrel 'CHANNEL'
LIMIT product TO product_levelrel 'ITEM'
LIMIT customer TO customer_levelrel 'SHIP_TO'

" Populate variables using calculations
ACROSS time units_cube_composite DO -
     'extended_cost_variable = units_variable * unit_cost_variable'
ACROSS time units_cube_composite DO -
     'sales_variable = units_variable * unit_price_variable'
ACROSS time units_cube_composite DO -
     'margin_variable = sales_variable - extended_cost_variable'

" Save the new variables
UPDATE
COMMIT
```

Creating Measure Formulas

Follow these steps to create and register the SALES formula. Repeat them for EXTENDED_COST and MARGIN.

1. In the Object View of Analytic Workspace Manager, expand the Formulas folder for the GLOBAL analytic workspace.
2. Right-click UNITS, and choose **Create Like** from the menu.

   The Create Like dialog is displayed.
3. Type SALES in the Destination Name box, and click **OK**.

   SALES is added to the list in the Formulas folder.
4. Click SALES and make these changes to the property pages:

   On the Expression page, change UNITS_VARIABLE to SALES_VARIABLE in the AGGREGATE function call.

   On the Properties page, change the values of AW$LOGICAL_NAME and SOURCE_NAME to SALES.

   Click **Apply** to save changes to the property pages.
5. To save the new definitions, choose **Save** from the File menu.
6. To register the SALES measure, open OLAP Worksheet and issue the following commands:

```
" Add SALES to the ALL_MEASURES dimension
MAINTAIN ALL_MEASURES ADD 'global_aw.units_cube.sales.measure'

" Add descriptions to the ALL_DESCRIPTIONS variable
LIMIT all_measures TO 'global_aw.units_cube.sales.measure'
LIMIT all_objects TO all_measures
LIMIT all_languages TO 1
all_descriptions(all_desctypes, 'SHORT')= 'Sales'
all_descriptions(all_desctypes, 'LONG')= 'Sales as Units * Price'
all_descriptions(all_desctypes, 'PLURAL')= 'Sales'

" Add measure name to the AW_NAMES variable

aw_names = 'GLOBAL_AW.GLOBAL!SALES'

" Add measure to the CUBE_MEASURES valueset
LIMIT all_cubes TO 'GLOBAL_AW.UNITS_CUBE.CUBE'
LIMIT cube_measures ADD 'GLOBAL_AW.UNITS_CUBE.SALES.MEASURE'

" Save these changes
UPDATE
COMMIT
```

Aggregating the New Global Variables

After you have created a standard form measure, you can aggregate it the same as any other measure. Because the new measures were added to an existing cube, you can either modify an existing aggregation plan or create a new one for the new measures. Follow these steps:

1. In the OLAP Catalog View, expand the Cubes folder sufficiently to see `UNITS_CUBE` in the `GLOBAL` analytic workspace.
2. To modify an existing aggregation plan, do the following:
   a. Expand the Aggregation Plans folder under `UNITS_CUBE` and right-click the plan.
   b. Choose **Edit** from the menu.
   c. Add `SALES`, `EXTENDED_COST`, and `MARGIN` to the plan.

   or

   To create a new aggregation plan, right-click `UNITS_CUBE` and choose **Create Aggregation Plan Using Wizard**. Follow the steps of the wizard, and choose Help for additional information.

3. To deploy the aggregation plan, right-click it and choose **Deploy Aggregation Plan** from the menu.
4. Choose **Save** from the File menu.

Adding More Custom Measures to GLOBAL

The remaining measures can be calculated at runtime using any of the available methods. The following steps create a new formula object and register it as a measure, using the method described in this chapter. Alternatively, you can use the `DBMS_AW_UTILITIES` package to define permanent custom measures, as described in Chapter 7.

To define the `SALES_PP` measure, take these steps:

1. In the Object View, right-click the `SALES` formula and choose **Create Like** from the menu.
2. Type `SALES_PP` as the destination name in the Create Like dialog.
3. Click the new `SALES_PP` formula and make these changes to the property pages:

   On the Properties page, change `AW$LOGICAL_NAME` and `SOURCE_NAME` to `SALES_PP`.

   On the Expression page, replace the `AGGREGATE` function with this `LAG` function:

   ```
   LAG(sales, 1, time, LEVELREL time_levelrel)
   ```

4. Register `SALES_PP` the same as the other measures.

Repeat these steps for the other measures listed in Table 9–1.

Using an OLAP DML Program to Add Measures to GLOBAL

The previous examples showed how to define measures manually using Analytic Workspace Manager. Another option is to use an OLAP DML program. Example 9–3 shows a sample program for adding measures. It takes three arguments:

- The name of the measure
- The name of the source variable
- The name of the cube for the measure

This is the command to run the program:

```
CALL create_measure('display_name' 'source_variable')
```

For example, `CALL create_measure('Sales' 'sales_variable')`

All of the other information is provided in local variables at the beginning of the program. If you use this program as a template for creating measures in your own analytic workspace, then either change the settings of these local variables or change the variables to command-line arguments.

This is the basic sequence of the program:

- Checks that the source variable exists and creates it if it does not. The program does not populate the variable; it just creates the object definition.
- Creates a formula with an `AGGREGATE` function in the equation. You can alter the equation at any time.
- Registers the new measure in the database standard form catalogs.

Comments are used throughout the program to help you understand how it works. You will also see these symbols:

" (double quote) begins or ends a comment
' (single quote) encloses literal text
& (ampersand) substitutes the value of an expression for the expression itself
\ (backslash) identifies the next character as literal, not part of command syntax
= (equal) sets the variable on the left to the value of the expression on the right
- (hyphen) continues command onto next line
: (colon) follows the name of label used to redirect processing

For the full syntax and usage of the commands and functions in this program, refer to the *Oracle OLAP DML Reference*.

Example 9–3 DML Program for Adding Measures to UNITS_CUBE

```
DEFINE CREATE_MEASURE PROGRAM
PROGRAM
ARG _displayname          text
ARG _measvar              text
ARG _cube                 text
VARIABLE _schema          text
VARIABLE _aw              text
VARIABLE _fullname        text
VARIABLE _measure         text
VARIABLE _datatype        text
```

```
VARIABLE _dims            text
VARIABLE _segwidth        text
VARIABLE _aggmap          text
VARIABLE _createdby       text
VARIABLE _fullmeas        text

TRAP ON OOPS              "Redirect processing on error to OOPS label

" Check for measure name argument on command line
IF _displayname EQ na
  THEN SIGNAL noarg 'You must supply a measure name.'
  ELSE _measure = UPCASE(_displayname)

IF _measvar EQ na
  THEN _measvar = JOINCHARS(_measure, '_VARIABLE')

IF _cube EQ na
  THEN _cube = 'UNITS_CUBE'
  ELSE _cube = UPCASE(_cube)

" Change these local variables for your data
_schema = 'GLOBAL_AW'                " Name of the schema that owns the analytic workspace
_aw = 'GLOBAL'                       " Name of the analytic workspace
_segwidth = '85 1000000'             " Segment size appropriate for measures in this cube
_aggmap = JOINCHARS(_cube '_AGGMAP_AWCREATEDDEFAULT_1')  " Name of default aggmap for cube
_datatype = 'DECIMAL'

_createdby = 'AW$CREATE'
_fullname = UPCASE(JOINCHARS(_schema, '.', _cube, '.', _measvar))
_dims = OBJ(PROPERTY, 'SYS_DIMS', _cube)
_fullmeas = UPCASE(JOINCHARS(_schema, '.', _cube, '.', _measure, '.MEASURE'))

" Redirect processing to FORMULA label if variable already exists
IF EXISTS(_measvar)
 THEN GOTO FORMULA

" Define the variable
&JOINCHARS('DEFINE ', _measvar, ' VARIABLE ' , _datatype , ' <', _dims, '>')

" Set Database Standard Form metadata required to register a measure variable
&JOINCHARS('CONSIDER ', _measvar)
PROPERTY 'AW$CLASS' 'EXTENSION'
PROPERTY 'AW$CREATEDBY' _createdby
PROPERTY 'AW$LASTMODIFIED' JOINCHARS(today , '_' , tod)
PROPERTY 'AW$PARENT_NAME' _measure
```

```
PROPERTY 'AW$ROLE' 'VARIABLE'
PROPERTY 'AW$STATE' 'CREATED'
PROPERTY 'AW$SEGWDTH_CMD' JOINCHARS(-
     'CHGDFN ' , _schema , '.' , _aw , '!' , _measure , ' SEGWIDTH ' , _segwidth)

FORMULA:
" Check if the measure is already defined
IF EXISTS(_measure)
   THEN SIGNAL measexists JOINCHARS(_measure ' already exists.')
" Create the formula
&JOINCHARS('DEFINE ' , _measure , ' FORMULA ' , _measvar)
" Define the calculation equation
&JOINCHARS('EQ AGGREGATE(', _schema, '.', _aw, '!', _measvar, ' USING ', _aggmap, ')' )

" Set properties needed by the BI Beans enablement process
&JOINCHARS('CONSIDER ' , _measure)
PROPERTY 'AW$CLASS' 'IMPLEMENTATION'
PROPERTY 'AW$COMPSPEC' _aggmap
PROPERTY 'AW$CREATEDBY' _createdby
PROPERTY 'AW$LASTMODIFIED' JOINCHARS(TODAY , '_' , TOD)
PROPERTY 'AW$LOGICAL_NAME' _measure
PROPERTY 'AW$PARENT_NAME' _cube
PROPERTY 'AW$ROLE' 'MEASUREDEF'
PROPERTY 'AW$STATE' 'CREATED'

" Register measure in standard form catalogs
&JOINCHARS('MAINTAIN all_measures ADD ', '\'', _fullmeas, '\'')
&JOINCHARS('all_descriptions(all_objects \'<ALL_MEASURES: ' , _fullmeas ' >\' all_desctypes
\'SHORT\') = \'' , _displayname , '\'')
&JOINCHARS('aw_names(all_objects \'<ALL_MEASURES: ' , _fullmeas ' >\') = \'', _schema, '.', _aw,
'!', _measure, '\'')
&JOINCHARS('LIMIT all_cubes to \'', _schema, '.', _cube, '.CUBE\'')
&JOINCHARS('LIMIT cube_measures add  ' '\'', _fullmeas, '\'')
RETURN

OOPS:
show 'Program ended in an error.'
END
```

10

Predicting Future Performance

This chapter introduces the tools available in an analytic workspace to generate a forecast. It explains how to store the forecast in a standard form measure, and how to create a standard form cube for forecast results.

This chapter contains the following chapters:

Creating a Forecast

The OLAP DML supports simple linear regressions, several non-linear regression methods, single exponential smoothing, double exponential smoothing, and the Holt-Winters method. If you are unsure of which method to use, you can have the OLAP engine decide the best fit for your data based on past performance.

Most forecasts are calculated at the base level. You then aggregate the base-level forecast data to generate forecast aggregates. Typically, you do not generate forecast aggregates from the aggregates of actual data. The examples in this chapter assume that you wish to generate forecast aggregates in this way.

However, at times you may want to generate forecasts at the aggregate level and then allocate the data to lower levels. This method of forecasting is also supported.

Steps for Creating a Forecast

These are the steps for creating a forecast. Each one is discussed in more detail in the sections that follow.

1. Verify that the time periods for the forecast have been created in your time dimension. Add them if necessary.
2. Define the variables that will be used to store the results.
3. Write a program that generates the forecast.
4. Compile and run the program.
5. Check the results.
6. Add the results measure to a cube. Optionally, first create a new cube for forecasting results.
7. Create a new aggregation plan or modify an existing one to include the measure containing the forecast results. Deploy the aggregation plan.
8. Enable the analytic workspace for your applications.

Creating the Forecast Time Periods

The future time periods that you want to forecast must be defined as members of the time dimension in your analytic workspace. If they do not exist there already, you must:

1. Add the new members and their attributes to the Time dimension table in the source schema.
2. Use the Refresh wizard in Analytic Workspace Manager to add the new members to the dimension in the analytic workspace.

You should use whatever mechanism guarantees that these Time dimension members will be identical when you load actual data.

Defining Variables for the Results

A forecast requires a minimum of one variable for the results, and up to three variables if you want seasonal and smoothed seasonal forecasts. These variables typically have the same dimensions and data type as the variable used to generate the forecast.

Take these steps to define the variables for a forecast:

1. Define the results variable as a standard form measure.

   Refer to "Adding Custom Measures to a Cube" on page 9-8 for instructions on defining the variable and the aggregate formula, and for registering the measure.

2. For a seasonal forecast, define a second variable for the seasonal factors. Do *not* assign standard form properties to this variable. Instead, do the following:

   a. In the Object View, expand the folder for your analytic workspace.

   b. Right-click Variables and choose **Create Variable** from the menu.

   c. Define the variable with a `DECIMAL` data type.

   d. On the Dimensions page, list the dimensions in the appropriate order for variables in your cube, typically Time first, then a composite dimension.

3. For a smoothed seasonal forecast, define a third variable for the smoothing factors. Copy the seasonal factors variable by right-clicking the variable and choosing **Create Like**.

Developing a Forecast Program

A forecast uses several related commands that are always executed from within an OLAP DML program. These commands define a **forecasting context**. Use the following commands in the order they are listed here.

1. `FCOPEN` function. Opens a forecasting context and returns its handle.
2. `FCSET` command. Specifies the characteristics of a forecast.
3. `FCEXEC` command. Executes a forecast and populates Oracle OLAP variables with forecasting data.
4. `FCQUERY` function (optional). Retrieves information about the characteristics of a forecast or a trial of a forecast.
5. `FCCLOSE` command. Closes a forecasting context.

> **See Also:** For descriptions of the various forecasting methods, information about querying forecast trials, and the full syntax of these commands and functions, refer to the *Oracle OLAP DML Reference*.

Example 10–1 provides a template for these commands and others that are typically used in a forecast.

Example 10–1 Template for a Forecast

```
VARIABLE handle INTEGER    " Define a local variable
TRAP ON OOPS                 " Redirect processing on error to OOPS label

" Select base level time periods
LIMIT time_dim TO levelrel_time 'base_data'
" Keep historical and forecast periods
LIMIT time_dim KEEP LAST n

" Open a handle for the forecast
handle = FCOPEN('forecast_name')
" Specify the forecast method
FCSET handle METHOD 'method' descriptors
" Execute the forecast and identify source and target variables
FCEXEC handle TIME time_dim INTO target_var1 SEASONAL -
     target_var2 SMSEASONAL target_var3 source_var
FCCLOSE handle             " Close the forecast
RETURN

OOPS:
SHOW 'Error running program'
```

Generating a Forecast

To generate the forecast data, run the forecast program, using a command like this one.

```
CALL forecast_sales
```

Defining a New Cube

Cubes provide a method of organizing measures with similar characteristics. There is no practical limit on the number of measures that you can associate with a particular cube. However, you may prefer to create a separate cube for some calculated measures, even though they have the same characteristics as an existing cube. For example, while you can add forecast measures to an existing cube with actual measures, you might not want to risk confusing them.

The metadata for cubes includes information about the source variables for its measures, such as the names of the composite and the aggregation maps. The following discussion assumes that these objects already exist. If not, refer to Chapter 6.

Creating a Cubedef Object

A *cubedef* object is a text dimension that lists the names of a cube's dimensions, as described in "Standard Form Cubes" on page 8-25Take the following steps to create a *cubedef* object using Analytic Workspace Manager.

1. Open the Object View and expand the folder for your analytic workspace.
2. Expand the Dimension folder and right-click a *cubedef* dimension for an existing cube.
3. Choose **Create Like** from the menu.

   The Create Like dialog is displayed.
4. Type a name for the new cube.

   To conform with the other cube names in the workspace, the name should end with `_CUBE`, such as `SALES_CUBE`.
5. Select the new *cubedef* dimension from the Dimension folder, and make these changes in the property viewer:
   - **Basic page:** Type a new description.
   - **Properties page:** Delete these properties: `AW$LOADPRGS`, `LOAD_TYPE`, `SOURCE_NAME`, `SOURCE_OWNER`. Then edit the other property values so they are appropriate for the new object. For more information about these properties, refer to Table 8–13 on page 8-26.

     The new cube does not use a load program to obtain data, and it should not appear in the Refresh wizard.
6. Open OLAP Worksheet. Use commands such as the following to add the names of dimensions as values:

   ```
   MAINTAIN cube ADD 'dimension' 'dimension' ...
   ```

   For example: `MAINTAIN new_cube ADD 'PRODUCT' 'TIME'`
7. From the File menu, choose **Save** to update the analytic workspace and all objects in the current schema.

Creating a Default Aggregation Map

All cubes must have a default aggregation map, which is used initially to guarantee that all queries are answered by fully solved measures. An aggregation map (or

aggmap object in the language of the OLAP DML) contains all of the rules for aggregation.

To create a default aggregation map for a cube, take these steps:

1. In the Object View, expand the Aggregation Maps folder.
2. Right-click the default aggregation for a similar cube and choose **Create Like** from the menu.

   A default aggregation map has a name such as `UNITS_CUBE_AGGMAP_AWCREATEDDEFAULT_1`.
3. Give the new aggregation map a similar name, such as `FORECAST_CUBE_AGGMAP_AWCREATEDDEFAULT_1`.
4. Select the new aggregation map and make these changes:
   - **Properties page:** Change the value of `AW$PARENT_NAME` to the name of the new cube.
   - **Aggmap page:** Verify that there is exactly one `RELATION` command for each dimension of the new cube. If changes are needed, make them and then click Compile to check the syntax of your changes.
5. Click **Apply** to save these changes in your session.
6. From the File menu, choose **Save** to save these changes for future sessions.

Registering a New Cube

Registering a cube is very similar to registering a measure, as described in "Registering a New Measure" on page 9-12. It involves most of the same catalogs. You can examine their property sheets in Analytic Workspace Manager, or you can issue this command in OLAP Worksheet to see their definitions:

```
DESCRIBE all_cubes all_descriptions aw_names cube_measures
```

For more information about these catalogs, refer to "Standard Form Catalogs" on page 8-30.

Adding a Cube to the ALL_CUBES Dimension

The `ALL_CUBES` dimension is a list of all cubes in the analytic workspace. To see its contents, issue this OLAP DML command:

```
REPORT W 40 all_cubes
```

The names of the cubes have this detailed format:

```
schema.cube.CUBE
```

To add a new cube to ALL_CUBES, use this command syntax:

```
MAINTAIN all_cubes ADD detailed_cube_name
```

For example:

```
MAINTAIN all_cubes ADD 'GLOBAL.ANALYTICS_CUBE.CUBE'
```

Issue another REPORT command to make sure that the change was made correctly, then issue these commands to save it:

```
UPDATE; COMMIT
```

Adding a Cube to the ALL_DESCRIPTIONS Variable

The ALL_DESCRIPTIONS variable stores the short, long, and plural descriptions of each object, as described in "ALL_DESCRIPTIONS Variable" on page 9-13.

Use commands such as these to add descriptions of your new cube:

```
LIMIT all_languages TO 'AMERICAN.AMERICA'
LIMIT all_cubes TO 'detailed_mcubename'
LIMIT all_objects TO all_cubes
all_descriptions(all_desctypes, 'SHORT')= 'short description'
all_descriptions(all_desctypes, 'LONG')= 'long description'
all_descriptions(all_desctypes, 'PLURAL')= 'plural description'
```

Issue another REPORT command to make sure that the changes were made correctly, then issue these commands to save them:

```
UPDATE; COMMIT
```

Adding a Cube to the AW_NAMES Variable

The AW_NAMES variable identifies the full name of objects in the analytic workspace, as described in "AW_NAMES Variable" on page 9-15.

Use commands such as these to add the workspace name of a new cube:

```
LIMIT all_cubes TO 'detailed_cube_name'
LIMIT all_objects TO all_cubes
aw_names = 'full workspace object name'
```

Issue another REPORT command to make sure that the changes were made correctly, then issue these commands to save them:

```
UPDATE; COMMIT
```

Adding Measures to the New Cube in the CUBE_MEASURES Valueset

The CUBE_MEASURES **valueset** identifies the measures in each cube, as described in "CUBE_MEASURES Valueset" on page 9-16.

Use commands such as these to add measures to the new cube:

```
LIMIT all_cubes TO cube
LIMIT cube_measures ADD 'detailed measure name . . .'
```

For example:

```
LIMIT all_cubes TO 'ANALYTICS_CUBE'
LIMIT cube_measures ADD 'GLOBAL.ANALYTICS_CUBE.PROFIT.MEASURE' -
   'GLOBAL.ANALYTICS_CUBE.SALES_PCTCHG.MEASURE'
```

Issue another REPORT command to make sure that the changes were made correctly, then issue these commands to save them:

```
UPDATE; COMMIT
```

Troubleshooting a Hand-Crafted Cube

If you made errors in creating a cube, then errors will occur when you try to aggregate the cube, or refresh or enable your analytic workspace. Follow this check list to identify the cause of failure.

- Check the properties of the cube dimension against those listed in "Standard Form Cubes" on page 8-25. If you copied another cube, make sure that you made all of the appropriate changes to the property values.
- Verify that you populated the cube dimension.

```
REPORT forecast_cube

FORECAST_CUBE
--------------
CHANNEL
CUSTOMER
PRODUCT
TIME
```

- Verify that you added the cube to the ALL_CUBES dimension.

```
REPORT W 30 all_cubes

ALL_CUBES
------------------------------
GLOBAL_AW.PRICE_CUBE.CUBE
GLOBAL_AW.UNITS_CUBE.CUBE
GLOBAL_AW.FORECAST_CUBE.CUBE
```

- Verify that you added the cube to the AW_NAMES variable.

```
LIMIT all_objects TO all_cubes
REPORT W 42 DOWN all_objects W 35 aw_names

ALL_OBJECTS                                          AW_NAMES
---------------------------------------- ----------------------------------
<ALL_CUBES: GLOBAL_AW.PRICE_CUBE.CUBE>     GLOBAL_AW.GLOBAL!PRICE_CUBE
<ALL_CUBES: GLOBAL_AW.UNITS_CUBE.CUBE>     GLOBAL_AW.GLOBAL!UNITS_CUBE
<ALL_CUBES: GLOBAL_AW.FORECAST_CUBE.CUBE>  GLOBAL_AW.GLOBAL!FORECAST_CUBE
```

Case Study: Forecasting Global Sales

While you could add the forecast measure to the Units cube, which contains the actual data, this example will create a new cube for it named FORECAST_CUBE. FORECAST_CUBE has the same dimensions as UNITS_CUBE, so the two cubes will share a composite dimension, UNITS_CUBE_COMPOSITE. The forecast will populate a single measure in the Forecast cube.

This example assumes that you have created the SALES measure, as described in Chapter 9.

Defining a New Cube for Forecast Measures

These are the basic steps to create a new cube named FORECAST_CUBE:

1. In the Object View of Analytic Workspace Manager, copy UNITS_CUBE as FORECAST_CUBE using Create Like. This step copies the object definition, but not the contents, of UNITS_CUBE.
2. On the Properties page for FORECAST_CUBE, change the following properties, then click **Apply**:
   - AW$LOADPRGS, LOAD_TYPE, SOURCE_NAME, SOURCE_OWNER: Delete so that cube will be ignored by the Refresh wizard.

- AW$LOGICAL_NAME: Set to FORECAST_CUBE
- AGGMAPLIST: Set to FORECAST_CUBE_AGGMAP_AWCREATEDDEFAULT_1
- DISPLAY_NAME: Set to Sales Forecast Cube

3. Right-click UNITS_CUBE_AGGMAP_AWCREATEDDEFAULT_1, and choose **Create Like** to create a default aggregation map named FORECAST_CUBE_AGGMAP_AWCREATEDDEFAULT_1.
4. On the Properties page, change the value of AW$PARENT_NAME to FORECAST_CUBE.

   The new cube has the same dimensions as the Units cube. Otherwise, you would need to edit the aggregation map.
5. To save the new definitions, choose **Save** from the File menu.
6. To add the dimensions of the Forecast cube, issue this command:

```
MAINTAIN forecast_cube ADD 'CHANNEL' 'CUSTOMER' 'PRODUCT' 'TIME'
```

7. To register the FORECAST_CUBE cube, open OLAP Worksheet and issue the following commands:

```
" Add FORECAST_CUBE to the ALL_CUBES dimension
MAINTAIN ALL_CUBES ADD 'GLOBAL_AW.FORECAST_CUBE.CUBE'

" Add descriptions to the ALL_DESCRIPTIONS variable
LIMIT all_cubes TO 'GLOBAL_AW.FORECAST_CUBE.CUBE'
LIMIT all_objects TO all_cubes
LIMIT all_languages TO 1
all_descriptions(all_desctypes, 'SHORT')= 'Sales Fcast'
all_descriptions(all_desctypes, 'LONG')= 'Sales Forecast
all_descriptions(all_desctypes, 'PLURAL')= 'Sales Forecasts'

" Add cube name to the AW_NAMES variable

aw_names = 'GLOBAL_AW.GLOBAL!FORECAST_CUBE'

" Save these changes
UPDATE
COMMIT
```

Defining the Forecasting Measures for Global Sales

The results of this forecast are stored in three variables. Only one is of interest to analysts; the other two hold the seasonal and smoothing adjustment factors used to create the forecast.

The quickest way to define the standard form measure is using the CREATE_MEASURE program shown in Example 9–3, "DML Program for Adding Measures to UNITS_CUBE". Take these steps:

1. In the Object View of Analytic Workspace Manager, attach GLOBAL in read/write mode. If the program is in a separate workspace, then it must be attached also, in either read-only or read/write mode.
2. Open OLAP Worksheet and issue the following command:

   ```
   AW LIST
   ```

   The GLOBAL analytic workspace must be listed first because the new workspace objects will be created in the first one listed. If GLOBAL is not first, then issue this command:

   ```
   AW ATTACH global FIRST
   ```

3. Issue this command to create a standard form measure for the forecast results:

   ```
   CALL create_measure('sales_fcast', na, 'forecast_cube')
   ```

   The arguments specify a new measure named SALES_FCAST, a new variable whose name is constructed from the measure name, and a cube named FORECAST_CUBE.
4. Create SALES_FCAST_SEASONAL by taking these steps:
   a. In the Object View, right-click **Create Variable**.

      The Create Variable dialog is displayed.

   b. On the Basic page, define SALES_FCAST_SEASONAL with a DECIMAL data type.
   c. On the Dimensions page, list TIME first, then UNITS_CUBE_COMPOSITE.
   d. Click **Create** to save this variable definition in the current session.
5. Create SALES_FCAST_SMOOTHED by right-clicking SALES_FCAST_SEASONAL and choosing **Create Like** from the menu.
6. From the File menu, choose **Save**.

Developing a Forecasting Program for Global Sales

Example 10–2 shows a program named FORECAST_SALES, which forecasts sales in the GLOBAL analytic workspace. You can use it as the basis of forecast programs in your analytic workspace.

The forecast itself requires only four commands. The default forecast method is AUTOMATIC, which permits the OLAP engine to select the best method based on the data. Seasonality is also specified, and both seasonal and smoothed seasonal variables are targeted.

Identifying Historical and Forecast Time Periods

In the GLOBAL analytic workspace, there are 65 historical periods (Jan-98 to May-03) and 12 forecast periods (Jun-03 to May-04). Because the base time period is a month, seasonal adjustments are based on a 12-period cycle. The program uses the INTEGER argument of the LIMIT function to obtain the numeric position of the last historical time period, and sets the status of TIME relative to that position.

Arguments to the FORECAST_SALES Sample Program

The FORECAST_SALES program takes five arguments:

- The forecasting method (AUTOMATIC, LINREF, NLREL1 to NLREG5, SESMOOTH, DESMOOTH, or HOLT/WINTERS). These methods are described in the *Oracle OLAP DML Reference*.
- The long description of the last time period for which there is data.
- The number of historical periods to be used in the forecast.
- The number of periods to forecast.
- The number of periods in a seasonal cycle.

Default values are set for these arguments so that they can be omitted from the command line. These are some of the ways you can run this program:

```
CALL forecast_sales
CALL forecast_sales('holt/winters')
CALL forecast_sales(na, na, 36, 6)
```

Because arguments are passed sequentially to the program, you may need to pass an NA as a placeholder value for some arguments, as shown in the third example. Later arguments can simply be omitted.

The program arguments, along with some preset local variables, are used to select the dimension members in the status. All dimensions are limited to the base level, so that precalculated aggregates will not be used in the forecast. In addition, the TIME dimension must be limited so that only the source historical periods and the target forecast periods are in status.

Example 10–2 Forecasting Program for Global Sales

```
DEFINE FORECAST_SALES PROGRAM
PROGRAM
ARG _method                TEXT   " Forecasting method
ARG _last_time             TEXT   " Long desc of last hist time period
ARG _histperiods           INT    " Number of historical periods
ARG _fcast_periods         INT    " Number of forecast periods
ARG _periodicity           INT    " Number of periods in a cycle
VARIABLE _time_level       TEXT   " Base level of time dimension
VARIABLE _channel_level    TEXT   " Base level of channel dimension
VARIABLE _product_level    TEXT   " Base level of product dimension
VARIABLE _customer_level   TEXT   " Base level of customer dimension
VARIABLE _last_time_pos    INT    " Numeric position of _last_time in time dim
VARIABLE _handle           INT    " Forecast handle

TRAP ON OOPS              " Divert processing on error to OOPS label

" Set default values for args
if _method eq na
  then _method = 'AUTOMATIC'
if _last_time eq na
  then _last_time = 'May-03'
if _histperiods eq na
  then _histperiods = 48
if _fcast_periods eq na
  then _fcast_periods = 12
if _periodicity eq na
  then _periodicity = 12

" Identify base levels of dimensions
 _time_level='MONTH'
 _channel_level='CHANNEL'
 _product_level= 'ITEM'
 _customer_level='SHIP_TO'

" Set dimension status to base level
PUSH time channel product customer
LIMIT channel TO channel_levelrel EQ _channel_level
```

```
LIMIT product TO product_levelrel EQ _product_level
LIMIT customer TO customer_levelrel EQ _customer_level
LIMIT time TO time_levelrel EQ _time_level

" Check time parameters of forecast and refine status of time dimension
_last_time_pos = LIMIT(INTEGER time TO time_long_description EQ _last_time)
IF _histperiods + _fcast_periods GT STATLEN(time)
  THEN SIGNAL toosmall 'You specified more time periods than are defined.'
IF _last_time_pos - _histperiods lt 0
  THEN SIGNAL nohist 'You specified too many historical periods.'
IF _last_time_pos + _fcast_periods GT STATLAST(time)
  THEN SIGNAL nofuture 'You specified too many forecast periods.'
  ELSE LIMIT time KEEP -
     (_last_time_pos - _histperiods + 1) TO (_last_time_pos + _fcast_periods)

" Run the forecast
_handle = FCOPEN('sales')
FCSET _handle METHOD _method HISTPERIODS _histperiods PERIODICITY _periodicity
FCEXEC _handle TIME time INTO sales_fcast_variable -
  SEASONAL sales_fcast_seasonal_variable -
  SMSEASONAL sales_fcast_smoothed_variable sales
FCCLOSE _handle

POP time channel product customer
RETURN

OOPS:
SHOW 'Program ended in an error.'
END
```

Reviewing the Forecast Data for Global Sales

Example 10–3 shows partial results from running the FORECAST_SALES program with the default settings. The SALES measure has data only for historical time periods (May-03 and earlier), and the SALES_FCAST measure has data only for forecast time periods (Jun-03 and later). SALES_FCAST_SEASONAL and SALES_FCAST_SMOOTHED store the factors in the cells for the first seasonal cycle (12 months).

Locating data in a very sparse measure can be a challenge. Limit the time dimension to the periods of interest for the forecast, and limit all of the other dimensions to one member that you know has data. Example 10–3 also shows how to limit a dimension by level, attribute value, position, or value.

Example 10–3 Viewing Forecast Results for Global Sales

```
LIMIT time TO time_levelrel EQ 'MONTH'     "Select base level time periods
"Remove periods not used in forecast
LIMIT time REMOVE time_end_date LT '30JUN00'
"Select base level channels and products
LIMIT channel TO channel_long_description EQ 'Direct Sales'
LIMIT product TO product_levelrel EQ 'ITEM'
LIMIT product KEEP FIRST 1         "Keep just the first product
LIMIT customer TO '51'             "Select customer 51
REPORT W 5 DOWN time W 12 <time_long_description sales -
   sales_fcast sales_fcast_seasonal sales_fcast_smoothed>

CHANNEL: 2
PRODUCT: 13
CUSTOMER: 51
ALL_LANGUAGES: AMERICAN_AMERICA
      -------------------------TIME_HIERLIST--------------------------
      ----------------------------CALENDAR----------------------------
      TIME_LONG_DE                              SALES_FCAST_ SALES_FCAST_
TIME   SCRIPTION      SALES     SALES_FCAST    SEASONAL     SMOOTHED
----- ------------ ---------- ------------ ------------ ------------
48    Jun-00          2,893.68           NA         0.32         0.70
49    Jul-00          2,840.35           NA         0.78         0.70
50    Aug-00          5,739.92           NA         1.16         0.70
51    Sep-00          5,821.08           NA         0.74         0.70
52    Oct-00          5,034.92           NA         0.32         0.69
53    Nov-00          2,488.27           NA         0.74         1.37
54    Dec-00          5,100.34           NA         1.36         1.22
55    Jan-01                NA           NA         0.70         1.24
56    Feb-01          4,903.58           NA         1.35         1.28
57    Mar-01          4,893.34           NA         1.39         1.32
58    Apr-01          4,824.84           NA         1.49         1.37
59    May-01          4,791.26           NA         1.65         0.70
                            .
                            .
                            .
91    Jun-03                NA         0.98           NA           NA
92    Jul-03                NA         1.01           NA           NA
93    Aug-03                NA         1.21           NA           NA
94    Sep-03                NA         1.12           NA           NA
95    Oct-03                NA         1.07           NA           NA
96    Nov-03                NA         1.05           NA           NA
97    Dec-03                NA         1.06           NA           NA
103   Jan-04                NA         1.79           NA           NA
```

```
104   Feb-04              NA          1.84          NA          NA
105   Mar-04              NA          1.89          NA          NA
106   Apr-04              NA          1.93          NA          NA
107   May-04              NA          1.96          NA          NA
108   Jun-04              NA            NA          NA          NA
```

Aggregating and Enabling the Forecast Measure

You can create and deploy an aggregation plan for the new Forecast cube the same as any other cube:

1. In the OLAP Catalog View, expand the `GLOBAL` analytic workspace folder.
2. Right-click `FORECAST_CUBE` and choose **Create Aggregation Plan Using Wizard** from the menu. Follow the steps of the wizard.
3. After creating the aggregation plan, expand the Aggregation Plans folder.
4. Right-click the name of the aggregation plan and choose **Deploy Aggregation Plan Using Wizard**.

To make the forecast available to applications, re-enable the `GLOBAL` analytic workspace.

If you experience problems with running any of these wizards, refer to "Troubleshooting a Hand-Crafted Cube" on page 10-8.

11

Acquiring Data From Other Sources

Oracle OLAP provides data acquisition facilities so that you can create a standard form analytic workspace, or add data to an existing workspace, from sources other than a star or snowflake schema. This chapter introduces those facilities. It contains the following topics:

Overview of OLAP Data Acquisition Subsystems

Oracle Warehouse Builder can transform a wide variety of data sources into a star schema and, from the star schema, into an analytic workspace. As an alternative method, you can create an analytic workspace containing empty standard form objects, and populate these objects directly from the data sources using the facilities of the OLAP DML.

Even if you have successfully built your analytic workspace using either Analytic Workspace Manager or Oracle Warehouse Builder, you may want to add measures from other sources, such as syndicated data or government business statistics. In that case, you can use the information provided in Chapter 8, "Exploring a Standard Form Analytic Workspace", and Chapter 9, "Adding Measures to a Standard Form Analytic Workspace" to define and register the standard form workspace objects.

Then you can use one of the methods introduced in this chapter to populate the objects.

This chapter shows how to use OLAP tools to generate a standard form analytic workspace, then how to populate it manually from various sources using the OLAP DML.

The OLAP DML has facilities to load data from these sources:

- **Flat files.** The file reader commands load data from flat files, so that you can use data from spreadsheets, syndicated and government sources, or legacy database systems.
- **Relational tables.** The OLAP DML `SQL` command enables you to issue most `SQL` commands from an analytic workspace. Using the `SQL` command, you can fetch data with appropriate data types from any relational table or view into an analytic workspace. OLAP tools, such as the `DBMS_AWM` package, use the SQL command to populate analytic workspaces.
- **EIF files.** The `IMPORT` and `EXPORT` commands enable you to create standard form analytic workspaces from legacy Express databases. A conversion program is available for those containing Oracle Express Objects metadata. Refer to

How to Manually Create a Standard Form Analytic Workspace

The steps that you take to create a standard form analytic workspace from alternative sources is basically the same as from a star or snowflake schema. There are two primary differences:

- Instead of loading your data into the star or snowflake schema of a data warehouse, you simply create the empty tables. These tables provide the basis for defining the OLAP Catalog metadata required by the `DBMS_AWM` package to create an analytic workspace. The data remains in its original form until it is loaded directly into an analytic workspace.
- The `DBMS_AWM` package uses characteristics of the data in the initial load to make default choices such as dimension order and segment size. Since there is no data from which it can make appropriate choices, you must specify the correct values. Using the `DBMS_AWM` package directly provides you with the most control. However, you can still use Analytic Workspace Manager or Oracle Warehouse Builder if you wish, and modify the results where necessary before loading the data.

Take these steps to generate a standard form analytic workspace from flat files, relational tables, or an Express database with no Oracle Express Objects metadata. (If you are converting an Oracle Express Objects database, skip these instructions and go to Appendix B.)

1. Identify the dimensions, attributes, measures, and cubes for your data, and use this information to design a star schema.

   You can use pencil and paper, a database design software package, or any other method that suites you.

2. Implement your design by creating the dimension tables and fact tables.

   You can issue SQL `CREATE TABLE` statements directly in SQL*Plus, or use a graphical interface such as Oracle Enterprise Manager to create the tables. Note that you are creating the tables, but not populating them.

3. Create OLAP Catalog metadata for the star schema.

   Use any of these methods for creating either `CWM1` or `CWM2` metadata: the OLAP Management tool in Oracle Enterprise Manager; the OLAP Bridge in Oracle Warehouse Builder; or the `CWM2` PL/SQL package.

4. Create a standard form analytic workspace from the OLAP Catalog metadata.

   Use any of these methods: the Create Analytic Workspace wizard in Analytic Workspace Manager; the OLAP Bridge in Oracle Warehouse Builder; or the `DBMS_AWM` PL/SQL package. Specify a full load, even though the tables do not contain data, so that all catalogs are populated correctly in the analytic workspace.

5. Review the analytic workspace and make any changes to the object definitions. In particular, look at the dimension order for composites and data variables, and set an appropriate segment size on the target variables.

   Refer to "Case Study: Creating the GLOBALX Workspace From Alternative Sources" on page 11-19 for examples of these types of changes.

6. Load data into the dimensions, relations, and variables of the standard form analytic workspace.

   Use any of the methods described in this chapter.

7. Make any additional changes to the workspace metadata.

You now have a standard form analytic workspace, and you can use any of the tools for aggregation and deployment provided for standard form workspaces. However,

you must refresh the data using whatever OLAP DML programs you created for that purpose.

Reading Flat Files

You can use file reader OLAP DML commands to acquire data from external files in various formats: binary, packed decimal, or text. While you can use some of the file reader commands individually, it is best to place them in a program. You can thereby minimize mistakes in typing and test your commands on smaller sets of data. A program also enables you to perform operations in which several commands are used together to loop over many records in a file. Afterward, you can use the program to refresh your data.

About the File Reader Programs

Table 11–1 describes the OLAP DML file reader commands. Refer to the *Oracle OLAP DML Reference* for the complete syntax, usage notes, and examples of these commands and functions.

Table 11–1 OLAP DML File Reader Commands

Command	Description
`FILECLOSE` command	Closes an open file.
`FILEERROR` function	Returns information about the first error that occurred when you are processing a record from an input file with the `FILEREAD` and `FILEVIEW` commands.
`FILEGET` function	Returns text from a file that has been opened for reading.
`FILENEXT` function	Makes a record available for processing by the `FILEVIEW` command. It returns `YES` when it is able to read a record and `NO` when it reaches the end of the file.
`FILEOPEN` function	Opens a file, assigns it a fileunit number (an arbitrary integer), and returns that number.
`FILEPUT` command	Writes data that is specified in a text expression to a file that is opened in `WRITE` or `APPEND` mode.
`FILEQUERY` function	Returns information about one or more files.
`FILEREAD` command	Reads records from an input file, processes the data, and stores the data in workspace dimensions, composites, relations, and variables, according to descriptions of the fields in the input record.

Table 11–1 (Cont.) OLAP DML File Reader Commands

Command	Description
FILESET command	Sets the paging attributes of a specified fileunit
FILEVIEW command	Works in conjunction with the FILENEXT function to read one record at a time of an input file, process the data, and store the data in workspace dimensions and variables according to the descriptions of the fields.
RECNO function	Reports the current record number of a file opened for reading.

Writing a Program for Reading Files

While reading from a file, you can format the data from each field individually, and use DML functions to process the information before assigning it to a workspace object. Reading a file generally involves the following steps.

1. Open the data file.
2. Read data from the file one record or line at a time.
3. Process the data and assign it to one or more workspace objects.
4. Close the file.

The FILEREAD and FILEVIEW commands have the same attributes and can do the same processing on your data. However, they differ in important ways:

- The FILEREAD command loops automatically over all records in the file and processes them automatically. Use FILEREAD when all records that you wish to read in the file are the same. FILEREAD is easier to use and faster than FILEVIEW.

  Because FILEREAD is able to process most files, it is shown in the examples in this chapter.

- The FILEVIEW command processes one record at a time. FILEVIEW is the more powerful of the two file-reading commands; it can process all of files that FILEREAD can, plus process different types of records.

Example 11–1 provides a template for a developing a file-reading program in the OLAP DML. Refer to "Fetching Dimensions Members From Tables" on page 11-14 for strategies for reading dimension members.

Example 11–1 Template for Reading Flat Files

```
VARIABLE funit INTEGER          "Define local variable for file handle
TRAP ON CLEANUP                 "Divert processing on error to CLEANUP label
funit = FILEOPEN('directory/datafile' READ)  "Open the file

"Read the file with FILEREAD
FILEREAD funit
   .
   .
   .
CLEANUP:                                    "Cleanup label
IF funit NE na                              "Close the file
   THEN FILECLOSE funit
```

Mapping Fields to Workspace Objects

The FILEREAD command maps fields to workspace objects. A source file can be structured with records in any of the following ways:

- Ruled files contain data in columns, with fields defined by a starting position and a width.
- Structured PRN files contain strings of text or numbers. A text field is enclosed in quotation marks. A number field can contain periods (.) in addition to numbers, but any other character, including spaces and commas, terminates the field.
- CSV files (for Comma-Separated Values) use a special character, the delimiter, to separate the fields in a record.

The target for the data in an analytic workspace is either a dimension, a relation, or a variable. Dimensions can either be maintained by adding new members, or they can be used just to align incoming data with existing dimension members. In standard form analytic workspaces, a variable is typically an attribute or a measure.

Reading Ruled Files

The basic syntax of FILEREAD for mapping the data in ruled files is:

```
COLUMN n WIDTH n workspace_object
```

The following is an example of four records from a data file. From left to right, the columns are channels, products, customers, time periods, and units. The first

column (channels) is 10 characters wide, and the other columns are 11 characters wide.

```
2          13          51          54          2
2          13          51          56          2
2          13          51          57          2
2          13          51          58          2
```

The following FILEREAD command reads the data from the last column into the UNITS_VARIABLE variable, aligning the values in the other four columns with existing dimension members. The RULED keyword is optional, since it is the default record format.

```
FILEREAD funit RULED -
   COLUMN  1 WIDTH 10 channel -
   COLUMN 11 WIDTH 11 product -
   COLUMN 22 WIDTH 11 customer -
   COLUMN 33 WIDTH 11 time -
   COLUMN 44 WIDTH 11 units_variable
```

Reading Structured PRN Files

The basic syntax in FILEREAD for mapping structured data is:

```
FIELD n workspace_object
```

The same data file shown previously in "Reading Ruled Files" can be read with the following command:

```
FILEREAD funit STRUCTURED -
   FIELD 1 channel -
   FIELD 2 product -
   FIELD 3 customer -
   FIELD 4 time -
   FIELD 5 units_variable
```

Reading CSV Files

The basic syntax for reading a CSV file is the same as for structured PRN files:

```
FIELD n workspace_object
```

The following is an example of four records from a CSV file, in which a comma is the delimiter. The fields are the same as the previous data file shown in "Reading Ruled Files": channels, products, customers, time periods, and units.

```
2,13,51,54,2
2,13,51,56,2
2,13,51,57,2
2,13,51,58,2
```

This file can be read with the following command; the DELIMITER clause is optional in this case, because a comma is the default delimiter.

```
FILEREAD funit CSV DELIMITER ',' -
   FIELD 1 channel -
   FIELD 2 product -
   FIELD 3 customer -
   FIELD 4 time -
   FIELD 5 units_variable
```

Setting Dimension Status for Reading Measures

Whenever you read data values into a variable, you must set the status of each dimension. Typically, the incoming records contain a field for each dimension; when a record is processed in the analytic workspace, the dimensions are temporarily limited to these values so that data targeted at a variable or relation is stored in the correct cell. However, if the records omit one or more dimensions, then you must set them manually before reading the file.

For example, if your file contains data only for the Direct Sales channel for August 2003, and thus does not have fields specifying the channel or time, then your program must limit the CHANNEL and TIME dimensions before reading the file. Otherwise, the data is aligned with the first member of those dimensions (All Channels and Jan-98).

Optimizing a Data Load

Your data will load fastest if the variables in your analytic workspace are defined with fastest and slowest varying dimensions that match the order of records in the source data file. If you have control over the order of records in the source data file, then you can create the data file to match the variables in your analytic workspace. Otherwise, you may need to choose between optimizing for loads and optimizing for queries when defining the dimension order of variables in your analytic workspace.

For example, a data file might have records sorted in this order:

- Lists all records for the first channel, then all records for the second channel, and so forth.
- Lists all products for the first channel, then all products for the second channel, and so forth.
- Lists all customers for the first product, then lists all customers for the second product, and so forth.
- Lists all time periods for the first customer, then all time periods for the second customer, and so forth.

In a workspace variable definition, the order of the dimensions identifies the way the data is stored. The fastest-varying dimension is listed first, and the slowest-varying dimension is listed last.

For this sample file, the data load will proceed fastest if the target variable is defined with `TIME` as the fastest varying dimension and `CHANNEL` as the slowest varying dimension, so the dimensions are listed in this order: `TIME PRODUCT CUSTOMER CHANNEL`. With a composite dimension, the definition looks like this:

```
DEFINE UNITS_VARIABLE VARIABLE DECIMAL <TIME UNITS_CUBE_COMPOSITE <CUSTOMER
PRODUCT CHANNEL>>
```

Having the `TIME` dimension as the fastest varying dimension outside the composite also provides good run-time performance for time-based analysis, because the time periods are clustered together. This is a best-case scenario, in which the workspace variables are optimized for queries, and the data file is sorted correctly for the fastest loads.

However, if you have a separate data file for each time period, then `TIME` becomes the slowest-varying dimension for the load. In this case, there is a conflict between the dimension order that optimizes queries, and the dimension order that optimizes data loads. You need to choose which dimension order is best under these circumstances.

If you have a small batch window in which to load data, you may need to optimize for the data load by defining variables with `TIME` as the last dimension, as shown here:

```
DEFINE UNITS_VARIABLE VARIABLE DECIMAL <UNITS_CUBE_COMPOSITE <CUSTOMER PRODUCT
CHANNEL> TIME>
```

Reading and Maintaining Dimension Members

The records in a data file typically contain fields for dimension values that identify the cell in which the data values should be stored. When all of the dimension values in the file already exist in your analytic workspace, you can use the default attribute MATCH in the dimension field description. MATCH accepts only dimension values that already are in the analytic workspace.

When an incoming value does not match, the command signals an error. Your file reader program can handle the error by skipping the record and continuing processing, or by halting the processing and letting you check the validity of the data file. The test for whether the error is caused by a new dimension member or another type of error is based on the transient value of the ERRORNAME option when the error is signaled.

Example 11–2 provides a template for error handling that permits processing to continue when a dimension value in the data file does not match a dimension value in the analytic workspace.

When your data file contains all new, or a mixture of new and existing dimension values, you can add the new values and all the associated data to the analytic workspace by using the APPEND attribute in the field description, as shown here:

```
FILEREAD funit -
     COLUMN n APPEND WIDTH n dimension
```

Example 11–2 Template for Skipping Records With New Dimension Members

```
VARIABLE funit INTEGER          "Define local variable for file handle
TRAP ON oops                    "Divert processing on error to oops label
funit = FILEOPEN('directory/datafile' READ)  "Open the file

next:                                        "Resume processing label
FILEREAD funit                               "Read the file with FILEREAD
   .
   .
   .

WHILE FILENEXT(funit)                        "Or read it with FILEVIEW
   DO
   FILEVIEW funit...
   DOEND

FILECLOSE funit                              "Close the file
RETURN                                       "End normal processing
oops:                                        "Error label
```

```
IF funit NE na AND ERRORNAME NE 'ATTN'
   THEN DO
   TRAP ON oops
   GOTO next                                "Resume processing at next label
   DOEND
IF funit NE na                              "Close the file on error
   THEN FILECLOSE funit
```

Transforming Incoming Values

The FILEREAD command enables you to modify values as they are read into the analytic workspace.

Basic Transformations

To simply add characters before or after a value, use the LSET and RSET clauses. For example, if the incoming time periods are only the months (such as JAN, FEB, MAR), you can add the year before storing the values in the TIME dimension:

```
FILEREAD funit -
     COLUMN 1 WIDTH 15 RSET '-04' time
```

For other transformations, you can use the FILEVIEW command or any of the data manipulation functions of the OLAP DML. The object of these manipulations is the keyword VALUE, which represents the value of the current field. In this example, the incoming values are converted to upper case:

```
FILEREAD funit -
     COLUMN 1 WIDTH 15 time = UPCASE(VALUE)
```

Using Relations to Align Dimension Values

If you need to match existing dimension values and a simple transformation cannot create a match, then you can create a relation in the analytic workspace that correlates the two sets of values. Take these steps:

1. Create a new dimension for the incoming dimension values.

   You can define the dimension in Analytic Workspace Manager or with a command like this in OLAP Worksheet:

   ```
   DEFINE new_dimension DIMENSION TEXT
   ```

2. Read the dimension values from the file.
3. Create a relation between the two dimensions.

You can define the relation in Analytic Workspace Manager or with a command like this in OLAP Worksheet:

```
DEFINE relation RELATION dimension <new_dimension>
```

4. Read the data from the file, using the relation to align the data.

   Use syntax like this:

```
FILEREAD funit CSV -
     FIELD 1 dimension = relation(new_dimension VALUE)
```

For example, if your Product dimension uses SKUs (stock keeping units) as dimension members, and you want to add data that uses bar codes, then you create a dimension for the bar codes and a relation between the bar codes and the SKUs of the Product dimension. You can populate the relation from a file that correlates bar codes and SKUs.

> **See Also:** *Oracle OLAP DML Reference* under the entries for `FILEREAD`, `FILEVIEW`, and `DEFINE RELATION` for the complete syntax, usage notes, and examples of these commands.

Fetching Data From Relational Tables

You can embed SQL statements in OLAP DML programs using the OLAP DML SQL command.

```
SQL sql_statement
```

When formatting a SQL statement that is an argument to the OLAP DML, be sure to use single quotes (') wherever you need quotes. In the OLAP DML, a double quote (") indicates the beginning of a comment.

OLAP DML Support for SQL

You can use almost any `SQL` statement that is supported by Oracle in the OLAP DML `SQL` command. You can use `SELECT` to copy data from relational tables into analytic workspace objects. You can use the `INSERT` command to copy data from analytic workspace objects into relational tables.

The following Oracle SQL extensions are also supported:

- The `FOR UPDATE` clause in the `SELECT` statement of a cursor declaration, so that you can update or delete data associated with the cursor

- The `WHERE CURRENT OF` cursor clause in `UPDATE` and `DELETE` statements for interactive modifications to a table
- Stored procedures and triggers

`COMMIT` and `ROLLBACK` are ignored as arguments to the `SQL` command. To commit your changes, issue the OLAP DML `UPDATE` and `COMMIT` commands. You cannot roll back using the OLAP DML.

Most SQL commands are submitted directly to the SQL command processor; however, a small number are first processed in the OLAP DML, and their syntax may be slightly different from standard SQL. Refer to the *Oracle OLAP DML Reference* for further details.

Table 11–2 describes the OLAP DML commands that support embedded SQL.

Table 11–2 OLAP DML Commands for Embedded SQL

Statement	Description
`SQL` command	Passes SQL commands to the database SQL command processor
`SQLBLOCKMAX` option	Controls the maximum number of records retrieved from a table at one time
`SQLCODE` option	Holds the value returned by the database after the most recently attempted SQL operation
`SQLERRM` option	Contains an error message when `SQLCODE` has a nonzero value
`SQLMESSAGES` option	Controls whether error messages are sent to the current output file

Process: Copying Data From Relational Tables Into Analytic Workspace Objects

Using the OLAP DML, you can populate a standard form analytic workspace from relational tables by taking the following steps:

1. Define the analytic workspace objects that will hold the relational table data.

   Follow the steps listed in "How to Manually Create a Standard Form Analytic Workspace" on page 11-2. Then browse the analytic workspace to identify the objects you need to populate.

2. Write an OLAP DML program for each dimension. Compile and run the programs.

   Read the following instructions in "Fetching Data From Relational Tables".

3. Write an OLAP DML program for each cube. Compile and run the programs.

   Read the instructions in "Fetching Measures From Tables" on page 11-16.

Fetching Dimensions Members From Tables

There are several strategies for fetching dimension members. The best practice is to fetch just the dimension members first, and fetch their attributes as a separate step. For Time members, the best practice is to fetch one level at a time, making a separate pass through the source table for each level. This practice enables you to fetch the Time members in the correct order so that they do not need to be sorted afterward.

However, the simplest method, and the one shown here, populates dimension members at all levels, and all of the objects that support hierarchies, at the same time. Before using this method, be sure that `SEGWIDTH` is set correctly, as discussed in "Setting the Segment Size" on page 6-9.

Example 11–3 is a template that you can use for fetching dimensions in one pass. The program does the following:

- Reads the level columns one at a time, with their attribute columns, beginning with the top level (or most aggregate) and concluding with the base level. The syntax supports one hierarchy; refer to Example 11–18 on page 11-35 for the equivalent syntax in `FILEREAD` for handling multiple hierarchies.

  Because the parent relation is being populated at the same time, the parents must be added to the workspace dimension before their children. Otherwise, an error will occur, because a relation only accepts as legitimate values the members of a particular dimension. This is not an issue if you load all dimension members first.

- Populates a Boolean *member_inhier* variable manually. The *n* shown in the syntax is either a 1 (for yes) or a 0 (for no).

- Populates the *member_levelrel* relation manually with the appropriate level name for the column. Level names must exactly match the members of the *levellist* dimension.

- Populates the *member_parentrel* relation with the parent dimension member from the appropriate column.

- Includes commands for handling errors, which are omitted from other examples so they are easier to read.

Example 11–3 Template for Fetching Dimension Members

```
SQLMESSAGES=YES        " Display error messages on the screen
TRAP ON CLEANUP        " Go to the CLEANUP label if an error occurs
SQL DECLARE cursor CURSOR FOR SELECT -
    top_level, n, 'levelname', parent_level, attribute, attribute,...-
                .
                .
                .
    base_level, n, 'levelname', attribute, attribute,... -
    FROM table -
    WHERE where_clause

" Signal an error if the command failed
IF SQLCODE NE 0
   THEN SIGNAL declerr 'Define cursor failed'

" Open the cursor
SQL OPEN cursor
IF SQLCODE NE 0
   THEN SIGNAL openerr 'Open cursor failed'

" Fetch the data
SQL IMPORT cursor INTO -
   :APPEND dimension, :inhier, :levelrel, :parentrel, :attribute, :attribute,
...

IF SQLCODE NE 0 AND SQLCODE NE 100
   THEN SIGNAL geterr 'Fetch failed'

" Save these changes
UPDATE
COMMIT

CLEANUP:
SQL CLOSE cursor
SQL CLEANUP
```

Sorting Dimension Members

When you fetch dimension members at all levels in a single pass through the source table, they are mixed together in the target workspace dimension. For most dimensions, the order does not affect processing, although you can sort the members by level if you wish.

However, it is very important for Time dimension members to be ordered chronologically within levels so that the Time dimension supports time series analysis. Functions such as LEAD, LAG, and MOVINGAVERAGE use the relative position of Time members in their calculations. For example, LAG returns the dimension member that is a specified number of values before the current member. If the time periods are not in chronological order, the returned value is meaningless.

Your analytic workspace will perform better if you load the dimensions in the correct order instead of sorting them afterward.

Example 11–4 contains an OLAP DML program template for sorting the Time dimension. It does the following:

- Defines a valueset in which to hold the sorted values.
- Sorts the Time dimension by level first, then by end date within level.
- Stores the sorted values in the valueset.
- Reorders the Time dimension members.

Example 11–4 Template for Sorting the Time Dimension

```
IF NOT EXISTS('valueset')
  THEN DEFINE valueset VALUESET time_dim
LIMIT time_dim TO ALL
"Sort levels in descending order and time periods in ascending order
SORT time_dim D time_dim_LEVELREL A time_dim_END_DATE
LIMIT valueset TO time_dim
MAINTAIN time_dim MOVE VALUES(valueset) FIRST

"Save these changes
UPDATE
COMMIT
```

Fetching Measures From Tables

To fetch data from relational tables, you must write a program that defines a SQL cursor with the selection of data that you want to fetch, then retrieves the data into the analytic workspace objects that you previously created.

Example 11–5 is a template that you can use for fetching all of the measures for a particular cube. It does the following:

- Identifies a key column with the members of each dimension. When the data is fetched into a variable, the dimensions are limited to the appropriate cell by these values.

- Orders the source data to match the target variable.

  The `ORDER BY` clause in the `SELECT` statement is the reverse of the dimension list for the variable. If a variable is dimensioned by `<Product Geography Time>` so that Product is the fastest varying and Time is the slowest, then the `ORDER BY` clause sorts the rows by Time, Geography, and Product. This ordering speeds the data load.

- Requires that values in the key columns match existing members in the target workspace dimensions.

  A required match prevents new dimension members from being created without their level, parentage, attributes, and so forth.

Example 11–5 Template for Fetching Measures

```
SQLMESSAGES=YES        " Display error messages on the screen
TRAP ON CLEANUP         " Go to the CLEANUP label if an error occurs
" Define a cursor for selecting data
SQL DECLARE cursor CURSOR FOR SELECT -
    key1, key2, key3, keyn -
    meas1, meas2, meas3, measn -
    FROM table
    ORDER BY slowest_dim, ..., fastest_dim
" Signal an error if the command failed
IF SQLCODE NE 0
   THEN SIGNAL declerr 'Define cursor failed'

" Open the cursor
SQL OPEN cursor
IF SQLCODE NE 0
   THEN SIGNAL openerr 'Open cursor failed'

" Fetch the data
SQL IMPORT cursor INTO :MATCH dim1, :MATCH dim2, -
  :MATCH dim3, :MATCH dimn, -
  :var1, :var2 :var3
IF SQLCODE NE 0 AND SQLCODE NE 100
   THEN SIGNAL geterr 'Fetch failed'

" Save these changes
UPDATE
COMMIT

CLEANUP:
```

```
SQL CLOSE cursor  " Close the cursor
SQL CLEANUP       " Free resources
```

Populating Additional Metadata Objects

Some of the metadata objects in an analytic workspace can only be populated after loading the data. Since you are creating a standard form analytic workspace, you can use the same OLAP DML programs as the DBMS_AWM package. These programs are stored in an analytic workspace named AWCREATE, which is owned by SYS. You can access the programs by attaching the workspace with this OLAP DML command:

```
AW ATTACH sys.awcreate
```

This chapter describes two programs:

- ___POP.FMLYREL populates the *member_gid* variables and *member_familyrel* relations.
- ___ORDER.HIERARCHIES populates the *default_order* variables.

The program names have three initial underscores.

Using ___POP.FMLYREL

The ___POP.FMLYREL program populates the *member_gid* variable and *member_familyrel* variable for a dimension of a data cube. You must execute ___POP.FMLYREL for each dimension. Use this syntax to call ___POP.FMLYREL:

```
CALL ___POP.FMLYREL(aw, aw!dim, aw!dim_HIERLIST, aw!dim_LEVELLIST,
aw!dim_LEVELREL, dim, aw!dim_PARENTREL, aw!dim_INHIER)
```

Where:

aw is the name of the analytic workspace.
dim is the name of a *dimdef* dimension.

All arguments are text expressions, so you must enclose literal text in single quotes. Use all upper-case letters for the arguments.

For an example, see "Populating Additional Standard Form Metadata Objects" on page 11-39.

Using ___ORDR.HIERARCHIES

The ORDR.HIERARCHIES program populates the *default_order* attribute of a data dimension. You must run it for each dimension of a data cube. Use this syntax to run ORDR.HIERARCHIES:

```
CALL ___ordr.hierarchies('aw!dim', 'aw!dim_HIERLIST', 'aw!dim_HIER_CREATEDBY',
'dim_PARENTREL', 'dim_ORDER', 'dim_INHIER')
```

Where:

aw is the name of the analytic workspace.
dim is the name of a *dimdef* dimension.

All arguments are text expressions, so you must enclose literal text in single quotes. Use all upper-case letters for the arguments.

For an example, see "Populating Additional Standard Form Metadata Objects" on page 11-39.

Case Study: Creating the GLOBALX Workspace From Alternative Sources

This example shows how to create an analytic workspace by acquiring data from relational tables and flat files. It uses Global data simply because you are already familiar with this data set; if you want to create an analytic workspace directly from the Global star schema, refer to Chapter 6.

These are the basic steps:

1. Create the GLOBALX user and a default tablespace.
2. Create a star schema in GLOBALX.
3. Create OLAP Catalog metadata for the GLOBALX star schema that defines all of the dimensions, levels, hierarchies, attributes, and measures.
4. Define the GLOBALX_AW user and default tablespace.
5. Create the GLOBALX standard form analytic workspace.
6. Modify the GLOBALX analytic workspace, such as redefining composites and setting the segment size.
7. Populate the Price cube from relational tables using the OLAP DML SQL command.

8. Populate the Units cube from a flat file using the OLAP DML File Reader commands.
9. Aggregate the data.
10. Enable the GLOBALX analytic workspace for use by the BI Beans.

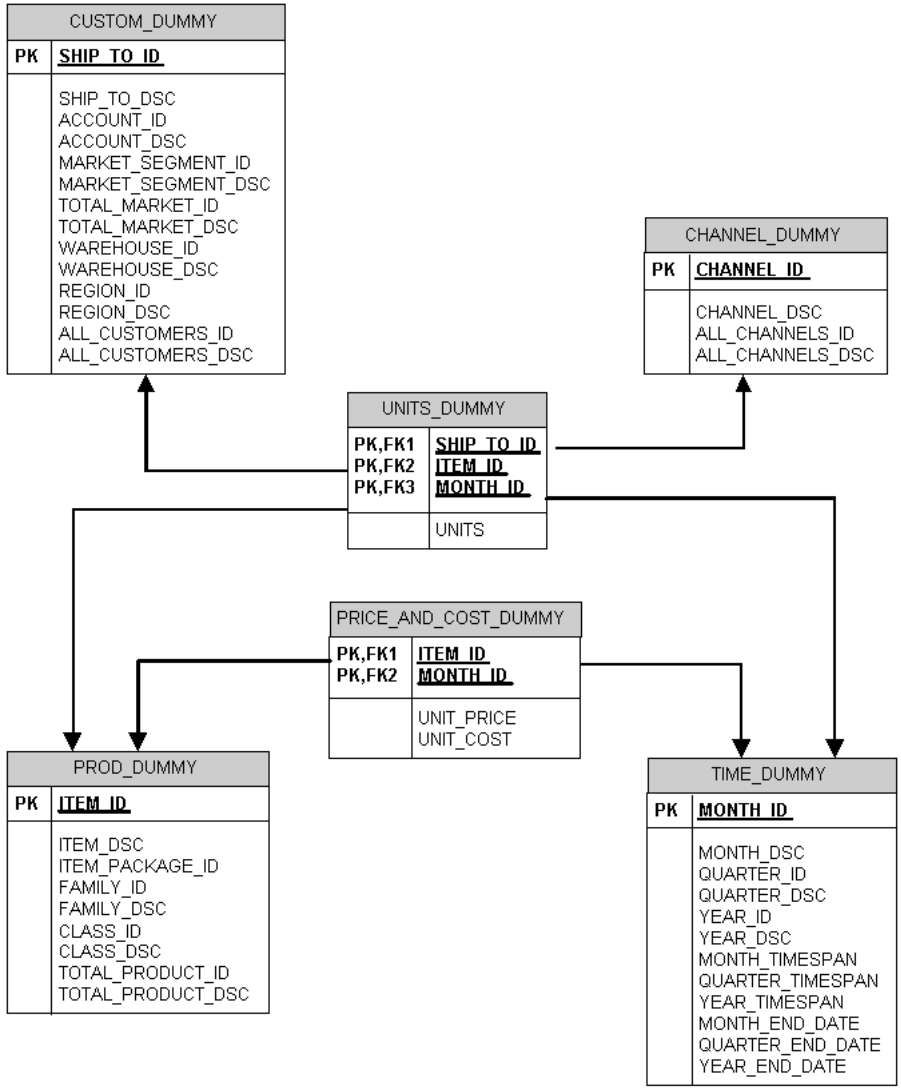
Designing and Implementing the GLOBALX Star Schema

Because Global data is already stored in the GLOBAL star schema, GLOBALX can simply mimic its design for the Price and Units cubes. The only difference is that while GLOBAL is populated with data, GLOBALX contains empty tables.

GLOBALX Schema Diagram

Example 11–6 diagrams the schema relationships.

Example 11–6 GLOBALX Schema Diagram



Procedure: Creating the GLOBALX Sample Schema

Take these steps to create the sample GLOBALX schema:

1. Create the GLOBALX user and a default tablespace. Sample scripts are shown in "SQL Scripts for Defining Users and Tablespaces" on page C-1.
2. Create the SQL scripts listed in "SQL Scripts for the GLOBALX Star Schema" on page C-3.
3. Log in to SQL*Plus or a similar SQL command processor as the GLOBALX user.
4. Execute the scripts using the SQL @ command.
5. After the scripts execute without errors, issue a SQL COMMIT statement.

Creating OLAP Catalog Metadata for the GLOBALX Schema

The metadata for the GLOBALX star schema can be generated by any available method: the OLAP Management tools in Oracle Enterprise Manager, the OLAP Bridge in Oracle Warehouse Builder, or the CWM2 PL/SQL package. This example arbitrarily uses the CWM2 packages.

Take these steps to create OLAP Catalog metadata for the GLOBALX schema:

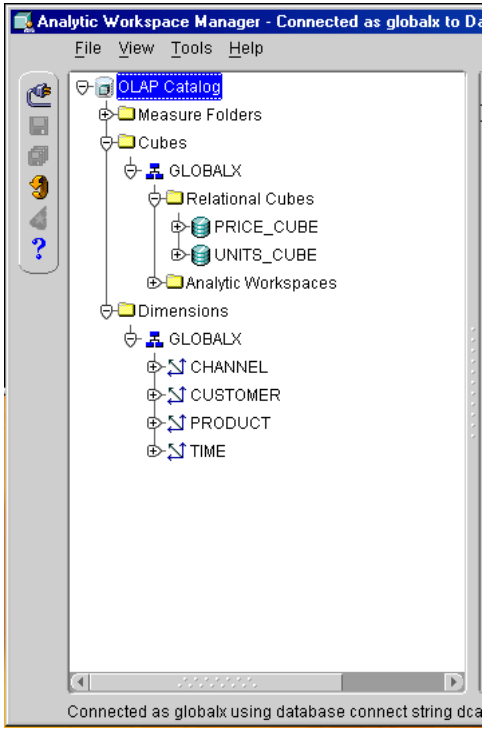
1. Create the SQL scripts listed in "SQL Scripts for OLAP Catalog Metadata" on page C-4.
2. Log in to SQL*Plus or a similar SQL command processor as the GLOBALX user.
3. Issue these SQL commands so that you can see the full report from the metadata validator, both on the screen and saved to a file:

   ```
   SET LINESIZE 135
   SET SERVEROUT ON SIZE 999999
   EXECUTE cwm2_olap_manager.set_echo_on
   SPOOL filepath
   SET ECHO ON
   ```

   The buffer for server output holds a maximum of 1,000,000 characters. If you are building a large application, you may need to control the size of the output with a combination of SET_ECHO_ON, SET_ECHO_OFF, BEGIN_LOG, and END_LOG commands.
4. Execute the CWM2 scripts using the SQL @ command.
5. After the scripts execute without errors, issue a SQL COMMIT statement.

6. Examine the metadata in Analytic Workspace Manager, as shown in Figure 11–1.
   a. Open Analytic Workspace Manager and connect as the GLOBALX user.
   b. In the OLAP Catalog view, expand the **Cubes**, **GLOBALX**, and **Relational Cubes** folders.

Figure 11–1 GLOBALX Metadata Displayed in Analytic Workspace Manager



Creating the GLOBALX Analytic Workspace

You can create the GLOBALX analytic workspace from the empty tables and OLAP Catalog metadata using any of the available methods: The Create Analytic Workspace wizard in Analytic Workspace Manager, the OLAP bridge in Oracle Warehouse Builder, or the DBMS_AWM PL/SQL procedures. This example uses the wizard to generate a script containing calls to DBMS_AWM, then modifies the script.

Take these steps to create the GLOBALX analytic workspace.

1. Create the GLOBALX_AW user with access rights to the GLOBALX tables and tablespaces, using a script like the one in "SQL Scripts for Defining Users and Tablespaces" on page C-1.
2. Open Analytic Workspace Manager and log in to the database as the GLOBALX_AW user.
3. From the Tools menu, choose **Create Analytic Workspace Using Wizard**.
4. Make these choices in the wizard:
   - Specify Analytic Workspace page: Type GLOBALX as the workspace name, and select GLOBALX_AW as the schema.
   - Select Cubes page: Select all cubes in the GLOBALX relational schema.
   - Choose Data Loading Options page: Select **Build analytic workspace and load dimensions and facts**. Clear the **Generate unique keys** box.

     There is no data in the tables to load; however, this choice populates more catalogs in the analytic workspace than the other choices. The lack of data does not cause the build to fail.
   - Choose Advanced Storage and Naming Options page: Select **Display the pages for setting the advanced storage options**. Clear the **Prefix measure names with cube names** box.

     Because no data is available, the tools cannot determine the correct order of the dimensions, nor an appropriate segment size. You must provide this information or the analytic workspace will run slower than it should.
   - Create Composite Dimension and following pages: Create a composite for the Units Cube named UNITS_CUBE_COMPOSITE with the dimensions in this order: CUSTOMER PRODUCT CHANNEL. Omit TIME from the composite.
   - Specify Segment Width and Dimension Order page: For the Units Cube, specify the dimensions like this.

     ```
     TIME                        85
     <CUSTOMER PRODUCT CHANNEL>  1000000
     ```
5. Save the new analytic workspace.
6. Open the GLOBALX analytic workspace in OLAP Worksheet, and make the following modifications:

a. Delete and redefine UNIT_COST_VARIABLE and UNIT_PRICE_VARIABLE so they are dimensioned by <TIME PRODUCT>.

These 80% dense, two-dimensional measures will perform better without a composite. Refer to "Examining Sparsity Characteristics for GLOBAL" on page 6-12 for a discussion of composite definitions. Follow the instructions in "Manually Changing Object Definitions" on page 6-13.

b. Delete PRICE_CUBE_COMPOSITE.

c. Set the segment size on either one of the variables with a command like this:

```
CHGDFN unit_price_variable SEGWIDTH 85 50
```

These settings reserve contiguous disk space for 85 time periods and 50 products. A single command changes the segment size on all measures in the same cube.

d. Set the segment size on the dimension attributes with the following commands:

```
CHGDFN time_end_date SEGWIDTH 85 1
CHGDFN time_long_description SEGWIDTH 85 1 1
CHGDFN customer_long_description SEGWIDTH 80 2 1
CHGDFN product_long_description SEGWIDTH 50 1 1
CHGDFN channel_long_description SEGWIDTH 3 1 1
```

e. Save these changes by issuing UPDATE and COMMIT commands.

Fetching the Price Cube From Relational Tables

The Price cube has two measures, UNIT_COST and UNIT_PRICE, with two dimensions, PRODUCT and TIME. In this example, the data is loaded manually from the GLOBAL star schema. However, it could be loaded from any form of relational tables using the method described here.

Take these steps to populate the Price cube in the GLOBALX analytic workspace:

1. Open Analytic Workspace Manager and connect to the database as the GLOBALX_AW user.
2. In the Object View, open the GLOBALX analytic workspace in read/write mode.
3. Create the OLAP DML programs for fetching the PRODUCT and TIME dimension members.

You can create and compile programs in the Object View or in OLAP Worksheet. You can execute programs, and view the contents of the objects they populated, only in OLAP Worksheet.

4. Open OLAP Worksheet and execute the programs using the CALL command.

   ```
   CALL program_name
   ```

5. After the programs run without error, check the contents of the target workspace objects to verify that they are populated correctly.
6. Issue UPDATE and COMMIT commands to save the loaded data.
7. Create and execute a data load program for the Price cube.
8. After that program runs without error, check the data in the target variables
9. Issue UPDATE and COMMIT commands to save the loaded data.

Loading Products From GLOBAL.PRODUCT_DIM

The GETPROD program shown in Example 11–7 fetches data into the PRODUCT dimension, the *member_parentrel* relation and the *long_description* variable. Note that parent values must be added to the PRODUCT dimension before their children, otherwise an error will occur in populating the parent relation. Thus the SELECT statement lists the level columns from the highest level of aggregation to the lowest.

The *member_inhier* Boolean variable is populated with values of 1 for true and 0 for false. The *member_levelrel* relation is also populated with text values that match the values of the *levellist* dimension.

Define the example program, then execute it with this command:

```
CALL getprod
```

Example 11–7 OLAP DML Program for Loading Products From GLOBAL.PRODUCT_DIM

```
DEFINE GETPROD PROGRAM
PROGRAM
TRAP ON CLEANUP
" Define cursor c1
SQL DECLARE c1 CURSOR FOR SELECT -
    total_product_id, 1, 'TOTAL_PRODUCT', total_product_dsc, -
    class_id, 1, 'CLASS', total_product_id, class_dsc, -
    family_id, 1, 'FAMILY', class_id, family_dsc, -
    item_id, 1, 'ITEM', family_id, item_dsc, item_package_id -
    FROM global.product_dim
```

```
" Open the cursor
SQL OPEN c1

" Fetch the data
SQL FETCH c1 LOOP INTO -
   :APPEND product, :product_inhier, :product_levelrel, :product_long_description, -
   :APPEND product, :product_inhier, :product_levelrel, :product_parentrel, -
          :product_long_description, -
   :APPEND product, :product_inhier, :product_levelrel, :product_parentrel,-
          :product_long_description, -
   :APPEND product, :product_inhier, :product_levelrel, :product_parentrel, -
          :product_long_description, :product_package

" Save these changes
UPDATE
COMMIT

CLEANUP:
SQL CLOSE c1
SQL CLEANUP
END
```

Example 11–8 shows a selection of the data to verify that the load was successful.

Example 11–8 Viewing the PRODUCT Dimension and Attributes

```
LIMIT product TO product_levelrel EQ 'ITEM'
LIMIT product KEEP FIRST 2
LIMIT product ADD ANCESTORS USING product_parentrel
REPORT W 8 DOWN product W 16 <product_long_description product_levelrel>
      W 10 <product_parentrel product_inhier>

ALL_LANGUAGES: AMERICAN_AMERICA
         -------------------PRODUCT_HIERLIST--------------------
         --------------------PRODUCT_ROLLUP---------------------
         PRODUCT_LONG_DES                 PRODUCT_PA PRODUCT_IN
PRODUCT      CRIPTION     PRODUCT_LEVELREL  RENTREL      HIER
-------- ---------------- ---------------- ---------- ----------
13       Envoy Standard   ITEM             4                 yes
14       Envoy Executive  ITEM             4                 yes
4        Portable PCs     FAMILY           2                 yes
2        Hardware         CLASS            1                 yes
1        Total Product    TOTAL_PRODUCT    NA                yes
```

Loading Time From GLOBAL.TIME_DIM

The program to fetch TIME members, shown in Example 11–9, is very similar to the previous program for fetching PRODUCT members. It differs only in the addition of time span and end date attributes.

However, TIME members must be sorted chronologically within levels in order to support time series analysis functions. Each row contains dimension members at every level, so the TIME dimension is populated with the levels completely mixed. Example 11–10 shows a program that sorts the TIME dimension. It uses the SORT command to order the current, temporary status of the TIME dimension, saves this order in a valueset, then loops over the valueset with the MAINTAIN command to reorder the values permanently.

Define the example programs, then execute them with these commands:

```
CALL gettime
CALL timesort
```

Example 11–9 OLAP DML Program for Loading Time From GLOBAL.TIME_DIM

```
DEFINE GETTIME PROGRAM
PROGRAM
TRAP ON CLEANUP
SQL DECLARE c1 CURSOR FOR SELECT -
    year_id, 1, 'YEAR', year_dsc, year_timespan, year_end_date, -
    quarter_id, 1, 'QUARTER', year_id, quarter_dsc, quarter_timespan, quarter_end_date, -
    month_id, 1, 'MONTH', quarter_id, month_dsc, month_timespan, month_end_date -
    FROM global.time_dim -
    ORDER BY month_end_date

" Open the cursor
SQL OPEN c1

" Fetch the data
SQL FETCH c1 LOOP INTO -
   :APPEND time, :time_inhier, :time_levelrel, :time_long_description, -
          :time_time_span, :time_end_date,-
   :APPEND time, :time_inhier, :time_levelrel, :time_parentrel, -
          :time_long_description, :time_time_span, :time_end_date,-
   :APPEND time, :time_inhier, :time_levelrel, :time_parentrel, -
          :time_long_description, :time_time_span, :time_end_date

" Save these changes
UPDATE
COMMIT
```

```
CLEANUP:
SQL CLOSE c1
SQL CLEANUP
END
```

Example 11–10 OLAP DML Program for Sorting TIME Dimension Members

```
DEFINE TIMESORT PROGRAM
PROGRAM
" Create a valueset to hold the sorted values
IF NOT EXISTS('timeset')
  THEN DEFINE timeset VALUESET time
LIMIT time TO ALL
" Sort by descending levels and ascending end-dates
SORT time D time_LEVELREL A time_end_date
" Save sorted values in the valueset
LIMIT timeset TO time
" Reorder the dimension members
MAINTAIN time MOVE VALUES(timeset) FIRST
END
```

The TIME dimension has too many members to list in its entirety, but selecting members by ancestry (as shown for PRODUCT) temporarily reorders the dimension. The results will show whether the objects were populated correctly, but not necessarily whether the members are sorted correctly. Example 11–11 uses LIMIT commands that do not change the original order. The report shows the correct sort order.

Example 11–11 Viewing the TIME Dimension and Attributes

```
LIMIT time TO FIRST 10
LIMIT time ADD LAST 3
REPORT W 5 DOWN time W 8 <time_long_description time_parentrel time_levelrel
time_inhier time_end_date time_time_span>
```

```
ALL_LANGUAGES: AMERICAN_AMERICA
      --------------------TIME_HIERLIST--------------------
      ----------------------CALENDAR----------------------
      TIME_LON
      G_DESCRI TIME_PAR TIME_LEV TIME_INH TIME_END TIME_TIM
TIME   PTION    ENTREL   ELREL     IER     _DATE    E_SPAN
----- -------- -------- -------- -------- -------- --------
1     1998     NA       YEAR          yes 31DEC98    365.00
2     1999     NA       YEAR          yes 31DEC99    365.00
3     2000     NA       YEAR          yes 31DEC00    366.00
4     2001     NA       YEAR          yes 31DEC01    365.00
85    2002     NA       YEAR          yes 31DEC02    365.00
102   2003     NA       YEAR          yes 31DEC03    365.00
119   2004     NA       YEAR          yes 31DEC04    366.00
5     Q1-98    1        QUARTER       yes 31MAR98     90.00
6     Q2-98    1        QUARTER       yes 30JUN98     91.00
7     Q3-98    1        QUARTER       yes 30SEP98     92.00
106   Apr-04   116      MONTH         yes 30APR04     30.00
107   May-04   116      MONTH         yes 31MAY04     31.00
108   Jun-04   116      MONTH         yes 30JUN04     30.00
```

Loading the PRICE Cube From PRICE_AND_COST_HISTORY_FACT

Example 11–12 shows the program for fetching data into UNIT_PRICE_VARIABLE and UNIT_COST_VARIABLE. Note that the data must be loaded into the variables, not into the *measuredef* formulas, which have the same names as the logical measures. These are the definitions for these variables:

```
DEFINE UNIT_PRICE_VARIABLE VARIABLE DECIMAL <TIME PRODUCT>
LD IMPLEMENTATION Variable for UNIT_PRICE Measure

DEFINE UNIT_COST_VARIABLE VARIABLE DECIMAL <TIME PRODUCT>
LD IMPLEMENTATION Variable for UNIT_COST Measure
```

The ORDER BY clause in the DECLARE CURSOR SELECT statement sorts the rows so that PRODUCT (ITEM_ID) is the slower varying dimension and TIME (MONTH_ID) is the faster varying dimension. This organization corresponds to the order in which the values are stored in the workspace variables, as shown by their definitions. This sort order enables the data to be loaded as quickly as possible.

All of the dimension members must already exist in the analytic workspace. If a value is found without a match among the dimension members, then the program fails with an error.

Define the example program, then execute it with this command:

```
CALL getpricecube
```

Example 11–12 OLAP DML Program to Load the PRICE Cube From PRICE_AND_COST_HISTORY_FACT

```
DEFINE GETPRICECUBE PROGRAM
PROGRAM
" Define a cursor for selecting data
SQL DECLARE c1 CURSOR FOR SELECT -
    item_id, month_id, unit_price, unit_cost -
    FROM global.price_and_cost_history_fact -
    ORDER BY item_id, month_id

" Open the cursor
SQL OPEN c1

" Fetch the data
SQL FETCH c1 LOOP INTO :MATCH product, :MATCH time, -
  :unit_price_variable, :unit_cost_variable

" Save these changes
UPDATE
COMMIT
SQL CLOSE c1  " Close the cursor
SQL CLEANUP
END
```

Unlike most measures, those from the Price cube are dense so that it is easy to check the data. The `LIMIT` commands in Example 11–13 select members at all levels of the `PRODUCT` and `TIME` hierarchies. There is only data at the lowest levels, so the other levels are calculated on demand. Notice that the *measuredef* formulas are shown, not their underlying variables.

To make a quick check for any values in a variable, use the `ANY` function:

```
SHOW ANY(variable NE NA)
```

For example:

```
SHOW ANY(unit_price_variable NE NA)
```

A return value of `YES` indicates that at least one cell has data; a value of `NO` indicates that all cells are empty.

Example 11–13 Validating the PRICE_CUBE Data Load

```
LIMIT time TO '44' '45'
LIMIT time ADD ANCESTORS USING time_parentrel
LIMIT product TO '13'
LIMIT product ADD ANCESTORS USING product_parentrel
REPORT unit_price unit_cost
```

```
                ----------------------------------------PRODUCT----------------------------------------
                ---------13---------- ----------4---------- ----------2---------- ----------1----------
TIME            UNIT_PRICE UNIT_COST  UNIT_PRICE UNIT_COST  UNIT_PRICE UNIT_COST  UNIT_PRICE UNIT_COST
--------------- ---------- ---------- ---------- ---------- ---------- ---------- ---------- ----------
44                3,008.91   2,862.51   9,483.71   8,944.35  19,163.02  18,071.18  19,960.72  18,714.88
45                3,142.99   2,926.79   9,590.01   9,024.35  18,969.89  17,924.55  19,735.20  18,542.48
13                9,152.01   8,655.17  28,452.92  26,883.70  57,229.23  53,982.32  59,577.63  55,873.16
3                34,314.40  32,681.09 111,333.24 105,481.00 224,713.71 211,680.12 234,516.47 219,574.10
```

Loading the Units Cube From Flat Files

The Units cube has one measure, UNITS, and four dimensions, TIME, CUSTOMER, PRODUCT, and CHANNEL. The TIME and PRODUCT dimensions have already been added to the analytic workspace in "Fetching the Price Cube From Relational Tables" on page 11-25, so unless additional dimension members are contained in the flat files, these two dimensions do not need to be maintained. However, the CUSTOMER and CHANNEL dimensions must be fully populated before loading the UNITS measure.

This example loads data from three flat files:

- CHANNEL.DAT contains all CHANNEL dimension members and their attributes. It is equivalent to the CHANNEL_DIM dimension table in the GLOBAL star schema, described in Chapter 3.
- CUSTOMER.DAT contains all CUSTOMER dimension members and their attributes. It is equivalent to the CUSTOMER_DIM dimension table in the GLOBAL star schema, described in Chapter 3.
- UNITS.DAT contains the base-level data for the UNITS measure. It is equivalent to the UNITS_HISTORY_FACT fact table in the GLOBAL star schema, described in Chapter 3.

The basic process for loading from flat files is the same as loading from relational tables, as described earlier in "Fetching the Price Cube From Relational Tables" on page 11-25. The difference is only in the OLAP DML programs.

Loading Channels From CHANNELS.DAT

CHANNELS.DAT is a comma-delimited file as shown in Example 11–14. It has fields that correspond to the columns of the CHANNELS_DIM dimension table in the Global star schema:

Channel ID
Channel Description
All Channels ID
All Channels Description

With these fields, you can populate the CHANNEL dimension, the CHANNEL_LONG_DESCRIPTION attribute, the CHANNEL_PARENTREL relation, and the CHANNEL_LEVELREL relation. In addition, you can populate CHANNEL_INHIER and CHANNEL_LEVELREL with literal text during the data load.

Example 11–14 CHANNELS.DAT Flat File

```
2,Direct Sales,1,All Channels
3,Catalog,1,All Channels
4,Internet,1,All Channels
```

Loading the dimension values is straightforward except for the All Channels dimension member (1), which appears only in the third field. It must be added to the CHANNEL dimension before it can be used as the parent of other dimension members in CHANNEL_PARENTREL. For this reason, the third field is read first as a dimension member that has no parent, and again as a parent value. Example 11–15 shows the program for loading the data.

Define the sample program, then execute it with this command:

```
CALL read_channels
```

Example 11–15 OLAP DML Program for Loading Channels from CHANNELS.DAT

```
DEFINE READ_CHANNELS PROGRAM
PROGRAM
VARIABLE funit INTEGER          "Define local variable for file handle
TRAP ON CLEANUP                 "Divert processing on error to CLEANUP label
funit = FILEOPEN('gx/channels.dat' READ)  "Open the file

FILEREAD funit CSV -
   FIELD 3 APPEND channel channel_inhier=yes channel_levelrel='ALL_CHANNELS' -
   FIELD 4 channel_long_description -
   FIELD 1 APPEND channel channel_inhier=YES channel_levelrel='CHANNEL' -
```

```
   FIELD 2 channel_long_description -
   FIELD 3 channel_parentrel

CLEANUP:
IF funit NE na
   THEN FILECLOSE funit
END
```

CHANNEL is a very small dimension with only four members, so you can review the results of the load without selecting a sample. Example 11–16 shows the results of the load.

Example 11–16 Viewing the CHANNEL Dimension and Attributes

```
REPORT W 8 DOWN channel W 12 <channel_long_description channel_parentrel
channel_levelrel channel_inhier>

ALL_LANGUAGES: AMERICAN_AMERICA
         -----------------CHANNEL_HIERLIST------------------
         ------------------CHANNEL_ROLLUP-------------------
         CHANNEL_LONG CHANNEL_PARE CHANNEL_LEVE CHANNEL_INHI
CHANNEL  _DESCRIPTION    NTREL         LREL          ER
-------- ------------ ------------ ------------ ------------
1        All Channels NA           ALL_CHANNELS          yes
2        Direct Sales 1            CHANNEL               yes
3        Catalog      1            CHANNEL               yes
4        Internet     1            CHANNEL               yes
```

Loading Customers From CUSTOMERS.DAT

CUSTOMERS.DAT is a structured file, so that text columns are enclosed in double quotes. It has fields that correspond to the columns in the CUSTOMERS_DIM dimension table in the GLOBAL star schema:

Ship_To ID
Ship_To Description
Account ID
Account Description
Market Segment ID
Market Segment Description
Total Market ID
Total Market Description
Warehouse ID
Warehouse Description

Region ID
Region Description
All Customers ID
All Customers Description

Example 11–17 shows the first six fields of a few sample records. It contains the same types of information as CHANNELS.DAT, so that all of the equivalent workspace objects are populated. The one significant difference is that the data supports two hierarchies.

Example 11–17 CUSTOMERS.DAT Flat File

```
49 "Bavarian Indust, GmbH Rome"     22 "Bavarian Industries"         5 "Manufacturing" ...
50 "Bavarian Indust, GmbH London"   22 "Bavarian Industries"         5 "Manufacturing" ...
55 "CiCi Douglas Chattanooga"       24 "CiCi Douglas"                5 "Manufacturing" ...
                          .
                          .
                          .
```

The load program for CUSTOMERS.DAT, like the one for CHANNELS.DAT, must read parent dimension members before their children. Field 13 contains the most aggregate level, All Customers ID, so it is loaded first. The program shown in Example 11–18 loads the parent members for the SHIPMENTS_ROLLUP hierarchy first, then the parent members for the MARKET_ROLLUP hierarchy. The base level, SHIP_TO, belongs to both hierarchies.

Define the example program, then execute it with this command:

```
CALL read_customers
```

Example 11–18 OLAP DML Program for Reading CUSTOMERS.DAT

```
DEFINE READ_CUSTOMERS PROGRAM
PROGRAM
VARIABLE funit INTEGER          "Define local variable for file handle
TRAP ON CLEANUP                 "Divert processing on error to CLEANUP label
funit = FILEOPEN('gx/customers.dat' READ)  "Open the file

FILEREAD funit STRUCTURED -
   FIELD 13 APPEND customer -
      customer_inhier(customer_hierlist 'SHIPMENTS_ROLLUP')=yes -
      customer_levelrel='ALL_CUSTOMERS' -
   FIELD 14 customer_long_description(customer_hierlist 'SHIPMENTS_ROLLUP') -
   FIELD 11 APPEND customer -
      customer_inhier(customer_hierlist 'SHIPMENTS_ROLLUP')=yes -
```

```
      customer_levelrel='REGION' -
   FIELD 12 customer_long_description(customer_hierlist 'SHIPMENTS_ROLLUP') -
   FIELD 13 customer_parentrel(customer_hierlist 'SHIPMENTS_ROLLUP') -
   FIELD 9 APPEND customer -
      customer_inhier(customer_hierlist 'SHIPMENTS_ROLLUP')=yes -
      customer_levelrel='WAREHOUSE' -
   FIELD 10 customer_long_description(customer_hierlist 'SHIPMENTS_ROLLUP') -
   FIELD 11 customer_parentrel(customer_hierlist 'SHIPMENTS_ROLLUP') -
   FIELD 7 APPEND customer -
      customer_inhier(customer_hierlist 'MARKET_ROLLUP')=yes -
      customer_levelrel='TOTAL_MARKET' -
   FIELD 8 customer_long_description(customer_hierlist 'MARKET_ROLLUP') -
   FIELD 9 customer_parentrel(customer_hierlist 'MARKET_ROLLUP') -
   FIELD 5 APPEND customer -
      customer_inhier(customer_hierlist 'MARKET_ROLLUP')=yes -
      customer_levelrel='MARKET_SEGMENT' -
   FIELD 6 customer_long_description(customer_hierlist 'MARKET_ROLLUP') -
   FIELD 7 customer_parentrel(customer_hierlist 'MARKET_ROLLUP') -
   FIELD 3 APPEND customer -
      customer_inhier(customer_hierlist 'MARKET_ROLLUP')=yes -
      customer_levelrel='ACCOUNT' -
   FIELD 4 customer_long_description(customer_hierlist 'MARKET_ROLLUP') -
   FIELD 5 customer_parentrel(customer_hierlist 'MARKET_ROLLUP') -
   FIELD 1 APPEND customer -
      customer_inhier(customer_hierlist 'SHIPMENTS_ROLLUP')=yes -
      customer_inhier(customer_hierlist 'MARKET_ROLLUP')=yes -
      customer_levelrel='SHIP_TO' -
   FIELD 2 customer_long_description(customer_hierlist 'SHIPMENTS_ROLLUP') -
   FIELD 2 customer_long_description(customer_hierlist 'MARKET_ROLLUP') -
   FIELD 3 customer_parentrel(customer_hierlist 'MARKET_ROLLUP') -
   FIELD 9 customer_parentrel(customer_hierlist 'SHIPMENTS_ROLLUP')

CLEANUP:
IF funit NE na
   THEN FILECLOSE funit
END
```

CUSTOMER is too large a dimension to show the complete results of the load. Example 11–19 shows how to select a few base-level dimensions and their ancestors, so that you can check that the supporting objects were populated correctly.

Example 11–19 Viewing the CUSTOMER Dimension and Attributes

```
LIMIT customer TO customer_levelrel 'SHIP_TO'  "Select base-level members
LIMIT customer KEEP FIRST 2                    "Keep the first 2 base-level members
LIMIT customer ADD ANCESTORS USING customer_parentrel  "Add all their ancestors
SORT customer A customer_levelrel A CUSTOMER           "Sort the selected members within levels

REPORT W 8 DOWN customer W 16 <customer_long_description customer_levelrel> -
   W 6 <customer_parentrel customer_inhier>

ALL_LANGUAGES: AMERICAN_AMERICA
        ---------------------------------------CUSTOMER_HIERLIST---------------------------------------
        ----------------MARKET_ROLLUP---------------- ---------------SHIPMENTS_ROLLUP----------------
                                          CUSTOM CUSTOM                                     CUSTOM CUSTOM
        CUSTOMER_LONG_DE CUSTOMER_LEVELRE ER_PAR ER_INH CUSTOMER_LONG_DE CUSTOMER_LEVELRE ER_PAR ER_INH
CUSTOMER    SCRIPTION            L         ENTREL  IER      SCRIPTION            L         ENTREL  IER
-------- ---------------- ---------------- ------ ------ ---------------- ---------------- ------ ------
22       Bavarian         ACCOUNT          5         yes NA               ACCOUNT          NA        NA
         Industries
1        All Customers    ALL_CUSTOMERS    NA         NA All Customers    ALL_CUSTOMERS    NA        yes
5        Manufacturing    MARKET_SEGMENT   7         yes NA               MARKET_SEGMENT   NA        NA
9        Europe           REGION           1          NA Europe           REGION           1         yes
49       Bavarian Indust, SHIP_TO          22        yes Bavarian Indust, SHIP_TO          16        yes
         GmbH Rome                                        GmbH Rome
50       Bavarian Indust, SHIP_TO          22        yes Bavarian Indust, SHIP_TO          20        yes
         GmbH London                                      GmbH London
7        Total Market     TOTAL_MARKET     14        yes NA               TOTAL_MARKET     NA        NA
14       Germany          WAREHOUSE        9          NA Germany          WAREHOUSE        9         yes
16       Italy            WAREHOUSE        9          NA Italy            WAREHOUSE        9         yes
20       United Kingdom   WAREHOUSE        9          NA United Kingdom   WAREHOUSE        9         yes
```

Reading the UNITS_CUBE.DAT File

`UNITS_CUBE.DAT` contains just the Units measure with columns for each dimension key. Example 11–20 shows several sample rows.

Example 11–20 UNITS_CUBE.DAT Flat File

```
CHANNEL_ID    ITEM_ID SHIP_TO_ID   MONTH_ID      UNITS
---------- ---------- ---------- ---------- ----------
         2         13         51         54          2
         2         13         51         56          2
         2         13         51         57          2
         2         13         51         58          2
         2         13         51         59          2
         2         13         51         61          1
```

The data is written to the UNITS_VARIABLE variable, not to the UNITS formula. This is the definition of UNITS_VARIABLE:

```
DEFINE UNITS_VARIABLE VARIABLE DECIMAL <TIME UNITS_CUBE_COMPOSITE <CUSTOMER
PRODUCT CHANNEL>>
```

Notice that it is dimensioned by UNITS_CUBE_COMPOSITE, but the incoming data is aligned with the base dimensions, as shown in Example 11–21. All four base dimensions are already populated.

Define the example program, then execute it with this command:

```
CALL read_units
```

Example 11–21 OLAP DML Program For Reading UNITS_CUBE.DAT

```
DEFINE READ_UNITS PROGRAM
PROGRAM
VARIABLE funit INTEGER          "Define local variable for file handle
TRAP ON cleanup                 "Divert processing on error to cleanup label
funit = FILEOPEN('gx/units_cube.dat' READ)  "Open the file

FILEREAD funit STRUCTURED -
    FIELD 1 channel -
    FIELD 2 product -
    FIELD 3 customer -
    FIELD 4 time -
    FIELD 5 units_variable

cleanup:
IF funit NE na
   THEN FILECLOSE funit
END
```

Measures typically contain vast amounts of data but are quite sparse, so you must target specific cells to verify that the data was loaded correctly. You can do this by selecting a row or two from the source file and limiting the workspace dimensions to those values, as shown in Example 11–22.

Example 11–22 Validating the UNITS_CUBE Data Load

```
limit time to '50' to '60'
limit channel to '2'
limit product to '13' '14'
limit customer to '51'
report down time across product: units_variable
```

```
CHANNEL: 2
CUSTOMER: 51
                ---UNITS_VARIABLE----
                -------PRODUCT-------
TIME               13         14
--------------- ---------- ----------
50                    2.00       2.00
51                    2.00         NA
52                    2.00       2.00
53                    1.00       2.00
54                    2.00       2.00
55                      NA       2.00
56                    2.00       2.00
57                    2.00         NA
58                    2.00       1.00
59                    2.00       2.00
60                      NA       1.00
```

Populating Additional Standard Form Metadata Objects

If you enable the GLOBALX analytic workspace for the BI Beans now, the dimension views will have many empty columns. For example, the view of the CHANNEL dimension has these empty columns:

```
CHANNEL_GID
CHANNEL_PARENTGID
ALL_CHANN_ALL_CHANNELS
CHANNEL_CHANNEL
AW_MEMBER_ORDER
```

The ___POP_FMLYREL and ___ORDR.HIERARCHIES programs populate the workspace objects that are displayed by these columns. Example 11–23 shows the commands used for the CHANNEL dimension. Repeat these commands for the PRODUCT, CUSTOMER, and TIME dimensions.

You do not need to re-enable the GLOBALX workspace after populating these objects. The data is available through the views as soon as you commit the changes to the database.

Example 11–23 OLAP DML Commands to Populate CHANNEL Metadata Objects

```
" Populate CHANNEL_GID and CHANNEL_FAMILYREL
CALL ___POP.FMLYREL('GLOBALX', 'GLOBALX!CHANNEL', 'GLOBALX!CHANNEL_HIERLIST',
```

```
'GLOBALX!CHANNEL_LEVELLIST', 'GLOBALX!CHANNEL_LEVELREL', 'CHANNEL',
'GLOBALX!CHANNEL_PARENTREL', 'GLOBALX!CHANNEL_INHIER')

" Populate CHANNEL_ORDER
call ___ordr.hierarchies('GLOBALX!CHANNEL', 'GLOBALX!CHANNEL_HIERLIST',
'GLOBALX!CHANNEL_HIER_CREATEDBY', 'CHANNEL_PARENTREL', 'CHANNEL_ORDER',
'CHANNEL_INHIER')
```

Using Tools with the GLOBALX Analytic Workspace

You can now use the Create and Deploy Aggregation Plan wizards and the enablers.

For refreshing the data, you must revise your data loading programs to access new data sources or to restrict the load to new time periods.

Part IV

Database Administration for OLAP

Part IV provides information for database administrators on administrative tasks associated with Oracle OLAP. It contains the following chapter:

- Chapter 12, "Administering Oracle OLAP"
- Chapter 13, "Materialized Views for the OLAP API"

12

Administering Oracle OLAP

This chapter describes the various administrative tasks that are associated with Oracle OLAP. It contains the following topics:

Administration Overview

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, a database administrator or applications developer needs to address management tasks in the specific context of Oracle OLAP, in addition to creating and maintaining analytic workspaces. Following is a list of these tasks.

- Tablespaces. Create permanent and temporary tablespaces to prevent I/O bottlenecks, as described in "Creating Tablespaces for Analytic Workspaces" on page 12-2.
- Database configuration. Set initialization parameters to optimize performance, as described in "Initialization Parameters for Oracle OLAP" on page 12-7 and "Initialization Parameters for the BI Beans" on page 12-9.

- Security. Users of OLAP applications must have database identities that have been granted the appropriate access rights. For users to have access to files, you must define database directory objects and grant users access to them. Refer to "Setting Up User Names" on page 12-5.
- Performance. Database monitoring tools can identify recommended changes to the database configuration based on past usage, as described in "Monitoring Performance" on page 12-15.

> **See Also:** *Oracle Database Administrator's Guide* for detailed information about managing the Oracle Database.

Creating Tablespaces for Analytic Workspaces

Before you create an analytic workspace, you should create undo, permanent, and temporary tablespaces dedicated to their use. Analytic workspaces are created in the user's default tablespace, unless the user specifies otherwise. The default tablespace for all users is set initially to `SYS`. Creating analytic workspaces in the `SYS` tablespace can degrade overall performance. Similarly, analytic workspaces should *not* share tablespaces with relational tables, especially not the source star or snowflake schema.

Oracle OLAP makes heavy use of temporary tablespaces, so it is particularly important that they be set up correctly to prevent I/O bottlenecks.

The tablespaces that you set up for use by Oracle OLAP are used by SQL for tasks such as creating and maintaining OLAP Catalog metadata and views of workspace data, in addition to their use by analytic workspaces.

If possible, you should stripe the datafiles and temporary files across as many controllers and drives as are available.

Creating an UNDO Tablespace

The following SQL commands create an undo tablespace.

```
CREATE UNDO TABLESPACE tablespace DATAFILE 'pathname'
     SIZE size REUSE AUTOEXTEND ON NEXT size
     MAXSIZE UNLIMITED EXTENT MANAGEMENT LOCAL;
```

Where:

tablespace is the name of the tablespace
pathname is the fully qualified file name
size is an appropriate number of bytes

For example:

```
CREATE UNDO TABLESPACE olapundo DATAFILE '$ORACLE_HOME/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 these settings, then restart the database as described in "Initialization Parameters for Oracle OLAP" on page 12-7.

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

Creating a Permanent Tablespace for Analytic Workspaces

When a user creates an analytic workspace, it is created in the user's default tablespace, which is initially set to the SYS tablespace. The following SQL statements create a tablespace appropriate for storing analytic workspaces.

```
CREATE TABLESPACE tablespace DATAFILE 'pathname'
    SIZE size REUSE AUTOEXTEND ON NEXT size MAXSIZE UNLIMITED
    EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO;

ALTER USER username DEFAULT TABLESPACE tablespace
```

Where:

tablespace is the name of the tablespace
pathname is the fully qualified file name
size is an appropriate number of bytes
username is the name of a database user

For example:

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

If your computer has multiple disks, then you can stripe the tablespace across them. The next example shows SQL statements that distribute the GLO tablespace across three physical disks:

```
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;
```

Creating a Temporary Tablespace for Analytic Workspaces

Oracle OLAP uses temporary tablespace to store all changes to the data in an analytic workspace, whether the changes are the result of a data load, what-if analysis, forecasting, aggregation, or some other analysis. An OLAP DML `UPDATE` command moves the changes into the permanent tablespace and clears the temporary tablespace.

Oracle OLAP also uses temporary tablespace to maintain different generations of an analytic workspace. This enables it to present a consistent view of the analytic workspace when one or more users are reading it while the contents are being updated. This usage creates numerous extensions within the tablespace, so be sure to specify a small `EXTENT MANAGEMENT` size.

The following commands create a temporary tablespace suitable for use by Oracle OLAP.

```
CREATE TEMPORARY TABLESPACE tablespace TEMPFILE 'pathname'
   SIZE size REUSE AUTOEXTEND ON NEXT size MAXSIZE UNLIMITED
   EXTENT MANAGEMENT LOCAL UNIFORM SIZE size;
```

Where:

pathname is a fully qualified file name
size is an appropriate number of bytes
tablespace is the name of the tablespace
username is a database user

For example:

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

You can stripe temporary tablespaces across several disks the same as permanent tablespaces. The next example shows the GLOTMP temporary tablespace striped across three physical disks.

```
CREATE TEMPORARY TABLESPACE glotmp TEMPFILE
   'disk1/oradata/glotmp1.tmp' SIZE 50M REUSE AUTOEXTEND ON NEXT 5M MAXSIZE 1024M
   EXTENT MANAGEMENT LOCAL UNIFORM SIZE 256K;
```

```
ALTER TABLESPACE glotmp ADD TEMPFILE
   'disk2/oradata/glotmp2.tmp' SIZE 50M REUSE AUTOEXTEND ON NEXT 5M MAXSIZE 1024M,
   'disk3/oradata/glotmp3.tmp' SIZE 50M REUSE AUTOEXTEND ON NEXT 5M MAXSIZE UNLIMITED;
```

Querying the Size of an Analytic Workspace

To find out the size of the tablespace extensions for a particular analytic workspace, use the following SQL statements:

```
COLUMN DBMS_LOB.GETLENGTH(AWLOB) HEADING "Bytes";
SELECT EXTNUM, DBMS_LOB.GETLENGTH(AWLOB) FROM AW$awname;
```

Where:

awname is the name of the analytic workspace.

Setting Up User Names

To connect to the database, a user must present a user name and password that can be authenticated by database security. All users must have the `CONNECT` role. The additional privileges associated with that user name control the user's access to data. As a database administrator, you must set up user names with appropriate credentials for all users of Oracle OLAP applications.

You can define user names and grant them these rights from the Security folder of Oracle Enterprise Manager or by using SQL commands.

Two roles are defined on installation of the database explicitly to support Oracle OLAP:

- `OLAP_USER` role provides users with the privileges to create and manage OLAP Catalog metadata and standard form analytic workspaces in their own schemas. Any OLAP user who will be performing these tasks should have the `OLAP_USER` role or equivalent privileges.
- `OLAP_DBA` role provides a DBA or system administrator with privileges to create and manage OLAP Catalog metadata and standard form analytic workspaces in any schema. The `OLAP_DBA` role is granted with the DBA role. Care should be taken in extending this privilege to additional users.

  See Also: *Oracle Database SQL Reference* for more information about granting privileges.

SQL Access For DBAs and Application Developers

To create OLAP Catalog metadata, users must be granted the OLAP_USER role. To create analytic workspaces, users also need SELECT privileges on the source schema tables, and an unlimited quota on the tablespace in which the workspace is created. Example 12–1 shows the SQL statements for creating the GLOBAL_AW user.

Example 12–1 SQL Statements for Creating the GLOBAL_AW User

```
CREATE USER 'GLOBAL_AW' IDENTIFIED BY 'global_aw'
   DEFAULT TABLESPACE glo
   TEMPORARY TABLESPACE glotmp
   QUOTA UNLIMITED ON glo
   QUOTA UNLIMITED ON glotmp
   ACCOUNT UNLOCK;

GRANT SELECT ON global.channel_dim TO global_aw;
GRANT SELECT ON global.customer_dim TO global_aw;
GRANT SELECT ON global.product_dim TO global_aw;
GRANT SELECT ON global.time_dim TO global_aw;
GRANT SELECT ON global.price_and_cost_history_fact TO global_aw;
GRANT SELECT ON global.price_and_cost_update_fact TO global_aw;
GRANT SELECT ON global.units_history_fact TO global_aw;
GRANT SELECT ON global.units_update_fact TO global_aw;
```

SQL Access for Analysts

To access an existing analytic workspace, users must have these access privileges on the table in which the workspace is stored:

- To read from the analytic workspace, SELECT privileges.
- To write to the analytic workspace, SELECT, INSERT, and UPDATE privileges.

Note that the name of the table is the same as the name of the analytic workspace, with the addition of an AW$ prefix. For example, the XADEMO analytic workspace is stored in the AW$XADEMO relational table.

For users to access views of workspace data, they must be granted EXECUTE privileges explicitly on those views.

Example 12–2 shows the SQL statements that gives all users read-only privileges to the GLOBAL analytic workspace, and user SCOTT read/write privileges.

Example 12–2 Granting Access Rights to the GLOBAL Analytic Workspace

```
GRANT SELECT ON global_aw.aw$ global TO PUBLIC;
GRANT INSERT ON global_aw.aw$ global TO scott;
GRANT UPDATE ON global_aw.aw$ global TO scott;
```

Access to Database Objects Using the BI Beans

To connect to the database using the BI Beans, users must have the following access rights to the database:

- `CONNECT` role
- `QUERY REWRITE` system privilege
- `SELECT` privileges on the database objects containing the data to be analyzed, whether the data is stored in an analytic workspace or in relational tables. Refer to the previous topic, "SQL Access for Analysts", for information about granting access to analytic workspaces.

Initialization Parameters for Oracle OLAP

Table 12–1 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.

Table 12–1 Initial Settings for Database Parameter Files

Parameter	Setting
`DB_CACHE_SIZE`	Half of physical memory
`OLAP_PAGE_POOL_SIZE`	For queries, 2-8MB; enlarge temporarily for data loads
`PARALLEL_MAX_SERVERS`	The number of processors minus one This parameter limits the number of processes that are used for a parallel update and for SQL `SELECT` operations when reading from relational tables. The number of parallel processes is also dependent on the number of analytic workspace extension files that are being updated.
`PGA_AGGREGATE_TARGET`	200-400 MB
`SESSIONS`	2.5 * maximum number of simultaneous OLAP users

Table 12–1 (Cont.) Initial Settings for Database Parameter Files

Parameter	Setting
UTL_FILE_DIR	Directory path where the Oracle Database can write to a file.
UNDO_MANAGEMENT	AUTO
UNDO_TABLESPACE	Name of the undo tablespace, which must be defined first as shown in "Creating an UNDO Tablespace" on page 12-2

See Also: *Oracle Database Performance Tuning Guide* for information about these parameters.

Procedure: Setting System Parameters for OLAP

Take the following steps to set system parameters:

1. Open the `init`*`sid`*`.ora` initialization file in a text editor.

   The initialization file is located in `$ORACLE_HOME/admin/`*`sid`*`/pfile`, where *`sid`* is the system identifier as defined in `$ORACLE_HOME/network/admin/tnsnames.ora`.

2. Add or change the settings in the file.

   For example, you might enter a command like this so that Oracle can write files to the `olapscripts` directory:

   ```
   UTL_FILE_DIR=/users/oracle/olapscripts
   ```

3. Stop and restart the database, using commands such as the following. Be sure to identify the initialization file in the `STARTUP` command.

   ```
   SQLPLUS '/ AS SYSDBA'
   SHUTDOWN IMMEDIATE
   STARTUP pfile=$ORACLE_HOME/admin/rel10g/pfile/initrel10g.ora
   ```

About the OLAP_PAGE_POOL_SIZE Setting

`OLAP_PAGE_POOL_SIZE` is an initialization parameter that is specific to Oracle OLAP. This parameter specifies in bytes the maximum size of the paging cache to be allocated to each OLAP session. The minimum value of `OLAP_PAGE_POOL_SIZE` is 2 MB. The default value is 32 MB.

These are the basic guidelines for setting `OLAP_PAGE_POOL_SIZE`:

- In the database initialization file, set OLAP_PAGE_POOL_SIZE to a value based on the maximum number of simultaneous OLAP users. The setting should be in the order to 2-8MB; 4MB is typical. Larger is better, but remember that each user is allocated that amount.
- For data loads, use a SQL ALTER SESSION statement to enlarge the OLAP page pool as much as possible just for the duration of the load, on the basis that the page pool is not shared with other users at this time. The setting should be in the order of 100-400MB, but smaller than DB_CACHE_SIZE.

The OLAP page pool is allocated at the start of an OLAP session and released when the user closes the session. An OLAP session can be initiated by the OLAP_TABLE function, the DBMS_AWM PL/SQL package, or using the command line in OLAP Worksheet.

The OLAP page pool is allocated from the User Global Area (UGA). When the database is running in dedicated mode, the UGA is part of the Process Global Area (PGA). When the database is running as a shared server process, the UGA is part of the Shared Global Area (SGA).

When the OLAP page pool is full, it uses the DB cache as a swap space. This in-memory swapping is a relatively fast operation. When the DB cache is full, it swaps to disk, which is a relatively slow operation. If the DB cache must swap to disk frequently, then performance will suffer significantly.

About the PGA_AGGREGATE_TARGET Setting

PGA_AGGREGATE_TARGET is used by SQL statements, particularly when performing SELECT statements with GROUP BY and ORDER BY clauses. It is not used by the OLAP engine. However, PGA_AGGREGATE_TARGET can affect the performance of the BI Beans when selecting data from relational tables. If your Oracle Database supports this type of application, set PGA_AGGREGATE_TARGET initially to 200-400MB, and use the database performance monitoring tools to recommend adjustments.

Initialization Parameters for the BI Beans

The BI Beans will perform best if the configuration parameters for the database are optimized for this type of use. During installation of the Oracle Database, an OLAP configuration table is created and populated with ALTER SESSION commands that have been tested to optimize the performance of the BI Beans. Each time the BI Beans opens a session, it executes these ALTER SESSION commands.

If a database instance is being used only to support Java applications that use the BI Beans, then you can modify your server parameter file or `init.ora` file to include these settings. Alternatively, you might want to include some of the settings in the server parameter file and leave others in the table, depending upon how your database instance is going to be used. These are your choices:

- Keep all of the parameters in the configuration table, so that they are set as part of the initialization of a BI Beans session. This method fully isolates these configuration settings solely for the BI Beans. (Default)
- Add some of the configuration parameters to the server parameter file or `init.ora` file, and delete those rows from the configuration table. This is useful if your database is being used by other applications that require the same settings.
- Add all of the configuration parameters to the server parameter file or `init.ora` file, and delete all rows from the configuration table. This is the most convenient if your database instance is being used only by the BI Beans.

Regardless of where these parameters are set, you should check the Oracle Technology Network for updated recommendations.

> **See Also:** *Oracle Database SQL Reference* for descriptions of initialization parameters that can be set by the `ALTER SESSION` command

Permitting Access to External Files

The OLAP DML contains three types of commands that read from and write to external files:

- File read commands that copy data from flat files to workspace objects.
- Import and export commands that copy workspace objects and their contents to files for transfer to another database instance.
- File input and output commands that read and execute DML commands from a file and redirect command output to a file.

These commands control access to files by using `BFILE` security. This database security mechanism creates a logical database directory to represent a physical disk directory. Permissions are assigned to the database directory, which control access to files within the associated physical directory.

You use PL/SQL statements to create a database directory and grant permissions. The relevant syntax of these SQL statements is provided in this chapter.

See Also: *Oracle Database SQL Reference* under the entries for `CREATE DIRECTORY` and `GRANT` for the full syntax and usage notes.

Creating a Database Directory

To create a database directory, you must have `CREATE ANY DIRECTORY` system privileges.

Use a `CREATE DIRECTORY` statement to create a new directory, or a `REPLACE DIRECTORY` statement to redefine an existing directory, using the following PL/SQL syntax:

```
{CREATE | REPLACE | CREATE OR REPLACE} DIRECTORY directory AS 'pathname';
```

Where:

directory is the name of the logical database directory
pathname is the physical directory path

Granting Access Rights to a Database Directory

After you create a directory, grant users and groups access rights to the files contained in that directory, using the following PL/SQL syntax:

```
GRANT permission ON DIRECTORY directory TO {user | role | PUBLIC};
```

Where:

permission is one of the following:

`READ` for read-only access
`WRITE` for write-only access
`ALL` for read and write access

directory is the name of the database directory

user is a database user who gets immediate access rights to *directory*

role is a database role that gets immediate access rights to *directory*

`PUBLIC` gives all database users immediate access rights to *directory*

Example: Creating and Using a Database Directory

The following SQL commands create a directory named OLAPFILES to control access to a directory named /users/oracle/OraHome1/olap and grant read access to all users.

```
CREATE DIRECTORY olapfiles as '/users/oracle/OraHome1/olap';
GRANT READ ON DIRECTORY olapfiles TO PUBLIC;
```

Users access files located in /users/oracle/OraHome1/olap with DML commands such as this one:

```
IMPORT ALL FROM EIF FILE 'olapfiles/salesq2.eif' DATA DFNS
```

Understanding Data Storage

Oracle OLAP multidimensional data is stored in analytic workspaces, which are, in turn, stored in relational tables. An analytic workspace can contain a variety of objects, such as dimensions, variables, and OLAP DML programs. These objects typically support a particular application or set of data.

Whenever an analytic workspace is created, modified, or accessed, the information is stored in a table in the relational database.

Important: These tables are vital for the operation of Oracle OLAP. Do not delete them or attempt to modify them directly unless you are fully aware of the consequences.

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_AW user creates two analytic workspaces, one named GLOBAL and the other named GLOBAL_PROGRAMS, then these tables will be created in the GLOBAL_AW schema:

```
AW$GLOBAL
AW$GLOBAL_PROGRAMS
```

Tables are created by default with eight partitions. You can manage these partitions the same as you would for any other table in your database.

The tables store all of the object definitions and data. Each object in an analytic workspace is stored in one or more page spaces, and each page space is stored in a separate row of the table. A page space is grouping of related data pages; a page is a unit for swapping data in and out of memory.

For example, a dimension is stored in three page spaces and thus has three rows (one each for dimension members, a hash index, and a logical-to-physical map). A variable is stored in one row; a partitioned variable has a row for each partition.

Table 12–2 describes the columns of a table that stores an analytic workspace.

Table 12–2 Column Descriptions for Analytic Workspace Tables

Column	Data Type	NULL	Description
EXTNUM	NUMBER(8)	-	Extension number Analytic workspaces are stored in physical LOBs (called extensions), which have a default maximum size of 500MB. The first extension is 0, the second is 1, and so forth.
PS#	NUMBER(10)	-	Page space number Each object is stored in at least one page space.
GEN#	NUMBER(10)	-	Generation number A generation (a snapshot of the page space) is maintained for each reader to assure a consistent view of the analytic workspace throughout a session.
AWLOB	BLOB	-	Analytic workspace LOB Actual storage of the analytic workspace object.
OBJNAME	VARCHAR2(60)	-	Object name The name of the object in the analytic workspace.
PARTNAME	VARCHAR2(60)	-	Partition name A name for the page space in which the object is stored. Each object is stored in its own page space. A partitioned variable is stored with a page space for each partition. The number of partitions and their names are specified when a partition template is created in the analytic workspace.

Table 12–3 shows a few sample rows of an analytic workspace table, which are the results of the following query.

```
SELECT * FROM aw$global WHERE
     OBJNAME = 'TIME' OR
     OBJNAME = 'UNITS_VARIABLE'
     ORDER BY GEN#, PS#;
```

Table 12–3 Sample Rows From AW$GLOBAL

EXTNUM	PS#	GEN#	AWLOB	OBJNAME	PARTNAME
0	2515	0	-	TIME	TIME
0	2516	0	-	TIME	TIME
0	2517	0	-	TIME	TIME
0	2745	0	-	UNITS_VARIABLE	UNITS_VARIABLE
0	2515	9	-	TIME	TIME
0	2516	9	-	TIME	TIME
0	2517	9	-	TIME	TIME

See Also: *Oracle OLAP DML Reference* for information about managing analytic workspaces.

System Tables

The SYS user owns several tables associated with analytic workspaces:

```
AW$EXPRESS
AW$AWCREATE
AW$AWMD
AW$
PS$
```

- AW$EXPRESS stores the EXPRESS analytic workspace. This workspace contains objects and programs that support the OLAP DML. The EXPRESS workspace is used any time that a session is open.
- AW$AWCREATE stores the AWCREATE analytic workspace, which contains programs for creating and managing standard form analytic workspaces.
- AW$AWMD stores the AWMD analytic workspace, which contains programs for creating standard form catalogs.

- AW$ maintains a record of all analytic workspaces in the database, recording its name, owner, and other information.
- PS$ maintains a history of all page spaces. A page 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 the 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$.

The CWM1 and CWM2 read APIs are tables owned by the OLAPSYS user. Public synonyms provide user access to these tables.

Monitoring Performance

Each Oracle Database instance maintains a set of virtual tables that record current database activity. These tables are called dynamic performance tables. The dynamic performance tables collect data on internal disk structures and memory structures. Among them are tables that collect data on Oracle OLAP. By monitoring these tables, you can detect usage trends and diagnose system bottlenecks. Refer to the *Oracle OLAP Reference* for information about the OLAP dynamic performance views.

13

Materialized Views for the OLAP API

This chapter explains how to create materialized views specific to the requirements of the OLAP API and the BI Beans. If you are using analytic workspaces, then you can skip this information because an analytic workspace generates and stores aggregate data so that materialized views are unnecessary. However, if you are developing a strictly relational application, then you must create materialized views using the methods described here. Otherwise, the SQL used to create the materialized views will not match the SQL generated by the OLAP API, and Query Rewrite will not use the materialized views to formulate the answer set to a query.

> **See Also:** *Oracle Data Warehousing Guide* for information on managing materialized views.

This chapter includes the following topics:

Summary Management with Oracle OLAP

A basic feature of online analytical processing (OLAP) is the ability to analyze and view various levels of aggregate data. With Oracle OLAP, you can choose to store aggregate data within analytic workspaces or within materialized views.

Summary management for relational warehouses is managed by Oracle's query rewrite facility. Query rewrite enables a query to fetch aggregate data from materialized views rather than recomputing the aggregates at runtime.

When the OLAP API queries a warehouse stored in relational tables, it uses query rewrite whenever possible. To prepare your relational warehouse for access by the OLAP API, you need to establish materialized views according to the guidelines described in this chapter.

Overview and Requirements

The OLAP API requires a specific set of materialized views for each OLAP Catalog cube that maps to a star schema. The cube must be mapped to a single fact table, and the fact table may contain only lowest-level data.

For each cube, there must be a separate dimension materialized view for each hierarchy of each of the cube's dimensions. For the cube's fact table, there is a single materialized view, created with `GROUP BY GROUPING SETS` syntax.

Use the Oracle Data Management package, `DBMS_ODM`, to create materialized views.

Important: Do not use the `DBMS_OLAP` package to create materialized views for the OLAP API. Query rewrite will not map the SQL generated by the OLAP API to the materialized views generated by this package.

The `DBMS_OLAP` package is described in the Oracle Data Warehousing Guide.

Materialized Views Required for a Cube

The OLAP API requires a dimension materialized view for each hierarchy associated with a cube. For example, the `SALES_CUBE` cube in the Sales History (`SH`) schema requires seven dimension materialized views, as illustrated in Table 13–1.

For the cube's fact table, the OLAP API requires a single grouping set materialized view.

Table 13–1 Number of Dimension Materialized Views for SH.SALES_CUBE

SALES_CUBE Dimensions	Hierarchies	Number of MVs
SH.CHANNELS_DIM	CHANNEL_ROLLUP	1
SH.CUSTOMERS_DIM	CUST_ROLLUP	2
	GEOG_ROLLUP	
SH.PRODUCTS_DIM	PROD_ROLLUP	1
SH.PROMOTIONS_DIM	PROMO_ROLLUP	1
SH.TIMES_DIM	CAL_ROLLUP	2
	FIS_ROLLUP	

Materialized Views and OLAP Metadata

Before creating materialized views, you must create OLAP metadata for the star schema. You can use Oracle Enterprise Manager, or you can write a script using the CWM2 packages. Refer to Chapter 5 for information about the OLAP Catalog.

Example: Dimension Materialized View

The SQL script for creating dimension materialized views includes a CREATE MATERIALIZED VIEW statement, and statements for generating statistics and bitmap indexes.

CREATE Materialized View for a Dimension Hierarchy

The basic syntax of the CREATE MATERIALIZED VIEW statement for a dimension hierarchy is as follows.

```
CREATE MATERIALIZED VIEW mv_name
PARTITION BY RANGE (gid)
   (partition values less than(1) ,
     .
     .
     partition values less than(MAXVALUE))
TABLESPACE tblspace_name
BUILD IMMEDIATE
USING NO INDEX
REFRESH FORCE
ENABLE QUERY REWRITE
AS
```

```
SELECT
   COUNT(*) COUNT_STAR,
   GROUPING_ID(level_columns) gid,
   MAX(attribute_column_1)
   .
   .
   MAX(attribute_column_n)
   level_cols
FROM
   dimension_tables
GROUP BY
   hierarchy1_level1, ROLLUP(hierarchy1_level2,... hierarchy1_leveln),
   hierarchy2_level1, ROLLUP(hierarchy2_level2,... hierarchy2_leveln),
     .
     .
   hierarchyn_level1, ROLLUP(hierarchyn_level2,... hierarchyn_leveln);
```

In the `GROUP BY` clause, level columns are listed in order from most aggregate (level1) to least aggregate (level*n*). The least aggregate level, or "leaf node", is also the key column. Note that level1 is excluded from the `ROLLUP` list.

Bitmap Indexes for a Dimension Hierarchy

The script includes statements like the following to generate bitmap indexes for the level columns and the `GID` column. It also calculates a bitmap index for the parent `GID` and parent `ET` key.

```
CREATE BITMAP INDEX index_name ON mv_name(level_column)
PCTFREE 0
COMPUTE STATISTICS
LOCAL
NOLOGGING;
```

Statistics for a Dimension Hierarchy

The script includes statements like the following to generate statistics.

```
execute dbms_stats.gather_table_stats(mv_owner, mv_name,
degree=>dbms_stats.default_degree,method_opt=>
  'for all columns size skewonly') ;
ALTER TABLE mv_name MINIMIZE RECORDS_PER_BLOCK ;
```

Example: Fact Materialized View

The SQL script generated by the DBMS_ODM package for creating fact materialized views includes a CREATE MATERIALIZED VIEW statement and statements for generating statistics and bitmap indexes.

CREATE Fact Materialized View

The basic syntax of the CREATE MATERIALIZED VIEW statement with grouping sets for a fact table is as follows.

```
CREATE MATERIALIZED VIEW mv_name
PARTITION BY RANGE (gid)
   (partition values less than(1) ,
     .
     .
     partition values less than(MAXVALUE))
PCTFREE x PCTUSED y
BUILD IMMEDIATE
USING NO INDEX
REFRESH FORCE
ENABLE QUERY REWRITE
AS
SELECT
   GROUPING_ID(level_columns) gid,
   agg_method(measure_1),
    .
    .
   agg_method(measure_n),
   COUNT(*) COUNT_OF_STAR,
   level_columns
FROM
   dimension_tables, fact_table
WHERE
   (dimension_primary_key_1 = fact_foreign_key_1) AND
    .
    .
   (dimension_primary_key_n = fact_foreign_key_n)
GROUP BY GROUPING SETS (
   (level columns in grouping set_1),
   .
   .
   (level columns in grouping set_n);
```

Each grouping set contains a combination of levels specified for aggregation. For example, a grouping set could specify that the cube's data be aggregated by month for all products in each region. The procedures in the DBMS_ODM package use two tables, SYS.OLAPTABLEVELS and SYS.OLAPTABLEVELTUPLES, to construct the level combinations in each grouping set. For information on generating and editing these tables, see "Procedure: Create Grouping Set Materialized Views" on page 13-7.

The SELECT clause lists the levels from the dimension tables and the measures from the fact table. The selected measures will be aggregated over each combination of these levels that has been specified for aggregation. The aggregation method is typically addition (SUM), but it may be a method such as average or weighted average. The aggregation method associated with each measure is specified in the OLAP Catalog metadata for the measure.

Bitmap Indexes for Fact Materialized Views

The script includes statements like the following to generate bitmap indexes for each level chosen for inclusion in the materialized view. It also creates a bitmap index for all higher aggregate levels within the dimension. For example, if you chose to aggregate to the quarter level of a time calendar hierarchy, a bitmap index would be created for year and quarter.

```
CREATE BITMAP INDEX index_name ON mv_name(level_col)
LOCAL
COMPUTE STATISTICS
PARALLEL PCTFREE 0
NOLOGGING;
```

Statistics for Fact Materialized Views

The script includes statements like the following to generate statistics.

```
execute dbms_stats.gather_table_stats(mv_owner, mv_name,
   degree=>dbms_stats.default_degree, estimate_percent=>
   dbms_stats.auto_sample_size, method_opt=>
  'for all columns size 1 for columns size 254 GID' , granularity=>'GLOBAL') ;
ALTER TABLE mv_name MINIMIZE RECORDS_PER_BLOCK ;
```

Using the DBMS_ODM Package

The procedures in the OLAP Data Management package, DBMS_ODM, generate scripts that create dimension materialized views and fact materialized views in grouping set form. You can run these scripts in their original form, modify the

scripts before executing them, or use them simply as models for writing your own SQL scripts.

> **Important:** If you choose to modify the scripts, take care to generate materialized views with the same structure as those generated by DBMS_ODM. Otherwise the materialized views may not be accessible to the OLAP API.

Procedure: Create Grouping Set Materialized Views

Follow these steps to create grouping set materialized views for a cube:

1. Create a cube in the OLAP Catalog. You can use Enterprise Manager, or you can use the CWM2 procedures. If you use the CWM2 procedures, be sure to map the cube to a star schema.
2. Enable your database to write scripts to a file by setting the UTL_FILE_DIR parameter to a valid directory.
3. Log in to SQL*Plus using the identity of the metadata owner.
4. Delete any materialized views that currently exist for the cube.
5. Create scripts to generate the dimension materialized views. Execute DBMS.CREATEDIMMV_GS for each of the cube's dimensions.
6. Use the following three step procedure to create a script to generate a grouping set materialized view for the cube's fact table:
   a. Execute DBMS_ODM.CREATEDIMLEVTUPLE to create the table SYS.OLAPTABLEVELS. This table lists all the dimensions of the cube and all the levels of each dimension.

      By default, all the levels of all the dimensions are selected for inclusion in the materialized view. If you know that you will not need to store aggregate data for some levels, you can edit the table to deselect those levels.

   b. Execute DBMS_ODM.CREATECUBELEVELTUPLE to create the table SYS.OLAPTABLEVELTUPLES. This table lists all the possible combinations (grouping sets) of the cube's levels. Only the grouping sets that include the levels selected in SYS.OLAPTABLEVELS are selected for inclusion in the materialized view. If you know that you will not need to store aggregate data for some of these level combinations, you can edit the table to deselect those combinations

c. Execute DBMS_ODM.CREATEFACTMV_GS to create the script.

7. Optionally, edit the scripts using any text editor.
8. Run the scripts in SQL*Plus, using commands such as the following:

```
@/users/oracle/OraHome1/olap/mvscript.sql;
```

Example: Create Grouping Set Materialized Views for a Sales Cube

Let's assume that you want to create materialized views for the DRUGSTORE cube in the DRUG_DEPOT schema. The cube contains sales, cost, quantity, and forecasting data. It is mapped to a fact table containing only lowest-level data and to dimension tables for CHANNEL, GEOGRAPHY, PRODUCT, and TIME. Each dimension has a single hierarchy.

1. First generate the scripts for the dimension materialized views. The following statements create the scripts chanmv, prodmv, geogmv, and timemv in /dat1/scripts/drug_depot.

```
EXEC DBMS_ODM.CREATEDIMMV_GS
   ('drug_depot', 'channel','chanmv','/dat1/scripts/drug_depot');
EXEC DBMS_ODM.CREATEDIMMV_GS
   ('drug_depot', 'product','prodmv','/dat1/scripts/drug_depot');
EXEC DBMS_ODM.CREATEDIMMV_GS
   ('drug_depot', 'geography','geogmv','/dat1/scripts/drug_depot');
EXEC DBMS_ODM.CREATEDIMMV_GS
   ('drug_depot', 'time','timemv','/dat1/scripts/drug_depot');
```

2. Run the scripts to create the dimension materialized views.
3. Next create the table of dimension levels for the fact materialized view.

```
EXEC DBMS_ODM.CREATEDIMLEVTUPLE('drug_depot', 'drugstore');
```

The table of levels, SYS.OLAPTABLEVELS, is a temporary table specific to your session. You can view the table as follows.

```
select * from SYS.OLAPTABLEVELS;

SCHEMA_NAME    DIMENSION_NAME  CUBE_NAME     LEVEL_NAME       SELECTED
-----------    --------------  ----------    ----------       --------
DRUG_DEPOT     CHANNEL         DRUGSTORE     TOTAL               1
DRUG_DEPOT     CHANNEL         DRUGSTORE     CHANNEL_CLASS       1
DRUG_DEPOT     CHANNEL         DRUGSTORE     CHANNEL_ID          1
DRUG_DEPOT     GEOGRAPHY       DRUGSTORE     TOTAL               1
DRUG_DEPOT     GEOGRAPHY       DRUGSTORE     REGION              1
```

```
DRUG_DEPOT     GEOGRAPHY       DRUGSTORE     SUB_REGION          1
DRUG_DEPOT     GEOGRAPHY       DRUGSTORE     COUNTRY             1
DRUG_DEPOT     GEOGRAPHY       DRUGSTORE     STATE_PROVINCE      1
DRUG_DEPOT     PRODUCT         DRUGSTORE     TOTAL               1
DRUG_DEPOT     PRODUCT         DRUGSTORE     PROD_CATEGORY       1
DRUG_DEPOT     PRODUCT         DRUGSTORE     PROD_SUBCATEGORY    1
DRUG_DEPOT     PRODUCT         DRUGSTORE     ID                  1
DRUG_DEPOT     TIME            DRUGSTORE     Year                1
DRUG_DEPOT     TIME            DRUGSTORE     Quarter             1
DRUG_DEPOT     TIME            DRUGSTORE     Month               1
```

All the levels in SYS.OLAPTABLEVELS are initially selected with "1" in the SELECTED column.

4. Let's assume that you want to store aggregate data for each region and sub-region, across all channels and all categories of products. You do not care about data at the month level, you only want to store quarter and year data in the materialized view.

   Edit SYS.OLAPTABLEVELS to deselect all `CHANNEL` levels except total, the state-province level of `GEOGRAPHY`, sub-categories and individual product IDs in `PRODUCT`, and month in `TIME`.

```
update SYS.OLAPTABLEVELS set selected = 0
   where LEVEL_NAME in ('CHANNEL_ID','CHANNEL_CLASS', 'STATE_PROVINCE',
                        'ID','PROD_SUBCATEGORY','Month');
select * from sys.olaptablevels;

SCHEMA_NAME    DIMENSION_NAME  CUBE_NAME     LEVEL_NAME       SELECTED
-----------    --------------  ----------    ----------       --------
DRUG_DEPOT     CHANNEL         DRUGSTORE     TOTAL               1
DRUG_DEPOT     CHANNEL         DRUGSTORE     CHANNEL_CLASS       0
DRUG_DEPOT     CHANNEL         DRUGSTORE     CHANNEL_ID          0
DRUG_DEPOT     GEOGRAPHY       DRUGSTORE     TOTAL               1
DRUG_DEPOT     GEOGRAPHY       DRUGSTORE     REGION              1
DRUG_DEPOT     GEOGRAPHY       DRUGSTORE     SUB_REGION          1
DRUG_DEPOT     GEOGRAPHY       DRUGSTORE     COUNTRY             1
DRUG_DEPOT     GEOGRAPHY       DRUGSTORE     STATE_PROVINCE      0
DRUG_DEPOT     PRODUCT         DRUGSTORE     TOTAL               1
DRUG_DEPOT     PRODUCT         DRUGSTORE     PROD_CATEGORY       1
DRUG_DEPOT     PRODUCT         DRUGSTORE     PROD_SUBCATEGORY    0
DRUG_DEPOT     PRODUCT         DRUGSTORE     ID                  0
DRUG_DEPOT     TIME            DRUGSTORE     Year                1
DRUG_DEPOT     TIME            DRUGSTORE     Quarter             1
DRUG_DEPOT     TIME            DRUGSTORE     Month               0
```

5. Next create the table SYS.OLAPTABLEVELTUPLES. This table, which is also a session-specific temporary table, contains all the possible combinations of the cube's levels. Each combination of four levels, or grouping set, has an identification number. The grouping sets that include the levels you selected in SYS.OLAPTABLEVELS are marked with a 1 in the SELECTED column.

```
exec dbms_odm.createcubeleveltuple('drug_depot','drugstore');
select * from sys.olaptableveltuples;

ID       SCHEMA_NAME CUBE_NAME  DIMENSION_NAME   LEVEL_NAME         SELECTED
--       ----------- ---------  --------------   -----------        --------
1        DRUG_DEPOT  DRUGSTORE  GEOGRAPHY        STATE_PROVINCE         0
1        DRUG_DEPOT  DRUGSTORE  PRODUCT          ID                     0
1        DRUG_DEPOT  DRUGSTORE  CHANNEL          CHANNEL_ID             0
1        DRUG_DEPOT  DRUGSTORE  TIME             Month                  0
2        DRUG_DEPOT  DRUGSTORE  GEOGRAPHY        COUNTRY                0
2        DRUG_DEPOT  DRUGSTORE  PRODUCT          ID                     0
2        DRUG_DEPOT  DRUGSTORE  CHANNEL          CHANNEL_ID             0
2         DRUG_DEPOT  DRUGSTORE  TIME                Month                   0
.
.
.
112         DRUG_DEPOT  DRUGSTORE  GEOGRAPHY       COUNTRY                 1
112         DRUG_DEPOT  DRUGSTORE  PRODUCT         PROD_CATEGORY           1
112         DRUG_DEPOT  DRUGSTORE  CHANNEL         TOTAL                   1
112         DRUG_DEPOT  DRUGSTORE  TIME            Quarter                 1
113         DRUG_DEPOT  DRUGSTORE  GEOGRAPHY       SUB_REGION              1
113         DRUG_DEPOT  DRUGSTORE  PRODUCT         PROD_CATEGORY           1
113         DRUG_DEPOT  DRUGSTORE  CHANNEL         TOTAL                   1
113         DRUG_DEPOT  DRUGSTORE  TIME            Quarter                 1
.
.
.
179         DRUG_DEPOT  DRUGSTORE  GEOGRAPHY       REGION                  1
179         DRUG_DEPOT  DRUGSTORE  PRODUCT         TOTAL                   1
179         DRUG_DEPOT  DRUGSTORE  CHANNEL         TOTAL                   1
179         DRUG_DEPOT  DRUGSTORE  TIME            Year                    1
180         DRUG_DEPOT  DRUGSTORE  GEOGRAPHY       TOTAL                   1
180         DRUG_DEPOT  DRUGSTORE  PRODUCT         TOTAL                   1
180         DRUG_DEPOT  DRUGSTORE  CHANNEL         TOTAL                   1
180         DRUG_DEPOT  DRUGSTORE  TIME            Year                    1
```

The `SYS.OLAPTABLEVELTUPLES` table has 720 rows, identifying 180 unique level tuples, or grouping sets. 180 is the product of the number of levels for each of the

cube's dimensions, 3*5*4*3. There are 3 levels in CHANNEL, 5 levels in GEOGRAPHY, 4 levels in PRODUCT, and 3 levels in TIME

Of the 180 grouping sets, only 16 are selected for inclusion in the materialized view. You can display the 64 selected rows (16*4) with the following statement.

```
select * from sys.olaptableveltuples where SELECTED = 1;
```

ID	SCHEMA_NAME	CUBE_NAME	DIMENSION_NAME	LEVEL_NAME	SELECTED
112	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	COUNTRY	1
112	DRUG_DEPOT	DRUGSTORE	PRODUCT	PROD_CATEGORY	1
112	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
112	DRUG_DEPOT	DRUGSTORE	TIME	Quarter	1
113	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	SUB_REGION	1
113	DRUG_DEPOT	DRUGSTORE	PRODUCT	PROD_CATEGORY	1
113	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
113	DRUG_DEPOT	DRUGSTORE	TIME	Quarter	1
114	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	REGION	1
114	DRUG_DEPOT	DRUGSTORE	PRODUCT	PROD_CATEGORY	1
114	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
114	DRUG_DEPOT	DRUGSTORE	TIME	Quarter	1
115	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	TOTAL	1
115	DRUG_DEPOT	DRUGSTORE	PRODUCT	PROD_CATEGORY	1
115	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
115	DRUG_DEPOT	DRUGSTORE	TIME	Quarter	1
117	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	COUNTRY	1
117	DRUG_DEPOT	DRUGSTORE	PRODUCT	TOTAL	1
117	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
117	DRUG_DEPOT	DRUGSTORE	TIME	Quarter	1
118	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	SUB_REGION	1
118	DRUG_DEPOT	DRUGSTORE	PRODUCT	TOTAL	1
118	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
118	DRUG_DEPOT	DRUGSTORE	TIME	Quarter	1
119	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	REGION	1
119	DRUG_DEPOT	DRUGSTORE	PRODUCT	TOTAL	1
119	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
119	DRUG_DEPOT	DRUGSTORE	TIME	Quarter	1
120	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	TOTAL	1
120	DRUG_DEPOT	DRUGSTORE	PRODUCT	TOTAL	1
120	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
120	DRUG_DEPOT	DRUGSTORE	TIME	Quarter	1
172	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	COUNTRY	1
172	DRUG_DEPOT	DRUGSTORE	PRODUCT	PROD_CATEGORY	1
172	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1

```
172        DRUG_DEPOT  DRUGSTORE  TIME          Year                1
173        DRUG_DEPOT  DRUGSTORE  GEOGRAPHY     SUB_REGION          1
173        DRUG_DEPOT  DRUGSTORE  PRODUCT       PROD_CATEGORY       1
173        DRUG_DEPOT  DRUGSTORE  CHANNEL       TOTAL               1
173        DRUG_DEPOT  DRUGSTORE  TIME          Year                1
174        DRUG_DEPOT  DRUGSTORE  GEOGRAPHY     REGION              1
174        DRUG_DEPOT  DRUGSTORE  PRODUCT       PROD_CATEGORY       1
174        DRUG_DEPOT  DRUGSTORE  CHANNEL       TOTAL               1
174        DRUG_DEPOT  DRUGSTORE  TIME          Year                1
175        DRUG_DEPOT  DRUGSTORE  GEOGRAPHY     TOTAL               1
175        DRUG_DEPOT  DRUGSTORE  PRODUCT       PROD_CATEGORY       1
175        DRUG_DEPOT  DRUGSTORE  CHANNEL       TOTAL               1
175        DRUG_DEPOT  DRUGSTORE  TIME          Year                1
177        DRUG_DEPOT  DRUGSTORE  GEOGRAPHY     COUNTRY             1
177        DRUG_DEPOT  DRUGSTORE  PRODUCT       TOTAL               1
177        DRUG_DEPOT  DRUGSTORE  CHANNEL       TOTAL               1
177        DRUG_DEPOT  DRUGSTORE  TIME          Year                1
178        DRUG_DEPOT  DRUGSTORE  GEOGRAPHY     SUB_REGION          1
178        DRUG_DEPOT  DRUGSTORE  PRODUCT       TOTAL               1
178        DRUG_DEPOT  DRUGSTORE  CHANNEL       TOTAL               1
178        DRUG_DEPOT  DRUGSTORE  TIME          Year                1
179        DRUG_DEPOT  DRUGSTORE  GEOGRAPHY     REGION              1
179        DRUG_DEPOT  DRUGSTORE  PRODUCT       TOTAL               1
179        DRUG_DEPOT  DRUGSTORE  CHANNEL       TOTAL               1
179        DRUG_DEPOT  DRUGSTORE  TIME          Year                1
180        DRUG_DEPOT  DRUGSTORE  GEOGRAPHY     TOTAL               1
180        DRUG_DEPOT  DRUGSTORE  PRODUCT       TOTAL               1
180        DRUG_DEPOT  DRUGSTORE  CHANNEL       TOTAL               1
180        DRUG_DEPOT  DRUGSTORE  TIME          Year                1
```

6. Suppose you want to store product totals by year for each sub-region. You do not want to store aggregates for any other grouping sets that contain the sub-region level.

Grouping sets 113, 118, 173, and 178 all use the `SUB_REGION` level of `GEOGRAPHY`.

```
ID      GEOGRAPHY       PRODUCT            CHANNEL      TIME
--      ----------      -------            ------       -----
113     SUB_REGION      PROD_CATEGORY      TOTAL        Quarter
118     SUB_REGION      TOTAL              TOTAL        Quarter
173     SUB_REGION      PROD_CATEGORY      TOTAL        Year
178     SUB_REGION      TOTAL              TOTAL        Year
```

You could edit the SYS.OLAPTABLEVELTUPLES table with a statement like the following.

```
update SYS.OLAPTABLEVELTUPLES set selected = 0
       where ID in ('113','118', '173');
select * from sys.olaptableveltuples where SELECTED = 1;

ID       SCHEMA_NAME CUBE_NAME  DIMENSION_NAME   LEVEL_NAME          SELECTED
--       ----------- ---------  --------------   -----------         --------
112        DRUG_DEPOT  DRUGSTORE  GEOGRAPHY      COUNTRY                  1
112        DRUG_DEPOT  DRUGSTORE  PRODUCT        PROD_CATEGORY            1
112        DRUG_DEPOT  DRUGSTORE  CHANNEL        TOTAL                    1
112        DRUG_DEPOT  DRUGSTORE  TIME           Quarter                  1
114        DRUG_DEPOT  DRUGSTORE  GEOGRAPHY      REGION                   1
114        DRUG_DEPOT  DRUGSTORE  PRODUCT        PROD_CATEGORY            1
114        DRUG_DEPOT  DRUGSTORE  CHANNEL        TOTAL                    1
114        DRUG_DEPOT  DRUGSTORE  TIME           Quarter                  1
115        DRUG_DEPOT  DRUGSTORE  GEOGRAPHY      TOTAL                    1
115        DRUG_DEPOT  DRUGSTORE  PRODUCT        PROD_CATEGORY            1
115        DRUG_DEPOT  DRUGSTORE  CHANNEL        TOTAL                    1
115        DRUG_DEPOT  DRUGSTORE  TIME           Quarter                  1
117        DRUG_DEPOT  DRUGSTORE  GEOGRAPHY      COUNTRY                  1
117        DRUG_DEPOT  DRUGSTORE  PRODUCT        TOTAL                    1
117        DRUG_DEPOT  DRUGSTORE  CHANNEL        TOTAL                    1
117        DRUG_DEPOT  DRUGSTORE  TIME           Quarter                  1
119        DRUG_DEPOT  DRUGSTORE  GEOGRAPHY      REGION                   1
119        DRUG_DEPOT  DRUGSTORE  PRODUCT        TOTAL                    1
119        DRUG_DEPOT  DRUGSTORE  CHANNEL        TOTAL                    1
119        DRUG_DEPOT  DRUGSTORE  TIME           Quarter                  1
120        DRUG_DEPOT  DRUGSTORE  GEOGRAPHY      TOTAL                    1
120        DRUG_DEPOT  DRUGSTORE  PRODUCT        TOTAL                    1
120        DRUG_DEPOT  DRUGSTORE  CHANNEL        TOTAL                    1
120        DRUG_DEPOT  DRUGSTORE  TIME           Quarter                  1
172        DRUG_DEPOT  DRUGSTORE  GEOGRAPHY      COUNTRY                  1
172        DRUG_DEPOT  DRUGSTORE  PRODUCT        PROD_CATEGORY            1
172        DRUG_DEPOT  DRUGSTORE  CHANNEL        TOTAL                    1
172        DRUG_DEPOT  DRUGSTORE  TIME           Year                     1
174        DRUG_DEPOT  DRUGSTORE  GEOGRAPHY      REGION                   1
174        DRUG_DEPOT  DRUGSTORE  PRODUCT       PROD_CATEGORY             1
174        DRUG_DEPOT  DRUGSTORE  CHANNEL       TOTAL                     1
174        DRUG_DEPOT  DRUGSTORE  TIME          Year                      1
175        DRUG_DEPOT  DRUGSTORE  GEOGRAPHY     TOTAL                     1
175        DRUG_DEPOT  DRUGSTORE  PRODUCT       PROD_CATEGORY             1
175        DRUG_DEPOT  DRUGSTORE  CHANNEL       TOTAL                     1
175        DRUG_DEPOT  DRUGSTORE  TIME          Year                      1
```

```
177        DRUG_DEPOT  DRUGSTORE  GEOGRAPHY     COUNTRY                 1
177        DRUG_DEPOT  DRUGSTORE  PRODUCT       TOTAL                   1
177        DRUG_DEPOT  DRUGSTORE  CHANNEL       TOTAL                   1
177        DRUG_DEPOT  DRUGSTORE  TIME          Year                    1
178        DRUG_DEPOT  DRUGSTORE  GEOGRAPHY     SUB_REGION              1
178        DRUG_DEPOT  DRUGSTORE  PRODUCT       TOTAL                   1
178        DRUG_DEPOT  DRUGSTORE  CHANNEL       TOTAL                   1
178        DRUG_DEPOT  DRUGSTORE  TIME          Year                    1
179        DRUG_DEPOT  DRUGSTORE  GEOGRAPHY     REGION                  1
179        DRUG_DEPOT  DRUGSTORE  PRODUCT       TOTAL                   1
179        DRUG_DEPOT  DRUGSTORE  CHANNEL       TOTAL                   1
179        DRUG_DEPOT  DRUGSTORE  TIME          Year                    1
180        DRUG_DEPOT  DRUGSTORE  GEOGRAPHY     TOTAL                   1
180        DRUG_DEPOT  DRUGSTORE  PRODUCT       TOTAL                   1
180        DRUG_DEPOT  DRUGSTORE  CHANNEL       TOTAL                   1
180        DRUG_DEPOT  DRUGSTORE  TIME          Year                    1
```

7. To create the script that will generate the fact materialized view, run the `CREATEFACTMV_GS` procedure.

```
exec dbms_odm.createfactmv_gs
      ('drug_depot','drugstore',
       'drugstore_mv','/dat1/scripts/drug_depot',TRUE);
```

The `CREATE MATERIALIZED VIEW` statement in the script contains the following grouping sets in the `GROUP BY GROUPING SETS` clause.

```
(TIMES.CALENDAR_YEAR, TIMES.CALENDAR_QUARTER, CHANNELS.TOTAL,
    PRODUCTS.TOTAL, PRODUCTS.PROD_CATEGORY, GEOGRAPHIES.TOTAL,
    GEOGRAPHIES.REGION, GEOGRAPHIES.SUB_REGION, GEOGRAPHIES.COUNTRY ),
(TIMES.CALENDAR_YEAR, TIMES.CALENDAR_QUARTER, CHANNELS.TOTAL,
    PRODUCTS.TOTAL, PRODUCTS.PROD_CATEGORY, GEOGRAPHIES.TOTAL,
    GEOGRAPHIES.REGION),
(TIMES.CALENDAR_YEAR, TIMES.CALENDAR_QUARTER, CHANNELS.TOTAL,
    PRODUCTS.TOTAL, PRODUCTS.PROD_CATEGORY, GEOGRAPHIES.TOTAL),
(TIMES.CALENDAR_YEAR, TIMES.CALENDAR_QUARTER, CHANNELS.TOTAL,
    PRODUCTS.TOTAL, GEOGRAPHIES.TOTAL , GEOGRAPHIES.REGION,
    GEOGRAPHIES.SUB_REGION, GEOGRAPHIES.COUNTRY),
(TIMES.CALENDAR_YEAR, TIMES.CALENDAR_QUARTER, CHANNELS.TOTAL,
    PRODUCTS.TOTAL, GEOGRAPHIES.TOTAL, GEOGRAPHIES.REGION),
(TIMES.CALENDAR_YEAR, TIMES.CALENDAR_QUARTER, CHANNELS.TOTAL,
    PRODUCTS.TOTAL, GEOGRAPHIES.TOTAL),
(TIMES.CALENDAR_YEAR, CHANNELS.TOTAL, PRODUCTS.TOTAL,
    PRODUCTS.PROD_CATEGORY, GEOGRAPHIES.TOTAL, GEOGRAPHIES.REGION,
    GEOGRAPHIES.SUB_REGION, GEOGRAPHIES.COUNTRY),
(TIMES.CALENDAR_YEAR, CHANNELS.TOTAL, PRODUCTS.TOTAL,
```

```
    PRODUCTS.PROD_CATEGORY, GEOGRAPHIES.TOTAL, GEOGRAPHIES.REGION),
(TIMES.CALENDAR_YEAR, CHANNELS.TOTAL, PRODUCTS.TOTAL,
    PRODUCTS.PROD_CATEGORY, GEOGRAPHIES.TOTAL),
(TIMES.CALENDAR_YEAR, CHANNELS.TOTAL, PRODUCTS.TOTAL, GEOGRAPHIES.TOTAL,
    GEOGRAPHIES.REGION, GEOGRAPHIES.SUB_REGION, GEOGRAPHIES.COUNTRY),
(TIMES.CALENDAR_YEAR, CHANNELS.TOTAL, PRODUCTS.TOTAL, GEOGRAPHIES.TOTAL,
    GEOGRAPHIES.REGION, GEOGRAPHIES.SUB_REGION),
(TIMES.CALENDAR_YEAR, CHANNELS.TOTAL, PRODUCTS.TOTAL, GEOGRAPHIES.TOTAL,
    GEOGRAPHIES.REGION),
(TIMES.CALENDAR_YEAR, CHANNELS.TOTAL, PRODUCTS.TOTAL, GEOGRAPHIES.TOTAL)
```

The following statement at the end of the script sets the `MV_SUMMARY_CODE` associated with the cube in the OLAP Catalog. This setting indicates that the materialized view associated with this cube is in grouping set form.

```
execute cwm2_olap_cube.set_mv_summary_code
     ('DRUG_DEPOT', 'DRUGSTORE', 'GROUPINGSET') ;
```

8. Run the `drugstore_mv` script to create the fact materialized view.

A

Database Standard Form for Analytic Workspaces

Database standard form is a set of conventions describing the objects in an analytic workspace that can be managed by various Oracle OLAP utilities. This appendix describes database standard form conventions. It has the following sections:

Overview of Database Standard Form

An analytic workspace that conforms to database standard form has objects that implement a logical model for cubes, dimensions, and measures. Standard form includes naming conventions for workspace objects, and it specifies object properties relating the logical model to the workspace objects that implement it.

Purpose of Database Standard Form

The purpose of the standard form conventions is to provide an agreed-upon logical model and workspace implementation to be used by related Oracle OLAP utilities. Because these utilities work with data and metadata that is in the standard form, the

utilities are compatible with one another. Therefore, a DBA who uses these utilities can create and maintain analytic workspaces that can be accessed through more than one analytic tool. Currently, access is available through the OLAP API, BI Beans, and Discoverer.

The Analytic Workspace Manager provides the following utilities, which either create or depend upon the presence of a standard form workspace:

- Analytic Workspace Creation Wizard creates a new workspace in standard form from OLAP Catalog metadata.
- Analytic Workspace Refresh Wizard refreshes an existing workspace in standard form from OLAP Catalog metadata.
- Enable for OLAP API and BI Beans feature enables a standard form workspace for access through the OLAP API and BI Beans.
- Enable for Discoverer Wizard enables a standard form workspace for access through Discoverer.

Analytic Workspace Manager uses the PL/SQL package `DBMS_AWM` to create, refresh, and enable standard form analytic workspaces for access by the OLAP API. You can use the Analytic Workspace Manager to manage your analytic workspaces, or you can develop your own scripts using the `DBMS_AWM` procedures.

Audience for Database Standard Form

Ordinarily, you create, manage, and enable standard form workspaces using the tools and procedures provided. Therefore, you will typically have no need for detailed knowledge of the standard form. However, such knowledge is necessary under the following circumstances:

- When you want to make manual additions to an existing standard form workspace. For example, the OLAP Catalog metadata might not have included a plural description for a dimension, and the DBA might want to add one in the workspace rather than in the OLAP Catalog.
- When you are developing an application that uses standard form workspaces, and you want it to discover information at run-time, such as which measures are available for analysis in a particular workspace, how they are dimensioned, and what levels and hierarchies are defined.

This appendix describes the standard form so that if you have these requirements, you can understand the conventions of a standard form workspace. To understand these conventions, you must be familiar with multidimensional OLAP concepts and should be experienced in using the OLAP DML.

Logical Model and Workspace Objects

The standard form logical model includes cubes, measures, and dimensions, as well as the hierarchies, levels, and attributes that are associated with dimensions. A cube is considered to be the parent of the measures that it contains, and a dimension is considered to be the parent of its hierarchies, levels, and attributes. A cube has dimensionality; that is, it is associated with its list of dimensions.

Implementation of a Cube

The primary workspace object that implements a logical cube is a workspace dimension referred to as the *cubedef* dimension. The values of this dimension are the names of the cube's dimensions.

Secondary workspace objects that implement a logical cube are an aggmap (referred to as the *comspec* aggmap) and a composite (referred to as the *loopspec* composite).

For more information about these objects, see "Cube Objects" on page A-15.

Implementation of a Measure

The primary workspace object that implements a logical measure is a workspace variable, formula, or relation referred to as the *measuredef* object. The values of this object are the values of the logical measure.

The only secondary workspace object for a measure is the *compspec* aggmap for its cube.

For more information about these objects, see "Measure Objects" on page A-17.

Implementation of a Dimension

The primary workspace object that implements a logical dimension is a workspace dimension referred to as the *dimdef* dimension. The values of this dimension are the values of the logical dimension.

Hierarchies and levels of a dimension do not have primary objects of their own. Instead, the following objects provide the implementation:

- *hierlist* dimension lists the dimension's hierarchies
- *levellist* dimension lists the dimension's levels
- *parentrel* relation records the parent for each member of the dimension
- *levelrel* relation records the level for each member of the dimension
- *hier_levels* valueset records the levels in each hierarchy

The primary workspace object that implements an attribute of a dimension is a workspace variable, formula, or relation referred to as the *attrdef* object. The values of this object are the values of the logical attribute.

For more information about these objects, see "Dimension Objects" on page A-18.

Classes of Workspace Objects

Each standard form workspace object belongs to one of four classes:

- **Implementation class.** Objects in this class implement the logical model. They include all the workspace objects described in the section "Logical Model and Workspace Objects" on page A-3, for example the *cubedef*, *measuredef*, *dimdef*, and *hierlist* objects.
- **Catalogs class.** Objects in this class hold information about the logical model. They include a list of all the cubes in the workspace, a list of all the measures in the workspace, a list of all the dimensions in the workspace, and other lists that can facilitate the work of various utilities.
- **Features class.** Objects in this class hold information about specific objects in the logical model. For example, one object stores the descriptions of all the logical objects, while another indicates whether the object is intended to be visible to the user.
- **Extensions class.** Objects in this class are defined and maintained by the Oracle OLAP utilities. They are proprietary extensions to the standard form, and there is no commitment on the part of Oracle to maintain them from release to release.

  Do not define, modify, or depend on objects in the Extensions class.

Properties of Workspace Objects

A fundamental feature of standard form is that it depends on the OLAP DML properties of workspace objects for the implementation of the logical model. OLAP DML properties are assigned using the OLAP DML `PROPERTY` command.

Object Naming Conventions

There are no restrictions on the names of the workspace objects that implement a standard form logical model, other than the rules imposed by the OLAP DML. For logical objects, however, standard form imposes strict naming rules. This is because the utilities that depend on standard form reference objects by their logical names.

Standard form naming conventions for logical names are consistent with those of the Oracle Database. They establish name spaces within which logical names must be unique, and they provide rules for constructing full names to reflect the name space organization. Logical names are sometimes referred to as "simple logical names" in order to distinguish them from full names.

Logical Names

In general, the simple logical name for an object, such as a cube or dimension, conforms to the rules for a SQL simple expression, with minor differences. The rules for standard form logical names require that a name:

1. Have 1 to 30 bytes.
2. Cannot be an Oracle reserved word.
3. Is not case-sensitive.
4. Cannot contain quotation marks.
5. Must begin with an alphabetic character from your database character set.
6. Must contain only alphanumeric characters from your database character set and the underscore (_), dollar sign ($), and pound sign (#). However, Oracle strongly discourages you from using the dollar or pound sign. If your database character set contains multi byte characters, Oracle recommends that you include at least one single-byte character in each logical name.

The `AW$LOGICAL_NAME` property of a workspace object contains the simple logical name of the object that it implements. An example of a simple logical name is `PRODUCT`.

Name Space Organization

Standard form naming conventions impose an organization of logical objects that defines the following name spaces:

- **Schema.** The logical names of cubes and dimensions must be unique within the schema that owns the analytic workspace.
- **Cube.** The logical names of measures must be unique within a given cube.
- **Dimension.** The logical names of hierarchies must be unique within a dimension. The logical names of levels must be unique within a dimension. The logical names of attributes must be unique within a dimension. Within a given dimension, a hierarchy can have the same name as a level or attribute.

The name space organization reflects an ownership, or parent, relationship among the logical objects. For example, a measure has a cube as its parent object, and an attribute has a dimension as its parent object. The AW$PARENT_NAME property on workspace objects records these relationships.

Simple Logical Names and Full Names

Because simple logical names are not unique outside their name space, standard form conventions specify a full name for each logical object. This full name includes the simple logical name, but also indicates the name space to which the object belongs and its object type. The following is an example of a full name for an attribute whose simple name is TIME_SPAN and whose parent object is a dimension called TIME.

```
GLOBAL_AW.TIME.TIME_SPAN.ATTRIBUTE
```

The final component of a full name is the object type. In this example, it is ATTRIBUTE. All the possible types are listed in the *all_objtypes* dimension, which is described in "ALL_OBJTYPES Dimension" on page A-29.

Full names are used in the catalog class objects that list various object types. For example, the values of the *all_dimensions*, *all_cubes*, and *all_attributes* dimensions are the full names of logical objects.

Workspace Object Properties

The section "Properties of Workspace Objects" on page A-4 introduced the use of properties in the standard form. Properties are the primary method by which logical objects are implemented by workspace objects. The properties are created on the workspace objects using the OLAP DML PROPERTY command.

Workspace objects in the standard form have well-defined properties that fall into three groups:

- Properties specific to implementation class objects.
- System properties on all workspace objects.

  These properties are created and given values by Oracle OLAP utilities, either DBMS_AWM or the utilities offered by Analytic Workspace Manager. You must never modify or delete these properties.
- Role property on all workspace objects.

All objects that are in the standard form have a property called AW$ROLE. It indicates the role (or function) that is played by the object in the standard form.

Properties Specific to Implementation Class Objects

Properties for the logical name and parent name are on all implementation class objects. Three additional properties might or might not be present depending on the role of the object.

Table A–1 lists the implementation class properties and describes each one.

Table A–1 Implementation Class Properties

Property	Description
AW$LOGICAL_NAME	The simple logical name of the logical object that is implemented by this workspace object. The value is set only for objects whose role is CUBEDEF, MEASUREDEF, DIMDEF, and ATTRDEF. The property exists, but the value is NA, for all other roles in the implementation class.
AW$PARENT_NAME	The simple logical name of the parent of the logical object that is implemented by this workspace object. The value is set for all implementation class objects except for those whose roles are CUBEDEF and DIMDEF. The value is NA for these two, because they have no parent.
AW$LOOPSPEC	For objects with role CUBEDEF, the name of the composite for the cube. This is the name of a workspace object, not the logical name of an object. For all other roles, this property is missing.
AW$COMPSPEC	For objects with role MEASUREDEF, the name of the AGGMAP object for the measure. This is the name of a workspace object, not the logical name of an object. For all other roles, this property is missing.
AW$TYPE	For objects with role DIMDEF and ATTRDEF, the type of the dimension or attribute. For all other roles, this property is missing. If the role is DIMDEF, this property indicates whether the dimension is a time dimension. Values are TIME or NA. If the role is ATTRDEF, this property indicates a special use for the attribute by Oracle OLAP. Values that indicate special use are DEFAULT_ORDER, END_DATE, TIME_SPAN, MEMBER_LONG_DESCRIPTION, MEMBER_SHORT_DESCRIPTION, MEMBER_VISIBLE. If the value is USER or NA, then the attribute has no special meaning for Oracle OLAP.

System Properties on All Workspace Objects

All workspace objects that are part of the standard form have four system properties.

Table A–2 lists the system properties and describes each one.

Table A–2 System Properties

Property	Description
AW$CLASS	The class of the workspace object. Possible values are IMPLEMENTATION, CATALOGS, FEATURES, and EXTENSIONS. For a description of these classes, see "Classes of Workspace Objects" on page A-4.
AW$CREATEDBY	The entity that created the workspace object. For example, if it was created by DBMS_AWM, then the value is AW$CREATE.
AW$LASTMODIFIED	The date and time when the workspace object was last registered.
AW$STATE	The state of the workspace object with respect to the standard form, for example, VALID_MEMBER.

Role Property on All Workspace Objects

All workspace objects that are part of the standard form have a role property.

Table A–3 describes the role property.

Table A–3 Role Property

Property	Description
AW$ROLE	The role (that is, function) that is performed by this object. The possible values are different for each object class. For information on property values, see "Role Property Values for Implementation Class Objects" on page A-8, "Role Property Values for Catalogs Class Objects" on page A-10, "Role Property Values for Features Class Objects" on page A-12, and "Role Property Values for Extensions Class Objects" on page A-13.

Role Property Values for Implementation Class Objects

The AW$ROLE property indicates the function (that is, role) that is performed by the workspace object. For implementation class objects, roles indicate fundamental building blocks of the logical model, such as cubes, measures, and dimensions.

There can be several implementation class objects that have the same role in a standard form workspace. For example, there are several objects with the role of DIMDEF because there is one such object for each dimension in the logical model.

Table A–4 lists the possible values and describes each role.

Table A–4 Role Property Values: Implementation Class

Role Property Value	Role Description
CUBEDEF	Implements a cube whose logical name is in the AW$LOGICAL_NAME property. For information about objects with this role, see "Cubedef Dimension" on page A-15.
MEASUREDEF	Implements a measure whose logical name is in the AW$LOGICAL_NAME property. For information about objects with this role, see "Measuredef Object" on page A-17.
DIMDEF	Implements a dimension whose logical name is in the AW$LOGICAL_NAME property. For information about objects with this role, see "Dimdef Dimension" on page A-19.
HIERLIST	Lists the names of the hierarchies of the dimension whose name is in the AW$PARENT_NAME property. For information about objects with this role, see "Hierlist Dimension" on page A-20.
LEVELLIST	Lists the names of the levels of the dimension whose name is in the AW$PARENT_NAME property. For information about objects with this role, see "Levellist Dimension" on page A-20.
MEMBER_LEVELREL	Records the level for each member of the dimension whose name is in the AW$PARENT_NAME property. For information about objects with this role, see "Member_Levelrel Relation" on page A-21.
MEMBER_PARENTREL	Records the parent for each member of the dimension whose name is in the AW$PARENT_NAME property. For information about objects with this role, see "Member_Parentrel Relation" on page A-22.
HIER_LEVELS	Lists the levels that are included in each hierarchy of the dimension whose name is in the AW$PARENT_NAME property. For information about objects with this role, see "Hier_Levels Valueset" on page A-23.
ATTRDEF	Implements an attribute whose logical name is in the AW$LOGICAL_NAME property. For information about objects with this role, see "Attrdef Object" on page A-23.

Role Property Values for Catalogs Class Objects

The AW$ROLE property indicates the function (or role) that is performed by the workspace object. For catalogs class objects, the objects with various roles provide information about the logical model such as a list of cubes, a list of object types, or a list of measures.

There is only one catalogs class object with a given role in a standard form workspace. For example, there is only one object that lists all the dimensions in the workspace.

Table A–5 lists the possible values and describes each role.

Table A–5 Role Property Values: Catalogs Class

Role Property Value	Role Description
ALL_OBJECTS	Lists the full names of all the objects that have been registered with the standard form in this workspace. For information about the object with this role, see "ALL_OBJECTS Dimension" on page A-28.
ALL_CUBES	Lists the full names of all the cubes that have been registered with the standard form in this workspace. For information about the object with this role, see "ALL_CUBES Dimension" on page A-25.
ALL_MEASURES	Lists the full names of all the measures that have been registered with the standard form in this workspace. For information about the object with this role, see "ALL_MEASURES Dimension" on page A-26.
ALL_DIMENSIONS	Lists the full names of all the dimensions that have been registered with the standard form in this workspace. For information about the object with this role, see "ALL_DIMENSIONS Dimension" on page A-26
ALL_HIERARCHIES	Lists the full names of all the hierarchies that have been registered with the standard form in this workspace. For information about the object with this role, see "ALL_HIERARCHIES Dimension" on page A-26.
ALL_LEVELS	Lists the full names of all the levels that have been registered with the standard form in this workspace. For information about the object with this role, see "ALL_LEVELS Dimension" on page A-27.
ALL_ATTRIBUTES	Lists the full names of all the attributes that have been registered with the standard form in this workspace. For information about the object with this role, see "ALL_ATTRIBUTES Dimension" on page A-28.

Table A–5 (Cont.) Role Property Values: Catalogs Class

Role Property Value	Role Description
ALL_OBJTYPES	Lists types of objects currently supported by the standard form: CUBE, MEASURE, DIMENSION, HIERARCHY, LEVEL, and ATTRIBUTE. For information about the object with this role, see "ALL_OBJTYPES Dimension" on page A-29.
ALL_DESCTYPES	Lists the types of descriptions currently supported by the standard form: SHORT, LONG, and PLURAL. For information about the object with this role, see "ALL_DESCTYPES Dimension" on page A-30.
ALL_ATTRTYPES	Lists all the attribute types that are currently supported by the standard form. These are valid values for the AW$TYPE property of an object with the ATTRDEF role. For information about the object with the ALL_ATTRTYPES role, see "ALL_ATTRTYPES Dimension" on page A-30.
AW_ROLES	Lists all values for the AW$ROLE property currently supported by the standard form. The list includes roles for objects of all classes. For information about the object with this role, see "AW_ROLES Dimension" on page A-30.
ALL_LANGUAGES	Lists the names of all the languages that a DBA has included in the workspace. For information about the object with this role, see "ALL_LANGUAGES Dimension" on page A-32.
CUBE_MEASURES	Lists the full names of the measures that belong to each cube in the workspace. For information about the object with this role, see "CUBE_MEASURES Valueset" on page A-32.
DIM_HIERARCHIES	Lists the full names of the hierarchies that belong to each dimension in the workspace. For information about the object with this role, see "DIM_HIERARCHIES Valueset" on page A-33.
DIM_LEVELS	Lists the full names of the levels that belong to each dimension in the workspace. For information about the object with this role, see "DIM_LEVELS Valueset" on page A-33.
DIM_ATTRIBUTES	Lists the full names of the attributes that belong to each dimension in the workspace. For information about the object with this role, see "DIM_ATTRIBUTES Valueset" on page A-34.

Table A–5 (Cont.) Role Property Values: Catalogs Class

Role Property Value	Role Description
AW_NAMES	Records the name of the workspace object that implements each logical cube, measure, dimension, and attribute. For other logical objects, there is no single corresponding workspace object, so the value is NA. For information about the object with this role, see "AW_NAMES Variable" on page A-34.
AW_COMPSPECS	Records, for each dimension, the names of all AGGMAP objects that reference the dimension. For information about the object with this role, see "AW_COMPSPECS Variable" on page A-35.
AW_LOOPSPECS	Records, for each cube, the name of its composite. For information about the object with this role, see "AW_LOOPSPECS Variable" on page A-35.

Role Property Values for Features Class Objects

The AW$ROLE property indicates the function (or role) that is performed by the workspace object. For features class objects, roles provide various types of supplementary data for logical objects such as descriptions.

For many roles, there is a single features class object in a standard form workspace. However, for the roles that have MEMBER in their names, there is one object for each dimension.

Table A–6 lists the possible values and describes each role that applies to features class objects.

Table A–6 Role Property Values: Features Class

Role Property Value	Role Description
ALL_DESCRIPTIONS	Records short, long, and plural descriptions for all objects. For information about the object with this role, see "ALL_DESCRIPTIONS Variable" on page A-36.
ATTR_INHIER	Indicates whether a given attribute is associated with a given hierarchy. For information about the object with this role, see "ATTR_INHIER Variable" on page A-36.
DEFAULT_HIER	Records the full name of the default hierarchy for each dimension. For information about the object with this role, see "DEFAULT_HIER Relation" on page A-37.

Table A–6 (Cont.) Role Property Values: Features Class

Role Property Value	Role Description
MEMBER_CREATEDBY	Records the entity that created each member of a given dimension. For information about the object with this role, see "Member_Createdby Variable" on page A-38.
MEMBER_FAMILYREL	Records the family relation for each hierarchy of a given dimension. For information about the object with this role, see "Member_Familyrel Relation" on page A-39.
MEMBER_GID	Records the grouping id for each hierarchy of a given dimension. For information about the object with this role, see "Member_Gid Variable" on page A-39.
MEMBER_INHIER	Indicates whether a given member of a dimension is in a given hierarchy. For information about the object with this role, see "Member_Inhier Variable" on page A-38.
OBJ_CREATEDBY	Records the entity that created each object. For information about the object with this role, see "OBJ_CREATEDBY Variable" on page A-39.
OBJ_STATE	Records the current state of each object that has ever been registered. For information about the object with this role, see "OBJ_STATE Variable" on page A-40.
VERSION	Records the number of the standard form version under which the workspace is being managed. For information about the object with this role, see "VERSION Variable" on page A-40.
VISIBLE	Indicates whether a given object should be made visible to the user by Oracle OLAP enabling utilities. For information about the object with this role, see "VISIBLE Variable" on page A-37.

Role Property Values for Extensions Class Objects

The AW$ROLE property indicates the function (or role) that is performed by the workspace object. For Extensions class objects, roles are for internal use of Oracle OLAP utilities such as DBMS_AWM and the enablers.

DBAs and users must not create, modify, or depend on objects that are in the Extensions class. The AW$ROLE property, and all properties, for objects in this class are for proprietary use only. Oracle makes no commitment to maintain the roles and relationships of these objects.

Terminology: Using Role Names to Describe Objects

Because the standard form conventions have no conventions that govern the names of workspace objects, documentation cannot refer to the objects by name. Instead, the objects are discussed using the values of their AW$ROLE properties as descriptors.

For example, we refer to the *cubedef* dimension, the *aw_names* variable, and the *default_hier* relation. These references are to the workspace objects whose AW$ROLE property is set to CUBEDEF, AW_NAMES, and DEFAULT_HIER, respectively. The actual names of the workspace objects for most classes are typically similar to, but not identical to, their roles.

The sections that follow describe each object that has a role in the standard form conventions.

Implementation Class Objects

The objects in the implementation class provide the implementation for the logical objects in a given workspace. In general, they hold the data that users see as dimensions and measures. Implementation class objects differ from workspace to workspace. For example, one workspace might have measures called SALES and COST, while another workspace might have measures called BUDGET and ACTUAL.

The *cubedef*, *measuredef*, and *dimdef* objects implement cubes, measures, and dimensions respectively. In addition, each of these objects have implementation class helper objects. An overview of the objects is provided in the section "Logical Model and Workspace Objects" on page A-3.

The rest of this section describes each of the implementation class objects. Note that the examples in this section show the properties required by the standard form. If you examine a workspace that was created by Analytic Workspace Manager or the DBMS_AWM package, you might find some additional properties on various objects. These are not required for compliance with the standard form.

For information about the values that should be assigned to the properties, see Chapter 8.

To list all the objects that have a given role, limit the NAME dimension to all the objects that have that role and then report the values of the NAME dimension. For example, execute the following OLAP DML commands to list all the *cubedef* objects.

```
LIMIT name TO OBJ(PROPERTY 'AW$ROLE') EQ 'CUBEDEF'
REPORT name
```

```
NAME
--------------
UNITS_CUBE
PRICE_CUBE
```

Cube Objects

A cube is implemented by a *cubedef* dimension. It also has a *loopspec* composite.

Cubedef Dimension

A logical cube is implemented by a workspace dimension that has the value CUBEDEF in its AW$ROLE property. The values of a given *cubedef* dimension are the names of the logical dimensions of the cube.

A *cubedef* dimension has no parent, so its AW$PARENT_NAME property is set to NA. A logical cube is the parent of the measures that belong to it.

The following is a full description of a *cubedef* dimension called UNITS_CUBE.

```
FULLDSC units_cube

DEFINE UNITS_CUBE DIMENSION TEXT
LD IMPLEMENTATION UNITS_CUBE Cube
PROPERTY 'AGGMAPLIST' 'GLOBAL_AW.GLOBAL!UNITS_CUBE_AGGMAP_AWCREATEDDEFAULT_1'
PROPERTY 'AW$CLASS' 'IMPLEMENTATION'
PROPERTY 'AW$CREATEDBY' 'AW$CREATE'
PROPERTY 'AW$LASTMODIFIED' '03SEP03_15:28:35'
PROPERTY 'AW$LOADPRGS' 'GLOBAL_AW.GLOBAL!___GET.CUBE.DATA_UNITS_CUBE_1'
PROPERTY 'AW$LOGICAL_NAME' 'UNITS_CUBE'
PROPERTY 'AW$LOOPSPEC' 'GLOBAL_AW.GLOBAL!UNITS_CUBE_COMPOSITE'
PROPERTY 'AW$PARENT_NAME' NA
PROPERTY 'AW$ROLE' 'CUBEDEF'
```

The following is a report that shows the values of the UNITS_CUBE dimension. The values are the names of the *dimdef* dimensions that implement the cube's logical dimensions.

```
REPORT units_cube

UNITS_CUBE
--------------
CHANNEL
CUSTOMER
PRODUCT
TIME
```

Loopspec Composite

A logical cube has a *loopspec* composite, which facilitates efficient data access for the cube's measures. The *loopspec* composite is particularly useful when looping through sparse data is required. For information about composites, see the *Oracle OLAP DML Reference*.

Typically, the *loopspec* composite includes all the dimensions of the cube, except for any time dimension that might be present. The parent of a *loopspec* is the logical cube that it supports.

The following is a full description of a *loopspec* composite for the logical cube called `UNITS_CUBE`. The composite includes all the dimensions that are listed as values of the *cubedef* dimension, except for the `TIME` dimension.

```
FULLDSC units_cube_composite

DEFINE UNITS_CUBE_COMPOSITE COMPOSITE <CUSTOMER PRODUCT CHANNEL>
LD IMPLEMENTATION Composite for UNITS_CUBE cube
PROPERTY 'AW$CLASS' 'IMPLEMENTATION'
PROPERTY 'AW$CREATEDBY' 'AW$CREATE'
PROPERTY 'AW$LASTMODIFIED' '03SEP03_15:28:40'
PROPERTY 'AW$PARENT_NAME' 'UNITS_CUBE'
PROPERTY 'AW$ROLE' 'LOOPSPEC'
PROPERTY 'AW$STATE' 'CREATED'
```

The following is a report that shows the first nine values of this *loopspec* composite.

```
REPORT units_cube_composite

 CUSTOMER   PRODUCT    CHANNEL
---------- ---------- ----------
51         13         2
51         14         2
51         15         2
51         16         2
65         17         2
65         18         2
65         19         2
65         20         2
61         20         2
.          .          .
.          .          .
.          .          .
```

Measure Objects

A measure is implemented by a *measuredef* object. A measure also has a *compspec* aggmap, which provides aggregation rules for the measure.

Measuredef Object

A logical measure is implemented by a workspace object that has the value MEASUREDEF in its AW$ROLE property. The *measuredef* object can be a variable, formula, or relation.

The values of the *measuredef* object are the values of the logical measure, and its parent is the logical cube.

The following is a full description of a *measuredef* object for the logical measure called UNITS. The object is a formula that is dimensioned by the dimensions of the parent cube, which is called UNITS_CUBE. The formula includes fully qualified object names, but this type of specification is optional.

```
FULLDSC units

DEFINE UNITS FORMULA DECIMAL <TIME CUSTOMER PRODUCT CHANNEL>
LD IMPLEMENTATION UNITS Measure in UNITS_CUBE Cube
EQ aggregate( GLOBAL_AW.GLOBAL!UNITS_VARIABLE using
GLOBAL_AW.GLOBAL!UNITS_CUBE_AGGMAP_AWCREATEDDEFAULT_1)
PROPERTY 'AW$CLASS' -'IMPLEMENTATION'
PROPERTY 'AW$COMPSPEC' 'GLOBAL_AW.GLOBAL!UNITS_CUBE_AGGMAP_AWCREATEDDEFAULT_1'
PROPERTY 'AW$CREATEDBY' 'AW$CREATE'
PROPERTY 'AW$LASTMODIFIED' '03SEP03_15:28:41'
PROPERTY 'AW$LOGICAL_NAME' 'UNITS'
PROPERTY 'AW$PARENT_NAME' 'UNITS_CUBE'
PROPERTY 'AW$ROLE' 'MEASUREDEF'
PROPERTY 'AW$STATE' 'CREATED'
```

The AW$COMPSPEC property holds the name of the measure's *compspec* aggmap. If this property value is NA, then measure values that are not stored will not be calculated; they will be NA.

COMPSPEC Aggmap

A logical measure can have a *compspec* object, which is an aggmap that specifies the rules for calculating aggregates. For information about aggmaps, see the *Oracle OLAP DML Reference.*

The following is a full description of a *compspec* aggmap for the logical measure called UNITS. The aggmap includes fully qualified object names, but this type of specification is optional.

```
FULLDSC units_cube_aggmap_awcreateddefault_1

DEFINE UNITS_CUBE_AGGMAP_AWCREATEDDEFAULT_1 AGGMAP
LD IMPLEMENTATION Default aggmap created by dbms_awm.refresh_awcube for
UNITS_CUBE cube
AGGMAP
RELATION GLOBAL_AW.GLOBAL!CHANNEL_PARENTREL OPERATOR SUM PRECOMPUTE(NA)
RELATION GLOBAL_AW.GLOBAL!CUSTOMER_PARENTREL OPERATOR SUM PRECOMPUTE(NA)
RELATION GLOBAL_AW.GLOBAL!PRODUCT_PARENTREL OPERATOR SUM PRECOMPUTE(NA)
RELATION GLOBAL_AW.GLOBAL!TIME_PARENTREL OPERATOR SUM PRECOMPUTE(NA)
AGGINDEX NO
END
PROPERTY 'AW$CLASS' 'IMPLEMENTATION'
PROPERTY 'AW$CREATEDBY' 'AW$CREATE'
PROPERTY 'AW$LASTMODIFIED' '03SEP03_15:29:24'
PROPERTY 'AW$PARENT_NAME' 'UNITS_CUBE'
PROPERTY 'AW$ROLE' 'COMPSPEC'
PROPERTY 'AW$STATE' 'CREATED'
```

Dimension Objects

A dimension is implemented by a *dimdef* object. In addition, a dimension has one each of the following supporting objects:

- *Hierlist* dimension
- *Levellist* dimension
- *Member_levelrel* relation
- *Member_parentrel* relation
- *Hier_levels* valueset

Optionally, a dimension can have one or more *attrdef* objects.

For each of these objects, its AW$ROLE property records the object's function. For example, the AW$ROLE property of a *hierlist* dimension is set to HIERLIST. In addition, the AW$PARENT property for each of these objects contains the name of the logical dimension to which the object belongs.

If a dimension does not have a hierarchy, or it does not have levels, or it has neither, then these supporting objects exist but they are not populated.

Dimdef Dimension

A logical dimension is implemented by a workspace dimension that has the value DIMDEF in its AW$ROLE property. The values of a given *dimdef* dimension are the values of the logical dimension.

A *dimdef* dimension has no parent, so its AW$PARENT_NAME property is set to NA. The AW$TYPE property is set to TIME for time dimensions, and it is set to NA for all other dimensions.

The following is a full description of a *dimdef* dimension for the logical dimension called PRODUCT.

```
FULLDSC product

DEFINE PRODUCT DIMENSION TEXT
LD IMPLEMENTATION PRODUCT Dimension
PROPERTY 'AW$CLASS' 'IMPLEMENTATION'
PROPERTY 'AW$CREATEDBY' 'AW$CREATE'
PROPERTY 'AW$LASTMODIFIED' '03SEP03_15:27:42'
PROPERTY 'AW$LOGICAL_NAME' 'PRODUCT'
PROPERTY 'AW$PARENT_NAME' NA
PROPERTY 'AW$ROLE' 'DIMDEF'
PROPERTY 'AW$STATE' 'ACTIVE'
PROPERTY 'AW$TYPE' NA
```

The following is a report that shows sample values of this *dimdef* dimension from all the levels. This is an **embedded totals** dimension. In this example, the use of surrogate keys ensures uniqueness among the values from all levels. When surrogate keys are not used, another strategy must be used to insure uniqueness. For example, you can use the level as a prefix, such as ITEM.46 and FAMILY.7. The example includes an *attrdef* variable and *member_levelrel* relation to clarify the results.

```
LIMIT product TO '46'
LIMIT product ADD ANCESTORS USING product_parentrel
REPORT DOWN product W 25 <product_long_description product_levelrel>
```

```
ALL_LANGUAGES: AMERICAN_AMERICA
               -----------------PRODUCT_HIERLIST------------------
               ------------------PRODUCT_ROLLUP-------------------
PRODUCT        PRODUCT_LONG_DESCRIPTION      PRODUCT_LEVELREL
-------------- ------------------------ ------------------------
46             Standard Mouse            ITEM
7              Accessories               FAMILY
3              Software/Other            CLASS
1              Total Product             TOTAL_PRODUCT
```

Hierlist Dimension

A *hierlist* dimension lists the names of the hierarchies of its parent dimension. That is, the values of the *hierlist* dimension are the names of hierarchies, such as the CALENDAR and FISCAL hierarchies for a time dimension. The hierarchies do not have one-to-one implementations as workspace objects, so the names refer to logical hierarchies not to workspace objects.

The following is a full description of a *hierlist* dimension called TIME_HIERLIST.

```
DEFINE TIME_HIERLIST DIMENSION TEXT
LD IMPLEMENTATION List of Hierarchies for  TIME
PROPERTY 'AW$CLASS' 'IMPLEMENTATION'
PROPERTY 'AW$CREATEDBY' 'AW$CREATE'
PROPERTY 'AW$LASTMODIFIED' '03SEP03_15:27:47'
PROPERTY 'AW$PARENT_NAME' 'TIME'
PROPERTY 'AW$ROLE' 'HIERLIST'
PROPERTY 'AW$STATE' 'CREATED'
```

The following is a report that shows the values of this *hierlist* dimension. TIME has one hierarchy, which is named CALENDAR.

```
REPORT time_hierlist

TIME_HIERLIST
--------------
CALENDAR
```

Levellist Dimension

A *levellist* dimension lists the names of the levels of its parent dimension. That is, the values of the *levellist* dimension are the names of levels, such as the CITY, STATE, and COUNTRY levels for a geography dimension. The levels do not have one-to-one implementations as workspace objects, so the names refer to logical levels not to workspace objects. The logical level for each dimension value is identified in the dimension's MEMBER_LEVELREL relation.

The following is a full description of a *levellist* dimension called TIME_LEVELLIST.

```
FULLDSC time_levellist

DEFINE TIME_LEVELLIST DIMENSION TEXT
LD IMPLEMENTATION List of levels for TIME
PROPERTY 'AW$CLASS' 'IMPLEMENTATION'
PROPERTY 'AW$CREATEDBY' 'AW$CREATE'
PROPERTY 'AW$LASTMODIFIED' '03SEP03_15:27:47'
PROPERTY 'AW$LOGICAL_NAME' NA
PROPERTY 'AW$PARENT_NAME' 'TIME'
PROPERTY 'AW$ROLE' 'LEVELLIST'
PROPERTY 'AW$STATE' 'CREATED'
```

The following is a report that shows the values of this *levellist* dimension. The levels are YEAR, QUARTER, and MONTH.

```
REPORT time_levellist

TIME_LEVELLIST
--------------
YEAR
QUARTER
MONTH
```

Member_Levelrel Relation

A *member_levelrel* relation records the level for each value of the relation's parent dimension. For example, for a geography dimension, the *member_levelrel* relation might record the fact that BOSTON belongs to the CITY level and IOWA belongs to the STATE level.

The following is a full description of a *member_levelrel* relation called TIME_LEVELREL.

```
FULLDSC time_levelrel

DEFINE TIME_LEVELREL RELATION TIME_LEVELLIST <TIME>
LD IMPLEMENTATION Level of each dimension member for TIME
PROPERTY 'AW$CLASS' 'IMPLEMENTATION'
PROPERTY 'AW$CREATEDBY' 'AW$CREATE'
PROPERTY 'AW$LASTMODIFIED' '03SEP03_15:27:47'
PROPERTY 'AW$PARENT_NAME' 'TIME'
PROPERTY 'AW$ROLE' 'MEMBER_LEVELREL'
PROPERTY 'AW$STATE' 'CREATED'
```

The following is a report that shows sample values of this *member_levelrel* relation. The levels are MONTH, QUARTER, and YEAR.

```
LIMIT time TO '75'
LIMIT time ADD ANCESTORS USING time_parentrel
REPORT DOWN time W 15 time_levelrel

TIME            TIME_LEVELREL
--------------- ---------------
75              MONTH
83              QUARTER
85              YEAR
```

Member_Parentrel Relation

A *member_parentrel* relation records the parent dimension value for each value of the relation's parent dimension. For example, for a geography dimension, the *member_parentrel* relation might record the fact that the parent of BOSTON is MASSACHUSETTS, and the parent of IOWA is USA.

The following is a full description of a *member_parentrel* relation called TIME_PARENTREL.

```
FULLDSC time_parentrel

DEFINE TIME_PARENTREL RELATION TIME <TIME TIME_HIERLIST>
PROPERTY 'AW$CLASS' 'IMPLEMENTATION'
PROPERTY 'AW$CREATEDBY' 'AW$CREATE'
PROPERTY 'AW$LASTMODIFIED' '03SEP03_15:27:47'
PROPERTY 'AW$PARENT_NAME' 'TIME'
PROPERTY 'AW$ROLE' 'MEMBER_PARENTREL'
PROPERTY 'AW$STATE' 'CREATED'
```

The following is a report that shows the values of this *member_parentrel* relation. The parent of a given value can be different, depending on which hierarchy is being considered.

```
REPORT DOWN time W 20 time_parentrel

                ---TIME_PARENTREL---
                ---TIME_HIERLIST----
TIME                  CALENDAR
--------------- --------------------
75              83
83              85
85              NA
```

Hier_Levels Valueset

A *hier_levels* valueset lists the levels that are included in each hierarchy of the parent dimension.

The following is a full description of a *hier_levels* valueset called TIME_HIER_LEVELS.

```
FULLDSC time_hier_levels

DEFINE TIME_HIER_LEVELS VALUESET TIME_LEVELLIST <TIME_HIERLIST>
LD IMPLEMENTATION Ordered from Bottom to Top list of levels in a hierarchy for
TIME
PROPERTY 'AW$CLASS' 'IMPLEMENTATION'
PROPERTY 'AW$CREATEDBY' 'AW$CREATE'
PROPERTY 'AW$LASTMODIFIED' '03SEP03_15:27:47'
PROPERTY 'AW$PARENT_NAME' 'TIME'
PROPERTY 'AW$ROLE' 'HIER_LEVELS'
PROPERTY 'AW$STATE' 'CREATED'
```

The following command present the list of levels for each hierarchy, as recorded in this *hier_levels* valueset.

```
REPORT W 25 VALUES(time_hier_levels)

TIME_HIERLIST  VALUES(TIME_HIER_LEVELS)
-------------- -------------------------
CALENDAR       MONTH
               QUARTER
               YEAR
```

Attrdef Object

A logical attribute is implemented by a workspace object that has the value *attrdef* in its AW$ROLE property. The *attrdef* object can be a variable, formula, or relation. The values of the *attrdef* object are the values of the logical attribute, and its parent is the logical dimension to which it belongs.

The AW$TYPE property indicates whether Oracle OLAP has a special use for the attribute. Property values that indicate such a special use are DEFAULT_ORDER, END_DATE, TIME_SPAN, MEMBER_LONG_DESCRIPTION, MEMBER_SHORT_DESCRIPTION, and MEMBER_VISIBLE. If the value is USER or NA, then the attribute has no special meaning for Oracle OLAP.

An *attrdef* object must be dimensioned by its parent *dimdef* dimension. In addition, it can be dimensioned by the *hierlist* dimension or the ALL_LANGUAGES dimension, or both.

The following is a full description of an *attrdef* object called TIME_LONG_DESCRIPTION. This long description attribute is implemented as a variable.

```
FULLDSC time_long_description

DEFINE TIME_LONG_DESCRIPTION VARIABLE TEXT <TIME TIME_HIERLIST ALL_LANGUAGES>
LD IMPLEMENTATION LONG_DESCRIPTION Attribute for TIME Dimension
PROPERTY 'AW$CLASS' 'IMPLEMENTATION'
PROPERTY 'AW$CREATEDBY' 'AW$CREATE'
PROPERTY 'AW$LASTMODIFIED' '03SEP03_15:28:25'
PROPERTY 'AW$LOGICAL_NAME' 'LONG_DESCRIPTION'
PROPERTY 'AW$PARENT_NAME' 'TIME'
PROPERTY 'AW$ROLE' 'ATTRDEF'
PROPERTY 'AW$STATE' 'CREATED'
PROPERTY 'AW$TYPE' 'Long Description'
```

The following is a report that shows selected values of this *attrdef* object at each level.

```
LIMIT time TO time_levelrel EQ 'YEAR'
LIMIT time KEEP LAST 1
LIMIT time ADD DESCENDANTS USING time_parentrel
REPORT DOWN time W 25 time_long_description

ALL_LANGUAGES: AMERICAN_AMERICA
               --TIME_LONG_DESCRIPTION--
               ------TIME_HIERLIST------
TIME                    CALENDAR
--------------- -------------------------
119             2004
115             Q1-04
116             Q2-04
103             Jan-04
104             Feb-04
105             Mar-04
106             Apr-04
107             May-04
108             Jun-04
```

Catalogs Class Objects

Catalogs class objects hold information about the logical objects in the workspace. Catalog class objects include a list of all the cubes in the workspace, a list of all the measures in the workspace, a list of all the dimensions in the workspace, and other lists that can facilitate the work of various utilities. A given workspace has a single instance of each Catalog class object. DBMS_AWM creates these objects using the role as the name, so that the *all_languages* dimension is named ALL_LANGUAGES. For this reason, the names of objects in the CATALOGS class are shown here in capital letters to indicate actual names.

In this section, Catalogs class objects are discussed in the following groups:

- Lists of Objects
- Lists of Types, Roles, and Languages
- Lists of Cube and Dimension Objects
- Supporting Object Information

Lists of Objects

The Catalogs class includes a set of dimensions, each of which lists all the objects of a given kind. For example, the ALL_MEASURES dimension lists all the logical measures.

ALL_CUBES Dimension

The ALL_CUBES dimension lists the full names of all the logical cubes in the workspace. The following is a full description of an ALL_CUBES dimension.

```
FULLDSC all_cubes

DEFINE ALL_CUBES DIMENSION TEXT
LD CATALOGS List of all cubes in the aw
PROPERTY 'AW$CLASS' 'CATALOGS'
PROPERTY 'AW$CREATEDBY' 'AW$CREATE'
PROPERTY 'AW$LASTMODIFIED' '04DEC02_13:09:14'
PROPERTY 'AW$ROLE' 'ALL_CUBES'
PROPERTY 'AW$STATE' 'CREATED'
```

The following is a report of the values of this ALL_CUBES dimension.

```
REPORT W 30 all_cubes
```

```
ALL_CUBES
-----------------------------
GLOBAL_AW.UNITS_CUBE.CUBE
GLOBAL_AW.PRICE_CUBE.CUBE
```

ALL_MEASURES Dimension

The ALL_MEASURES dimension lists the full names of all the logical measures in the workspace.

A full description for this dimension is similar to those presented for the ALL_CUBES dimension in "ALL_CUBES Dimension" on page A-25. The following is a report of the values of an ALL_MEASURES dimension.

```
REPORT W 40 all_measures

ALL_MEASURES
----------------------------------------
GLOBAL_AW.UNITS_CUBE.UNITS.MEASURE
GLOBAL_AW.PRICE_CUBE.UNIT_COST.MEASURE
GLOBAL_AW.PRICE_CUBE.UNIT_PRICE.MEASURE
```

ALL_DIMENSIONS Dimension

The ALL_DIMENSIONS dimension lists the full names of all the logical dimensions in the workspace.

A full description for this dimension is similar to those presented for the ALL_CUBES dimension in "ALL_CUBES Dimension" on page A-25. The following is a report of the values of an ALL_DIMENSIONS dimension.

```
REPORT W 40 all_dimensions

ALL_DIMENSIONS
----------------------------------------
GLOBAL_AW.CHANNEL.DIMENSION
GLOBAL_AW.CUSTOMER.DIMENSION
GLOBAL_AW.PRODUCT.DIMENSION
GLOBAL_AW.TIME.DIMENSION
```

ALL_HIERARCHIES Dimension

The ALL_HIERARCHIES dimension lists the full names of all the hierarchies in the workspace.

A full description for this dimension is similar to those presented for the ALL_CUBES dimension in "ALL_CUBES Dimension" on page A-25. The following is a report of the values of an ALL_HIERARCHIES dimension.

```
REPORT W 45 all_hierarchies

ALL_HIERARCHIES
-------------------------------------------
GLOBAL_AW.CHANNEL.CHANNEL_ROLLUP.HIERARCHY
GLOBAL_AW.CHANNEL.AW$NONE.HIERARCHY
GLOBAL_AW.CUSTOMER.SHIPMENTS_ROLLUP.HIERARCHY
GLOBAL_AW.CUSTOMER.MARKET_ROLLUP.HIERARCHY
GLOBAL_AW.CUSTOMER.AW$NONE.HIERARCHY
GLOBAL_AW.PRODUCT.PRODUCT_ROLLUP.HIERARCHY
GLOBAL_AW.PRODUCT.AW$NONE.HIERARCHY
GLOBAL_AW.TIME.CALENDAR.HIERARCHY
GLOBAL_AW.TIME.AW$NONE.HIERARCHY
```

Hierarchies with a simple name of AW$NONE indicate that a dimension has no hierarchy.

ALL_LEVELS Dimension

The ALL_LEVELS dimension lists the full names of all the levels in the workspace.

A full description for this dimension is similar to those presented for the ALL_CUBES dimension in "ALL_CUBES Dimension" on page A-25. The following is a report of the values of an ALL_LEVELS dimension.

```
REPORT W 40 all_levels

ALL_LEVELS
----------------------------------------
GLOBAL_AW.CHANNEL.ALL_CHANNELS.LEVEL
GLOBAL_AW.CHANNEL.CHANNEL.LEVEL
GLOBAL_AW.CHANNEL.AW$NONE.LEVEL
GLOBAL_AW.CUSTOMER.ALL_CUSTOMERS.LEVEL
GLOBAL_AW.CUSTOMER.REGION.LEVEL
GLOBAL_AW.CUSTOMER.WAREHOUSE.LEVEL
GLOBAL_AW.CUSTOMER.TOTAL_MARKET.LEVEL
GLOBAL_AW.CUSTOMER.MARKET_SEGMENT.LEVEL
GLOBAL_AW.CUSTOMER.ACCOUNT.LEVEL
GLOBAL_AW.CUSTOMER.SHIP_TO.LEVEL
GLOBAL_AW.CUSTOMER.AW$NONE.LEVEL
GLOBAL_AW.PRODUCT.TOTAL_PRODUCT.LEVEL
GLOBAL_AW.PRODUCT.CLASS.LEVEL
```

```
GLOBAL_AW.PRODUCT.FAMILY.LEVEL
GLOBAL_AW.PRODUCT.ITEM.LEVEL
GLOBAL_AW.PRODUCT.AW$NONE.LEVEL
GLOBAL_AW.TIME.YEAR.LEVEL
GLOBAL_AW.TIME.QUARTER.LEVEL
GLOBAL_AW.TIME.MONTH.LEVEL
GLOBAL_AW.TIME.AW$NONE.LEVEL
```

ALL_ATTRIBUTES Dimension

The ALL_ATTRIBUTES dimension lists the full names of all the attributes in the workspace.

A full description for this dimension is similar to those presented for the ALL_CUBES dimension in "ALL_CUBES Dimension" on page A-25. The following is a report of the values of an ALL_ATTRIBUTES dimension.

```
REPORT W 50 all_attributes

ALL_ATTRIBUTES
--------------------------------------------------
GLOBAL_AW.CHANNEL.LONG_DESCRIPTION.ATTRIBUTE
GLOBAL_AW.CHANNEL.SHORT_DESCRIPTION.ATTRIBUTE
GLOBAL_AW.CUSTOMER.LONG_DESCRIPTION.ATTRIBUTE
GLOBAL_AW.CUSTOMER.SHORT_DESCRIPTION.ATTRIBUTE
GLOBAL_AW.PRODUCT.LONG_DESCRIPTION.ATTRIBUTE
GLOBAL_AW.PRODUCT.SHORT_DESCRIPTION.ATTRIBUTE
GLOBAL_AW.PRODUCT.PACKAGE.ATTRIBUTE
GLOBAL_AW.TIME.LONG_DESCRIPTION.ATTRIBUTE
GLOBAL_AW.TIME.SHORT_DESCRIPTION.ATTRIBUTE
GLOBAL_AW.TIME.END_DATE.ATTRIBUTE
GLOBAL_AW.TIME.TIME_SPAN.ATTRIBUTE
```

ALL_OBJECTS Dimension

The ALL_OBJECTS dimension lists the full names of all the logical objects in the workspace.

The following is a full description of an ALL_OBJECTS dimension.

```
FULLDSC all_objects

DEFINE ALL_OBJECTS DIMENSION CONCAT (ALL_DIMENSIONS ALL_CUBES ALL_MEASURES
ALL_HIERARCHIES ALL_LEVELS ALL_ATTRIBUTES)
LD CATALOGS List of all objects in the aw
PROPERTY 'AW$CLASS''CATALOGS'
PROPERTY 'AW$CREATEDBY' 'AW$CREATE'
```

```
PROPERTY 'AW$LASTMODIFIED' '04DEC02_13:07:35'
PROPERTY 'AW$ROLE' 'ALL_OBJECTS'
PROPERTY 'AW$STATE' 'CREATED'
```

ALL_OBJECTS is a concat dimension of the ALL_CUBES, ALL_MEASURES, ALL_HIERARCHIES, ALL_LEVELS, and ALL_ATTRIBUTES dimensions. Its dimension members are a concatenated list of the members of those dimensions, as shown by this example.

```
LIMIT all_cubes TO FIRST 2
LIMIT all_measures TO FIRST 2
LIMIT all_hierarchies TO FIRST 2
LIMIT all_levels TO FIRST 2
LIMIT all_attributes TO FIRST 2
LIMIT all_objects TO all_cubes
LIMIT all_objects ADD all_measures
LIMIT all_objects ADD all_hierarchies
LIMIT all_objects ADD all_levels
LIMIT all_objects ADD all_attributes
REPORT W 70 all_objects

ALL_OBJECTS
------------------------------------------------------------------------
<ALL_CUBES: GLOBAL_AW.UNITS_CUBE.CUBE>
<ALL_CUBES: GLOBAL_AW.PRICE_CUBE.CUBE>
<ALL_MEASURES: GLOBAL_AW.UNITS_CUBE.UNITS.MEASURE>
<ALL_MEASURES: GLOBAL_AW.PRICE_CUBE.UNIT_COST.MEASURE>
<ALL_HIERARCHIES: GLOBAL_AW.CHANNEL.CHANNEL_ROLLUP.HIERARCHY>
<ALL_HIERARCHIES: GLOBAL_AW.CHANNEL.AW$NONE.HIERARCHY>
<ALL_LEVELS: GLOBAL_AW.CHANNEL.ALL_CHANNELS.LEVEL>
<ALL_LEVELS: GLOBAL_AW.CHANNEL.CHANNEL.LEVEL>
<ALL_ATTRIBUTES: GLOBAL_AW.CHANNEL.LONG_DESCRIPTION.ATTRIBUTE>
<ALL_ATTRIBUTES: GLOBAL_AW.CHANNEL.SHORT_DESCRIPTION.ATTRIBUTE>
```

Lists of Types, Roles, and Languages

The Catalogs class includes dimensions that list types and roles that are supported by the current version of the standard form. In addition, there is a dimension that lists the languages supported by the current analytic workspace.

ALL_OBJTYPES Dimension

The ALL_OBJTYPES dimension lists all the object types that are supported in the current version of the standard form. The following report lists the types.

```
REPORT all_objtypes

ALL_OBJTYPES
--------------
CUBE
MEASURE
DIMENSION
LEVEL
HIERARCHY
ATTRIBUTE
```

ALL_DESCTYPES Dimension

The ALL_DESCTYPES dimension lists all the description types that are recognized in the current version of the standard form. The following report lists the types.

```
REPORT all_desctypes

ALL_DESCTYPES
--------------
SHORT
LONG
PLURAL
```

ALL_ATTRTYPES Dimension

The ALL_ATTRTYPES dimension lists all the attribute types that are recognized in the current version of the standard form. The following report lists the types.

```
REPORT W 40 all_attrtypes

ALL_ATTRTYPES
----------------------------------------
DEFAULT_ORDER
END_DATE
TIME_SPAN
MEMBER_LONG_DESCRIPTION
MEMBER_SHORT_DESCRIPTION
MEMBER_VISIBLE
USER
```

AW_ROLES Dimension

The AW_ROLES dimension lists all the roles that are recognized in the current version of the standard form. The following report lists the roles.

```
REPORT W 30 aw_roles

AW_ROLES
------------------------------
LANGUAGEDEF
ADTVIEWLIST
ADTLIST
ADTTBLLIST
ADTREL
ADTTBLREL
ADTLMTMAP
DIMDEF
MEMBER_CREATEDBY
LEVELLIST
MEMBER_LEVELREL
LEVEL_CREATEDBY
LEVELCOLLIST
LEVELCOLNUM
LEVELCOLMAP
HIERLIST
HIER_CREATEDBY
MEMBER_INHIER
MEMBER_PARENTREL
ATTRDEF
SRCCOMPOSITE
SRCLVLOWNER
SRCLVLTBL
SRCLVLCOL
SRCLVLPNTCOL
MEMBER_FAMILYREL
HIER_LEVELS
MEMBER_GID
ALL_LANGUAGES
ALL_DIMENSIONS
ALL_CUBES
ALL_MEASURES
ALL_HIERARCHIES
ALL_LEVELS
ALL_ATTRIBUTES
AW_ROLES
ALL_DESCTYPES
ALL_OBJTYPES
ALL_OBJECTS
AW_NAMES
AW_COMPSPECS
```

```
AW_LOOPSPECS
```

ALL_LANGUAGES Dimension

The ALL_LANGUAGES dimension lists all the languages that are implemented in the current analytic workspace. The following report lists the single language that is implemented in a sample workspace. Language names should follow Globalization Support standards.

```
REPORT W 30 all_languages

ALL_LANGUAGES
------------------------------
AMERICAN_AMERICA
```

Lists of Cube and Dimension Objects

The Catalogs class includes valuesets that list the measures in each cube, as well as the hierarchies, levels, and attributes in each dimension. These lists are specific to a given workspace.

CUBE_MEASURES Valueset

The CUBE_MEASURES valueset lists the measures that belong to each cube in the current analytic workspace. The valueset is dimensioned by ALL_CUBES, so that each cube has its own list. The following is a full description of a CUBE_MEASURES valueset in a sample workspace.

```
FULLDSC cube_measures

DEFINE CUBE_MEASURES VALUESET ALL_MEASURES <ALL_CUBES>
PROPERTY 'AW$CLASS' 'CATALOGS'
PROPERTY 'AW$CREATEDBY' 'AW$CREATE'
PROPERTY 'AW$LASTMODIFIED' '03SEP03_15:27:47'
PROPERTY 'AW$ROLE' 'CUBE_MEASURES'
PROPERTY 'AW$STATE' 'CREATED'
```

The following commands present the list of measures associated with each cube.

```
LCOLWIDTH=30             "Widen the label column
REPORT W 40 VALUES(cube_measures)

ALL_CUBES                              VALUES(CUBE_MEASURES)
------------------------------ ----------------------------------------
GLOBAL_AW.UNITS_CUBE.CUBE      GLOBAL_AW.UNITS_CUBE.UNITS.MEASURE
```

```
GLOBAL_AW.PRICE_CUBE.CUBE      GLOBAL_AW.PRICE_CUBE.UNIT_COST.MEASURE
                               GLOBAL_AW.PRICE_CUBE.UNIT_PRICE.MEASURE
```

DIM_HIERARCHIES Valueset

The DIM_HIERARCHIES valueset lists the hierarchies that belong to each dimension in the current analytic workspace. The valueset is dimensioned by ALL_DIMENSIONS, so that each dimension has its own list. The following commands present the list of hierarchies for each dimension.

```
REPORT W 45 VALUES(dim_hierarchies)

ALL_DIMENSIONS                          VALUES(DIM_HIERARCHIES)
------------------------------ ----------------------------------------
GLOBAL_AW.CHANNEL.DIMENSION    GLOBAL_AW.CHANNEL.CHANNEL_ROLLUP.HIERARCHY
GLOBAL_AW.CUSTOMER.DIMENSION   GLOBAL_AW.CUSTOMER.SHIPMENTS_ROLLUP.HIERARCHY
                               GLOBAL_AW.CUSTOMER.MARKET_ROLLUP.HIERARCHY
GLOBAL_AW.PRODUCT.DIMENSION    GLOBAL_AW.PRODUCT.PRODUCT_ROLLUP.HIERARCHY
GLOBAL_AW.TIME.DIMENSION       GLOBAL_AW.TIME.CALENDAR.HIERARCHY
```

DIM_LEVELS Valueset

The DIM_LEVELS valueset lists the levels that belong to each dimension in the current analytic workspace. The valueset is dimensioned by ALL_DIMENSIONS, so that each dimension has its own list. The following commands present the list of levels for each dimension.

```
REPORT W 45 VALUES(dim_levels)

ALL_DIMENSIONS                            VALUES(DIM_LEVELS)
------------------------------ ----------------------------------------
GLOBAL_AW.CHANNEL.DIMENSION    GLOBAL_AW.CHANNEL.ALL_CHANNELS.LEVEL
                               GLOBAL_AW.CHANNEL.CHANNEL.LEVEL
GLOBAL_AW.CUSTOMER.DIMENSION   GLOBAL_AW.CUSTOMER.ALL_CUSTOMERS.LEVEL
                               GLOBAL_AW.CUSTOMER.REGION.LEVEL
                               GLOBAL_AW.CUSTOMER.WAREHOUSE.LEVEL
                               GLOBAL_AW.CUSTOMER.TOTAL_MARKET.LEVEL
                               GLOBAL_AW.CUSTOMER.MARKET_SEGMENT.LEVEL
                               GLOBAL_AW.CUSTOMER.ACCOUNT.LEVEL
                               GLOBAL_AW.CUSTOMER.SHIP_TO.LEVEL
GLOBAL_AW.PRODUCT.DIMENSION    GLOBAL_AW.PRODUCT.TOTAL_PRODUCT.LEVEL
                               GLOBAL_AW.PRODUCT.CLASS.LEVEL
                               GLOBAL_AW.PRODUCT.FAMILY.LEVEL
                               GLOBAL_AW.PRODUCT.ITEM.LEVEL
GLOBAL_AW.TIME.DIMENSION       GLOBAL_AW.TIME.YEAR.LEVEL
```

```
                                GLOBAL_AW.TIME.QUARTER.LEVEL
                                GLOBAL_AW.TIME.MONTH.LEVEL
```

DIM_ATTRIBUTES Valueset

The DIM_ATTRIBUTES valueset lists the attributes that belong to each dimension in the current analytic workspace. The valueset is dimensioned by ALL_DIMENSIONS, so that each dimension has its own list. The following commands present the list of attributes for a dimension called TIME.

```
REPORT W 46 VALUES(dim_attributes)

ALL_DIMENSIONS                          VALUES(DIM_ATTRIBUTES)
------------------------------ ------------------------------------------
GLOBAL_AW.CHANNEL.DIMENSION    GLOBAL_AW.CHANNEL.LONG_DESCRIPTION.ATTRIBUTE
                               GLOBAL_AW.CHANNEL.SHORT_DESCRIPTION.ATTRIBUTE
GLOBAL_AW.CUSTOMER.DIMENSION   GLOBAL_AW.CUSTOMER.LONG_DESCRIPTION.ATTRIBUTE
                               GLOBAL_AW.CUSTOMER.SHORT_DESCRIPTION.ATTRIBUTE
GLOBAL_AW.PRODUCT.DIMENSION    GLOBAL_AW.PRODUCT.LONG_DESCRIPTION.ATTRIBUTE
                               GLOBAL_AW.PRODUCT.SHORT_DESCRIPTION.ATTRIBUTE
                               GLOBAL_AW.PRODUCT.PACKAGE.ATTRIBUTE
GLOBAL_AW.TIME.DIMENSION       GLOBAL_AW.TIME.LONG_DESCRIPTION.ATTRIBUTE
                               GLOBAL_AW.TIME.SHORT_DESCRIPTION.ATTRIBUTE
                               GLOBAL_AW.TIME.END_DATE.ATTRIBUTE
                               GLOBAL_AW.TIME.TIME_SPAN.ATTRIBUTE
```

Supporting Object Information

The Catalogs class includes variables and formulas that list the objects that support various other objects.

AW_NAMES Variable

The AW_NAMES variable is dimensioned by ALL_OBJECTS. It contains the name of the workspace object that implements each logical object. If no workspace object implements a given logical object, the value is NA.

The following is a full description of an AW_NAMES variable.

```
FULLDSC aw_names

DEFINE AW_NAMES VARIABLE TEXT <ALL_OBJECTS>
PROPERTY 'AW$CLASS' 'CATALOGS'
PROPERTY 'AW$CREATEDBY' 'AW$CREATE'
PROPERTY 'AW$LASTMODIFIED' '04DEC02_13:09:14'
PROPERTY 'AW$ROLE' 'AW_NAMES'
```

```
PROPERTY 'AW$STATE' 'CREATED'
```

AW_COMPSPECS Variable

The AW_COMPSPECS variable is dimensioned by ALL_DIMENSIONS. For each logical dimension, the AW_COMPSPECS variable contains the names of the AGGMAP objects that must be modified when the dimension is modified.

The following is a full description of an AW_COMPSPECS variable.

```
FULLDSC aw_compspecs

DEFINE AW_COMPSPECS VARIABLE TEXT <ALL_DIMENSIONS>
PROPERTY 'AW$CLASS' 'CATALOGS'
PROPERTY 'AW$CREATEDBY' 'AW$CREATE'
PROPERTY 'AW$LASTMODIFIED' '04DEC02_13:09:14'
PROPERTY 'AW$ROLE' 'AW_COMPSPECS'
PROPERTY 'AW$STATE' 'CREATED'
```

AW_LOOPSPECS Variable

The AW_LOOPSPECS variable is dimensioned by ALL_CUBES. It contains the name of the workspace composite for each cube.

The following is a full description of an AW_LOOPSPECS variable.

```
FULLDSC aw_loopspecs

DEFINE AW_LOOPSPECS VARIABLE TEXT <ALL_CUBES>
PROPERTY 'AW$CLASS' 'CATALOGS'
PROPERTY 'AW$CREATEDBY' 'AW$CREATE'
PROPERTY 'AW$LASTMODIFIED' '04DEC02_13:09:14'
PROPERTY 'AW$ROLE' 'AW_LOOPSPECS'
PROPERTY 'AW$STATE' 'CREATED'
```

Features Class Objects

Features class objects hold information about specific logical objects and the workspace objects that implement them. For example, one object stores the descriptions of all the logical objects, while another indicates whether the object is intended to be visible to the user.

ALL_DESCRIPTIONS Variable

The ALL_DESCRIPTIONS variable contains the short, long, and plural descriptions of various logical objects. For search convenience it is dimensioned by a composite.

The following is a full description of an ALL_DESCRIPTIONS variable.

```
FULLDSC all_descriptions

DEFINE ALL_DESCRIPTIONS VARIABLE TEXT <SPARSE <ALL_OBJECTS ALL_DESCTYPES ALL_LANGUAGES>>
LD FEATURES Descriptions for all objects
PROPERTY 'AW$CLASS' 'FEATURES'
PROPERTY 'AW$CREATEDBY' 'AW$CREATE'
PROPERTY 'AW$LASTMODIFIED' '04DEC02_13:09:14'
PROPERTY 'AW$ROLE' 'ALL_DESCRIPTIONS'
PROPERTY 'AW$STATE' 'CREATED'
```

The following report shows sample values for ALL_DESCRIPTIONS.

```
report w 30 down all_dimensions all_descriptions

ALL_LANGUAGES: AMERICAN_AMERICA
                               --------ALL_DESCRIPTIONS--------
                               ---------ALL_DESCTYPES----------
ALL_DIMENSIONS                   SHORT       LONG      PLURAL
------------------------------ ---------- ---------- ----------
GLOBAL_AW.CHANNEL.DIMENSION    Channel    NA         CHANNEL
GLOBAL_AW.CUSTOMER.DIMENSION   Customer   NA         CUSTOMER
GLOBAL_AW.PRODUCT.DIMENSION    Product    NA         PRODUCT
GLOBAL_AW.TIME.DIMENSION       Time       NA         TIME
```

ATTR_INHIER Variable

The ATTR_INHIER variable is a boolean variable that indicates whether a given attribute is associated with a given hierarchy. The variable is dimensioned by ALL_ATTRIBUTES and ALL_HIERARCHIES.

The following is a full description of an ATTR_INHIER variable.

```
FULLDSC attr_inhier

DEFINE ATTR_INHIER VARIABLE BOOLEAN <ALL_ATTRIBUTES ALL_HIERARCHIES>
LD FEATURES Indicates if each attribute participates in each hierarchy
PROPERTY 'AW$CLASS' 'FEATURES'
PROPERTY 'AW$CREATEDBY' 'AW$CREATE'
PROPERTY 'AW$LASTMODIFIED' '04DEC02_13:09:14'
PROPERTY 'AW$ROLE' 'ATTR_INHIER'
PROPERTY 'AW$STATE' 'CREATED'
```

DEFAULT_HIER Relation

The DEFAULT_HIER relation records the full name of the default hierarchy for each dimension. The base dimension for the relation is ALL_DIMENSIONS.

The following is a full description of a DEFAULT_HIER relation.

```
FULLDSC default_hier

DEFINE DEFAULT_HIER RELATION ALL_HIERARCHIES <ALL_DIMENSIONS>
LD FEATURES Default hierarchy for each dimension
PROPERTY 'AW$CLASS' 'FEATURES'
PROPERTY 'AW$CREATEDBY' 'AW$CREATE'
PROPERTY 'AW$LASTMODIFIED' '04DEC02_13:09:14'
PROPERTY 'AW$ROLE' 'DEFAULT_HIER'
PROPERTY 'AW$STATE' 'CREATED'
```

The following report shows the default hierarchy for each dimension.

```
REPORT W 45 default_hier

ALL_DIMENSIONS                          DEFAULT_HIER
------------------------------ ----------------------------------------------
GLOBAL_AW.CHANNEL.DIMENSION    GLOBAL_AW.CHANNEL.CHANNEL_ROLLUP.HIERARCHY
GLOBAL_AW.CUSTOMER.DIMENSION   GLOBAL_AW.CUSTOMER.SHIPMENTS_ROLLUP.HIERARCHY
GLOBAL_AW.PRODUCT.DIMENSION    GLOBAL_AW.PRODUCT.PRODUCT_ROLLUP.HIERARCHY
GLOBAL_AW.TIME.DIMENSION       GLOBAL_AW.TIME.CALENDAR.HIERARCHY
```

VISIBLE Variable

The VISIBLE variable is a boolean that indicates whether the Oracle OLAP enabler utilities should expose or ignore the objects that are registered. The variable is dimensioned by ALL_OBJECTS so that each object has its own setting.

The following is a full description of a VISIBLE variable.

```
FULLDSC visible

DEFINE VISIBLE VARIABLE BOOLEAN <ALL_OBJECTS>
LD FEATURES Is the object visible
PROPERTY 'AW$CLASS' 'FEATURES'
PROPERTY 'AW$CREATEDBY' 'AW$CREATE'
PROPERTY 'AW$LASTMODIFIED' '04DEC02_13:09:14'
PROPERTY 'AW$ROLE' 'VISIBLE'
PROPERTY 'AW$STATE' 'CREATED'
```

Member_Inhier Variable

The *member_inhier* variable is a boolean variable that indicates whether a given member of a dimension is in a given hierarchy. There is one of these variables for each dimension in the workspace, and that dimension is the variable's parent.

The following is a full description of a *member_inhier* variable for the `TIME` dimension.

```
FULLDSC time_inhier

DEFINE TIME_INHIER VARIABLE BOOLEAN <TIME TIME_HIERLIST>
LD FEATURES Indicator of whether each dimension member participates in a
hierarchy for TIME
PROPERTY 'AW$CLASS' 'FEATURES'
PROPERTY 'AW$CREATEDBY' 'AW$CREATE'
PROPERTY 'AW$LASTMODIFIED' '03SEP03_15:27:47'
PROPERTY 'AW$PARENT_NAME' 'TIME'
PROPERTY 'AW$ROLE' 'MEMBER_INHIER'
PROPERTY 'AW$STATE' 'CREATED'
```

Member_Createdby Variable

The *member_createdby* variable records the entity that created each member of a given dimension. There is one of these variables for each dimension in the workspace, and that dimension is the variable's parent.

The following is a full description of a *member_createdby* variable for a dimension called `TIME`.

```
FULLDSC time_createdby

DEFINE TIME_CREATEDBY VARIABLE TEXT <TIME>
LD FEATURES Creator of each dimension member for TIME
PROPERTY 'AW$CLASS' 'FEATURES'
PROPERTY 'AW$CREATEDBY' 'AW$CREATE'
PROPERTY 'AW$LASTMODIFIED' '03SEP03_15:27:47'
PROPERTY 'AW$PARENT_NAME' 'TIME'
PROPERTY 'AW$ROLE' 'MEMBER_CREATEDBY'
PROPERTY 'AW$STATE' 'CREATED'
```

Member_Familyrel Relation

The *member_familyrel* relation records the ancestors of a given member of a dimension. There is one of these relations for each dimension in the workspace, and that dimension is the variable's parent. These relations are for internal use.

The following is a full description of a *member_familyrel* relation for the TIME dimension.

```
FULLDSC time_familyrel

DEFINE TIME_FAMILYREL RELATION TIME <TIME TIME_LEVELLIST TIME_HIERLIST>
LD FEATURES Family/Ancestry structure for  TIME
PROPERTY 'AW$CLASS' 'FEATURES'
PROPERTY 'AW$CREATEDBY' 'AW$CREATE'
PROPERTY 'AW$LASTMODIFIED' '03SEP03_15:27:47'
PROPERTY 'AW$PARENT_NAME' 'TIME'
PROPERTY 'AW$ROLE' 'MEMBER_FAMILYREL'
PROPERTY 'AW$STATE' 'CREATED'
```

Member_Gid Variable

The *member_gid* variable records the level depth of a given member of a dimension, within a given hierarchy. There is one of these relations for each dimension in the workspace, and that dimension is the variable's parent. These relations are for internal use.

The following is a full description of a *member_gid* relation for the TIME dimension.

```
FULLDSC time_gid

DEFINE TIME_GID VARIABLE INTEGER <TIME TIME_HIERLIST>
LD FEATURES Grouping id value for TIME
PROPERTY 'AW$CLASS' 'FEATURES'
PROPERTY 'AW$CREATEDBY' 'AW$CREATE'
PROPERTY 'AW$LASTMODIFIED' '03SEP03_15:27:47'
PROPERTY 'AW$PARENT_NAME' 'TIME'
PROPERTY 'AW$ROLE' 'MEMBER_GID'
PROPERTY 'AW$STATE' 'CREATED'
```

OBJ_CREATEDBY Variable

The OBJ_CREATEDBY variable records the entity that created each object that is registered in the standard form. The variable is dimensioned by ALL_OBJECTS.

The following is a full description of the OBJ_CREATEDBY variable.

```
FULLDSC obj_createdby

DEFINE OBJ_CREATEDBY VARIABLE TEXT <ALL_OBJECTS>
LD FEATURES Creator of each object
PROPERTY 'AW$CLASS' 'FEATURES'
PROPERTY 'AW$CREATEDBY' 'AW$CREATE'
PROPERTY 'AW$LASTMODIFIED' '04DEC02_13:09:14'
PROPERTY 'AW$ROLE' 'OBJ_CREATEDBY'
PROPERTY 'AW$STATE' 'CREATED'
```

OBJ_STATE Variable

The OBJ_STATE variable records the state for each registered object in the standard form. The variable is dimensioned by ALL_OBJECTS. The value for each object is either UNDER_CONSTRUCTION or ACTIVE.

The following is a full description of the OBJ_CREATEDBY variable.

```
FULLDSC obj_state

DEFINE OBJ_STATE VARIABLE TEXT <ALL_OBJECTS>
LD FEATURES State of each object
PROPERTY 'AW$CLASS' 'FEATURES'
PROPERTY 'AW$CREATEDBY' 'AW$CREATE'
PROPERTY 'AW$LASTMODIFIED' '04DEC02_13:09:14'
PROPERTY 'AW$ROLE' 'OBJ_STATE'
PROPERTY 'AW$STATE' 'CREATED'
```

VERSION Variable

The VERSION variable records the version number of the standard form convention under which the analytic workspace is being managed.

Extensions Class Objects

Extensions class objects are defined and maintained by the Oracle OLAP utilities. They are proprietary extensions to the standard form, and there is no commitment on the part of Oracle to maintain them from release to release.

Do not define, modify, or depend on objects in the extensions class.

B

Upgrading From Express Server

This appendix provides upgrade instructions and identifies some of the major differences between Oracle Express Server 6.3 and Oracle OLAP. It is intended to provide a frame of reference to help you understand the material presented in this guide.

This appendix includes the following topics:

Administration

Oracle OLAP is installed as an option in Oracle Enterprise Edition, and it is now integrated with Oracle Database. While Express Server runs in a service environment, Oracle OLAP runs within the Oracle kernel.

In Oracle, the term **database** refers only to the relational database. An Express database is now called an **analytic workspace**. In Oracle OLAP, an analytic workspace can be used either as a transient data cache or as a persistent data repository. A persistent analytic workspace is stored as a LOB in a relational table, which in turn is stored in a tablespace. There are no "`.db`" files.

The administrative tasks for Oracle OLAP are merged with the database tool set.

Management Tools

Oracle Enterprise Manager encompasses the tools for administering Oracle databases, providing a common user interface across all platforms. Performance data for OLAP can be collected in system tables the same as any other Oracle database performance statistics. Oracle Enterprise Manager provides a graphical interface to SQL. Because OLAP now runs within the Oracle Database kernel, many of the basic administrative tasks (such as starting, stopping, and configuring the process) are subsumed into database management.

Analytic Workspace Manager is the tool for creating and managing analytic workspaces.

OLAP Instance Manager, `oesmgr`, and `oescmd` are not available.

Authentication of Users

Oracle OLAP does not use operating system identities, except for the installation user under whose identity Oracle Database is installed. You can delete other operating system identities created for use by Express Server (such as the DBA user, the Initialize user, the Default user, and individual user names) if they have no other purpose.

All authentication is performed by Oracle Database. Applications must always present credentials before opening a session, and those credentials must match a user name and password stored in the relational database. Before users can access Oracle OLAP, you must define user names and passwords in the database.

For users to access operating system files, they must have access rights to a directory object that is mapped to the physical directory path. This access is granted either to an individual user ID or to a database role.

> **See Also:** Chapter 12 for more information about OLAP administration tasks

Data Transfer

An Oracle OLAP session is always connected to the database. You do not open a connection with the database as a separate or optional step.

You can copy data between an analytic workspace objects (such as variables and dimensions) and relational tables in the following ways:

- A PL/SQL package named DBMS_AWM provides procedures for creating an analytic workspace from relational tables. Analytic Workspace Manager provides a graphical interface to this package.
- The OLAP DML SQL command fetches data into dimensions and variables for further manipulation. A new SQL IMPORT command facilitates bulk data transfer from relational tables into the analytic workspace, and a new SQL INSERT DIRECT command facilitates data transfer from the analytic workspace into relational tables.
- Using SQL table functions, it is now possible for a SQL-based application to manipulate and extract data from an analytic workspace. Express Server did not permit a data transfer to be initiated externally. Analytic Workspace Manager provides a graphical interface to the OLAP_TABLE function.

ODBC is not available, and thus access to third-party databases is not available directly from Oracle OLAP.

Oracle Express Relational Access Administrator and Oracle Express Relational Access Manager are not available.

Localization

The Express Server language support has been replaced by Oracle Globalization Technology, which provides more extensive localization support and is much easier to administer than the localization features of Express Server. Oracle Database and Oracle OLAP use the same character set, which is selected during installation.

If you are upgrading Express databases that use translation tables, then you can delete those tables because they are not needed by Oracle OLAP. Likewise, you should check your Express programs for use of obsolete commands and keywords that supported translation tables.

Support for Globalization Technology has been added to the OLAP DML. These options enable an application to query the current localization settings and override the behaviors controlled by the default language and territory.

Table B–1 identifies the Unicode character sets available in Oracle that are equivalent to the Express Server character sets. If you plan to import Express databases or to use Oracle OLAP to access multibyte data in external files, then you might find this information helpful in identifying an appropriate database character set. Note that the Express CHARSET option is now obsolete.

Table B–1 Multibyte Character Set Equivalents

Express Server	Unicode Character Set
EUC	JA16EUC
SHIFTJIS	JA16SJIS
HANGEUL	KO16KSC5601
SCHINESE	ZHS16GBK
TCHINESE	ZHT16BIG5

Applications Support

Oracle OLAP enables applications to access its multidimensional data directly through either a Java API or SQL. Express SPL programs can be executed using either programming method. Be sure to review all SPL programs to remove commands that are no longer available and to take advantage of new functionality.

Analytic Workspace Manager provides wizards for creating a **database standard form** analytic workspace from relational tables, aggregating the data, and enabling the workspace for access by the BI Beans or Oracle Discoverer. Enablement involves generating relational views of data stored in an analytic workspace, and creating metadata for those views that is in the appropriate type for the application.

You cannot run Windows C++, HTML, or Java applications that were developed for use with Express Server.

> **See Also:** Chapter 6 for methods of creating standard form analytic workspaces from data in relational tables

Programming Environment

Applications for Oracle OLAP can be developed in Java using the BI Beans. SQL-based applications can access OLAP data through views or manipulate it directly through the `OLAP_TABLE` functions.

OLAP Worksheet provides an interactive environment for developing stored procedures in either the OLAP DML or SQL. The PL/SQL `DBMS_AW` procedure executes OLAP DML commands from a SQL environment.

You cannot connect to Oracle OLAP using Express Administrator, Personal Express, or the Express Connection Utility.

See Also:

- Chapter 4 for information about the BI Beans
- Chapter 9 for methods of executing OLAP DML commands

Communications

Oracle OLAP provides communications through Oracle Call Interface (OCI) and Java Database Connectivity (JDBC).

OLAP Worksheet uses XCA for communication with the analytic workspace. However, XCA is not supported for user-developed applications and may produce unexpected results.

SNAPI is no longer available. Session sharing is not supported.

Metadata

The BI Beans can query data that is stored either in an analytic workspace or in relational tables. The database administrator defines OLAP Catalog metadata for both types of data source. The metadata is stored in tables and views.

Database standard form is a type of metadata stored in analytic workspaces for use by the server tools provided in Analytic Workspace Manager. This metadata is stored in properties on workspace objects and in catalogs, which are implemented as special dimensions, variables, and valuesets.

Oracle Express Administrator is not available in Oracle OLAP, and the Oracle Express Objects metadata that it generated is not used by the BI Beans.

See Also:

- Chapter 5 for information about the OLAP Catalog.
- Chapter 8 for information about database standard form.

Programming Language Changes

Numerous changes have been made to the Express Stored Procedure Language (now called the OLAP Data Manipulation Language or OLAP DML).

New Commands

Support in the following areas has been added to the OLAP DML:

Allocation
Dynamic model execution
Bulk data transfers between analytic workspaces and relational tables
Byte manipulation functions
Data conversion functions
New data types

Obsolete Commands

Support in the following areas has been dropped:

EXTCALL
ODBC
SNAPI
XCA
Operating system commands

Conjoint dimensions and the ROLLUP command are still available, but composite dimensions and aggmaps are strongly recommended instead, because they are easier to manage and perform better.

> **See Also:** *OLAP DML Reference* for comprehensive lists of new, obsolete, and significantly revised commands

UPDATE and COMMIT

The UPDATE command moves analytic workspace changes from a temporary tablespace to a permanent tablespace. Your changes are not saved permanently until you execute a COMMIT command, either from your Oracle OLAP session or from SQL. A COMMIT writes the permanent tablespace to disk.

Changes that have not been moved to the permanent tablespace are not committed. If you issue a COMMIT without first updating your analytic workspace, then no changes to the analytic workspace that you made after your last UPDATE are committed to disk.

The COMMIT command executes a SQL COMMIT command. All changes made during your session are committed, whether they were made through Oracle OLAP or through another form of access (such as SQL) to the database.

Converting Oracle Express Databases to Standard Form

EIF files are used to transfer the contents of an analytic workspace from one database to another and to upgrade from an Express database. You can create an analytic workspace from an Express database simply by using EIF files to transfer the objects.

The more complex task is to create an analytic workspace in database standard form, so that you can use the current generation of Oracle OLAP tools. You may be able to leverage your investment in Express metadata to create standard form metadata. Otherwise, you must define a new logical metadata model.

Who Should Use CREATE_DB_STDFORM

If your Express database contains Oracle Express Objects metadata (that is, it was created by Oracle Express Administrator), then you can use a conversion program named `CREATE_DB_STDFORM`. Without Oracle Express Objects metadata, `CREATE_DB_STDFORM` cannot generate sufficient standard form metadata for the OLAP tools to work.

Especially if your source data is in flat files, then use `CREATE_DB_STDFORM` if possible. There are no tools currently available for creating a standard form analytic workspace directly from flat files.

If your source data is in tables or views, then you have a choice of using `CREATE_DB_STDFORM` to convert an Express database, or using other tools to create an analytic workspace directly from the source data. `CREATE_DB_STDFORM` enables you to use your Oracle Express Objects metadata instead of redefining the logical model in the OLAP Catalog. However, you must perform other steps manually, as described in "What CREATE_DB_STDFORM Does Not Do For You" on page B-8. You can choose which method best suits your needs.

Table B–2 identifies the upgrade options.

Table B–2 Choosing an Upgrade Path for Express Databases

If you have Oracle Express Objects metadata...	And your source data is located in...	THEN create a standard form analytic workspace using...
Yes	Tables or views	`CREATE_DB_STDFORM` or one of the methods described in Chapter 6.
Yes	Flat files	`CREATE_DB_STDFORM`.

Table B–2 (Cont.) Choosing an Upgrade Path for Express Databases

If you have Oracle Express Objects metadata...	And your source data is located in...	THEN create a standard form analytic workspace using...
No	Tables or views	One of the methods described in Chapter 6.
No	Flat files	Oracle Warehouse Builder as described in Chapter 6, or the method described in Chapter 11.

What CREATE_DB_STDFORM Does For You

`CREATE_DB_STDFORM` enables you to start using the BI Beans against your data in a matter of minutes. The conversion step from Oracle Express Objects metadata to database standard form metadata involves running a single program. You can then enable the analytic workspace for the BI Beans using a dialog in Analytic Workspace Manager. The entire process, from importing the EIF file to querying views of the analytic workspace using a BI Beans application, is very quick and fully automated.

If you load data only at the base level, then you can use the Aggregation wizards in Analytic Workspace Manager to create and deploy an aggregation plan. This method of aggregation is faster and more flexible than the `ROLLUP` command.

What CREATE_DB_STDFORM Does Not Do For You

The conversion process circumvents the usual first step in creating an analytic workspace: developing a logical data model in the OLAP Catalog and mapping the logical objects to the data source. In this usual scenario, if you want to modify the logical model, you modify the OLAP Catalog; the tools make the appropriate changes to the standard form catalogs by refreshing them from the OLAP Catalog. This maintenance process is not available to analytic workspaces converted by `CREATE_DB_STDFORM`. Thus, you must do the following tasks manually:

- If you want to perform time-based analysis on your data, you must identify all time dimensions and populate end date and time span attributes before using `CREATE_DB_STDFORM`. A sample program is provided in this appendix.
- Your analytic workspace may contain programs with references to obsolete commands. You must revise them. You may also want to use some of the new features. For example, you can handle sparse data with composites (instead of conjoints) if you are not doing so already. You must define new variables and

copy the data from the old variables (or reload it from the data source) to make this change.

- You cannot use the Refresh Wizard in Analytic Workspace Manager to copy new data into a converted analytic workspace. Instead, you must modify the load programs or create new ones, and run them manually.
- You must make any changes to the standard form metadata manually using the `MAINTAIN` command and qualified data references.

Converting From Oracle Express Objects Metadata

The Oracle Express Objects conversion tool operates on an analytic workspace. It uses the Oracle Express Objects metadata to identify the roles of various objects, and then does the following:

- Populates existing objects with the appropriate standard form properties. For example, the Oracle Express Objects language dimension is given the `AW$ROLE` value of `ALL_LANGUAGES`.
- Creates new standard form objects with the dimensions and properties required by standard form, and copies the data from existing objects into it. For example, standard form attributes are dimensioned by the *hierdim* dimension and Oracle Express Objects attributes are not. In an `XADEMO` analytic workspace, the conversion tool creates a variable named `CHANNEL_LONG_DESCRIPTION` dimensioned by `CHANNEL`, `C0.HIERDIM`, and `_XA_LANGDIM`, and populates it with values from `C0.LONGLABEL`.
- Creates and populates standard form metadata objects, such as the standard form catalogs, *member_gid* and *member_inhier* variables, and *member_familyrel* and *member_levelrel* relations. For descriptions of these standard form objects, refer to Appendix A.

The conversion tool adds standard form objects and properties; it does not delete any Oracle Express Objects objects or properties. You can delete them manually if you wish.

The BI Beans requires a level-sorted Time dimension with period end dates and time span attributes in order to support time-based analysis.

CREATE_DB_STDFORM Syntax

The `CREATE_DB_STDFORM` program runs the Oracle Express Objects conversion tool. It has this syntax:

```
CREATE_DB_STDFORM(aw, [mode], [debug], [directory], [filename], [metacheck])
```

Where:

aw is the name of the analytic workspace (TEXT)

mode is the attachment mode (RO | RW | RWX)

debug controls whether the debugger runs (YES | NO)

directory is a database directory where the debug file is written (TEXT)

filename is the name of the debug file (TEXT)

metacheck controls whether a metadata check precedes the conversion (YES | NO)

For example, the following command attaches XADEMO in read/write mode, verifies that the Oracle Express Objects metadata is complete, converts the analytic workspace to standard form, and sends status messages to the screen:

```
CALL CREATE_DB_STDFORM('xademo')
```

The next command attaches XADEMO in read/write exclusive mode and redirects the status messages to a file named xademo.log in a database directory named xademo_dir. It also performs the metadata check.

```
CALL CREATE_DB_STDFORM('xademo', 'rwx', yes, 'xademo_dir', 'xademo.log')
```

Procedure: Converting From Oracle Express Objects to Standard Form

Most of the steps for converting to standard form (such as creating a new analytic workspace and importing the EIF file) can be done using the Object View in Analytic Workspace Manager. However, this procedure uses the command-line interface provided by OLAP Worksheet, on the basis that users making this conversion are already familiar with OLAP DML commands.

Follow these steps to use the Oracle Express Objects metadata conversion tool to create a standard form analytic workspace.

1. Create an EIF file from your Oracle Express Objects database, and copy the file to a physical directory that is mapped to a database directory.

   For information about database directories, refer to Permitting Access to External Files on page 12-10.

2. Open Analytic Workspace Manager and attach to Oracle Database, as described in "Introduction to Analytic Workspace Manager" on page 6-3.

3. From the Tools menu, choose OLAP Worksheet.

OLAP Worksheet opens in a separate window. For information about using OLAP Worksheet, refer to "Using OLAP Worksheet to Execute OLAP DML" on page 9-4.

4. Create a new analytic workspace from the EIF file using commands like these:

```
AW CREATE aw
IMPORT ALL FROM EIF FILE 'directory/filename.eif' DATA DFNS
UPDATE
COMMIT
```

5. Identify the time dimensions:

```
LIMIT name TO OBJ(PROPERTY 'DIMTYPE') EQ 1
REPORT name
```

6. Identify the hierarchy dimension for each time dimension:

```
SHOW OBJ(PROPERTY 'HIERDIM' timedim)
```

Note: The Oracle Express Objects metadata identifies all of the objects that support hierarchies and levels for a dimension. You can use the FULLDSC command to see all of the properties of a dimension, or use the OBJ function as shown here to obtain the value of particular properties, such as HIERDIM, LEVELDIM, and LEVELREL.

7. Create date and time span attributes for each dimension.

```
DEFINE TIME_TIME_SPAN VARIABLE INTEGER <timedim hierdim>
PROPERTY 'USERDATA' FALSE

DEFINE TIME_END_DATE VARIABLE DATE <timedim hierdim>
PROPERTY 'USERDATA' FALSE
```

8. Populate the end date and time span attributes, as described in "Populating Time Attributes" on page B-12.

9. Set properties on the Time dimension:

```
CONSIDER timedim
PROPERTY 'END_DATE' attribute_name
PROPERTY 'TIME_SPAN' attribute_name
```

The END_DATE and TIME_SPAN values (*attribute_name*) identify the names of the variables that you just created.

10. Run the conversion tool with a command like this:

```
CALL CREATE_DB_STDFORM('aw')
```

Refer to the syntax description in "CREATE_DB_STDFORM Syntax" on page B-9.

11. After the conversion tool completes successfully, save the changes:

```
UPDATE
COMMIT
```

You now have a standard form analytic workspace.

12. Enable the workspace for the BI Beans. Refer to "Enabling an Analytic Workspace for an Application" on page 6-24.

You can do this step now or after you have completed the other steps in this procedure.

13. To refresh the analytic workspace with new data, revise and run the data loader programs, as described in "Revising the Load Programs" on page B-13.

Populating Time Attributes

A standard form Time dimension has the following characteristics:

- Dimension members are sorted chronologically within level.
- The `AW$TYPE` property has a value of `'Time'`.
- Period end date and time span attributes are defined and populated.

The conversion process sets the `AW$TYPE` property, defines standard form attributes for period end dates and time span, and registers this information in the standard form catalogs. It does not change the order of the Time dimension members nor populate the attributes.

Sorting Time Dimension Members

If the Time members are not already sorted in chronological order within levels, then use a program like the one shown in "Sorting Dimension Members" on page 11-15 to sort them correctly. This topic assumes that your analytic workspace contains an **embedded total** dimension for time periods.

Creating and Populating End Date and Time Span Attributes

The end date and time span attributes are variables dimensioned by Time and Time's *hierdim* dimension. The end date variable must be defined with a `DATE` data type.

The method that you use to populate the end date and time span attributes depends on your data source and the format of your Time dimension members. If the information is available from your original data source (that is, the source from which you populated the Express database), then you can load the information using a file reader program like those discussed in "Reading Flat Files" on page 11-4. Otherwise, you must derive the information from the dimension members or their descriptions. An example of this method is shown in "Populating the XADEMO Time Attributes" on page B-17.

Setting Properties on Time Objects

You must define and set the following properties before running `CREATE_DB_STDFORM`:

- On the Time dimension, set the `END_DATE` and `TIME_SPAN` properties to the object names for these attributes. The `DIMTYPE` property should be set to `1` already.
- On the end-date and time-span attributes, set the `USERDATA` property to `FALSE`.

Revising the Load Programs

The Refresh wizard in Analytic Workspace Manager only operates on analytic workspaces created using `DBMS_AWM` procedures, as described in Chapter 6. When you create an analytic workspace using `CREATE_DB_STDFORM`, you circumvent the mechanisms that provide the Refresh wizard with the information it needs to acquire new data. You must refresh your analytic workspace manually using OLAP DML programs.

Your analytic workspace probably contains programs generated by Express Administrator for refreshing your Express database. You can begin by modifying these programs for use in your analytic workspace; they are unusable in their current state.

Delete the following code from your load programs:

- Calls to EDDE.MSG. This program displayed Express error messages in the Administrator graphical interface, and deleting calls to it does not affect the operation of your program.
- Calls to EDDE.HIERMNT. This program managed the metadata associated with dimension hierarchies. It is not available for use in analytic workspaces, nor is any of the information about your data that was stored in an XPDDDATA database. You must manage any changes to the standard form metadata manually.
- Code to establish a connection with Oracle. Since the analytic workspace is part of Oracle Database, a connection is always open.

The load programs only refresh the dimensions and measures. They do not refresh the dimension attributes, the hierarchy and level objects, or the standard form catalogs. Refer to Chapter 11 for information about writing load programs for standard form objects associated with dimensions. Refer to Appendix A for information about the standard form catalogs.

Example: Converting the XADEMO Database to Standard Form

This example uses an EIF file that contains objects and Oracle Express Objects metadata from an Express database named XADEMO. If you are converting an Express database, you are probably already familiar with XADEMO.

Creating a Standard Form XADEMO Analytic Workspace

Suppose that an EIF file named xademo.eif is located in a system directory named \users\oracle\xademo_files. Take these steps to create a standard form analytic workspace from this file.

1. Log in to your Oracle database as the SYSTEM user and create the XADEMO user, permanent and temporary tablespaces, and a database directory for access to the EIF file.

```
CREATE TABLESPACE olapdata DATAFILE '$ORACLE_HOME/oradata/olapdata.dbf'
   SIZE 5M REUSE AUTOEXTEND ON NEXT 5M MAXSIZE UNLIMITED
   EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO;

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

CREATE USER xademo IDENTIFIED BY 'xademo'
```

```
   DEFAULT TABLESPACE olapdata
   TEMPORARY TABLESPACE olaptmp
   QUOTA UNLIMITED ON olapdata
   QUOTA UNLIMITED ON olaptmp
   ACCOUNT UNLOCK;

CREATE DIRECTORY xademo_dir as '/users/oracle/OraHome1/xademo_files';
GRANT READ ON DIRECTORY xademo_dir TO xademo;
```

Refer to Chapter 12 for information about performing these tasks.

2. Open Analytic Workspace Manager and connect to your Oracle Database as the XADEMO user.
3. Open OLAP Worksheet.
4. Create an analytic workspace from the EIF file:

```
AW CREATE xademo
IMPORT ALL FROM EIF FILE 'olapdata/xademo.eif' DATA DFNS
UPDATE
COMMIT
```

5. Identify the time dimensions:

```
LIMIT name TO OBJ(PROPERTY 'DIMTYPE') EQ 1
REPORT name

NAME
--------------
TIME
QUARTER
YEAR
MONTH
```

This example shows how to provide support to the TIME dimension.

6. Identify the HIERDIM dimension for TIME.

```
SHOW OBJ(PROPERTY 'HIERDIM' 'TIME')

T0.HIERDIM
```

7. Create the TIME_END_DATE and TIME_TIME_SPAN variables.

```
DEFINE TIME_END_DATE VARIABLE DATE <TIME T0.HIERDIM>
PROPERTY 'USERDATA' FALSE
DEFINE TIME_TIME_SPAN VARIABLE INTEGER <TIME T0.HIERDIM>
```

```
PROPERTY 'USERDATA' FALSE
```

8. Populate the TIME_END_DATE and TIME_TIME_SPAN variables, as described in the following sections.
9. Set the properties on TIME.

```
CONSIDER time
PROPERTY 'END_DATE' 'TIME_END_DATE'
PROPERTY 'TIME_SPAN' 'TIME_TIME_SPAN'
```

10. Convert the analytic workspace to database standard form:

```
CALL CREATE_DB_STDFORM('xademo')
UPDATE
COMMIT
```

The XADEMO database does not have a data loader program, so no example is provided here. However, Chapter 11 includes examples of file load and SQL fetch programs.

About the Time Dimension in XADEMO

Oracle Express Objects metadata stores the names of supporting objects in properties on the TIME dimension, as shown in Table B–3.

Table B–3 Oracle Express Objects Properties for Hierarchy and Level Support

Property	Description
HIERDIM	List of hierarchies (dimension)
LEVELDIM	List of levels (dimension)
LEVELREL	Level associated with each dimension member (relation)
LEVELLABELFRM	Description of each level (formula)

By using the OBJ function, you can discover the names of objects that support the TIME dimension:

```
SHOW OBJ(PROPERTY 'LEVELDIM' 'TIME')
T0.LEVELDIM

SHOW OBJ(PROPERTY 'LEVELLABELFRM' 'TIME')
T0.LVLLABFRM
```

The TIME dimension has two hierarchies, which are listed in the T0.LEVELDIM dimension. They are named STANDARD and YTD. The following report shows sample TIME members at each level.

```
REPORT DOWN time t0.levelrel W 20 t0.lvllabfrm

           ----------------------------T0.HIERDIM-----------------------------
           ------------STANDARD------------- ---------------YTD---------------
TIME       T0.LEVELREL      T0.LVLLABFRM     T0.LEVELREL      T0.LVLLABFRM
---------- ------------ -------------------- ------------ --------------------
JAN96      L3           Month(s)             L5           YTD Month(s) Detail
FEB96      L3           Month(s)             L5           YTD Month(s) Detail
Q1.96      L2           Quarter(s)           NA           NA
LAST.YTD   NA           NA                   L4           YTD Summaries
1996       L1           Year(s)              NA           NA
```

Populating the XADEMO Time Attributes

The POP_TIME_ATTRS program shown in Example B–1 populates the TIME_END_DATE and TIME_TIME_SPAN variables.

For TIME_END_DATE, the program uses the ENDDATE function to identify the last day of each time period. The ENDDATE function only operates on dimensions with a time data type (such as MONTH and YEAR). However, the XADEMO TIME dimension has a TEXT data type. Several transformations are needed before the ENDDATE function can be used. The program takes these steps:

1. For each level, defines a dimension with the appropriate data type (MONTH, QUARTER, or YEAR). In the example, the dimensions are named M_TEMP, Q_TEMP, and Y_TEMP.
2. Stores the names of the dimension members for particular level in a valueset. In the example, the valueset is named T_LIST.
3. Uses the current status of the T_LIST valueset to add members to the new dimensions (M_TEMP, Q_TEMP, and Y_TEMP).

For TIME_TIME_SPAN, the program extracts the first two characters from TIME_END_DATE at the month level, which has values like 30APR96, to get the number of days in each month.

The program then uses the ROLLUP command to calculate the number of days in each quarter and year. T0.PARENT is a self-relation that identifies the parent-child relationships among dimension members. However, T0.PARENT and TIME_TIME_SPAN are both dimensioned by T0.HIERDIM, so ROLLUP cannot use T0.PARENT. Instead, the program creates a relation named TIME_PARENTREL

dimensioned only by TIME, populates it from T0.PARENT, and uses the new relation in the ROLLUP command.

Note that aggmaps are more efficient than ROLLUP, but since this case involves just a single dimension in which all aggregate values are stored, ROLLUP is slightly more convenient and the performance differences are negligible.

Example B–1 OLAP DML Program for Populating TIME Attributes

```
DEFINE POP_TIME_ATTRS PROGRAM
PROGRAM
VARIABLE _ytd TEXT          " Stores YTD time members
TRAP ON cleanup             " Divert processing on error to CLEANUP label

" Define dimensions for each level with date data types
IF NOT EXISTS('m_temp')
  THEN DEFINE m_temp DIMENSION MONTH
  ELSE MAINTAIN m_temp DELETE ALL

IF NOT EXISTS('q_temp')
  THEN DEFINE q_temp DIMENSION QUARTER
  ELSE MAINTAIN q_temp DELETE ALL

" Format years like TIME year members (1997 instead of YR97)
IF NOT EXISTS('y_temp')
  THEN DO
  DEFINE y_temp DIMENSION YEAR
  CONSIDER y_temp
  VNF <YYYY>
  DOEND
  ELSE MAINTAIN y_temp DELETE ALL

" Define a valueset to store time members
IF NOT EXISTS('t_list')
  THEN DEFINE t_list VALUESET TIME
  ELSE LIMIT t_list TO NA

" Define a one-dimensional time self-relation for rollup
IF NOT EXISTS('time_parentrel')
  THEN DEFINE time_parentrel RELATION time <time>
  ELSE time_parentrel = NA

" Initialize target variables
ALLSTAT
time_time_span = NA
```

```
time_end_date = NA
" ****************************************
"    Set values for the STANDARD hierarchy
" ****************************************
LIMIT t0.hierdim TO 'STANDARD'
" Select all time members at the month level
LIMIT time TO t0.levelrel 'L3'
" Store months in the valueset
LIMIT t_list TO time
" Populate M_TEMP so all months have a MONTH data type
MAINTAIN m_temp MERGE values(t_list)
" Calculate the end date
FOR m_temp
   time_end_date(time, m_temp) = ENDDATE(m_temp)
" Extract the number of days in each month
time_time_span = CONVERT(EXTCHARS(time_end_date, 1, 2), DECIMAL)

" Store quarters in q_temp
LIMIT time TO t0.levelrel 'L2'
LIMIT t_list TO time
MAINTAIN q_temp MERGE VALUES(t_list)
FOR q_temp
   time_end_date(time, q_temp) = ENDDATE(q_temp)

" Store years in y_temp
LIMIT time TO t0.levelrel 'L1'
LIMIT t_list TO time
MAINTAIN y_temp MERGE VALUES(t_list)
FOR y_temp
   time_end_date(time, y_temp) = ENDDATE(y_temp)
" ****************************************
"    Set values for the YTD hierarchy
" ****************************************
LIMIT t0.hierdim TO 'YTD'
" Limit status of months to YTD
LIMIT time TO t0.levelrel 'L5'
LIMIT t_list TO time
LIMIT m_temp TO t_list

" Calculate end date and time span for months
FOR m_temp
   time_end_date(time, m_temp) = ENDDATE(m_temp)
time_time_span = CONVERT(EXTCHARS(time_end_date, 1, 2), DECIMAL)

" Get current and previous YTD
```

```
LIMIT time TO t0.parent EQ 'LAST.YTD'
LIMIT time KEEP LAST 1
_ytd = time
time_end_date(time, 'LAST.YTD') = time_end_date(time, _ytd)
LIMIT time TO t0.parent EQ 'CURRENT.YTD'
LIMIT time KEEP LAST 1
_ytd = time
time_end_date(time, 'CURRENT.YTD') = time_end_date(time, _ytd)

" Rollup time span for quarters and years
LIMIT t0.hierdim TO ALL
LIMIT time TO ALL
FOR t0.hierdim
  DO
  time_parentrel = t0.parent
  ROLLUP time_time_span OVER time USING time_parentrel
  DOEND

CLEANUP:
" Delete temporary objects
DELETE m_temp q_temp y_temp t_list time_parentrel
END
```

C

Programs Used to Create GLOBALX

GLOBALX is a sample analytic workspace that is populated manually from sources other than a star schema, as described in Chapter 11. This appendix provides additional source code needed to replicate the examples in that chapter, but which are extraneous to the topics being discussed.

This appendix contains the following topics:

- SQL Scripts for Defining Users and Tablespaces
- SQL Scripts for the GLOBALX Star Schema
- SQL Scripts for OLAP Catalog Metadata

SQL Scripts for Defining Users and Tablespaces

Create the GLOBALX tablespace first, since it will be the default tablespace for both the GLOBALX and GLOBALX_AW users. Then create the GLOBALX user. Define the star schema before creating the GLOBALX_AW user.

Example C–1 Script for Creating the GLOBALX Tablespaces

```
CREATE TABLESPACE globalx LOGGING
   DATAFILE '/users/oracle/global_data/globalx.dbf'
   SIZE 5M REUSE AUTOEXTEND ON NEXT 5M MAXSIZE UNLIMITED
   EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO;

CREATE TEMPORARY TABLESPACE glotempx
   TEMPFILE '/users/oracle/global_data/glotempx.tmp'
   SIZE 5M REUNE AUTOEXTEND ON NEXT 5M MAXSIZE UNLIMITED
   EXTENT MANAGEMENT LOCAL UNIFORM SIZE 256K;
```

Example C–2 Script for Creating the GLOBALX User

```
/* Create the user and grant privileges */
CREATE USER globalx IDENTIFIED BY "globalx"
    DEFAULT TABLESPACE globalx
    TEMPORARY TABLESPACE glotempx
    QUOTA UNLIMITED ON globalx
    QUOTA UNLIMITED ON glotempx
    ACCOUNT UNLOCK;
GRANT CONNECT TO globalx;
GRANT OLAP_USER TO globalx;
GRANT CREATE ANY TYPE TO globalx;
GRANT CREATE ANY DIRECTORY TO globalx;

/* Create a database directory*/
CREATE OR REPLACE DIRECTORY gx AS '/users/oracle/globalx_files';
GRANT ALL ON DIRECTORY gx TO PUBLIC;
```

Example C–3 Script for Creating the GLOBALX_AW User

```
/* Create the user and grant privileges */
CREATE USER globalx_aw IDENTIFIED BY globalx_aw
    DEFAULT TABLESPACE globalx
    TEMPORARY TABLESPACE glotempx
    QUOTA UNLIMITED ON globalx
    QUOTA UNLIMITED ON glotempx
    ACCOUNT UNLOCK;
GRANT CONNECT TO globalx_aw;
GRANT OLAP_USER TO globalx_aw;
GRANT CREATE ANY TYPE TO globalx_aw;
GRANT CREATE ANY DIRECTORY TO globalx_aw;

/* Grant access to GLOBALX star schema */
GRANT SELECT ON GLOBALX.CHANNEL_DUMMY to GLOBALX_AW;
GRANT SELECT ON GLOBALX.PROD_DUMMY to GLOBALX_AW;
GRANT SELECT ON GLOBALX.CUSTOM_DUMMY to GLOBALX_AW;
GRANT SELECT ON GLOBALX.TIME_DUMMY to GLOBALX_AW;
GRANT SELECT ON GLOBALX.UNITS_DUMMY to GLOBALX_AW;
GRANT SELECT ON GLOBALX.PRICE_AND_COST_DUMMY to GLOBALX_AW;
```

SQL Scripts for the GLOBALX Star Schema

Example C–4 shows the SQL script used to create the empty GLOBALX tables.

Example C–4 Script for Creating the Empty GLOBALX Tables

```
CREATE TABLE GLOBALX.CHANNEL_DUMMY
   (CHANNEL_ID            NUMBER(5),
    CHANNEL_DSC           VARCHAR2(15),
    ALL_CHANNELS_ID       NUMBER(5),
    ALL_CHANNELS_DSC      VARCHAR2(15),
    CONSTRAINT UK_ON_CHANNEL_ID PRIMARY KEY(CHANNEL_ID));

CREATE TABLE GLOBALX.CUSTOM_DUMMY
   (SHIP_TO_ID            NUMBER(5),
    SHIP_TO_DSC           VARCHAR2(30),
    ACCOUNT_ID            NUMBER(5),
    ACCOUNT_DSC           VARCHAR2(30),
    MARKET_SEGMENT_ID     NUMBER(5),
    MARKET_SEGMENT_DSC    VARCHAR2(15),
    TOTAL_MARKET_ID       NUMBER(5),
    TOTAL_MARKET_DSC      VARCHAR2(15),
    WAREHOUSE_ID          NUMBER(5),
    WAREHOUSE_DSC         VARCHAR2(15),
    REGION_ID             NUMBER(5),
    REGION_DSC            VARCHAR(15),
    ALL_CUSTOMERS_ID      NUMBER(5),
    ALL_CUSTOMERS_DSC     VARCHAR2(15),
    CONSTRAINT UK_ON_SHIP_TO_ID PRIMARY KEY(SHIP_TO_ID));

CREATE TABLE GLOBALX.PROD_DUMMY
   (ITEM_ID               NUMBER(5),
    ITEM_DSC              VARCHAR2(31),
    ITEM_PACKAGE_ID       VARCHAR2(20),
    FAMILY_ID             NUMBER(5),
    FAMILY_DSC            VARCHAR2(20),
    CLASS_ID              NUMBER(5),
    CLASS_DSC             VARCHAR2(15),
    TOTAL_PRODUCT_ID      NUMBER(5),
    TOTAL_PRODUCT_DSC     VARCHAR2(15),
    CONSTRAINT UK_ON_ITEM_ID PRIMARY KEY(ITEM_ID));

CREATE TABLE GLOBALX.TIME_DUMMY
   (MONTH_ID              NUMBER(5),
    MONTH_DSC             VARCHAR2(10),
```

```
    QUARTER_ID            NUMBER(5),
    QUARTER_DSC           VARCHAR2(5),
    YEAR_ID               NUMBER(5),
    YEAR_DSC              VARCHAR2(5),
    MONTH_TIMESPAN        NUMBER(5),
    QUARTER_TIMESPAN      NUMBER(5),
    YEAR_TIMESPAN         NUMBER(5),
    MONTH_END_DATE        DATE,
    QUARTER_END_DATE      DATE,
    YEAR_END_DATE         DATE,
    CONSTRAINT UK_ON_MONTH_ID PRIMARY KEY(MONTH_ID));

CREATE TABLE GLOBALX.PRICE_AND_COST_DUMMY
   (ITEM_ID               NUMBER(5),
    MONTH_ID              NUMBER(5),
    UNIT_PRICE            NUMBER,
    UNIT_COST             NUMBER,
    CONSTRAINT FK_ON_ITEM_ID_01 FOREIGN KEY (ITEM_ID)
       REFERENCES PROD_DUMMY(ITEM_ID),
    CONSTRAINT FK_ON_MONTH_ID_01 FOREIGN KEY (MONTH_ID)
       REFERENCES TIME_DUMMY(MONTH_ID));

CREATE TABLE GLOBALX.UNITS_DUMMY
   (CHANNEL_ID            NUMBER(5),
    ITEM_ID               NUMBER(5),
    SHIP_TO_ID            NUMBER(5),
    MONTH_ID              NUMBER(5),
    UNITS                 NUMBER,
    CONSTRAINT FK_ON_CHANNEL_ID_02 FOREIGN KEY (CHANNEL_ID)
       REFERENCES CHANNEL_DUMMY(CHANNEL_ID),
    CONSTRAINT FK_ON_ITEM_ID_02 FOREIGN KEY (ITEM_ID)
       REFERENCES PROD_DUMMY(ITEM_ID),
    CONSTRAINT FK_ON_SHIP_TO_ID_02 FOREIGN KEY (SHIP_TO_ID)
       REFERENCES CUSTOM_DUMMY(SHIP_TO_ID),
    CONSTRAINT FK_ON_MONTH_ID_02 FOREIGN KEY (MONTH_ID)
       REFERENCES TIME_DUMMY(MONTH_ID));
```

SQL Scripts for OLAP Catalog Metadata

Example C–5 through Example C–10 are the scripts that define OLAP Catalog metadata for the GLOBALX star schema. This metadata enables the Create Analytic Workspace wizard to define a database standard form analytic workspace.

Example C–5 Script for Creating CHANNEL Metadata

```
EXECUTE CWM2_OLAP_DIMENSION.DROP_DIMENSION('GLOBALX','CHANNEL');
EXECUTE CWM2_OLAP_DIMENSION.CREATE_DIMENSION('GLOBALX','CHANNEL','Channel','CHANNEL',
   'Channel','Channel',NULL);
EXECUTE cwm2_olap_dimension_attribute.create_dimension_attribute_2('GLOBALX','CHANNEL',
   'Long Description','Long Description','Long Description','Long Description',1);
EXECUTE cwm2_olap_dimension_attribute.create_dimension_attribute_2('GLOBALX','CHANNEL',
   'Short Description','Short Description','Short Description','Short Description',1);
EXECUTE cwm2_olap_hierarchy.create_hierarchy('GLOBALX','CHANNEL','CHANNEL_ROLLUP',
   'Channel Rollup','Channel Rollup','Channel Rollup','UNSOLVED LEVEL-BASED');
EXECUTE cwm2_olap_dimension.set_default_display_hierarchy('GLOBALX','CHANNEL',
   CHANNEL_ROLLUP');
EXECUTE cwm2_olap_level.create_level('GLOBALX','CHANNEL','ALL_CHANNELS','All Channels',
   'All Channels','All Channels','All Channels');
EXECUTE cwm2_olap_level.create_level('GLOBALX','CHANNEL','CHANNEL','Channel','Channel',
   'Channel','Channel');
EXECUTE cwm2_olap_level.add_level_to_hierarchy('GLOBALX','CHANNEL','CHANNEL_ROLLUP',
   'ALL_CHANNELS',NULL);
EXECUTE cwm2_olap_level.add_level_to_hierarchy('GLOBALX','CHANNEL','CHANNEL_ROLLUP',
   'CHANNEL', 'ALL_CHANNELS');
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','CHANNEL',
   'Long Description','ALL_CHANNELS','Long Description','Long Description',
   'Long Description','Long Description',1);
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','CHANNEL',
   'Long Description','CHANNEL','Long Description','Long Description','Long Description',
   'Long Description',1);
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','CHANNEL',
   'Short Description','ALL_CHANNELS','Short Description','Short Description',
   'Short Description','Short Description',1);
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','CHANNEL',
   'Short Description','CHANNEL','Short Description','Short Description',
   'Short Description','Short Description',1);
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVEL('GLOBALX','CHANNEL','CHANNEL_ROLLUP',
   'ALL_CHANNELS','GLOBALX','CHANNEL_DUMMY','ALL_CHANNELS_ID',NULL);
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVEL('GLOBALX','CHANNEL','CHANNEL_ROLLUP',
   'CHANNEL','GLOBALX','CHANNEL_DUMMY','CHANNEL_ID','ALL_CHANNELS_ID');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','CHANNEL',
   'Long Description','CHANNEL_ROLLUP','CHANNEL','Long Description','GLOBALX',
   'CHANNEL_DUMMY','CHANNEL_DSC');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','CHANNEL','Long Description',
   'CHANNEL_ROLLUP','ALL_CHANNELS','Long Description','GLOBALX', 'CHANNEL_DUMMY',
   'ALL_CHANNELS_DSC');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','CHANNEL','Short Description',
   'CHANNEL_ROLLUP','CHANNEL','Short Description','GLOBALX', 'CHANNEL_DUMMY','CHANNEL_DSC');
```

```
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','CHANNEL','Short Description',
   'CHANNEL_ROLLUP','ALL_CHANNELS','Short Description', 'GLOBALX','CHANNEL_DUMMY',
   'ALL_CHANNELS_DSC');
EXECUTE cwm2_olap_validate.validate_dimension('GLOBALX', 'CHANNEL');
```

Example C–6 Script for Creating CUSTOMER Metadata

```
EXECUTE CWM2_OLAP_DIMENSION.DROP_DIMENSION('GLOBALX','CUSTOMER');
EXECUTE CWM2_OLAP_DIMENSION.CREATE_DIMENSION('GLOBALX','CUSTOMER','Customer','CUSTOMER',
   'Customer','Customer',NULL);
EXECUTE cwm2_olap_dimension_attribute.create_dimension_attribute_2('GLOBALX','CUSTOMER',
   'Long Description','Long Description','Long Description','Long Description',1);
EXECUTE cwm2_olap_dimension_attribute.create_dimension_attribute_2('GLOBALX','CUSTOMER',
   'Short Description','Short Description','Short Description','Short Description',1);
EXECUTE cwm2_olap_hierarchy.create_hierarchy('GLOBALX','CUSTOMER','MARKET_ROLLUP',
   'Market Rollup','Market Rollup','Market Rollup','UNSOLVED LEVEL-BASED');
EXECUTE cwm2_olap_hierarchy.create_hierarchy('GLOBALX','CUSTOMER','SHIPMENTS_ROLLUP',
   'Shipments Rollup','Shipments Rollup','Shipments Rollup','UNSOLVED LEVEL-BASED');
EXECUTE cwm2_olap_dimension.set_default_display_hierarchy('GLOBALX', 'CUSTOMER',
   'SHIPMENTS_ROLLUP');
EXECUTE cwm2_olap_level.create_level('GLOBALX','CUSTOMER','ALL_CUSTOMERS','All Customers',
   'All Customers','All Customers','All Customers');
EXECUTE cwm2_olap_level.create_level('GLOBALX','CUSTOMER','REGION','Region', 'Region','Region',
   'Region');
EXECUTE cwm2_olap_level.create_level('GLOBALX','CUSTOMER','WAREHOUSE', 'Warehouse','Warehouse',
   'Warehouse','Warehouse');
EXECUTE cwm2_olap_level.create_level('GLOBALX','CUSTOMER','TOTAL_MARKET','Total Market',
   'Total Market','Total Market','Total Market');
EXECUTE cwm2_olap_level.create_level('GLOBALX','CUSTOMER','MARKET_SEGMENT','Market Segment',
   'Market Segment','Market Segment','Market Segment');
EXECUTE cwm2_olap_level.create_level('GLOBALX','CUSTOMER','ACCOUNT', 'Account','Account',
   'Account','Account');
EXECUTE cwm2_olap_level.create_level('GLOBALX','CUSTOMER','SHIP_TO','Ship To','Ship To',
   'Ship To','Ship To');
EXECUTE cwm2_olap_level.add_level_to_hierarchy('GLOBALX','CUSTOMER', 'MARKET_ROLLUP',
   'TOTAL_MARKET',NULL);
EXECUTE cwm2_olap_level.add_level_to_hierarchy('GLOBALX','CUSTOMER', 'MARKET_ROLLUP',
   'MARKET_SEGMENT','TOTAL_MARKET');
EXECUTE cwm2_olap_level.add_level_to_hierarchy('GLOBALX','CUSTOMER','MARKET_ROLLUP',
   'ACCOUNT','MARKET_SEGMENT');
EXECUTE cwm2_olap_level.add_level_to_hierarchy('GLOBALX','CUSTOMER','MARKET_ROLLUP',
   'SHIP_TO','ACCOUNT');
EXECUTE cwm2_olap_level.add_level_to_hierarchy('GLOBALX','CUSTOMER','SHIPMENTS_ROLLUP',
   'ALL_CUSTOMERS',NULL);
```

```
EXECUTE cwm2_olap_level.add_level_to_hierarchy('GLOBALX','CUSTOMER','SHIPMENTS_ROLLUP',
   'REGION','ALL_CUSTOMERS');
EXECUTE cwm2_olap_level.add_level_to_hierarchy('GLOBALX','CUSTOMER','SHIPMENTS_ROLLUP',
   'WAREHOUSE','REGION');
EXECUTE cwm2_olap_level.add_level_to_hierarchy('GLOBALX','CUSTOMER','SHIPMENTS_ROLLUP',
   'SHIP_TO','WAREHOUSE');
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','CUSTOMER',
   'Long Description','ALL_CUSTOMERS','Long Description','Long Description',
   'Long Description','Long Description',1);
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','CUSTOMER',
   'Long Description','REGION','Long Description','Long Description','Long Description',
   'Long Description',1);
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','CUSTOMER',
   'Long Description','WAREHOUSE','Long Description','Long Description','Long Description',
   'Long Description',1);
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','CUSTOMER',
   'Long Description','TOTAL_MARKET','Long Description','Long Description','Long Description',
   'Long Description',1);
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','CUSTOMER',
   'Long Description','MARKET_SEGMENT','Long Description','Long Description',
   'Long Description','Long Description',1) ;
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','CUSTOMER',
   'Long Description','ACCOUNT','Long Description','Long Description','Long Description',
   'Long Description',1);
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','CUSTOMER',
   'Long Description','SHIP_TO','Long Description','Long Description','Long Description',
   'Long Description',1);
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','CUSTOMER',
   'Short Description','ALL_CUSTOMERS','Short Description','Short Description',
   'Short Description','Short Description',1);
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','CUSTOMER',
   'Short Description','REGION','Short Description','Short Description','Short Description',
   'Short Description',1);
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','CUSTOMER',
   'Short Description','WAREHOUSE','Short Description','Short Description','Short Description',
   'Short Description',1);
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','CUSTOMER',
   'Short Description','TOTAL_MARKET','Short Description','Short Description',
   'Short Description','Short Description',1);
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','CUSTOMER',
   'Short Description','MARKET_SEGMENT','Short Description','Short Description',
   'Short Description','Short Description',1);
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','CUSTOMER',
   'Short Description','ACCOUNT','Short Description','Short Description',
   'Short Description','Short Description',1);
```

```
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','CUSTOMER',
   'Short Description','SHIP_TO','Short Description','Short Description','Short Description',
   'Short Description',1);
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVEL('GLOBALX','CUSTOMER','MARKET_ROLLUP',
   'TOTAL_MARKET','GLOBALX','CUSTOM_DUMMY','ALL_CUSTOMERS_ID',NULL);
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVEL('GLOBALX','CUSTOMER','MARKET_ROLLUP',
   'MARKET_SEGMENT','GLOBALX','CUSTOM_DUMMY','MARKET_SEGMENT_ID','ALL_CUSTOMERS_ID');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVEL('GLOBALX','CUSTOMER','MARKET_ROLLUP',
   'ACCOUNT','GLOBALX','CUSTOM_DUMMY','ACCOUNT_ID','MARKET_SEGMENT_ID');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVEL('GLOBALX','CUSTOMER','MARKET_ROLLUP',
   'SHIP_TO','GLOBALX','CUSTOM_DUMMY','SHIP_TO_ID','ACCOUNT_ID');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVEL('GLOBALX','CUSTOMER','SHIPMENTS_ROLLUP',
   'ALL_CUSTOMERS','GLOBALX','CUSTOM_DUMMY','ALL_CUSTOMERS_ID',NULL);
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVEL('GLOBALX','CUSTOMER','SHIPMENTS_ROLLUP',
   'REGION','GLOBALX','CUSTOM_DUMMY','REGION_ID','ALL_CUSTOMERS_ID');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVEL('GLOBALX','CUSTOMER','SHIPMENTS_ROLLUP',
   'WAREHOUSE','GLOBALX','CUSTOM_DUMMY','WAREHOUSE_ID','REGION_ID');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVEL('GLOBALX','CUSTOMER','SHIPMENTS_ROLLUP',
   'SHIP_TO','GLOBALX','CUSTOM_DUMMY','SHIP_TO_ID','WAREHOUSE_ID');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','CUSTOMER','Long Description',
   'MARKET_ROLLUP','SHIP_TO','Long Description','GLOBALX', 'CUSTOM_DUMMY','SHIP_TO_DSC');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','CUSTOMER','Long Description',
   'MARKET_ROLLUP','ACCOUNT','Long Description','GLOBALX', 'CUSTOM_DUMMY','ACCOUNT_DSC');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','CUSTOMER','Long Description',
   'MARKET_ROLLUP','MARKET_SEGMENT','Long Description','GLOBALX', 'CUSTOM_DUMMY',
   'MARKET_SEGMENT_DSC');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','CUSTOMER','Long Description',
   'MARKET_ROLLUP','TOTAL_MARKET','Long Description','GLOBALX','CUSTOM_DUMMY',
   'TOTAL_MARKET_DSC');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','CUSTOMER','Long Description',
   'SHIPMENTS_ROLLUP','SHIP_TO','Long Description','GLOBALX', 'CUSTOM_DUMMY','SHIP_TO_DSC');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','CUSTOMER','Long Description',
   'SHIPMENTS_ROLLUP','WAREHOUSE','Long Description','GLOBALX', 'CUSTOM_DUMMY',
   'WAREHOUSE_DSC');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','CUSTOMER','Long Description',
   'SHIPMENTS_ROLLUP','REGION','Long Description','GLOBALX', 'CUSTOM_DUMMY','REGION_DSC');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','CUSTOMER','Long Description',
   'SHIPMENTS_ROLLUP','ALL_CUSTOMERS','Long Description','GLOBALX', 'CUSTOM_DUMMY',
   'ALL_CUSTOMERS_DSC');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','CUSTOMER','Short Description',
   'MARKET_ROLLUP','SHIP_TO','Short Description','GLOBALX', 'CUSTOM_DUMMY','SHIP_TO_DSC');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','CUSTOMER','Short Description',
   'MARKET_ROLLUP','ACCOUNT','Short Description','GLOBALX', 'CUSTOM_DUMMY','ACCOUNT_DSC');
```

```
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','CUSTOMER',
   'Short Description','MARKET_ROLLUP','MARKET_SEGMENT','Short Description','GLOBALX',
   'CUSTOM_DUMMY','MARKET_SEGMENT_DSC');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','CUSTOMER',
   'Short Description','MARKET_ROLLUP','TOTAL_MARKET','Short Description','GLOBALX',
   'CUSTOM_DUMMY','TOTAL_MARKET_DSC');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','CUSTOMER',
   'Short Description','SHIPMENTS_ROLLUP','SHIP_TO','Short Description','GLOBALX',
   'CUSTOM_DUMMY','SHIP_TO_DSC');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','CUSTOMER',
   'Short Description','SHIPMENTS_ROLLUP','WAREHOUSE','Short Description','GLOBALX',
   'CUSTOM_DUMMY','WAREHOUSE_DSC');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','CUSTOMER',
   'Short Description','SHIPMENTS_ROLLUP','REGION','Short Description','GLOBALX',
   'CUSTOM_DUMMY','REGION_DSC');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','CUSTOMER',
   'Short Description','SHIPMENTS_ROLLUP','ALL_CUSTOMERS','Short Description','GLOBALX',
   'CUSTOM_DUMMY','ALL_CUSTOMERS_DSC');
EXECUTE cwm2_olap_validate.validate_dimension('GLOBALX', 'CUSTOMER');
```

Example C–7 Script for Creating PRODUCT Metadata

```
EXECUTE CWM2_OLAP_DIMENSION.DROP_DIMENSION('GLOBALX','PRODUCT');
EXECUTE CWM2_OLAP_DIMENSION.CREATE_DIMENSION('GLOBALX','PRODUCT','Product','PRODUCT',
   'Product','Product',NULL);
EXECUTE cwm2_olap_dimension_attribute.create_dimension_attribute_2('GLOBALX','PRODUCT',
   'Long Description','Long Description','Long Description','Long Description',1);
EXECUTE cwm2_olap_dimension_attribute.create_dimension_attribute_2('GLOBALX','PRODUCT',
   'Short Description','Short Description','Short Description','Short Description',1);
EXECUTE cwm2_olap_dimension_attribute.create_dimension_attribute_2('GLOBALX',
   'PRODUCT','Package', 'Package','Package','Package',0);
EXECUTE cwm2_olap_hierarchy.create_hierarchy('GLOBALX','PRODUCT','PRODUCT_ROLLUP',
   'Product Rollup','Product Rollup','Product Rollup','UNSOLVED LEVEL-BASED');
EXECUTE cwm2_olap_dimension.set_default_display_hierarchy('GLOBALX','PRODUCT',
   'PRODUCT_ROLLUP');
EXECUTE cwm2_olap_level.create_level('GLOBALX','PRODUCT','TOTAL_PRODUCT','Total Product',
   'Total Product','Total Product','Total Product');
EXECUTE cwm2_olap_level.create_level('GLOBALX','PRODUCT','CLASS', 'Class','Class',
   'Class','Class');
EXECUTE cwm2_olap_level.create_level('GLOBALX','PRODUCT','FAMILY', 'Family','Family',
   'Family','Family');
EXECUTE cwm2_olap_level.create_level('GLOBALX','PRODUCT','ITEM','Item','Item','Item','Item');
EXECUTE cwm2_olap_level.add_level_to_hierarchy('GLOBALX','PRODUCT','PRODUCT_ROLLUP',
   'TOTAL_PRODUCT',NULL);
```

```
EXECUTE cwm2_olap_level.add_level_to_hierarchy('GLOBALX','PRODUCT','PRODUCT_ROLLUP',
   'CLASS','TOTAL_PRODUCT');
EXECUTE cwm2_olap_level.add_level_to_hierarchy('GLOBALX','PRODUCT','PRODUCT_ROLLUP',
   'FAMILY','CLASS');
EXECUTE cwm2_olap_level.add_level_to_hierarchy('GLOBALX','PRODUCT','PRODUCT_ROLLUP',
   'ITEM','FAMILY');
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','PRODUCT',
   'Long Description','TOTAL_PRODUCT','Long Description','Long Description',
   'Long Description','Long Description',1);
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','PRODUCT',
   'Long Description','CLASS','Long Description','Long Description','Long Description',
   'Long Description',1);
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','PRODUCT',
   'Long Description','FAMILY','Long Description','Long Description','Long Description',
   'Long Description',1);
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','PRODUCT',
   'Long Description','ITEM','Long Description','Long Description','Long Description',
   'Long Description',1);
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','PRODUCT',
   'Short Description','TOTAL_PRODUCT','Short Description','Short Description',
   'Short Description','Short Description',1);
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','PRODUCT',
   'Short Description','CLASS','Short Description','Short Description','Short Description',
   'Short Description',1);
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','PRODUCT',
   'Short Description','FAMILY','Short Description','Short Description','Short Description',
   'Short Description',1);
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','PRODUCT',
   'Short Description','ITEM','Short Description','Short Description','Short Description',
   'Short Description',1);
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','PRODUCT', 'Package',
   'ITEM','Package','Package','Package','Package',0);
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVEL('GLOBALX','PRODUCT','PRODUCT_ROLLUP',
   'TOTAL_PRODUCT','GLOBALX','PROD_DUMMY','TOTAL_PRODUCT_ID',NULL);
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVEL('GLOBALX','PRODUCT', 'PRODUCT_ROLLUP','CLASS',
   'GLOBALX','PROD_DUMMY','CLASS_ID','TOTAL_PRODUCT_ID');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVEL('GLOBALX','PRODUCT', 'PRODUCT_ROLLUP',
   'FAMILY', 'GLOBALX','PROD_DUMMY','FAMILY_ID','CLASS_ID');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVEL('GLOBALX','PRODUCT','PRODUCT_ROLLUP',
   'ITEM','GLOBALX','PROD_DUMMY','ITEM_ID','FAMILY_ID');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','PRODUCT','Long Description',
   'PRODUCT_ROLLUP','ITEM','Long Description','GLOBALX','PROD_DUMMY','ITEM_DSC');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','PRODUCT','Long Description',
   'PRODUCT_ROLLUP','FAMILY','Long Description','GLOBALX','PROD_DUMMY','FAMILY_DSC');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','PRODUCT',
```

```
   'Long Description','PRODUCT_ROLLUP','CLASS','Long Description','GLOBALX',
   'PROD_DUMMY','CLASS_DSC');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','PRODUCT','Long Description',
   'PRODUCT_ROLLUP','TOTAL_PRODUCT','Long Description','GLOBALX', 'PROD_DUMMY',
   'TOTAL_PRODUCT_DSC');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','PRODUCT','Short Description',
   'PRODUCT_ROLLUP','ITEM','Short Description','GLOBALX','PROD_DUMMY','ITEM_DSC');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','PRODUCT','Short Description',
   'PRODUCT_ROLLUP','FAMILY','Short Description','GLOBALX','PROD_DUMMY','FAMILY_DSC');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','PRODUCT','Short Description',
   'PRODUCT_ROLLUP','CLASS','Short Description','GLOBALX','PROD_DUMMY','CLASS_DSC');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','PRODUCT','Short Description',
   'PRODUCT_ROLLUP','TOTAL_PRODUCT','Short Description','GLOBALX','PROD_DUMMY',
   'TOTAL_PRODUCT_DSC');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','PRODUCT', 'Package',
   'PRODUCT_ROLLUP','ITEM','Package','GLOBALX','PROD_DUMMY','ITEM_PACKAGE_ID');
EXECUTE cwm2_olap_validate.validate_dimension('GLOBALX', 'PRODUCT');
```

Example C–8 Script for Creating TIME Metadata

```
EXECUTE CWM2_OLAP_DIMENSION.DROP_DIMENSION('GLOBALX','TIME');
EXECUTE CWM2_OLAP_DIMENSION.CREATE_DIMENSION('GLOBALX','TIME','Time', 'TIME','Time',
   'Time','Time');
EXECUTE cwm2_olap_dimension_attribute.create_dimension_attribute_2('GLOBALX','TIME',
   'Long Description','Long Description','Long Description','Long Description',1);
EXECUTE cwm2_olap_dimension_attribute.create_dimension_attribute_2('GLOBALX','TIME',
   'Short Description','Short Description','Short Description','Short Description',1);
EXECUTE cwm2_olap_dimension_attribute.create_dimension_attribute_2('GLOBALX','TIME',
   'End Date','End Date','End Date','End Date',1);
EXECUTE cwm2_olap_dimension_attribute.create_dimension_attribute_2('GLOBALX','TIME',
   'Time Span','Timespan','Timespan','Timespan',1);
EXECUTE cwm2_olap_hierarchy.create_hierarchy('GLOBALX','TIME','CALENDAR', 'Calendar',
   'Calendar','Calendar','UNSOLVED LEVEL-BASED');
EXECUTE cwm2_olap_dimension.set_default_display_hierarchy('GLOBALX','TIME','CALENDAR');
EXECUTE cwm2_olap_level.create_level('GLOBALX','TIME','YEAR','Year','Year','Year','Year');
EXECUTE cwm2_olap_level.create_level('GLOBALX','TIME','QUARTER', 'Quarter','Quarter',
   'Quarter','Quarter');
EXECUTE cwm2_olap_level.create_level('GLOBALX','TIME','MONTH','Month','Month','Month','Month');
EXECUTE cwm2_olap_level.add_level_to_hierarchy('GLOBALX','TIME','CALENDAR','YEAR',NULL);
EXECUTE cwm2_olap_level.add_level_to_hierarchy('GLOBALX','TIME','CALENDAR','QUARTER','YEAR');
EXECUTE cwm2_olap_level.add_level_to_hierarchy('GLOBALX','TIME','CALENDAR','MONTH','QUARTER');
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','TIME','Long Description',
   'YEAR','Long Description','Long Description','Long Description','Long Description',1);
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','TIME','Long Description',
   'QUARTER','Long Description','Long Description','Long Description','Long Description',1);
```

```
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','TIME','Long Description',
   'MONTH','Long Description','Long Description','Long Description','Long Description',1);
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','TIME','Short Description',
   'YEAR','Short Description','Short Description','Short Description','Short Description',1);
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','TIME','Short Description',
   'QUARTER','Short Description','Short Description','Short Description','Short Description',1);
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','TIME','Short Description',
   'MONTH','Short Description','Short Description','Short Description','Short Description',1);
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','TIME','End Date','YEAR',
   'End Date','End Date','End Date','End Date',1);
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','TIME','End Date','QUARTER',
   'End Date','End Date','End Date','End Date',1);
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','TIME','End Date','MONTH',
   'End Date','End Date','End Date','End Date',1);
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','TIME','Time Span','YEAR',
   'Time Span','Timespan','Timespan','Timespan',1);
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','TIME','Time Span',
   'QUARTER','Time Span','Timespan','Timespan','Timespan',1);
EXECUTE cwm2_olap_level_attribute.create_level_attribute_2('GLOBALX','TIME','Time Span','MONTH',
   'Time Span','Timespan','Timespan','Timespan',1);
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVEL('GLOBALX','TIME','CALENDAR', 'YEAR','GLOBALX',
   'TIME_DUMMY','YEAR_ID',NULL);
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVEL('GLOBALX','TIME','CALENDAR', 'QUARTER',
   'GLOBALX','TIME_DUMMY','QUARTER_ID','YEAR_ID');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVEL('GLOBALX','TIME','CALENDAR', 'MONTH','GLOBALX',
   'TIME_DUMMY','MONTH_ID','QUARTER_ID');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','TIME','Long Description',
   'CALENDAR','MONTH','Long Description','GLOBALX','TIME_DUMMY','MONTH_DSC');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','TIME','Long Description',
   'CALENDAR','QUARTER','Long Description','GLOBALX','TIME_DUMMY','QUARTER_DSC');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','TIME','Long Description',
   'CALENDAR','YEAR','Long Description','GLOBALX','TIME_DUMMY','YEAR_DSC');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','TIME','Short Description',
   'CALENDAR','MONTH','Short Description','GLOBALX','TIME_DUMMY','MONTH_DSC');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','TIME','Short Description',
   'CALENDAR','QUARTER','Short Description','GLOBALX','TIME_DUMMY','QUARTER_DSC');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','TIME','Short Description',
   'CALENDAR','YEAR','Short Description','GLOBALX','TIME_DUMMY','YEAR_DSC');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','TIME','End Date', 'CALENDAR',
   'MONTH','End Date','GLOBALX','TIME_DUMMY','MONTH_END_DATE');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','TIME','End Date', 'CALENDAR',
   'QUARTER','End Date','GLOBALX','TIME_DUMMY','QUARTER_END_DATE');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','TIME','End Date', 'CALENDAR',
   'YEAR','End Date','GLOBALX','TIME_DUMMY','YEAR_END_DATE');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','TIME','Time Span',
```

```
   'CALENDAR','MONTH','Time Span','GLOBALX','TIME_DUMMY','MONTH_TIMESPAN');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','TIME','Time Span',
   'CALENDAR','QUARTER','Time Span','GLOBALX','TIME_DUMMY','QUARTER_TIMESPAN');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_DIMTBL_HIERLEVELATTR('GLOBALX','TIME','Time Span',
   'CALENDAR','YEAR','Time Span','GLOBALX','TIME_DUMMY','YEAR_TIMESPAN');
EXECUTE cwm2_olap_validate.validate_dimension('GLOBALX', 'TIME');
```

Example C–9 Script for Creating UNITS Cube

```
EXECUTE CWM2_OLAP_CUBE.DROP_CUBE('GLOBALX','UNITS_CUBE');
EXECUTE CWM2_OLAP_CUBE.CREATE_CUBE('GLOBALX','UNITS_CUBE','UNITS CUBE','UNITS CUBE',
  'UNITS CUBE');
EXECUTE CWM2_OLAP_CUBE.ADD_DIMENSION_TO_CUBE('GLOBALX','UNITS_CUBE','GLOBALX','CHANNEL');
EXECUTE CWM2_OLAP_CUBE.ADD_DIMENSION_TO_CUBE('GLOBALX','UNITS_CUBE','GLOBALX','CUSTOMER');
EXECUTE CWM2_OLAP_CUBE.ADD_DIMENSION_TO_CUBE('GLOBALX','UNITS_CUBE','GLOBALX','PRODUCT');
EXECUTE CWM2_OLAP_CUBE.ADD_DIMENSION_TO_CUBE('GLOBALX','UNITS_CUBE','GLOBALX','TIME');
EXECUTE CWM2_OLAP_MEASURE.CREATE_MEASURE('GLOBALX','UNITS_CUBE','UNITS','UNITS','Units Sold',
   'Units Sold');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_FACTTBL_LEVELKEY('GLOBALX','UNITS_CUBE','GLOBALX','UNITS_DUMMY',
   'LOWESTLEVEL',
   'DIM:GLOBALX.CHANNEL/HIER:CHANNEL_ROLLUP/LVL:CHANNEL/COL:CHANNEL_ID;
    DIM:GLOBALX.CUSTOMER/HIER:MARKET_ROLLUP/LVL:SHIP_TO/COL:SHIP_TO_ID;
    DIM:GLOBALX.PRODUCT/HIER:PRODUCT_ROLLUP/LVL:ITEM/COL:ITEM_ID;
    DIM:GLOBALX.TIME/HIER:CALENDAR/LVL:MONTH/ COL:MONTH_ID;');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_FACTTBL_LEVELKEY('GLOBALX','UNITS_CUBE','GLOBALX',
   'UNITS_DUMMY','LOWESTLEVEL',
   'DIM:GLOBALX.CHANNEL/HIER:CHANNEL_ROLLUP/LVL:CHANNEL/COL:CHANNEL_ID;
    DIM:GLOBALX.CUSTOMER/HIER:SHIPMENTS_ROLLUP/LVL:SHIP_TO/COL:SHIP_TO_ID;
    DIM:GLOBALX.PRODUCT/HIER:PRODUCT_ROLLUP/LVL:ITEM/COL:ITEM_ID;
    DIM:GLOBALX.TIME/HIER:CALENDAR/LVL:MONTH/COL:MONTH_ID;');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_FACTTBL_MEASURE('GLOBALX','UNITS_CUBE','UNITS','GLOBALX',
   'UNITS_DUMMY','UNITS',
   'DIM:GLOBALX.CHANNEL/HIER:CHANNEL_ROLLUP/LVL:CHANNEL/COL:CHANNEL_ID;
    DIM:GLOBALX.CUSTOMER/HIER:MARKET_ROLLUP/LVL:SHIP_TO/COL:SHIP_TO_ID;
    DIM:GLOBALX.PRODUCT/HIER:PRODUCT_ROLLUP/LVL:ITEM/COL:ITEM_ID;
    DIM:GLOBALX.TIME/HIER:CALENDAR/LVL:MONTH/COL:MONTH_ID;');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_FACTTBL_MEASURE('GLOBALX','UNITS_CUBE','UNITS','GLOBALX',
   'UNITS_DUMMY','UNITS',
   'DIM:GLOBALX.CHANNEL/HIER:CHANNEL_ROLLUP/LVL:CHANNEL/COL:CHANNEL_ID;
    DIM:GLOBALX.CUSTOMER/HIER:SHIPMENTS_ROLLUP/LVL:SHIP_TO/COL:SHIP_TO_ID;
    DIM:GLOBALX.PRODUCT/HIER:PRODUCT_ROLLUP/LVL:ITEM/COL:ITEM_ID;
    DIM:GLOBALX.TIME/HIER:CALENDAR/LVL:MONTH/COL:MONTH_ID;');
EXECUTE cwm2_olap_validate.validate_Cube('GLOBALX', 'UNITS_CUBE');
```

Example C–10 Script for Create PRICE Cube

```
EXECUTE CWM2_OLAP_CUBE.DROP_CUBE('GLOBALX','PRICE_CUBE');
EXECUTE CWM2_OLAP_CUBE.CREATE_CUBE('GLOBALX','PRICE_CUBE','PRICE CUBE','PRICE CUBE',
   'PRICE CUBE');
EXECUTE CWM2_OLAP_CUBE.ADD_DIMENSION_TO_CUBE('GLOBALX','PRICE_CUBE','GLOBALX','TIME');
EXECUTE CWM2_OLAP_CUBE.ADD_DIMENSION_TO_CUBE('GLOBALX','PRICE_CUBE','GLOBALX','PRODUCT');
EXECUTE CWM2_OLAP_MEASURE.CREATE_MEASURE('GLOBALX','PRICE_CUBE','UNIT_COST','UNIT COST',
   'Unit Cost','Unit Cost');
EXECUTE CWM2_OLAP_MEASURE.CREATE_MEASURE('GLOBALX','PRICE_CUBE','UNIT_PRICE','UNIT PRICE',
   'Unit Price','Unit Price');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_FACTTBL_LEVELKEY('GLOBALX','PRICE_CUBE',
   'GLOBALX', 'PRICE_AND_COST_DUMMY','LOWESTLEVEL',
   'DIM:GLOBALX.PRODUCT/HIER:PRODUCT_ROLLUP/LVL:ITEM/COL:ITEM_ID;
    DIM:GLOBALX.TIME/HIER:CALENDAR/LVL:MONTH/COL:MONTH_ID;');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_FACTTBL_MEASURE('GLOBALX','PRICE_CUBE','UNIT_COST','GLOBALX',
   'PRICE_AND_COST_DUMMY','UNIT_COST',
   'DIM:GLOBALX.PRODUCT/HIER:PRODUCT_ROLLUP/LVL:ITEM/COL:ITEM_ID;
    DIM:GLOBALX.TIME/HIER:CALENDAR/LVL:MONTH/COL:MONTH_ID;');
EXECUTE CWM2_OLAP_TABLE_MAP.MAP_FACTTBL_MEASURE('GLOBALX','PRICE_CUBE','UNIT_PRICE','GLOBALX',
   'PRICE_AND_COST_DUMMY','UNIT_PRICE',
   'DIM:GLOBALX.PRODUCT/HIER:PRODUCT_ROLLUP/LVL:ITEM/COL:ITEM_ID;
    DIM:GLOBALX.TIME/HIER:CALENDAR/LVL:MONTH/COL:MONTH_ID;');
EXECUTE cwm2_olap_validate.validate_Cube('GLOBALX', 'PRICE_CUBE');
```

Glossary

abstract data type (ADT)

See **object type**.

additive

Describes a fact (or measure) that can be summarized through addition. An additive fact is the most common type of fact. Examples include sales, cost, and profit.

Contrast with **nonadditive**, **semi-additive**.

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. Aggregation is synonymous with summarization, and aggregate data is synonymous with summary data.

analytic workspace

A multidimensional schema stored in a LOB table in the relational database. An analytic workspace can contain a variety of objects. Some of these objects may be integrally connected to other objects, while others are totally independent. Some objects store data that is useful to applications, and other objects may only exist for the purposes of the DBA or developer. There are several basic types of objects which play a variety of roles in the multidimensional model. In these respects, an analytic workspace is very similar to a relational schema.

The OLAP DML is the basic, low-level language for working in an analytic workspace. Tools are available in PL/SQL and Java that provide an interface to the OLAP DML for users already familiar with those languages.

See also **OLAP DML**.

ancestor

A value at any level higher than a given value in a hierarchy. For example, in a Time dimension, the value 2002 might be the ancestor of the values Q1-02 and Jan-02. 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 descriptive characteristic of either a single dimension member or a group of dimension members. When applied to a single member, attributes 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 applied to a group, attributes represent logical groupings that enable users to select data based on like characteristics. For example, in a database representing footwear, you can use a shoe color attribute to select all boots, sneakers, and slippers that share the same color.

base level data

Data at the lowest level, often acquired from another source, such as a transactional database.

Contrast with **aggregation**.

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 variable with the dimensions `MONTH` and `DISTRICT`, then each combination of a month and a district identifies a separate cell of that variable.

See also **dimension**, **variable**.

child

A value at the level under a given value in a hierarchy. For example, in a Time dimension, the value Jan-02 might be the child of the value Q1-2002. A value can be a child for more than one parent if the child value belongs to multiple hierarchies.

Contrast with **parent**. See also **descendant**, **hierarchy**, **level**.

composite

An analytic workspace object that lists dimension-value combinations (also called a tuple) for which there is data. When a data value is added to a variable dimensioned by a composite, that action triggers the creation of a composite tuple. A composite is an index into one or more sparse data variables, and is used to store sparse data in a compact form.

See also **dimension**, **sparsity**, **variable**.

container

See **object**.

cube

A logical organization of measures with identical dimensions. The edges of the cube contain dimension members and the body of the cube contains 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. In a star schema, a cube is represented by a fact table.

custom measure

A derived measure that is calculated at run-time and presented as one or more additional columns of data added to a result set. The result set includes a value for each dimension member currently in status. A custom measure typically employs a single-row function to perform a calculation on one or more stored measures. For example, an analyst might create a custom measure for the difference in costs from the prior period, which uses the OLAP DML `LAGDIF` function on the `COSTS` measure. Another analyst might create a custom measure that calculates profits by subtracting the `COSTS` measure from the `SALES` measure.

See also **dimension member**, **OLAP DML**, **measure**, **status**.

custom member

A member of a dimension created at run-time and defined as the parent of one or more existing dimension members. The values of a measure for a custom member are calculated using the aggregation rules for that dimension.

See also **aggregation**, **dimension member**, **parent**.

data source

A database, application, repository, or file that provides data.

data warehouse

A relational database that is 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.

database standard form

An analytic workspace that has been constructed with a specific set of objects, such as hierarchy dimensions, level dimensions, parent relations, and level relations. Each object must be defined with a set of properties that identify its role and its relationships with other objects in the analytic workspace. The standard form is required for an analytic workspace to be accessible to OLAP tools, however, it is not a prerequisite for multidimensional analysis.

DBA

Database administrator. The person responsible for creating, installing, configuring and maintaining Oracle Databases.

definition

The description of an analytic workspace object. An object definition includes characteristics such as the object's name, type (for example, dimension or variable), data type, dimensions, long description, permission specifications, and properties.

See also **dictionary**, **object**, **property**.

denormalized

Permit redundancy in a table. Contrast with **normalize**.

derived fact (or measure)

A fact (or measure) that is generated from existing data using a mathematical operation or a data transformation. Examples include averages, totals, percentages, and differences.

descendant

A dimension member at any level below a particular member in a hierarchy. The level immediately below is the child.

Contrast with **ancestor**. See also **aggregation**, **child**, **hierarchy**, **level**.

dictionary

The collection of object definitions in an analytic workspace. The dictionary is also called the workspace dictionary.

See also **definition**, **object**.

dimension

A structure that categorizes data. Among the most common dimensions for sales-oriented data are time, geography, and product. Most dimensions have hierarchies.

In an analytic workspace, a dimension is a container for a list of values. A dimension acts as an index for identifying the values of a variable. For example, if sales data has a separate sales figure for each month, then the data has a month dimension; that is, the data is organized by month.

In SQL, a dimension is a type of object that defines hierarchical (parent/child) relationships between pairs of column sets.

See also **hierarchy**.

dimension member

One element in the list that makes up a dimension. Also called a dimension value. A computer company might have dimension members in the product dimension called LAPPC and DESKPC. Members in the geography dimension might include Boston and Paris. Members in the time dimension might include NOV02, DEC02, JAN03, FEB03, MAR03, and so forth.

dimension table

A relational table that stores all or part of the values for a logical dimension in a star or snowflake schema. Dimension tables describe the business entities of an enterprise, represented as hierarchical, categorical information such as time, departments, locations, and products. They are sometimes called lookup or reference tables.

dimension value

See **dimension member**.

dimension view

A relational view of data in an analytic workspace that contains the same types of data as a dimension table in a star schema, that is, columns for dimension members

and attributes. A dimension view typically lists all dimension members in the key column, regardless of their level in the dimension hierarchy.

See also **dimension table**, **star schema**.

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. When selecting data, you can expand or collapse a hierarchy by drilling down or up in it, respectively.

drill down

To expand the view to include child values that are associated with parent values in the hierarchy.

drill up

To collapse the list of descendant values that are associated with a parent value in the hierarchy.

edge

A set of one or more dimensions that are displayed together in a cube or document. Although there is no limit to the number of edges in a cube, data is often organized for display purposes along three edges, which are referred to as the row edge, the column edge, and the page edge.

In a cross-tab report, dimension members on the row edge appear in the first column and identify the rows, dimension members on the column edge appear in the first row and identify the columns, and dimension members on the page edge label the individual pages of the report.

See also **cube**.

EIF file

A specially formatted file for transferring data between analytic workspaces. Using the OLAP DML, you can create an EIF file using the `EXPORT` command and read an EIF file using the `IMPORT` command.

embedded total

A predefined level of aggregation built into a dimension for which a hierarchy exists. For example, in a time dimension, each quarter represents the total for the months in the quarter. Data for embedded totals is calculated in the analytic workspace by the aggregation system.

See also **aggregation**, **dimension**, **hierarchy**.

fact

See **measure**. See also **additive**, **derived fact (or measure)**.

fact table

A table in a star schema that contains facts. 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 level facts or facts that have been aggregated. 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.

family relation

An analytic workspace relation object that identifies the complete parentage of each dimension member. A family relation has at least two dimensions: the data dimension and a level dimension. The contents of the relation identify, for each dimension member, the ancestor at each level of the hierarchy.

See also **ancestor**, **level**, **relation**.

formula

A type of workspace object that represents a stored calculation, expression, or procedure that produces a value. A formula provides a way to define and save complex or frequently used relationships within the data without storing the result set. Each time you query a formula, the OLAP engine performs the calculation or procedure that is required to produce the value.

hierarchy

A logical structure that uses ordered levels as a means of organizing data. A hierarchy can be used to define data aggregation; for example, in a time dimension, a hierarchy might be used to aggregate data from the month level to the quarter level to the year level. A hierarchy can be used to define a navigational drill path, regardless of whether the levels in the hierarchy represent aggregated totals.

In the PL/SQL, hierarchies can be defined part of a dimension object.

level

A position in a hierarchy. For example, a time dimension might have a hierarchy that represents data at the month, quarter, and year levels.

level relation

An analytic workspace relation object that identifies the level of each dimension member.

See also **level**, **relation**.

mapping

The definition of the relationship and data flow between source and target objects.

materialized view

A precomputed relational table comprising aggregated or joined data from fact and possibly dimension tables. Also known as a summary or aggregate table.

measure

Data that can be examined and analyzed, such as sales or cost data. You can select and display the data in a measure. Measures can be stored as variables or relations, or measures can be calculated by means of formulas. 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.

There are both base measures and custom measures. Base measures, such as Volume Sales and Dollar Sales, are stored. Custom measures, such as Volume Share Year Ago, are calculated from base measures.

See also **formula**, **relation**, **variable**.

measure view

A relational view of data in analytic workspace that contains the same types of data as a fact table in a star schema. However, in addition to the base-level facts, a measure view also contains derived data, such as aggregates and inter-row calculations.

See also **fact table**, **star schema**.

metadata

Data that describes data and other structures, such as objects, business rules, and processes.

See also **OLAP Catalog**.

model

A type of analytic workspace object that contains a set of interrelated equations, which are used to calculate data and assign it to a variable or dimension value. Models are used frequently when working with financial data.

See also **dimension member**, **object**, **variable**.

NA value

A special data value that indicates that data is "not available" (NA). 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 fact (or measure) that cannot be summarized through addition, such as average. Contrast with **additive**, **semi-additive**.

normalize

In a relational database, the process of removing redundancy in data by separating the data into multiple tables. Contrast with **denormalized**.

object

In an analytic workspace, a distinct item in the workspace dictionary. Analytic workspaces consist of one or more objects, such as variables, formulas, dimensions, relations, and programs, which are used to organize, store, and retrieve data. Each object is created with a particular object type and stores a particular type of information. Objects that are the same type (for example, three variables) can have different roles within the analytic workspace.

See also **role**.

object type

In Oracle object technology, a form of user-defined data type that is an abstraction of a real-world entity. An object type is a schema object with the following components:

- A name, which identifies the object type uniquely within a schema
- Attributes, which model the structure and state of the real-world entity

- Methods, which implement the behavior of the real-world entity, in either PL/SQL or Java

OLAP Catalog

A metadata package consisting of a set of read and write APIs that describe data in multidimensional terms, such as cubes, measures, dimensions, and attributes.

See also **metadata**.

OLAP DML

The low-level data definition and manipulation language for analytic workspaces.

on-the-fly

Calculated at run-time in response to a specific query. In an analytic workspace, custom measures and custom members are typically calculated on the fly. Aggregate data can be precalculated, calculated on the fly, or a combination of the two methods.

Contrast with **precalculate**.

online analytical processing (OLAP)

Functionality characterized by dynamic, multidimensional 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

online transaction processing (OLTP)

Systems 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.

parent

A dimension member at the level 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**.

parent relation

An analytic workspace relation object that defines a dimension's hierarchies by storing the parent of each dimension member.

See also **parent**, **relation**.

precalculate

Calculated and stored as a data maintenance procedure. In an analytic workspace, aggregate data can be precalculated, calculated on the fly, or a combination of the two methods.

Contrast with **on-the-fly**.

program

A type of database object that contains a series of OLAP DML commands. A program executes a set of related commands. Programs can be nested, with one calling another. A program can return a value; in this case, it is called a user-defined function.

See also **object**.

property

A characteristic of an object or component. Properties provide identifiers and descriptions, define object features (such as the number of decimal places or the color), or define object behaviors (such as whether an object is enabled). Properties are used extensively in standard form analytic workspaces.

See also **object**.

QDR

See **qualified data reference**.

qualified data reference

A qualifier that limits one or more dimensions to a single value for the duration of an OLAP DML command. A QDR is useful when you want to temporarily reference

a value without affecting the current status. In the following example of an OLAP DML command, the QDR limits the MONTH dimension to JUN02.

```
SHOW sales(month 'JUN02')
```

See also **dimension**, **dimension member**, **status**.

relation

A type of workspace object that is similar to a variable, except that it restricts its data values to the members of a particular dimension (such as PRODUCT) instead of to a particular data type (such as NUMBER). A relation establishes a correspondence between the values of a given dimension and the values of that dimension or other dimensions in the database.

For example, you might have a relation between cities and sales regions, such that each city belongs to a particular region. In a relation between cities and sales regions, the relation is dimensioned by CITY. Each cell holds the corresponding value of the REGION dimension.

See also **cell**, **dimension**, **dimension member**, **variable**.

role

The function of a workspace object within its broader categorization of object type. For example, a variable that stores numeric business measures has a role of measure. A variable that stores descriptive product names has a role of attribute. Both are variables, but they contain different types of information and play different roles in the multidimensional model.

See also **object**.

rollup form

A table that displays the full ancestry of each dimension member within a row. The table provides a column for each level of the hierarchy.

For example, a row for base-level dimension member Florence has FLORENCE in the City column, ITALY in the Country column, and EUROPE in the Region column. A row for Italy has null in the City column, ITALY in the Country column, and EUROPE in the Region column.

Contrast with **embedded total**. See also **ancestor**, **dimension member**, **hierarchy**.

schema

A collection of related database objects. Relational schemas are grouped by database user ID and include tables, views, and other objects. Multidimensional schemas are

called analytic workspaces and include dimensions, relations, variables, and other objects.

See also **analytic workspace**, **snowflake schema**, **star schema**.

semi-additive

Describes a fact (or measure) that can be summarized through addition along some, but not all, dimensions. Examples include head count and on-hand stock.

Contrast with **additive**, **nonadditive**.

snowflake schema

A type of star schema in which the dimension tables are partly or fully normalized.

See also **normalize**, **schema**, **star schema**.

solved data

A result set in which all derived data has been calculated. Data fetched from an analytic workspace 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 analytic workspace is the same whether the data was precalculated or calculated on the fly.

See also **on-the-fly**, **precalculate**.

source

A database, application, file, or other storage facility from which the data in a data warehouse is derived.

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. Such "empty," or NA, values take up storage space in an analytic workspace. To handle sparse data efficiently, you can create a composite.

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 variable 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 contain NA values.
- Random sparsity occurs when NA values are scattered throughout the variable, usually because some combinations of dimension values never have any data.

For example, a district might only sell certain products and never have data for other products. Other districts might sell some of those products and other ones, too.

See also **composite**, **NA value**.

standard form

See **database standard form**.

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 multidimensional data model. The star schema consists of one or more fact tables and one or more dimension tables that are related through foreign keys.

See also **schema**, **snowflake schema**

status

The list of currently accessible values for a given dimension. If the status of a given dimension is limited to a subset of its stored values, then all expressions that are based on that dimension will be limited to the corresponding subset of data. 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 **dimension**, **dimension member**.

summary

See **aggregation**, **materialized view**.

update window

The length of time available for updating the data in your database.

valueset

A type of workspace object. A valueset contains a list of dimension members for a particular dimension. After defining a valueset, you use the LIMIT command to assign members from the dimension to the valueset. The values in a valueset can be saved across Oracle OLAP sessions.

When you begin a new Oracle OLAP session or start up a workspace, each dimension has all values in the status. You can then limit a dimension to the values stored in the valueset for that dimension.

See also **dimension**.

variable

A type of workspace object that stores data. The data type of a variable indicates the kind of data that it contains, such as numeric or text data.

If a variable has dimensions, then those dimensions organize its data, and there is one cell for each combination of dimension members. A dimensioned variable is an array whose cells are individual data values. If a variable has no dimensions, then it is a single-cell variable, which contains one data value.

See also **cell**, **dimension**, **dimension member**, **object**.

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X