Oracle9*i* OLAP

User's Guide

Release 2 (9.2.0.2)

Sept 2002

Part No. A95295-02

ORACLE

Oracle9*i* OLAP User's Guide, Release 2 (9.2.0.2)

Part No. A95295-02

Contents

2 Manipulating Multidimensional Data

3 Developing OLAP Applications

4 Designing Your Database for OLAP

5 Creating OLAP Catalog Metadata

Part II Oracle OLAP Administration

6 Administering Oracle OLAP

7 OLAP Dynamic Performance Views

8 OLAP_API_SESSION_INIT

9 Creating an Analytic Workspace From Relational Tables

10 Creating Materialized Views for the OLAP API

12 OLAP_TABLE

14 Viewing OLAP Catalog Metadata

15 CWM2_OLAP_AW_ACCESS

20 CWM2_OLAP_HIERARCHY

21 CWM2_OLAP_LEVEL

22 CWM2_OLAP_LEVEL_ATTRIBUTE

23 CWM2_OLAP_MEASURE

24 CWM2_OLAP_METADATA_REFRESH

25 CWM2_OLAP_PC_TRANSFORM

Send Us Your Comments

Oracle9*i* OLAP User's Guide, Release 2 (9.2.0.2)

Part No. A95295-02

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this document. 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?

If you find any errors or have any other suggestions for improvement, please indicate the document title and part number, 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 (optionally) electronic mail address.

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

Preface

The *Oracle9i OLAP User's Guide* describes how to use Oracle OLAP for business analysis. It introduces the concepts underlying analytical applications and multidimensional querying, and the tools used for application development and system administration.

This preface contains these topics:

Audience

This guide is intended for application developers and database administrators who perform the following tasks:

- Administer a database
- Build and maintain data warehouses or data marts
- Define metadata
- Develop analytical applications

To use this document, you need no prior knowledge of Oracle OLAP.

Organization

This document is organized in five parts.

Part 1: The Basics

Provides conceptual information of general interest to anyone planning to use Oracle OLAP.

Chapter 1, "Overview"

Explains the basics of using Oracle OLAP and related client software for analytical applications.

Chapter 2, "Manipulating Multidimensional Data"

Provides an overview of data manipulation using the OLAP DML.

Chapter 3, "Developing OLAP Applications"

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

Chapter 4, "Designing Your Database for OLAP"

Highlights some of the most important data warehousing concepts, and provides additional information that is specific to Oracle OLAP.

Chapter 5, "Creating OLAP Catalog Metadata"

Provides an overview of OLAP Catalog metadata and the APIs for working with it.

Part II: "Oracle OLAP Administration"

Provides information for database administrators on administrative tasks associated with Oracle OLAP.

Chapter 6, "Administering Oracle OLAP"

Describes the various administrative tasks that are associated with Oracle OLAP.

Chapter 7, "OLAP Dynamic Performance Views"

Describes the relational views that contain performance data on Oracle OLAP.

Chapter 8, "OLAP_API_SESSION_INIT"

Describes the `OLAP_API_SESSION_INIT` package, which contains procedures for maintaining a configuration table of initialization parameters.

Chapter 9, "Creating an Analytic Workspace From Relational Tables"

Describes how to create an analytic workspace from a star schema and OLAP Catalog metadata. Describes how to generate relational views of the workspace data.

Chapter 10, "Creating Materialized Views for the OLAP API"

Describes how to create materialized views for star schemas that will be used by the OLAP API.

Part III: "SQL Access Reference"

Provides information about SQL packages and procedures that either create relational views of multidimensional data or embed OLAP DML commands in their syntax.

Chapter 11, "DBMS_AW"

Contains reference information for the `DBMS_AW` package, which enables SQL programmers to issue OLAP DML statements against analytic workspace data.

Chapter 12, "OLAP_TABLE"

Describes how SQL programmers can use the `OLAP_TABLE` function in a SQL `SELECT` statement to query multidimensional data in an analytic workspace

Part IV: "OLAP Catalog Metadata API Reference"

Describes the OLAP Catalog views and the PL/SQL packages for creating OLAP Catalog metadata.

Chapter 13, "Using the OLAP Catalog Metadata APIs"

Describes how to use the `CWM2` PL/SQL packages.

Chapter 14, "Viewing OLAP Catalog Metadata"

Describes the views of OLAP Catalog metadata.

Chapter 15, "CWM2_OLAP_AW_ACCESS"

Describes procedures for creating generic views of data stored in analytic workspaces.

Chapter 16, "CWM2_OLAP_AW_CREATE"

Describes procedures for creating an analytic workspace from relational tables and generating views of the resulting data in the analytic workspace.

Chapter 17, "CWM2_OLAP_CUBE"

Describes procedures for creating, dropping, and locking cubes, for adding dimensions to cubes, and for setting general properties of cubes.

Chapter 18, "CWM2_OLAP_DIMENSION"

Describes procedures for creating, dropping, and locking dimensions, and for setting general dimension properties.

Chapter 19, "CWM2_OLAP_DIMENSION_ATTRIBUTE"

Describes procedures for creating, dropping, and locking dimension attributes, and for setting general properties of dimension attributes.

Chapter 20, "CWM2_OLAP_HIERARCHY"

Describes procedures for creating, dropping, and locking hierarchies, and for setting general hierarchy properties.

Chapter 21, "CWM2_OLAP_LEVEL"

Describes procedures for creating, dropping, and locking levels, for adding levels to hierarchies, and for setting the general properties of levels.

Chapter 22, "CWM2_OLAP_LEVEL_ATTRIBUTE"

Describes a procedure for creating level attributes, associating them with dimension attributes, and for dropping, locking, and setting the general properties of level attributes.

Chapter 23, "CWM2_OLAP_MEASURE"

Describes procedures for creating, dropping, and locking measures, and for setting general properties of measures.

Chapter 24, "CWM2_OLAP_METADATA_REFRESH"

Describes the procedure for refreshing metadata tables for the OLAP API.

Chapter 25, "CWM2_OLAP_PC_TRANSFORM"

Describes the procedure for converting a parent-child dimension table to an embedded-total dimension table.

Chapter 26, "CWM2_OLAP_TABLE_MAP"

Describes procedures for mapping OLAP metadata entities to columns in your data warehouse tables or views.

Chapter 27, "CWM2_OLAP_VALIDATE"

Describes procedures for validating OLAP metadata.

Chapter 28, "CWM_CLASSIFY"

Describes procedures for creating measure folders and populating them with measures.

Part V: "OLAP API Materialized View Reference"

Explains how to create materialized views for queries for aggregate data from the OLAP API.

Chapter 29, "Creating Dimension Materialized Views"

Explains how to create materialized views for dimensions.

Chapter 30, "Creating Fact Materialized Views With DBMS_ODM"

Explains how to use the `DBMS_ODM` package to create fact table materialized views in grouping set form.

Chapter 31, "Creating Fact Materialized Views With OLAP Summary Advisor"

Explains how to use OLAP Summary Advisor to create fact table materialized views in concatenated rollup form.

Appendix A, "Upgrading From Express Server"

Provides upgrading instructions and identifies some of the major differences between Oracle Express Server 6.3 and Oracle9*i* OLAP.

Related Documentation

For more information, see these Oracle resources:

- *Oracle9i OLAP Developer's Guide to the OLAP API*
- Oracle9i OLAP API Javadoc
- *Oracle9i OLAP Developer's Guide to the OLAP DML*
- Oracle9i OLAP DML Reference help

Many books in the documentation set use the sample schemas of the seed database, which is installed by default when you install Oracle. Refer to *Oracle9i Sample Schemas* for information on how these schemas were created and how you can use them yourself.

In North America, printed documentation is available for sale in the Oracle Store at

```
http://oraclestore.oracle.com/
```

Customers in Europe, the Middle East, and Africa (EMEA) can purchase documentation from

```
http://www.oraclebookshop.com/
```

Other customers can contact their Oracle representative to purchase printed documentation.

To download free release notes, installation documentation, white papers, or other collateral, please visit the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at

```
http://otn.oracle.com/admin/account/membership.html
```

If you already have a username and password for OTN, then you can go directly to the documentation section of the OTN Web site at

`http://otn.oracle.com/docs/index.htm`

To access the database documentation search engine directly, please visit

`http://tahiti.oracle.com`

Conventions

This section describes the conventions used in the text and code examples of this documentation set. It describes:

- Conventions in Text
- Conventions in Code Examples

Conventions in Text

We use various conventions in text to help you more quickly identify special terms. The following table describes those conventions and provides examples of their use.

Convention	Meaning	Example
Bold	Bold typeface indicates terms that are defined in the text or terms that appear in a glossary, or both.	When you specify this clause, you create an **index-organized table**.
Italics	Italic typeface indicates book titles or emphasis.	*Oracle9i Database Concepts* Ensure that the recovery catalog and target database do *not* reside on the same disk.
`UPPERCASE monospace (fixed-width) font`	Uppercase monospace typeface indicates elements supplied by the system. Such elements include parameters, privileges, datatypes, RMAN keywords, SQL keywords, SQL*Plus or utility commands, packages and methods, as well as system-supplied column names, database objects and structures, usernames, and roles.	You can specify this clause only for a `NUMBER` column. You can back up the database by using the `BACKUP` command. Query the `TABLE_NAME` column in the `USER_TABLES` data dictionary view. Use the `DBMS_STATS.GENERATE_STATS` procedure.

Convention	Meaning	Example
`lowercase monospace (fixed-width) font`	Lowercase monospace typeface indicates executables, filenames, directory names, and sample user-supplied elements. Such elements include computer and database names, net service names, and connect identifiers, as well as user-supplied database objects and structures, column names, packages and classes, usernames and roles, program units, and parameter values. **Note:** Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.	Enter `sqlplus` to open SQL*Plus. The password is specified in the `orapwd` file. Back up the datafiles and control files in the `/disk1/oracle/dbs` directory. The `department_id`, `department_name`, and `location_id` columns are in the `hr.departments` table. Set the `QUERY_REWRITE_ENABLED` initialization parameter to `true`. Connect as `oe` user. The `JRepUtil` class implements these methods.
`lowercase italic monospace (fixed-width) font`	Lowercase italic monospace font represents placeholders or variables.	You can specify the *`parallel_clause`*. Run `U`*`old_release`*`.SQL` where *`old_release`* refers to the release you installed prior to upgrading.

Conventions in Code Examples

Code examples illustrate SQL, PL/SQL, SQL*Plus, or other command-line statements. They are displayed in a monospace (fixed-width) font and separated from normal text as shown in this example:

```
SELECT username FROM dba_users WHERE username = 'MIGRATE';
```

The following table describes typographic conventions used in code examples and provides examples of their use.

Convention	Meaning	Example
[ ]	Brackets enclose one or more optional items. Do not enter the brackets.	`DECIMAL (`*`digits`*` [ , `*`precision`*` ])`
{ }	Braces enclose two or more items, one of which is required. Do not enter the braces.	`{ENABLE | DISABLE}`
|	A vertical bar represents a choice of two or more options within brackets or braces. Enter one of the options. Do not enter the vertical bar.	`{ENABLE | DISABLE}` `[COMPRESS | NOCOMPRESS]`

Convention	Meaning	Example
. . .	Horizontal ellipsis points indicate either: ■ That we have omitted parts of the code that are not directly related to the example ■ That you can repeat a portion of the code	`CREATE TABLE ... AS subquery;` `SELECT col1, col2, ... , coln FROM employees;`

.
.
.
	Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example.	`SQL> SELECT NAME FROM V$DATAFILE;`
		`NAME`
		`/fsl/dbs/tbs_01.dbf`
		`/fs1/dbs/tbs_02.dbf`
		`.`
		`.`
		`.`
		`/fsl/dbs/tbs_09.dbf`
		`9 rows selected.`

Other notation	You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as shown.	`acctbal NUMBER(11,2);` `acct CONSTANT NUMBER(4) := 3;`
Italics	Italicized text indicates placeholders or variables for which you must supply particular values.	`CONNECT SYSTEM/system_password` `DB_NAME = database_name`

UPPERCASE	Uppercase typeface indicates elements supplied by the system. We show these terms in uppercase in order to distinguish them from terms you define. Unless terms appear in brackets, enter them in the order and with the spelling shown. However, because these terms are not case sensitive, you can enter them in lowercase.	`SELECT last_name, employee_id FROM employees;`
		`SELECT * FROM USER_TABLES;`
		`DROP TABLE hr.employees;`

lowercase	Lowercase typeface indicates programmatic elements that you supply. For example, lowercase indicates names of tables, columns, or files.
	Note: Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.
		`SELECT last_name, employee_id FROM employees;`
		`sqlplus hr/hr`
		`CREATE USER mjones IDENTIFIED BY ty3MU9;`

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 Corporation 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 Corporation does not own or control. Oracle Corporation neither evaluates nor makes any representations regarding the accessibility of these Web sites.

What's New in Oracle OLAP?

Oracle9*i* Release 2 provides multidimensional analysis within the Oracle database. Oracle OLAP is the next generation of analytical engines and related software, providing an upgrade path from Oracle Express Server release 6.3.

See Also:

- Appendix A, "Upgrading From Express Server" for specific differences between Express Server and Oracle OLAP.
- *Oracle9i OLAP Developer's Guide to the OLAP DML* for changes to the OLAP data manipulation language.

The following sections describe the new features in Oracle9*i* OLAP:

Oracle9*i* Release 2 (9.2) New Features in Oracle OLAP

The following list briefly describes the new features of Oracle OLAP.

- **Oracle OLAP is integrated with the Oracle database**

  The OLAP engine runs in the Oracle kernel, and analytic workspaces are stored as LOBs in relational tables.

  See Also: "The Oracle9i Integrated Relational-Multidimensional Database" on page 1-4

- **Oracle OLAP management tools are integrated with Oracle**

  The Oracle DBA uses one set of management tools for both the Oracle database and Oracle OLAP.

  See Also: Chapter 6, "Administering Oracle OLAP"

- **SQL applications can access multidimensional data**

  SQL applications can use the database table functions to access and manipulate data directly in the multidimensional OLAP data cache. Alternatively, relational views can be created for multidimensional data, which provides access to standard SQL.

  See Also: Chapter 3, "Developing OLAP Applications"

- **Tools simplify creation of analytic workspaces and related views**

  Tools are available to help move data from relational tables into multidimensional objects in an analytic workspace, and to generate views of these objects so that applications can access workspace data using standard SQL.

  See Also:

  - Chapter 9, "Creating an Analytic Workspace From Relational Tables"
  - Chapter 15, "CWM2_OLAP_AW_ACCESS"

- **Applications can use OCI or JDBC to connect to Oracle OLAP**

  OLAP applications that used SNAPI communications in Express Server 6.3 and earlier can upgrade to Oracle OLAP without substantially changing the application's Express language-based architecture.

  See Also: Chapter 1, "Overview"

- **OLAP API is available for developing Java applications**

  The Oracle OLAP API is an all-Java application programming interface that is designed specifically to support multidimensional analysis.

  See Also: Chapter 3, "Developing OLAP Applications"

- **OLAP Catalog API supports third-party applications development**

  PL/SQL interfaces to the OLAP Catalog allow developers to query and update the logical multidimensional metadata model and map it to physical relational and analytic workspace data.

  See Also: Chapter 5, "Creating OLAP Catalog Metadata"

- **OLAP metadata provides extended schema support**

  The Oracle OLAP Catalog metadata supports star, snowflake, and multidimensional schema. The metadata supports level-based, parent-child, and complex dimension hierarchies.

  See Also: Chapter 5, "Creating OLAP Catalog Metadata"

- **Oracle Globalization Support extended to Oracle OLAP**

Oracle Globalization Support provides the Oracle standard for internationalizing and localizing Oracle products. The character set encoding supports Unicode using the UTF-8 standard, which is a format that transforms all Unicode characters into a variable-length encoding of bytes. Its use in the database and Oracle OLAP allows text data in native languages to be passed between them without data loss or performance degradation.

See Also: *Oracle9i Database Globalization Support Guide*

Part I

The Basics

Part I contains basic information about multidimensional analysis. It is of interest to anyone who may use Oracle OLAP as a database administrator, an applications developer, or an end user.

This part contains the following chapters:

1

Overview

This chapter explains the basics of using Oracle OLAP and related client software for analytical applications. By reading this chapter, you will get an overview of its features.

This chapter includes the following topics:

Why OLAP?

Relational databases have dominated database technology by providing 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 competitive edge in an increasingly competitive marketplace.

Analytical Processing Answers Business Questions

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. While a standard transactional query might ask, "When did order 84305 ship?" a typical series of analytical queries might ask, "How do sales in the Southwestern region 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?"

The transactional query involves simple data selection and retrieval. However, the analytical queries involve inter-row calculations, time series analysis, and access to aggregated historical and current data. This is online analytical processing — OLAP.

The data processing required to answer analytical questions is fundamentally different from the data processing required to answer transactional questions. Table 1–1 highlights the major differences.

Table 1–1 Characteristics of Transactional And Analytical Queries

Characteristic	Transactional Query	Analytical Query
Typical operation	Update	Analyze
Age of data	Current	Historical
Level of data	Detail	Aggregate
Data required per query	Minimal	Extensive
Querying pattern	Individual queries	Iterative queries

Types of OLAP Applications

Applications that support business analyses fall into these major groups:

- Standard reporting
- Ad-hoc query and reporting
- Multidimensional analytical reporting
- Predictive analysis and planning

Oracle provides the technology for all of these types of applications. Oracle OLAP and its development tools are particularly suited to analytical reporting and predictive analysis applications. This guide will introduce you to the tools for developing these types of applications.

Analytical Reporting

Analytic applications can support many facets of a business and offer high returns on the investment. Here are just a few examples of analytical applications:

- Accounting. Forecasting, budgeting, cost and profitability analyses, and consolidation
- Human Resources. Skills consolidation, labor scheduling and optimization
- Distribution. Scheduling and optimization
- Sales Force Automation. Cross-selling and territory analyses
- Marketing. Churn and market-based analyses
- Retailing. Site location and demographic analyses
- Manufacturing. Demand planning and forecasting
- Health Care. Outcomes analysis
- Financial Services. Risk assessment and management

Predictive Analysis

Planning applications allow 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 analyses systems allow organizations to analyze past performance, build revenue and spending plans, manage toward 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 allow organizations to predict market demand based on factors such as sales history, promotional plans, pricing models, and so forth. They can model different scenarios that forecast product demand and then determine appropriate manufacturing goals.

The Oracle9*i* Integrated Relational-Multidimensional Database

Oracle provides multidimensional technology within the database. Organizations no longer need to choose between a multidimensional OLAP database and a relational database. By integrating OLAP into the database, Oracle provides the power of a multidimensional database while retaining the manageability, scalability, and reliability of the Oracle database and the accessibility of SQL. The Oracle database provides the functionality of a specialized analytic database while eliminating the need for a separate database system.

The advantages of a single integrated relational-multidimensional database when compared to two separate relational and multidimensional databases are many:

- **Simplified management.** All management tasks are consolidated into a single database and can be managed through Oracle Enterprise Manager or PL/SQL.
- **High availability.** Oracle OLAP has the same scalability and high reliability as the Oracle database, including support for Real Application Clusters and Oracle Data Guard. Real Application Clusters allow multiple instances of the database to work cooperatively against a single disk image of the database. When more processing power is needed, another server can be added to the cluster. If a server fails, then another server automatically takes over. Oracle Data Guard protects against complete site failure, for instance, in the event of an unprotected power failure. In the event of site failure, Oracle Data Guard automatically switches to a backup instance at a different site.
- **High security.** Oracle provides complete security to all data in the database, including multidimensional data. All users are defined in a single user catalog and are assigned privileges using standard security features such as roles and privileges. More finely grained access privileges can also be granted.
- **Open access.** Both relational and multidimensional data can be accessed through SQL and the OLAP API. Application developers can choose to use the calculation and data navigation features of the OLAP API, or they can leverage

their investment in SQL to access multidimensional data. Any OLAP calculation can be queried using SQL. Standard reporting applications can present the results of complex multidimensional calculations. Ad-hoc querying tools can provide new calculation functions.

- **Reduced update time.** Oracle allows data to be stored in either relational or multidimensional tables and provides access to both through SQL and the OLAP API. Thus, data does not need to be replicated in two data stores. The typical two-step data maintenance process (update the data warehouse, then update the multidimensional database) is now reduced to a single step. The result is a corresponding reduction in the interval between the time the data is available from the source system and the time the data is available to users for analyses.
- **Improved data reliability.** Because data does not need to be replicated between the relational tables and multidimensional tables, it cannot get out of synchronization. All users have access to the same version of the data as soon as changes are committed to the database.

The Oracle relational database and Oracle OLAP provide complementary functionality to support the most versatile and high performance applications. The database and SQL engine provide detail data, summary management, and one-dimensional calculations using the SQL-99 OLAP extensions. Oracle OLAP expands these capabilities to provide forecasting, modeling, what-if scenarios, and multidimensional calculations.

Components of Oracle OLAP

Analytical queries and predictive analyses require a multidimensional OLAP solution. Oracle OLAP consists of the following components:

- Calculation engine
- Analytic workspaces
- OLAP DML
- PL/SQL table functions
- OLAP API
- OLAP Catalog metadata

This guide explains the relationships among these components from the perspectives of both database administrators and application developers.

Calculation Engine

The OLAP calculation 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. The OLAP engine runs within the Oracle kernel.

Analytic Workspace

An analytic workspace stores multidimensional data objects and procedures written in the OLAP DML. Within a single database, many analytic workspaces can be created and shared among users. Like relational tables, 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.

An analytic workspace can be temporary (that is, for the life of the session) or it can be persistent, that is, saved from one session to the next. When an analytic workspace is persistent, the data is stored as LOBs in database tables. Analytic workspaces also provide an alternative to materialized views as a means of storing aggregate data.

OLAP DML

The OLAP DML is a data manipulation language that is understood by the Oracle OLAP calculation engine. The OLAP DML extends the analytical capabilities of querying languages such as SQL and the OLAP API to include forecasting, modeling, and what-if scenarios. Application developers can create stored procedures that use conditional logic and the extensive library of DML commands and functions to perform complex analyses of data. Moreover, the OLAP DML is a very accessible calculation language, similar to that of a spreadsheet, which is easy for power users and DBAs to learn and use.

OLAP DML commands and functions include the following categories:

Aggregation
Allocation
Data Selection
Date and Time Operations
File Reading and Writing
Financial Operations
Forecasts and Regressions

Numeric Manipulation
Models
Statistical Operations
Text Manipulation
Time Series Manipulation

Both the OLAP API and PL/SQL can embed OLAP DML commands in their syntax.

Using the OLAP DML, database administrators and application developers can create multidimensional data objects that are stored in an analytic workspace. The OLAP DML operates on data that is stored (permanently or temporarily) in these multidimensional objects.

> **See Also:** Chapter 2, "Manipulating Multidimensional Data" for more information about using the OLAP DML.

SQL Table Functions

SQL table functions can take a set of rows as input and produce a set of rows as output that can be queried like a physical database table. Application developers who use SQL can access SQL packages that use table functions to create views of multidimensional data. SQL applications can then access these views. Thus, the calculation engine and multidimensional data sources are accessible to SQL, making analytic and predictive functions available to SQL-based applications. SQL applications can connect to the database using either the Oracle Call Interface (OCI) or Java Database Connectivity (JDBC).

> **See Also:** Chapter 3, "Developing OLAP Applications" for more information about using SQL table functions.

OLAP API

The Oracle OLAP API is an application programming interface to Oracle OLAP. It is a querying language that selects and manipulates data for display in a Java client. Because the OLAP API is all Java, it supports deployment of analytical applications to large, geographically distributed user communities on the Internet. It is object oriented, so that application developers define the results they want, not the process by which the results are obtained. The OLAP API connects to the database using JDBC.

The OLAP API is the technology underlying the Oracle BI Beans for access to relational and multidimensional data. JavaBeans are the building blocks of

application development. They are reusable pieces of Java code that can be assembled quickly into an application. The Oracle BI Beans provide pre-built OLAP-aware application building blocks: Connecting to a database; authenticating user credentials; selecting and fetching data; and displaying the data in a variety of tabular and graphical formats. Using the BI Beans, developers can create applications with a common "look and feel," enabling users to gain expertise quickly in the new product.

The BI Beans can be used within Oracle JDeveloper or other Java development environments to build analytical applications, which can be deployed as either Java or HTML clients.

> **See Also:** Chapter 3, "Developing OLAP Applications" for a more detailed introduction to the OLAP API and the BI Beans.

OLAP Catalog

Metadata is typically defined as "data about data." OLAP Catalog metadata is created and stored in relational tables in the database. OLAP applications can query this metadata repository to find out what data is available for analyses and display. The metadata contains information about the physical location of the data, that is, whether it is stored in a relational table or in an analytic workspace. The application does not need to be aware of the location of the data or alter its processing to accommodate the storage location. Since the data is queried using SQL, data from relational data and multidimensional data can be joined in a single SQL query.

Whether the data is stored in a relational schema or in an analytic workspace, the metadata identifies the data in terms of the multidimensional objects: measures, dimensions, levels, and attributes. The metadata provides information critical to the selection, manipulation, and display of that data.

> **See Also:** Chapter 4, "Designing Your Database for OLAP" for information about creating OLAP metadata.

Applications Access to Oracle OLAP

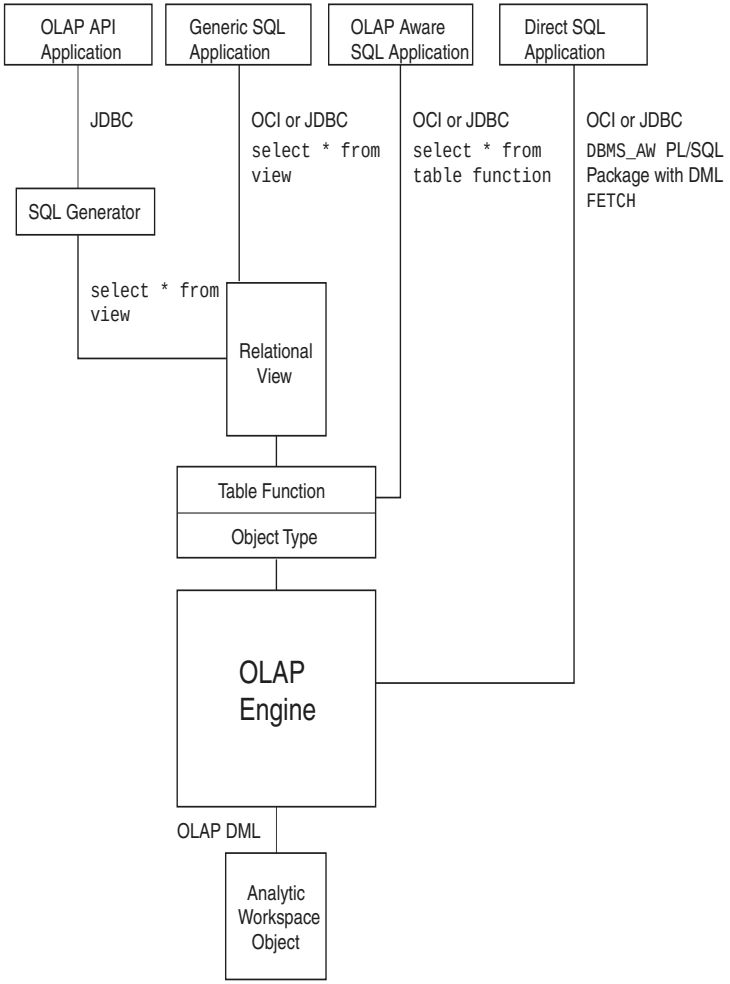
On a very basic level, all applications have access to analytic workspaces and the computational engine using SQL, but the application can be unaware of the SQL or unaware of the underlying OLAP facilities. They all use OCI or JDBC for their communications protocol.

However, at a higher level, different types of applications can access the computational power of Oracle OLAP in different ways.

- OLAP API clients are written in Java, which the SQL generator in Oracle OLAP converts to SQL. The application developer does not need to be familiar either with SQL or the OLAP DML.
- SQL-based applications can use pure SQL against relational views of multidimensional data. The application developer only needs to know SQL and the language of the user interface, such as C++. However, an application developer who is familiar with the OLAP DML can manipulate multidimensional data directly using DML commands embedded in SQL table functions.
- OLAP applications can operate directly on multidimensional data by making use of the conditional processing capabilities of stored procedures written in the OLAP DML.

Figure 1–1 illustrates these methods.

Figure 1–1 Methods of Querying Analytic Workspaces



2

Manipulating Multidimensional Data

This chapter provides an overview of data manipulation using the OLAP DML. It includes the following topics:

What Is the OLAP DML?

The OLAP DML is a data manipulation language. You can use DML commands and functions to perform complex analysis of data. You can also write stored procedures that contain DML commands and functions.

Extensive Analytic Capabilities

The OLAP DML enables application developers to extend the analytical capabilities of querying languages such as SQL and the OLAP API. These are some situations in which you might use the OLAP DML:

- When you need to calculate data that cannot be calculated as part of your data warehouse extraction, transformation, and load (ETL) process or in SQL. Examples include forecasts, solving a model, some types of consolidations (aggregations), and allocations.
- When your application needs to perform various calculations, but you do not want to immediately commit the results in SQL tables. For example, you might have a forecasting application where you want to allow users to save personal forecasts and reuse them during a later session, but you do not want users to commit the forecast to the SQL tables. Instead, you can just commit the data to the analytic workspace without committing it to SQL tables.
- When you want to manipulate data that is stored in an analytic workspace. An analytic workspace can be an alternative to materialized views for storing aggregate data. It may also be the preferred storage location for data that is frequently used in business analyses such as models and forecasts.

See Also:

- *Oracle9i OLAP Developer's Guide to the OLAP DML*
- Oracle9i OLAP DML Reference help

Features of the Multidimensional Model

There are inherent features of the multidimensional model that make it an appropriate environment for business intelligence. The multidimensional model:

- Enforces referential integrity. Each dimension member is unique and cannot be NA. If a measure has three dimensions, then each data value of that measure must be qualified by a member of each dimension.
- Promotes consistency. Dimensions are maintained as separate workspace objects and are shared by measures.
- Preserves the order of data. Each dimension has a default status list, which contains 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, the 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, the function call

  ```
  lag(sales, 12, month)
  ```

  compares the sales values of all months in the current status list against sales from a year ago (that is, 12 time periods earlier in the default status list for the `month` dimension).
- Presents data as fully solved. Applications do not need to define calculations. Because of the combination of power and ease-of-use of the OLAP DML, the analytic workspace can be prepared so that the data is presented as fully solved to the application.
- Manages calculated members and measures transparently. Users can define their own dimension members (often called **custom aggregates**), which function identically to the other dimension members and can be used transparently in any calculation. Similarly, users can define their own measures and assign values to them using any of the methods available in the OLAP DML. Throughout the session, these additions behave identically to the dimension members and objects originally provided in the workspace. Users can save their changes from one session to the next with a single DML command.

Basic Categories of OLAP DML Commands

Following are descriptions of some of the basic categories of OLAP DML commands and functions.

Aggregation

The OLAP DML supports a variety of aggregation methods including first, last, average, weighted average, and sum. In a multidimensional data object, the aggregation method can vary by dimension. Some of the data can be aggregated and stored, while other data is aggregated at runtime. A technique called "skip level" aggregation pre-aggregates every other level in a dimension hierarchy. The DBA can choose whatever method seems appropriate: by level, individual member, member attribute, time range, data value, or other criteria.

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. Some of the key features of the allocation system are:

- Support for hierarchies so the data is distributed based on parentage.
- Support for arbitrary selections so that data is distributed among selected members, regardless of parentage or in the absence of a hierarchy.
- A variety of allocation methods, including:
  - Copy methods (hierarchical copy, minimum, maximum, first, last)
  - Even distribution (even, hierarchical even)
  - Proportional distribution (including weighted distributions and user-defined multidimensional functions.
- Cell-level locking prevents certain cells from being overwritten by the allocation. This feature is used when some values for the planning period are known.
- Logging records how far an allocation has progressed and whether any errors have occurred.

Data Selection

Data selection within the analytic workspace is persistent throughout a session, which is a feature that supports the iterative nature of analytic queries. Users can select data in multiple steps, with each step refining the previous query. The OLAP DML provides data selection methods that are specifically designed for multidimensional data, such as hierarchical relations, levels of aggregation, attributes, time series functions, and data values.

Data Exchange

SQL statements can be embedded in the OLAP DML, which allows applications to select data from SQL tables and write data back to them. This can be done at runtime or as a data maintenance procedure. Access to SQL tables is controlled by the privileges and roles granted to the user's database ID.

The following embedded SQL statements define a cursor and fetch data from a relational table named `products` into a workspace dimension named `prod` and a measure named `prod_label`.

```
SQL DECLARE highprice CURSOR FOR SELECT prod_id, prod_name -
     FROM products WHERE suggested_price > :set_price
SQL OPEN highprice
SQL FETCH highprice LOOP INTO :prod, :prod_label
```

File Reading and Writing

Data can be read from flat files or spreadsheets into multidimensional objects. This is typically done as a data maintenance procedure. Access to external files is controlled by BFILE security. DBAs can set up aliases for directories and control which users and groups can use those aliases, as described in "Controlling Access to External Files" on page 6-10. The security system does not allow users to access directories without an alias.

The following program copies data from a file named `unit` and stores it in a dimensions named `month` and `productid` and variables named `productname` and `units.sold`. The DBA previously created a directory alias named `mydat`.

```
DEFINE read.product PROGRAM
PROGRAM
VARIABLE fi INT                  "Define a local integer variable
fi = FILEOPEN('mydat/unit' READ) "Store a file handle in the variable
```

```
FILEREAD fi COLUMN 1 WIDTH 5 month -
     COLUMN 6 WIDTH 6 productid -
     COLUMN 12 WIDTH 30 productname -
     COLUMN 44 WIDTH 22 units.sold
FILECLOSE fi
END
```

The next example creates a file named `custom.eif` as a private data store that contains the data and definitions for a custom measure named `mysales`. The user can import `mysales` during another session.

```
EXPORT mysales TO EIF FILE 'userdat/custom.eif' DATA DFNS
```

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)
```

Forecasts and Regressions

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

For example, the following FORECAST command uses the EXPONENTIAL method to forecast sales for the next 12 months based on historical data stored in the `sales` measure. It stores the results of the calculation in a second measure named `fcst.sales`.

```
FORECAST LENGTH 12 METHOD EXPONENTIAL FCNAME fcst.sales TIME month sales
```

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.

The following is an example of a modeling program.

```
'cost of goods' = 'raw materials'+labor+'fixed overhead'
'fixed overhead' = 'capital equipment'+'building costs'
'building costs' = 'building depreciation'+electric+heat+maintenance
'labor' = salary+benefits
'capital equipment' = 'equipment maintenance'+'equipment depreciation'
```

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 `CEIL` function returns the smallest whole number greater than or equal to a specified number. The function call

```
CEIL(-6.457)
```

returns a value of `-6`.

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.

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
```

Time Series Manipulation

The time series functions perform operations such as lead, lag, and moving average. For example, the MOVINGTOTAL function calculates a series of totals over time. The following example returns a 3-month total on the sales measure for all currently selected months.

```
MOVINGTOTAL(sales, -2, 0, 1, month)
```

Methods of Executing OLAP DML Commands

The OLAP DML can be used when you want to perform calculations that are not easily accomplished in the ETL process or using SQL (either directly or using the OLAP API). The results can be calculated as part of the data warehouse build and update process, and can optionally be written to SQL tables. Alternatively, applications developers can create OLAP DML programs using the OLAP Worksheet and execute them by embedding OLAP DML in their SQL- or Java-based applications.

OLAP Worksheet: The OLAP DML Development Tool

OLAP Worksheet is an interactive command line interface to Oracle that you can use to perform the following tasks:

- Connect to an analytic workspace
- Execute OLAP DML commands
- Execute SQL statements
- Create and populate data objects
- Create, modify, compile, and execute OLAP DML programs

OLAP Worksheet has a command input window and a program edit window.

Procedure: Open OLAP Worksheet

To open OLAP Worksheet, use the following steps:

1. In Enterprise Manager, connect to a database that is enabled for OLAP.
2. Expand the database.
3. Choose **Warehouse**.
4. From the right mouse menu, choose **Oracle OLAP Worksheet**.

Once you have opened OLAP Worksheet, you can use its menus to establish a connection to Oracle, open a workspace, execute OLAP DML commands, execute SQL statements, or write OLAP DML programs, save any changes, and close the connection.

Embedding OLAP DML Commands in Programs

Applications developers can embed OLAP DML in their SQL- or Java-based applications:

- 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.

See Also:

- *Oracle9i OLAP Developer's Guide to the OLAP DML* for further information about the OLAP DML and the OLAP Worksheet
- OLAP API Javadoc for a description of the `SPLExecutor` class.
- Chapter 11, "DBMS_AW" for descriptions of the procedures in the DBMS_AW package.

3

Developing OLAP Applications

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 SQL-Based OLAP Applications
- Building Analytical Java Applications
- Introducing the BI Beans
- Understanding the OLAP API

Building SQL-Based OLAP Applications

SQL-based applications can access multidimensional data, which is stored in analytic workspaces. Two mechanisms in the database's object technology make this possible:

- Object types (also called abstract data types or ADT) are the basis for object-oriented programming in PL/SQL. An object type encapsulates a data structure along with the functions and procedures needed to manipulate the data. When you define an object type using the `CREATE TYPE` statement, you create an abstract template that corresponds to a real-world object.

  In OLAP, these "real-world objects" are measures, dimensions, hierarchies, attributes, and so forth. By defining object types for the objects in an analytic workspace, you can describe the format of multidimensional data to SQL as rows and columns.

- Table functions produce a collection of rows that can be queried like a physical database table. You use a table function instead of the name of a database table, in the `FROM` clause of a query. A table function can take a collection of rows as input.

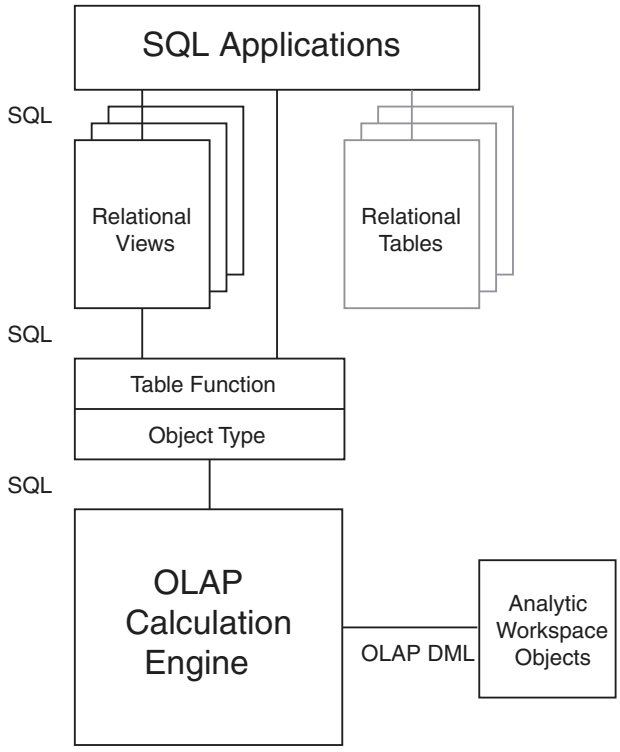
  You can use table functions to fetch data from objects in an analytic workspace. The table functions require arguments that are passed to the OLAP engine, which selects, manipulates, and returns the data. By incorporating table functions into your application, you have the most power and flexibility in selecting and manipulating data in the analytic workspace.

If you overlay the table functions with relational views, then you can make the table functions (and thus the source of the data) transparent to SQL-based applications. Your applications can use standard SQL to run against these views of multidimensional data, the same way that they access other relational tables and views in the database.

> **See Also:** *PL/SQL User's Guide and Reference* for detailed information about object types and table functions.

Figure 3–1 shows how a SQL application can access multidimensional data (using table functions and views) as well as relational data.

Figure 3–1 Components of a SQL-Based Analytical Application



Methods of Accessing Multidimensional Data From SQL

There are several ways that SQL can access the multidimensional data of an analytic workspace. An abstract data type and the table functions underlie all of them. The method that you choose depends on how you want to use the data.

- Use the `CWM2_OLAP_AW_CREATE` package to create the analytic workspace from a star schema. Use other procedures in this package to define a star schema of dimension views and fact views, which represent the measures, dimensions, hierarchies, and attributes in the analytic workspace. You can then query these views using standard SQL `SELECT` statements. You can use other `CWM2` APIs to create OLAP Catalog metadata based on these views.
- If your analytic workspace did not originate from the `AW_CREATE` process, you can use the `CWM2_OLAP_AW_ACCESS` PL/SQL package to generate views of the workspace.

- Write object type definitions and then make relational queries and views of the analytic workspace data by using the `OLAP_TABLE` function in SQL `SELECT` statements. This method is more complex than using `CWM2_OLAP_AW_CREATE` or `CWM2_OLAP_AW_ACCESS`, but it provides more flexibility and power in an application than using predefined views.

  See Also:

  - Chapter 9, "Creating an Analytic Workspace From Relational Tables"
  - Chapter 11, "DBMS_AW"
  - Chapter 15, "CWM2_OLAP_AW_ACCESS"

Embedding OLAP DML Commands in SQL

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 data from the analytic workspace back into relational tables.

While the data is in the analytic workspace, SQL programmers can also issue `SELECT` statements against the data in the analytic workspace using the `OLAP_TABLE` function.

See Also: Chapter 11, "DBMS_AW"

Building Analytical Java Applications

Java is the language of the Internet. Using Java, an application developer can write a standalone application or an *applet*, which is a program that can be included in an HTML page and executed in a browser.

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 allows 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 allows 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. For example, untrusted Java code cannot read to or write from files on the local file system, run programs, load libraries, define native method calls, or make network connections except to the originating host computer. A security manager determines the system resources that an applet can access. However, a signed applet, which identifies itself as being from a trusted source, has full access to system resources the same as local code.

Deploying Java Applications

With the rise in Internet technology, more and more businesses are recognizing the savings they can accrue just by changing the way they deploy their applications.

Traditional **thick client** applications implement many of their functions on the user's computer, thus requiring a large proportion of installed code. However, the days are gone when a team of technicians are required to install and maintain applications software on hundreds or thousands of individual desktop computers for a large user base. Instead, Java thick-client applications download the needed software to client computers automatically at run-time.

Alternatively, system administrators can deploy **thin client** applications that do not download any Java to client computers. These applications run on servers that users world wide can access using Java clients such as their Web browsers. By deploying thin client business intelligence applications on the Internet, businesses can distribute information both within their enterprise and externally to suppliers and customers.

Regardless of whether you choose a thick-client or a thin-client configuration, Java applications provide an immediate solution to the problems inherent in supporting large user communities, which typically are equipped with a variety of incompatible hardware and software platforms.

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 fetches data stored in a data warehouse into the OLAP multidimensional data cache for manipulation by its analytical engine. 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 applications and 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

For more information about the Java programming language, browse the Sun Microsystems Java Web site at `http://java.sun.com`. For information about JDeveloper, search the Oracle Web site at `http://www.oracle.com`.

> **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.

Two groups of BI Beans are currently available:

- Presentation Beans display the data in a rich variety of formats so that trends and variations can easily be detected. Among the Presentation BI Beans currently available are Graph, Table, and Crosstabs.

  The Presentation Beans can be implemented as a thick client or a thin client. Thick 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 thick 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.

- OLAP BI Beans acquire and manipulate the data. The OLAP BI 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 BI Beans for display.

You can use the BI Beans in either thick-client or thin-client applications.

> **See Also:** For more information about the BI Beans, go to the Oracle Web site at `http://www.oracle.com`.

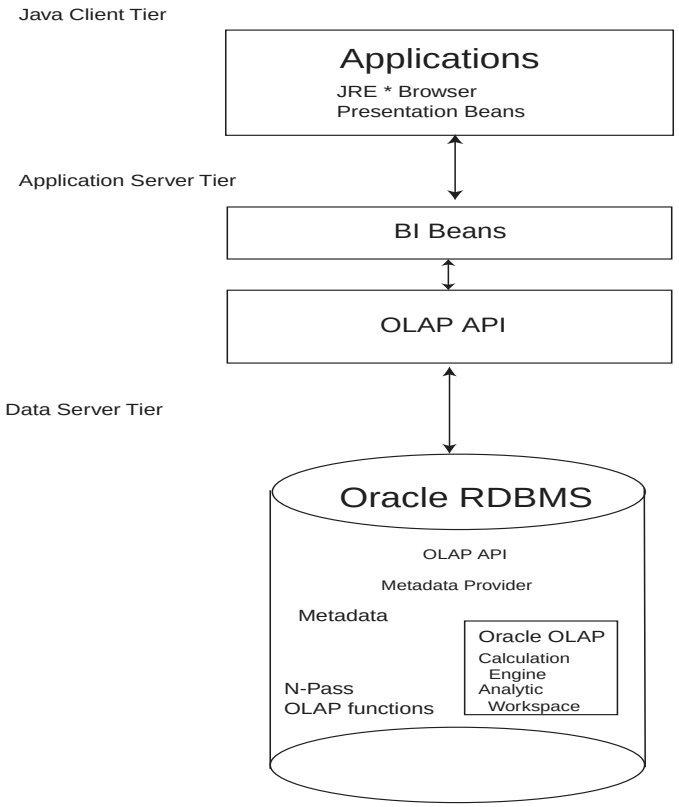
Thick-Client Configuration

The components of an OLAP thick-client application are grouped into three tiers, which can be on separate platforms or the same platform:

- Java client tier. A Java application can run either in a browser or directly in the Java Runtime Environment (JRE). The BI Beans that are dedicated to presenting the data and metadata also run on this tier.
- Application server tier. The "brains" of the application run on this tier, which includes the OLAP API and the OLAP BI Beans that are built using the OLAP API.
- Data server tier. The Oracle RDBMS and OLAP service form the data server tier, where the data is stored, selected, and manipulated. An OLAP API component also runs on the data server tier.

Figure 3–2 shows these relationships in a thick-client configuration.

Figure 3–2 Thick Client Configuration



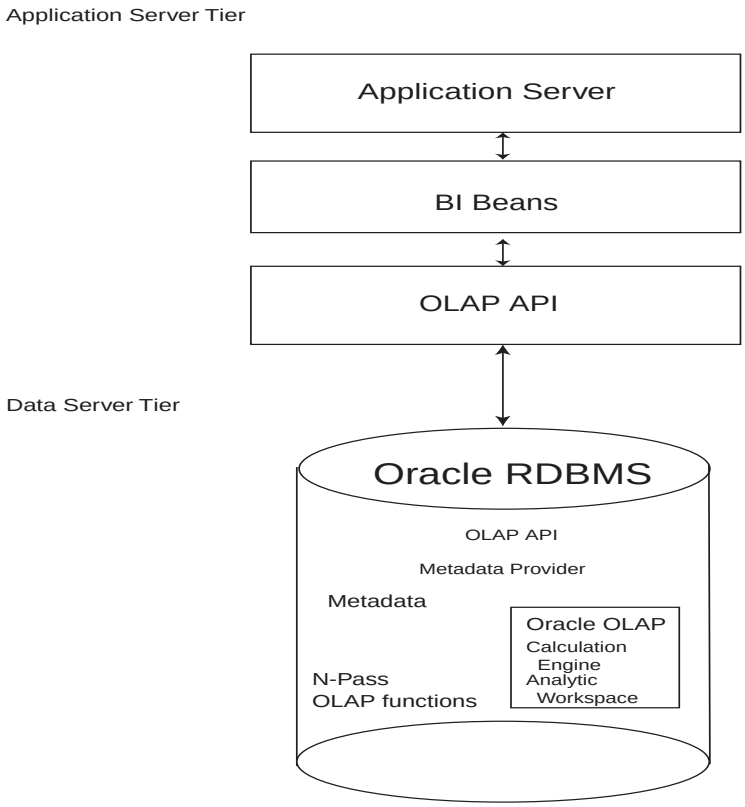
Thin-Client Configuration

The components of an OLAP thin-client application are grouped into two tiers, which can be on separate platforms or the same platform:

- Application server tier. The "brains" of the application run on this tier, which includes a Web server, the OLAP API and the OLAP BI Beans (both presentation and analytical).
- Data server tier. The Oracle RDBMS is the data server tier, where the data is stored, selected, and manipulated either in relational tables or in the OLAP analytic workspace. An OLAP API component also runs on the data server tier.

Figure 3–3 shows these relationships in a thin-client configuration.

Figure 3–3 Thin-Client Configuration



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. For information about metadata and other requirements, refer to Chapter 4, "Designing Your Database for OLAP".

Runtime Repository

The BI Beans employ a runtime repository in the Oracle database that allows users to save their personal analyses and to share their discoveries with other users.

Navigation

The Presentation BI 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 appeared along the X-axis of a graph now appear along the Y-axis, 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 BI Beans allow 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.

OLAP BI Beans

The OLAP BI 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

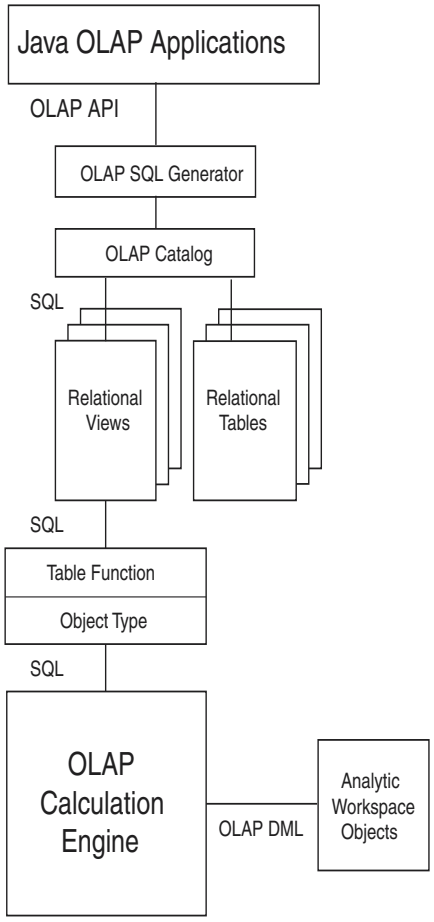
The OLAP API accesses the data through the OLAP Catalog, that is, the relational tables that contain OLAP metadata. The application does not need to be aware of whether the data is located in relational tables or in an analytic workspace, nor does it need to know the mechanism for accessing that data.

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.

Figure 3–4 shows how a query in the OLAP API that uses data from both a multidimensional workspace object and a relational table is resolved.

Figure 3–4 Accessing Relational and Multidimensional Data Using the OLAP API



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:

- *Oracle9i OLAP Developer's Guide to the OLAP API*
- OLAP API Javadoc

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 allows 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.

To acquire data from a data warehouse, the OLAP API generates SQL statements. Data fetches use many of the newest innovations in Oracle9*i*, including concatenated rollup, scrollable cursors, and query rewrite.

Calculation Capabilities

The OLAP API generates SQL commands to select and manipulate data stored in the relational tables. These SQL commands can include the "N-pass" functions, such as `RANK`, `PERCENTILE`, `TOPN`, `BOTTOMN`, `LAG`, `LEAD`, `SUM`, `AVG`, `MIN`, `MAX`, `COUNT`, and `STDDEV`.

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
- Row and column calculations
- 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

The OLAP engine performs additional calculations, such as:

- Modeling
- Forecasting
- What-if scenarios

These types of analysis can be performed on data in the analytic workspace.

See Also:

- *Oracle9i OLAP Developer's Guide to the OLAP API*
- *Oracle9i OLAP Developer's Guide to the OLAP DML*

Example 3–1 Selecting Values

This OLAP API code fragment demonstrates the selection of dimension values based on the data values of a measure. The Sales measure has four dimensions. The Geography, Channel, and Time dimensions are limited to one member each, then Product members are selected with Sales values greater than 20,000,000.

```
Source geographySel = geography.selectValue("BOSTON");
Source channelSel = channel.selectValue("TOTALCHANNEL");
Source timeSel = time.selectValue("1996");
Source prodSel = product.select(salesSel.gt(20000000));
Source result = sales.join(geographySel).
      join(channelSel).join(timeSel).join(prodSel);
```

4

Designing Your Database for OLAP

This chapter highlights some of the most important data warehousing concepts as they pertain to Oracle OLAP. It contains additional information that is specific to a data warehouse that will support applications that use OLAP Catalog metadata, such as the OLAP API and the BI Beans.

This chapter includes the following topics:

Overview

This chapter provides concepts and background to help you start the process of enabling your data warehouse for access by Oracle OLAP client applications. The OLAP API has special requirements that are discussed in this chapter. If you are developing a SQL application, you may still benefit from the discussion of OLAP concepts. Moreover, SQL applications can also be implemented to use OLAP Catalog metadata, like the OLAP API.

This chapter presumes that the relational data stores in your warehouse have already been generated. For this purpose, you may have used Oracle Warehouse Builder or some other Extraction Transformation Transport (ETT) tool. This chapter does not provide sufficient information for you to build a relational data warehouse of your own, or even to fully understand the issues involved in creating and maintaining the relational structures for storing warehouse data.

> **See Also:** *Oracle9i Data Warehousing Guide* for a detailed discussion of data warehousing concepts as they apply to storage in relational tables and data manipulation in SQL.

Preparing a Database for the OLAP API

Oracle provides specialized facilities for the development and deployment of Java-based OLAP clients: the OLAP API and the BI Beans (Business Intelligence Beans). The OLAP API directly queries the data warehouse. The BI Beans may be used as a layer between the end user and the OLAP API.

The OLAP API requires the presence of OLAP Catalog metadata in the database. You will need to take these steps to prepare your data warehouse for the Oracle OLAP API:

1. Design and implement the relational and/or multidimensional data stores to be used by analytical applications.
2. Create the OLAP Catalog metadata.
3. Create the special materialized views that are used by the Oracle OLAP API.

The information that you need to perform these steps is introduced in this chapter.

See Also:

- Chapter 5, "Creating OLAP Catalog Metadata" provides detailed information about the tools and APIs you can use to enable various warehouse configurations for OLAP access.
- The syntax of the PL/SQL APIs that create and display OLAP Catalog metadata are documented in Part IV, "OLAP Catalog Metadata API Reference".

Types of Data Stored in a Data Warehouse

The term **data warehouse** is used to distinguish a database that is used for business analysis (OLAP) rather than transaction processing (OLTP). While an OLTP database contains current low-level data and is typically optimized for the selection and retrieval of records, a data warehouse typically contains aggregated historical data and is optimized for particular types of analyses, depending upon the client applications.

The contents of your data warehouse depends on the requirements of your users. They should be able to tell you what type of data they want to view and at what levels of aggregation they want to be able to view it.

Your data warehouse will store these types of data:

- Historical data
- Derived data
- Metadata

These types of data are discussed individually.

Historical Data

A data warehouse typically contains several years of historical data. The amount of data that you decide to make available depends on available disk space and the types of analysis that you want to support. This data can come from your transactional database archives or other sources.

Some applications might perform analyses that require data at lower levels than users typically view it. You will need to check with the application builder or the application's documentation for those types of data requirements.

Derived Data

Derived data is generated from existing data using a mathematical operation or a data transformation. It can be created as part of a database maintenance operation or generated at run-time in response to a query.

Metadata

Metadata is data that describes the data and schema objects, and is used by applications to fetch and compute the data correctly.

OLAP Catalog metadata is designed specifically for use with Oracle OLAP. It is required by the Java-based Oracle OLAP API, and can also be used by SQL-based applications to query the database.

Data Structures in Relational and Multidimensional Data Stores

Oracle offers both relational and multidimensional storage within a single database. Historical and derived data can be stored either in relational tables or in multidimensional objects.

Relational Table Storage

The lowest level of historical data, as well as fully aggregated historical data, can be stored in **fact tables** in your data warehouse. The lowest level in a data warehouse is typically at a much higher level than in the transactional database. The transactional data should be aggregated to a base level where patterns and trends can emerge and analysis is meaningful, before being stored in the data warehouse. For example, individual purchase orders might be aggregated by sales representative, zip code, or some other demographic feature.

Dimension tables, also called **lookup tables**, are used to store the dimension members that determine the aggregation criteria for fact data. Dimension members are typically organized in levels that roll up within hierarchies.

The Oracle RDBMS provides **materialized views** for storing precomputed data derived from fact tables. Materialized views significantly improve querying times because the aggregates are computed and stored as a database administration task for everyone's use, that is, when the data is refreshed rather than each time the aggregates are needed.

Multidimensional Table Storage

As an alternative to relational table storage, data can be stored in multidimensional objects in analytic workspaces. **Analytic workspaces** are multidimensional structures that are designed specifically to support analytic processing. The equivalent of a relational table in an analytic workspace is a **variable**. You can think of variables as multidimensional tables. The historical and derived data in a data warehouse can be distributed between relational tables and workspace variables. Keep in mind that there is no need to duplicate data; it can be stored in tables or variables, but it does not need to be stored in both.

You can use the sophisticated analysis tools of the OLAP DML to generate new data such as forecasts. You have the option of copying this data into relational tables or keeping it exclusively in the analytic workspace. Analytic workspaces are also an alternative to materialized views for generating and storing aggregate data.

> **See Also:** "Choosing a Schema for Your Data" on page 4-7 for a discussion of the merits of these storage alternatives.

Temporary and Persistent Analytic Workspaces

Data can be loaded into analytic workspaces from SQL tables or from flat files. The analytic workspaces can be either temporary or persistent, depending on your needs. If an analytic workspace is needed only to perform a specific calculation and the results of the calculation does not need to persist in the workspace, the workspace can be discarded at the end of the session. This might occur if, for example, an application needs to forecast a small amount of sales data. Since the forecast can be rerun at any time, there might not be any point in saving the results.

Analytic workspaces can also persist across sessions. You might want to save data in an analytic workspace if you have calculated a significant amount of data (for example, a large forecast or the results of solving a model), or if you have aggregated data using non-additive aggregation methods.

Data in analytic workspaces can be shared by many different users. To share data in an analytic workspace, the workspace must be saved in the database during the period of time it is to be shared.

> **See Also:** *Oracle9i OLAP Developer's Guide to the OLAP DML* for detailed instructions on how to create and populate an analytic workspace, and how to manipulate data stored in it.

About Star, Snowflake, Parent-Child, and Multidimensional Schemas

A schema is a collection of database objects. The following types of schemas are characteristic of a relational data warehouse:

- **Star schema**. Consists of one or more fact tables related to one or more dimension tables. The relationships are defined through foreign keys, metadata, or both.
- **Snowflake schema**. A star schema that has been partially or fully normalized to reduce the number of duplicate values in the dimension tables. However, snowflake schema require more joins, which can slow performance.

For example, a star schema might have a single `geography` dimension table with four columns: `city`, `state`, `region`, and `country`. Only the `city` column has predominately unique values, while the other columns have increasing numbers of duplicate values.

A snowflake schema might have three related `geography` dimension tables: One table with two columns (`city` and `state`) that define the relationship between cities and states, a second table with two columns (`state` and `region`) that define the relationship between states and regions, and a third table with two columns (`region` and `country`) that define the relationship between regions and countries.

Star and snowflake schemas use **level-based dimensions**. Their hierarchies are defined by the relationship between levels, and their levels map to columns in dimension tables. Alternatively, a data warehouse schema may use **parent-child dimensions**. In this type of schema, dimension members map to a parent column and a child column. The parent-child combination in a given row expresses a hierarchical relationship.

Your relational tables can be organized in either a level-based schema (star or snowflake) or a parent-child schema.

With Oracle OLAP, your data warehouse storage options are extended to include:

- **Multidimensional schema.** You can think of analytic workspaces as multidimensional schema, since a workspace stores a collection of related objects.

With analytic workspace data, the data warehouse can support multidimensional and hybrid solutions in addition to pure relational storage models. Thus, an Oracle OLAP schema can contain multidimensional analytic workspace objects in addition to fact tables and dimension tables.

Choosing a Schema for Your Data

The types of analyses performed by the OLAP applications that your data warehouse will support determine the best choice of a data repository. You must examine the benefits of each storage method in light of these applications and decide which one most closely matches their requirements. You can choose to store the data for your business analysis applications from these alternatives:

- **Entirely in a relational schema.** During user sessions, SQL commands are used to select and manipulate the data in the relational database.

  Fact tables are the preferred data repository for most query and reporting applications that require read-only access to the data. For these applications, the relational database offers scalability in supporting very large data sets efficiently and manageability with a single set of administrative tools.

- **Entirely in a multidimensional schema.** As a routine maintenance task, data is loaded into dimensions and variables in the analytic workspace from one or more sources (including relational tables and flat files). During user sessions, data is selected and manipulated in the analytic workspace.

  Analytic workspaces should be used as a persistent data store for applications that support predictive analysis functions, such as models, forecasts, and what-if scenarios. Other design choices — such as the types of hierarchies, the use of non-additive aggregation methods, or storage issues concerning aggregate data — may make workspace objects the preferred data repository.

- **Distributed between a relational schema and a multidimensional schema.** The implementation of this model can, of course, vary widely since it encompasses any scheme that draws on the other two methods. A distributed solution may be desirable when an application requires the advanced calculation capabilities of the analytic workspace combined with the efficient storage of standard relational tables.

As explained in "How the OLAP API Accesses Multidimensional Data" on page 3-13, the storage location of data is transparent to applications that use OLAP metadata to identify data objects. Thus, database administrators can fine-tune the database by moving data between relational tables and analytic workspaces without breaking existing Java applications that use the OLAP API.

OLAP Metadata Model

The basic data model in a relational database is a table composed of one or more columns of data. All of the data is stored in columns. In contrast, the basic data model for multidimensional analysis is a **cube**, which is composed of Measures, Dimensions, and Attributes.

Within the OLAP Catalog, you identify whether the data will function as a measure, a dimension, or an attribute. Once these decisions are stored in the OLAP Catalog metadata, the OLAP API can access warehouse data without regard to its underlying storage format. Whether the data is stored in relational tables, analytic workspaces, or some combination of relational and multidimensional schemas, the OLAP Catalog presents the same logical model to applications that use the OLAP API.

The OLAP Catalog metadata informs applications about the data that is available within the database. The application can then define multidimensional objects to represent that data. When the application runs, it instantiates these objects and populates them with data.

Before you can create metadata, you must know what data users want to view and at what levels they want to view it. If you have already created a data warehouse, then you have already done most of this research. You only need to verify that the requirements haven't changed for the analytical applications that will be run using Oracle OLAP.

Note: The OLAP API uses OLAP metadata. Even if you have created other types of metadata to support other applications, you must create OLAP metadata for applications written in the OLAP API.

Keep in mind that the OLAP API only has access to objects in the database through the metadata definitions. Thus, if an object (such as a column in a table) has not been defined in the metadata, then it is not available to the OLAP API.

Mapping Data Objects to Metadata Objects

The objects comprising a data warehouse and Oracle OLAP metadata use different data structures. The data objects in your data warehouse are represented to the OLAP metadata catalog in the following relational objects, regardless of whether the data is actually stored in relational tables or workspace variables:

- Fact Tables or Views
- Level-based dimension Tables or Views

Oracle OLAP metadata catalog maps the data warehouse schema to these multidimensional data objects:

- Measures
- Dimensions
- Dimension attributes
- Levels
- Level attributes
- Hierarchies
- Cubes
- Measure folders

Measures

Measures are the same as facts. The term "fact" is typically used in relational databases, and the term "measure" is typically used in multidimensional applications.

Measures are thus located in fact tables. A fact table has columns that store measures (or facts) and foreign key columns that create the association with dimension tables.

Measures contain the data that you wish to analyze, such as Sales or Cost. OLAP Catalog metadata requires that a column have a numerical or a date data type to be identified as a measure. Most frequently, a measure is numerical and additive.

Note: The OLAP API supports native Java data types. It does not support the following Oracle data types: BLOB, CLOB, NCLOB, RAW, and LONG RAW. Do not create measures from facts with these unsupported data types.

The OLAP DML supports CLOB and NCLOB data types. Search for "SQL (FETCH)" in the Oracle9i OLAP DML Reference help for additional information about supported data types.

Dimensions

Dimensions identify and categorize your data. Dimension members are stored in a dimension table. Each column represents a particular level in a hierarchy. In a star schema, the columns are all in the same table; in a snowflake schema, the columns are in separate tables for each level.

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 `unit_cost` measure has two dimensions: `products_dim` and `times_dim`. A value of `unit_cost` (`21.60`) is only meaningful when it is qualified by a specific product code (`1575`) and a time period (`28-jan-1998`).

If you use Oracle Enterprise Manager to create OLAP metadata, then defining a dimension in your data warehouse creates a database dimension object, in addition to creating metadata. A dimension object contains the details of the parent-child relationship between columns in a dimension table; it does not contain data.

> **Note:** A dimension object is not created when you use the `CWM2` PL/SQL procedures to create OLAP metadata.

The database dimension object is used by the Summary Advisor and query rewrite to optimize your data warehouse.

Time Dimensions

OLAP metadata considers time dimensions to be distinct from other dimensions. When you specify a dimension in the OLAP metadata, you must identify whether it is a time dimension. A time dimension has special attributes that support both regular and irregular time periods.

Regular time periods, such as weeks, months, and years, are evident on standard calendars. Typically, they neither overlap nor have gaps between them.

Irregular time periods, such as promotional schedules and seasonal time periods, are not evident on standard calendars. They often overlap (even to the extent that one time period is a subset of another time period) or have gaps between them.

The time dimension table should contain the following columns to provide full time support:

- Values for all dimension members, with a column for each level of summarization (such as weeks, quarters, and years).

- An end-date attribute for each level, such as `WEEK_ENDDATE`, `QUARTER_ENDDATE`, and `YEAR_ENDDATE`. These columns must have a `DATE` data type. Their values identify the last day in the time period.
- A time-span attribute for each level, such as `WEEK_TIMESPAN`, `QUARTER_TIMESPAN`, and `YEAR_TIMESPAN`. These columns must have a `NUMBER` data type. Their values identify the number of days in the period.

> **Note:** The OLAP Management feature of Oracle Enterprise Manager provides support for creating and populating time dimension tables with these columns.

Example 4–1 Time Dimension in a Star Schema

The following table describes a dimension table in a star schema.

Column Name	Sample Value	Data Type	Comment
WEEK_ID	W12000	VARCHAR2	Level 1
WEEK_DESC	Week Ending January 8, 2000	VARCHAR2	Attribute
WEEK_ENDDATE	8-JAN-00	DATE	Attribute
WEEK_TIMESPAN	7	NUMBER	Attribute
QUARTER_ID	1QTR2000	VARCHAR2	Level 2
QUARTER_DESC	1st Quarter in Year 2000	VARCHAR2	Attribute
QUARTER_ENDDATE	31-MAR-00	DATE	Attribute
QUARTER_TIMESPAN	91	NUMBER	Attribute
YEAR_ID	YR2000	VARCHAR2	Level 3
YEAR_DESC	Year 2000	VARCHAR2	Attribute
YEAR_ENDDATE	31-DEC-00	DATE	Attribute
YEAR_TIMESPAN	366	NUMBER	Attribute

Example 4–2 Time Dimension in a Snowflake Schema

The following tables describe dimension tables in a snowflake schema. The first table defines weeks, which is the lowest level of time data.

Column Name	Sample Value	Data Type	Comment
WEEK_ID	W12000	VARCHAR2	Level 1
WEEK_DESC	Week Ending January 8, 2000	VARCHAR2	Attribute
WEEK_ENDDATE	8-JAN-00	DATE	Attribute
WEEK_TIMESPAN	7	NUMBER	Attribute

A second table defines quarters.

Column Name	Sample Value	Data Type	Comment
WEEK_ID	W12000	VARCHAR2	Foreign key
QUARTER_ID	1QTR2000	VARCHAR2	Level 2
QUARTER_DESC	1st Quarter in Year 2000	VARCHAR2	Attribute
QUARTER_ENDDATE	31-MAR-00	DATE	Attribute
QUARTER_TIMESPAN	91	NUMBER	Attribute

A third table defines years.

Column Name	Sample Value	Data Type	Comment
QUARTER_ID	1QTR2000	VARCHAR2	Foreign key
YEAR_ID	YR2000	VARCHAR2	Level 3
YEAR_DESC	Year 2000	VARCHAR2	Attribute
YEAR_ENDDATE	31-DEC-00	DATE	Attribute
YEAR_TIMESPAN	366	NUMBER	Attribute

Hierarchical Dimensions

A **hierarchy** is a way to organize data according to levels. Dimensions are structured hierarchically so that data at different levels of aggregation can be manipulated together efficiently for analysis and display. Dimension hierarchies enable users to recognize trends at one level of aggregation, 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. Levels group the data for aggregation and are used internally for computation. Each level above the base (or lowest) level represents the aggregate total of the levels below it. For example, a `time` dimension might have `day`, `week`, `quarter`, and `year` for the levels of a hierarchy. If data for the `sales` measure is stored in days, then the higher levels of the `time` dimension allow the `sales` data to be aggregated correctly into weeks, quarters, and years. Days roll up into weeks, weeks into quarters, and quarters into years.

The members of a hierarchy at different levels have a one-to-many *parent-child* relationship. For example, `qtr1` and `qtr2` are the children of `yr2001`, thus `yr2001` is the parent of `qtr1` and `qtr2`.

Attributes

Attributes provide descriptive information about the data and are typically used for display.

Level Attributes

Level attributes provide supplementary information about the dimension members at a particular level of a dimension hierarchy. The dimension members themselves may be meaningless, such as a value of "1296" for a time period. These cryptic values for dimension members are used internally for selecting and sorting quickly, but are meaningless to users.

For example, you might have columns for employee number (`ENUM`), last name (`LAST_NAME`), first name (`FIRST_NAME`), and telephone extension (`TELNO`). `ENUM` is the best choice for a level column, since it is a key column and its values uniquely identify the employees. `ENUM` also has a `NUMBER` data type, which makes it more efficient than a text column for the creation of indexes. `LAST_NAME`, `FIRST_NAME`, and `TELNO` are attributes. Even though they are dimensioned by `ENUM`, they do not make suitable measures because they are descriptive text rather than business measurements.

Dimension Attributes

Dimension attributes specify groupings of level attributes for a specific dimension. Whereas level attributes map to specific data values, dimension attributes are purely logical metadata objects.

An example of a dimension attribute is `end date`, which is required for time dimensions. If a time dimension has month, quarter, and year levels, `end date` identifies the last date of each month, each quarter, and each year. Within a relational schema, the three level attributes that make up the `end date` dimension attribute would be stored in columns with names like `month_end_date`, `quarter_end_date`, and `year_end_date`.

Cubes

Cubes are the metadata objects that associate measures with their dimensions. All the measures associated with a cube have the exact same dimensionality.

The edges of a cube are defined by its dimensions. Although there is no limit to the number of edges on a cube, data is often organized for display purposes along three edges, which are referred to as the row edge, column edge, and page edge. A single dimension or multiple dimensions can be placed on an edge. For example, sales data might be displayed with Product and Channel on the row edge, Time on the column edge, and Customer on the page edge.

Measure Folders

Measures can be organized within **measure folders,** which facilitate the browsing of data by business area. Measure folders are also known as **catalogs**.

Whereas dimensions and measures are associated with the schemas that contain their source data, measure folders are schema independent. Each OLAP client can view all measure folders defined within the Oracle instance.

5

Creating OLAP Catalog Metadata

This chapter describes the OLAP Catalog and the methods for working with OLAP metadata.

See Also:

- "OLAP Metadata Model" on page 4-8 for detailed descriptions of the logical entities in the OLAP Catalog.
- "Understanding the OLAP API" on page 3-12 for information on the OLAP API and its use of the OLAP Catalog.
- Part IV, "OLAP Catalog Metadata API Reference" for detailed descriptions of the APIs for creating `CWM2` metadata.

This chapter includes the following sections:

Overview of the OLAP Catalog

The repository for OLAP metadata is known as the **OLAP Catalog**. OLAP metadata represents warehouse data as logical cubes, as described in "OLAP Metadata Model" on page 4-8.

OLAP metadata must be defined and mapped to any data that will be accessed by the OLAP API. OLAP metadata may also be used by other types of analytical applications.

OLAP metadata maps to dimension tables and fact tables. The dimension tables must be organized in levels. The dimension tables and fact tables may be actual relational tables or they may be views representing data stored in analytic workspaces. A number of different warehouse configurations can be represented by OLAP metadata, as described in "Data Warehouse Requirements" on page 5-4.

Tools for Creating OLAP Metadata

There are two tools for creating OLAP metadata:

- Oracle Enterprise Manager.
- The `CWM2` PL/SQL APIs.

Note: Enterprise Manager currently uses a set of proprietary APIs to create OLAP metadata. It does not provide access to metadata created with the `CWM2` APIs.

However, the OLAP Catalog metadata views allow you to browse *all* the metadata in the OLAP Catalog. This includes metadata created by Enterprise Manager and metadata created by the `CWM2` APIs.

OLAP Catalog Components

The OLAP Catalog includes the following:

- **Metadata model tables** - A set of tables 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 warehouse data.

- **A 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.
- **A Read API** - A set of SQL views providing information about the metadata registered in the model tables.

Logical Steps for Creating OLAP Metadata

Whether you create OLAP metadata programmatically or by using Oracle Enterprise Manager, you follow the same logical steps.

To create OLAP metadata:

1. Create the dimensions. Specify the levels, attributes, and hierarchies associated with each one.
2. Create cubes and specify their edges (dimensions).
3. Create measures that represent the fact data. Associate each measure with a cube.
4. Map the metadata entities to the source data.
5. Create measure folders in which to store related measures. Populate the folders with measures.

Accessing the OLAP Catalog

To create OLAP metadata, you must be able to log into your database with credentials that have been granted the `OLAP_DBA` role. The OLAP Catalog is owned by the `OLAPSYS` user.

The `OLAP_DBA` role has system privileges associated with it, such as the ability to create and drop tables, indexes, and dimensions. For a list of these privileges, follow these steps:

1. Log into your database through Oracle Enterprise Manager.
2. Expand the Security branch.
3. Choose **OLAP_DBA**.
4. Display the Role and System Privileges pages.

If you have the system DBA role, then you also have the `OLAP_DBA` role. You must also have the `CONNECT` role.

Note: To view existing metadata, you only need the CONNECT and SELECT_CATALOG_ROLE roles.

Data Warehouse Requirements

The CWM2 APIs support and extend Enterprise Manager's warehouse requirements.

Basic Star or Snowflake Schema

Enterprise Manager creates OLAP metadata for star and snowflake schemas. It creates a database dimension object for each logical OLAP dimension. The database dimension object imposes the following restrictions on dimension tables and related fact tables:

- 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 NULLs.
- 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 data warehouse complies with these requirements, you can use either Enterprise Manager or the CWM2 APIs to create OLAP metadata.

Dimension Tables with Complex Hierarchies

If your dimension tables include any of the following variations, you must use the CWM2 APIs to create OLAP metadata:

- Level columns containing NULLs, such as in skip-level hierarchies
- Hierarchies with different lowest levels (sometimes called **ragged hierarchies**)
- Values mapped to different levels for multiple hierarchies

Solved and Unsolved Fact Data

Fact data is **unsolved** when it is stored at the lowest level of aggregation. Fact data is **solved** when it is stored with embedded totals.

Enterprise Manager creates OLAP metadata for unsolved fact data. With the CWM2 APIs, you can create OLAP metadata for both solved and unsolved fact data.

The CWM2 APIs also support multiple fact tables per cube. In this case, the data associated with a given combinations of hierarchies can be stored in a separate fact table. All the fact tables associated with a cube must have the same column structure.

Multidimensional Data

With the CWM2 APIs, you can create and populate analytic workspaces from a star schema and generate relational views of the resulting workspaces. You can then create OLAP metadata based on these views. Use the following procedure.

1. Use the CWM2_OLAP_AW_CREATE package, as described in Chapter 9 to create the workspace and create relational views of the data. These views take the place of fact tables and dimension tables.
2. Use other CWM2 APIs to create OLAP metadata based on these views.

Note: If your data is stored in an analytic workspace that was created in some other way, for example by using OLAP Worksheet or the DBMS_AW package, you can use the CWM2_OLAP_AW_ACCESS package to generate views of the workspace. Then use other CWM2 APIs to create OLAP metadata based on these views.

See Also: Chapter 3, "Developing OLAP Applications" and Chapter 12, "OLAP_TABLE" for more detailed explanations of the technology underlying views of analytic workspace data.

Parent-Child Dimensions

If the dimensions of your data are stored in parent-child dimension tables, then you must convert them to level-based dimensions before creating OLAP metadata. Use the following procedure:

1. Use the CWM2_OLAP_PC_TRANSFORM package, as described in Chapter 25 to convert the parent-child dimensions to a level-based dimensions.
2. Use other CWM2 APIs to create OLAP metadata based on the level-based dimensions.

Creating Metadata Using Oracle Enterprise Manager

If your data warehouse complies with the requirements listed in "Basic Star or Snowflake Schema" on page 5-4, 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.

Note: If you prefer to execute PL/SQL programs directly or your schema does not conform to the requirements of the OLAP Management tool, refer to "Creating Metadata Using PL/SQL" on page 5-9.

Procedure: Accessing OLAP Management

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

1. Open the Oracle Enterprise Manager console.

   You see the main page.

2. Expand **Databases** by clicking the plus sign next to it.

   You see the list of service names for Oracle databases for which you have defined a connection.

   If the database that you want to manage is not listed, then from the **Navigator** menu choose **Add Database to Tree**. You will need to supply the host name, port number, and SID.

3. Expand the database that you want to manage.

   You see the Database Connect Information dialog box.

4. Type in your user name (one with the appropriate credentials) and password for that database.

> **Tip:** Select the **Save as preferred credentials** box if you wish to eliminate this step in future sessions. Your user name and encrypted password will be saved in a local file. For security, make sure that *only you* can run Oracle Enterprise Manager with your stored credentials. Later, if you wish to change this information, then choose **Edit Local Preferred Credentials** from the Configuration menu.

The database folder will expand to show the various tools available for administering the database.

5. Expand **Warehouse**.
6. Expand **OLAP**.

   You see the types of objects that you can create. This part of Oracle Enterprise Manager is 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.

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

Business analysis is performed on historical data, so fully defined time periods are vital. Special support for time dimensions is built into the metadata to allow for time-dependent analyses, such as comparisons with earlier time periods.

Your time dimension table must have columns for end-date and time-span, as described in "Time Dimensions" on page 4-10. Typical levels and hierarchies for time dimensions are suggested by the Dimension Wizard, but you do not have to use them.

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 5-6.
2. To create a new dimension, right click on **Dimensions**, then choose one of the following:
   - **Create Using Wizard** to run the Dimension Wizard

     or

   - **Create** to edit a new dimension property sheet
3. Choose **Help** if you need additional 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. 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.

Follow these steps to create a cube:

1. Start Oracle Enterprise Manager and access OLAP Management, as described in "Procedure: Accessing OLAP Management" on page 5-6.
2. Right-click on **Cubes**, then choose one of the following:
   - **Create Using Wizard** to run the Cube Wizard

     or

   - **Create** to edit a new cube property sheet
3. Choose **Help** if you need additional information.

Viewing a Cube's Data

The Cube Viewer allows you to see the cube that you created in the same way that end-users might see it — with the data presented in a BI Beans crosstab, as described in "Crosstabs" on page 3-11. Moreover, you can select the data that you want to see by using the query builder.

Note: Only cubes created in Enterprise Manager are visible in the Cube Viewer.

Procedure: Viewing a Cube's Data

Follow these steps to view a cube:

1. Start Oracle Enterprise Manager and access OLAP Management, as described in "Procedure: Accessing OLAP Management" on page 5-6.
2. Expand the OLAP tree so that you can see the list of cubes.
3. Right-click on the cube you want to examine, then choose **Cube Viewer**.
4. If you need additional information, then search for the Help topic "Viewing a Cube's Data."

Creating Metadata Using PL/SQL

The `CWM2` PL/SQL packages contain stored procedures that can create OLAP metadata for a variety of data warehouses, as described in "Data Warehouse Requirements" on page 5-4.

Before using these packages, make sure that you have performed any required preprocessing steps, as described in "Multidimensional Data" and "Parent-Child Dimensions" on page 5-5.

> **See Also:** Part IV, "OLAP Catalog Metadata API Reference" for the comprehensive syntax of the CWM2 packages and examples of their use.

Views of OLAP Catalog Metadata

A set of views, identified by the ALL_OLAP2 prefix, presents the metadata in the OLAP Catalog. The metadata may have been created with the CWM2 PL/SQL packages or with Enterprise Manager. These views are described in Chapter 14, "Viewing OLAP Catalog Metadata".

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 Analytic Workspaces

The CWM2_OLAP_AW_CREATE package contains procedures for replicating a star schema within an analytic workspace and creating relational views of the workspace.

The CWM2_OLAP_AW_ACCESS package contains generic procedures for creating relational views of analytic workspaces. These workspaces do not have to be created by CWM2_OLAP_AW_CREATE.

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 OLAP API.

CWM2 Packages for Classification and Validation

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

- CWM_CLASSIFY
- CWM2_OLAP_VALIDATE

Part II

Oracle OLAP Administration

Part II provides information for database administrators on administrative tasks associated with Oracle OLAP.

This part contains the following chapters:

6

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 is 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. Following is a list of these tasks.

- Database configuration. Permanent and temporary tablespaces must be created to prevent I/O bottlenecks, as described in "Creating Tablespaces for Analytic Workspaces". Initialization parameters must also be set to optimize performance.
- Security. Users of OLAP applications must have database identities that have been granted the appropriate access rights. For users to have access to files, aliases for the directories must be created and the access rights must be granted. Refer to "Setting Up User Names" on page 6-9.
- Data maintenance. For data that will be stored in analytic workspaces, stored procedures must be developed in the OLAP DML for copying the data into multidimensional data objects and performing whatever aggregations or other data manipulations are required. Refer to the *Oracle9i OLAP Developer's Guide to the OLAP DML*.

  These tasks are typically performed during off-peak hours using a batch facility, as described in "Monitoring Performance" on page 6-13.
- Data Interfaces. For access by SQL, views of multidimensional objects can be created using table functions, as described in Chapter 3, "Developing OLAP Applications". For access by the OLAP API, OLAP metadata must be defined. Refer to Chapter 5, "Creating OLAP Catalog Metadata".
- Performance. Materialized views must be created for data stored in relational schema. All of the data, whether it is stored in relational tables or multidimensional tables, may require striping and partitioning to gain the best performance. For information about how analytic workspaces are stored in the database, refer to "Understanding Data Storage" on page 6-11. For information about striping and partitioning for relational tables, refer to the *Oracle9i Data Warehousing Guide*.

Initialization Parameters for Oracle OLAP

Several packages described in this guide require that `utl_file_dir` be set. This parameter enables the RDBMS to write to a file.

Table 6–1 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.

Table 6–1 Database Performance Initialization Parameter Settings

Parameter	Setting
`db_cache_size`	Half of physical memory
`parallel_max_servers`	The number of processors minus one This parameter limits the number of processes that are used for a parallel update. The number of parallel processes is also dependent on the number of analytic workspace extension files that are being updated.
`sessions`	2.5 * maximum number of OLAP users

> **See Also:** *Oracle9i SQL Reference* for information about these parameters.

Take the following steps to set system parameters:

1. Open the `init`*`sid`*`.ora` parameters file in a text editor.

   The parameters 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 `OraHome1/olap` directory:

   ```
   UTL_FILE_DIR=/users/oracle/OraHome1/olap
   ```

3. Stop and restart the database, using commands such as the following. Be sure to identify the parameters file in the `STARTUP` command.

   ```
   sqlplus '/ as sysdba'
   shutdown immediate
   startup pfile=/users/oracle/OraHome1/admin/rel9dw/pfile/initrel9dw.ora
   ```

OLAP_PAGE_POOL_SIZE

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 an OLAP session.

The OLAP paging cache is allocated at the start of an OLAP session and released when the user exits the database. An OLAP session can be initiated by the OLAP_TABLE function, the DBMS_AW PL/SQL package, or via command line in OLAP Worksheet.

The minimum value of OLAP_PAGE_POOL_SIZE is 2 MB. The default value is 32 MB.

The OLAP paging cache 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 in multi-threaded server mode (MTS), the UGA is part of the Shared Global Area (SGA).

When the database is running in dedicated mode, you can reset the value of OLAP_PAGE_POOL_SIZE in an ALTER SESSION statement. If you decrease the value, you should first do an UPDATE in the analytic workspace and a COMMIT in the database. If you increase the value to a size greater than the available memory, OLAP_PAGE_POOL_SIZE remains the same.

If OLAP_PAGE_POOL_SIZE is greater than available memory, OLAP session initialization will fail.

For performance reasons, it is generally preferable to use a small OLAP paging cache and a larger DB_CACHE_SIZE. An OLAP paging cache of 4 MB is fairly typical, with 2 MB used for systems with limited memory resources.

Initialization Parameters for the OLAP API

The OLAP API will perform best if the configuration parameters for the database are optimized for this type of use. During installation of the Oracle RDBMS, an OLAP configuration table is created and populated with ALTER SESSION commands that have been tested to optimize the performance of the OLAP API. Each time the OLAP API opens a session, it executes these ALTER SESSION commands.

If a database instance is being used only to support Java applications that use the OLAP API, 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 an OLAP API session. This method fully isolates these configuration settings solely for the OLAP API. (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 OLAP API.

Regardless of where these parameters are set, you should check the Oracle Technology Network for updated recommendations.

See Also:

- Chapter 8, "OLAP_API_SESSION_INIT" for information about the read and write APIs
- *Oracle9i SQL Reference* for descriptions of initialization parameters that can be set by the `ALTER SESSION` command

Creating Tablespaces for Analytic Workspaces

Before users begin creating analytic workspaces, you should create tablespaces that will be used for temporary and permanent storage of analytic workspaces. By default, these tablespaces are created in the SYS tablespace, which can degrade overall performance. Oracle OLAP makes heavy use of temporary tablespaces, so it is particularly important that they be set up correctly to prevent I/O bottlenecks.

These are some of the objects that Oracle OLAP stores in temporary tablespaces:

- The results of what-if analysis or other changes to the analytic workspace before they are committed to the database
- Output logs
- Views in a self-join
- Output of a table function when it exceeds 64KB

If possible, you should stripe the datafiles and temporary files across as many controllers and drives as are available.

Example 6–1 provides an example of a session in SQL*PLUS in which these tablespaces are created.

Example 6–1 Creating Tablespaces

The SQL commands in this example do the following:

- Create a tablespace named `OLAPUNDO` in a disk file named `olapundo.f`.
- Create and modify a rollback segment named `OLAPSEG` in the `OLAPUNDO` tablespace.
- Create a temporary tablespace named `OLAPTEMP` that uses up to four temporary disk files named `temp1.f`, `temp2.f`, `temp3.f`, and `temp4.f`. The additional disk files are located on separate physical disks (`user2`, `user3`, and `user4`).
- Grant the `SCOTT` user access rights to use `OLAPTEMP`.
- Create a tablespace named `OLAPTS` in up to three disk files named `olapdf1.f`, `olapdf2.f`, and `olapdf3.f`.

Following this example is an explanation of the statements beginning with "Creating a Tablespace for Rollbacks" on page 6-7.

```
SQL>  CREATE TABLESPACE olapundo DATAFILE '/user1/oracle/datafiles/olapundo.f'
   2   SIZE 200M REUSE AUTOEXTEND ON EXTENT MANAGEMENT LOCAL UNIFORM;

Tablespace created.

SQL> CREATE ROLLBACK SEGMENT olapseg TABLESPACE olapundo STORAGE (OPTIMAL 6M);

Rollback segment created.

SQL> ALTER ROLLBACK SEGMENT olapseg ONLINE;

Rollback segment altered.

SQL> CREATE TEMPORARY TABLESPACE olaptemp TEMPFILE
   2  '/user2/oracle/datafiles/temp1.f' SIZE 1024M REUSE
   3  AUTOEXTEND ON NEXT 100M MAXSIZE 2048M EXTENT MANAGEMENT LOCAL;

SQL> ALTER TABLESPACE olaptemp ADD TEMPFILE
   2 '/user2/oracle/datafiles/temp2.f' SIZE 1024M REUSE
```

```
AUTOEXTEND ON NEXT 100M MAXSIZE 4096,
   3 '/user3/oracle/datafiles/temp3.f' SIZE 1024M REUSE
AUTOEXTEND ON NEXT 100M MAXSIZE 4096,
   4 '/user4/oracle/datafiles/temp4.f' SIZE 1024M REUSE
AUTOEXTEND ON NEXT 100M MAXSIZE UNLIMITED;

Tablespace altered.

SQL> ALTER USER scott TEMPORARY TABLESPACE olaptemp;

User altered.

SQL> CREATE TABLESPACE olapts DATAFILE
  2  '/user1/oracle/olapdf1.f' SIZE 500M REUSE AUTOEXTEND ON NEXT 100M
MAXSIZE 4096M,
  3  '/user2/oracle/olapdf2.f' SIZE 500M REUSE AUTOEXTEND ON NEXT 100M
MAXSIZE 4096M,
  4  '/user3/oracle/olapdf3.f' SIZE 500M REUSE AUTOEXTEND ON NEXT 100M
MAXSIZE UNLIMITED;

Tablespace created.
```

Creating a Tablespace for Rollbacks

The following SQL commands create a tablespace that Oracle OLAP uses to store changes to active analytic workspaces so that the changes can be rolled back if necessary.

```
CREATE TABLESPACE tablespacename DATAFILE 'pathname' SIZE size REUSE
     AUTOEXTEND ON EXTENT MANAGEMENT LOCAL UNIFORM;

CREATE ROLLBACK SEGMENT segmentname TABLESPACE tablespacename
     STORAGE (OPTIMAL size);
```

Where:

segmentname is the name of the segment.

pathname is the fully qualified file name.

size is an appropriate size for these tablespaces.

tablespacename is the name of the tablespace being defined.

Creating a Temporary Tablespace

Oracle OLAP uses temporary tablespace to maintain different generations of an analytic workspace. This allows it to present a consistent view of the analytic workspace when one or more users are reading it while the contents are being updated.

```
CREATE TEMPORARY TABLESPACE tablespacename TEMPFILE 'pathname1'
   SIZE size REUSE AUTOEXTEND ON NEXT size MAXSIZE size EXTENT MANAGEMENT LOCAL;
ALTER TABLESPACE tablespacename ADD TEMPFILE
   'pathname2' SIZE size REUSE AUTOEXTEND ON NEXT size MAXSIZE size,
   'pathname3' SIZE size REUSE AUTOEXTEND ON NEXT size MAXSIZE size,
   'pathname4' SIZE size REUSE AUTOEXTEND ON NEXT size MAXSIZE size;

ALTER USER username TEMPORARY TABLESPACE tablespacename;
```

Where:

segmentname is the name of the segment.

pathname1... pathname4 are the fully qualified file names of files that located on separate disk drives if possible.

size is an appropriate size for these tablespaces.

tablespacename is the name of the tablespace being defined.

username is a user or group that you want to grant access rights to this tablespace.

workspacename is the name of a new analytic workspace.

Creating Tablespaces for Analytic Workspaces

When a user creates an analytic workspace, it is created by default in the SYS tablespace. The following commands create a tablespace that a user or group of users can specify as the storage location for their analytic workspaces. Using this temporary tablespace instead of the SYS tablespace will result in better performance. Note that this tablespace can be located on a separate disk drive.

```
CREATE TABLESPACE tablespacename DATAFILE
     'pathname1' SIZE size REUSE AUTOEXTEND ON NEXT size MAXSIZE size,
     'pathname2' SIZE size REUSE AUTOEXTEND ON NEXT size MAXSIZE size,
     'pathname3' SIZE size REUSE AUTOEXTEND ON NEXT size MAXSIZE UNLIMITED;
```

Where:

segmentname is the name of the segment.

pathname1... pathname3 are the fully qualified names of files located on separate disk drives if possible.

size is an appropriate size for these tablespaces.

tablespacename is the name of the tablespace.

username is a user or group that you want to grant access rights to this tablespace.

workspacename is the name of a new analytic workspace.

After creating this tablespace, be sure to instruct the users with access rights to create their analytic workspaces with OLAP DML commands such as the following one. Otherwise, their analytic workspaces will still be created in the SYS tablespace, even though you have created a separate tablespace for this purpose.

```
AW CREATE workspacename TABLESPACE tablespacename
```

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$workspacename;
```

Where:

workspacename 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. The 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.

To connect to the database using the OLAP API, 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

You can define user names and grant them these rights from the Security folder of Oracle Enterprise Manager.

Controlling 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 between flat files and 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 directory alias to represent a physical disk directory. Permissions are assigned to the alias, which control access to files within the associated physical directory.

You use PL/SQL statements to create a directory alias and grant permissions. The relevant syntax of these SQL statements is provided in this chapter.

> **See Also:** *Oracle9i SQL Reference* under the entries for `CREATE DIRECTORY` and `GRANT` for the full syntax and usage notes.

Creating a Directory Alias

To create a directory alias, you must have `CREATE ANY DIRECTORY` system privileges.

Use a `CREATE DIRECTORY` statement to create a new directory alias, or a `REPLACE DIRECTORY` statement to redefine an existing directory alias, using the following PL/SQL syntax:

```
{CREATE | REPLACE | CREATE OR REPLACE} DIRECTORY alias AS 'pathname';
```

Where:

alias is the name of the directory alias.

pathname is the physical directory path.

Granting Access Rights to a Directory Alias

After you create a directory alias, grant users and groups access rights to the files contained in that directory, using the following PL/SQL syntax:

```
GRANT permission ON DIRECTORY alias 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

alias is the name of the directory alias.

user is a database user name. That user gets immediate access rights.

role is a database role. All users who have been granted that role get immediate access rights.

PUBLIC is all database users. All users gets immediate access rights.

Example: Creating and Using a Directory Alias

The following SQL commands create a directory alias named olapdemo to control access to a directory named /users/oracle/OraHome1/demo and grant read access to all users.

```
CREATE DIRECTORY olapdemo as '/users/oracle/OraHome1/demo';
GRANT READ ON DIRECTORY olapdemo TO PUBLIC;
```

Users access files located in /users/oracle/OraHome1/demo with DML commands such as this one:

```
funit = FILEOPEN('olapdemo/units.dat' READ)
```

Understanding Data Storage

Oracle OLAP multidimensional data is stored in analytic workspaces. An analytic workspace can contain a variety of objects, such as dimensions, variables (also called measures), 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 tables 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.

User-Owned Tables

An analytic workspace is stored in a table in the Oracle database as a Binary Large Object (BLOB).

For example, if the SCOTT user creates two analytic workspaces, one named SALESDATA and the other named SALESPRGS, then these tables will be created in the SCOTT schema:

```
AW$SALESDATA
AW$SALESPRGS
```

These tables store all of the object definitions and data for these analytic workspaces.

> **See Also:** *Oracle9i OLAP Developer's Guide to the OLAP DML* for information about managing analytic workspaces.

System Tables

The SYS user owns several tables associated with analytic workspaces:

```
AW$EXPRESS
AW$
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$ 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.

One writer and many readers can use an analytic workspace at one time. 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$.

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.

OLAP dynamic performance tables and associated views are described in Chapter 7, "OLAP Dynamic Performance Views".

7

OLAP Dynamic Performance Views

Oracle collects performance statistics in fixed tables, and creates user-accessible views from these tables. This chapter describes the views that contain performance data on Oracle OLAP.

> **See Also:** For additional information about dynamic performance tables and views, refer to the following:
>
> - *Oracle9i Database Reference*
> - *Oracle9i Database Performance Guide and Reference*

This chapter contains the following topics:

System Tables Referenced by OLAP Performance Views

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. Dynamic performance tables are continuously updated while the database is in use. Among them are tables that collect data on Oracle OLAP.

The names of the OLAP dynamic performance tables begin with V$AW. The SYS user owns the dynamic performance tables. In addition, any user with the SELECT CATALOG role can access the tables.

The system creates views from these tables and creates public synonyms for the views. The views are sometimes called **fixed views** because they cannot be altered or removed by the database administrator. The synonym names also begin with V$AW. The views are also owned by SYS, but the DBA can grant access to them to a wider range of users.

The following sample SQL*Plus session shows the list of OLAP system tables.

```
% sqlplus '/ as sysdba'
      .
      .
      .
SQL> SELECT name FROM v$fixed_table WHERE name LIKE 'V$AW%';

NAME
- - - - - - - - - - - - - - - -
V$AW_OLAP
V$AW_CALC
V$AW_SESSION_INFO
```

Summary of OLAP Performance Views

Table 7–1 briefly describes each OLAP performance view.

Table 7–1 OLAP Performance Views

Fixed View	Description
V$AW_CALC	Collects information about the use of cache space.
V$AW_OLAP	Collects information about the status of active analytic workspaces.
V$AW_SESSION_INFO	Collects information about each active session.

V$AW_CALC

V$AW_CALC reports on the effectiveness of various caches used by Oracle OLAP. Because OLAP queries tend to be iterative, the same data is typically queried repeatedly during a session. The caches provide much faster access to data that has already been calculated during a session than would be possible if the data had to be recalculated for each query.

The more effective the caches are, the better the response time experienced by users. An ineffective cache (that is, one with few hits and many misses) probably indicates that the data is not being stored optimally for the way it is being viewed. To improve runtime performance, you may need to reorder the dimensions of the variables (that is, change the order of fastest to slowest varying dimensions).

Oracle OLAP uses the following caches:

- Aggregate cache. An optional cache used by the AGGREGATE function in the OLAP DML. The AGGREGATE function calculates aggregate data at runtime in response to a query. When a cache is maintained, AGGREGATE can retrieve data that was previously calculated during the session instead of recalculating it each time the data is queried.
- Session cache. Oracle OLAP maintains a cache for each session for storing the results of calculations. When the session ends, the contents of the cache are discarded.
- Page pool. A cache allocated from the program global area (PGA) in the database, which Oracle OLAP maintains for the session. The page pool is associated with a particular session and is shared by all attached analytic workspaces. If the page pool becomes too full, then Oracle OLAP writes some of the pages to the database cache. When an UPDATE command is issued in the OLAP DML, the changed pages associated with that analytic workspace are written to the permanent LOB, using temporary segments as the staging area for streaming the data to disk.
- Database cache. The larger cache maintained by the Oracle RDBMS for the database instance.

See Also:

- *Oracle9i OLAP Developer's Guide to the OLAP DML* for full discussions of data storage issues and aggregation.
- Oracle9i OLAP DML Reference help under the CACHE command for information about defining an aggregate cache.

Column	Datatype	Description
AGGREGATE_CACHE_HITS	NUMBER	The number of times a dimension member is found in the aggregate cache (a hit). The number of hits for run-time aggregation can be increased by fetching data across the dense dimension.
AGGREGATE_CACHE_MISSES	NUMBER	The number of times a dimension member is not found in the aggregate cache and must be read from disk (a miss).
SESSION_CACHE_HITS	NUMBER	The number of times the data is found in the session cache (a hit).
SESSION_CACHE_MISSES	NUMBER	The number of times the data is not found in the session cache (a miss).
POOL_HITS	NUMBER	The number of times the data is found in a page in the OLAP page pool (a hit).
POOL_MISSES	NUMBER	The number of times the data is not found in the OLAP page pool (a miss).
POOL_NEW_PAGES	NUMBER	The number of newly created pages in the OLAP page pool that have not yet been written to the workspace LOB.
POOL_RECLAIMED_PAGES	NUMBER	The number of previously unused pages that have been recycled with new data.
CACHE_WRITES	NUMBER	The number of times the data from the OLAP page pool has been written to the database cache.
POOL_SIZE	NUMBER	The number of pages in the OLAP page pool.

V$AW_OLAP

V$AW_OLAP provides a record of active sessions and their use with analytic workspaces. A row is generated whenever an analytic workspace is created or attached. The first row for a session is created when the first DML command is issued. It identifies the SYS.EXPRESS workspace, which is attached automatically to each session. Rows related to a particular analytic workspace are deleted when the workspace is detached from the session or the session ends.

Column	Datatype	Description
SESSION_ID	NUMBER	A unique numerical identifier for a session.
AW_NUMBER	NUMBER	A unique numerical identifier for an analytic workspace.
ATTACH_MODE	VARCHAR2(10)	READ ONLY or READ WRITE.
GENERATION	NUMBER	The generation of an analytic workspace. Each UPDATE creates a new generation. Sessions attaching the same workspace between UPDATE commands share the same generation.
TEMP_SPACE_PAGES	NUMBER	The number of pages stored in temporary segments for the analytic workspace.
TEMP_SPACE_READS	NUMBER	The number of times data has been read from a temporary segment and not from the page pool.
LOB_READS	NUMBER	The number of times data has been read from the table where the analytic workspace is stored (the permanent LOB).
POOL_CHANGED_PAGES	NUMBER	The number of pages in the page pool that have been modified in this analytic workspace.
POOL_UNCHANGED_PAGES	NUMBER	The number of pages in the page pool that have not been modified in this analytic workspace.

V$AW_SESSION_INFO

V$AW_SESSION_INFO provides information about each active session.

A transaction is a single exchange between a client session and Oracle OLAP. Multiple DML commands can execute within a single transaction, such as in a call to the DBMS_AW.EXECUTE procedure.

Column	Datatype	Description
CLIENT_TYPE	VARCHAR2(64)	OLAP
SESSION_STATE	VARCHAR2(64)	TRANSACTING, NOT_TRANSACTING, EXCEPTION_HANDLING, CONSTRUCTING, CONSTRUCTED, DECONSTRUCTING, or DECONSTRUCTED
SESSION_HANDLE	NUMBER	The session identifier
USERID	VARCHAR2(64)	The database user name under which the session opened
CURR_DML_COMMAND	VARCHAR2(64)	The DML command currently being executed
PREV_DML_COMMAND	VARCHAR2(64)	The DML command most recently completed.
TOTAL_TRANSACTION	NUMBER	The total number of transactions executed within the session; this number provides a general indication of the level of activity in the session
TOTAL_TRANSACTION_TIME	NUMBER	The total elapsed time in milliseconds in which transactions were being executed
AVERAGE_TRANSACTION_TIME	NUMBER	The average elapsed time in milliseconds to complete a transaction
TRANSACTION_CPU_TIME	NUMBER	The total CPU time in milliseconds used to complete the most recent transaction
TOTAL_TRANSACTION_CPU_TIME	NUMBER	The total CPU time used to execute all transactions in this session; this total does not include transactions that are currently in progress
AVERAGE_TRANSACTION_CPU_TIME	NUMBER	The average CPU time to complete a transaction; this average does not include transactions that are currently in progress

8

OLAP_API_SESSION_INIT

The `OLAP_API_SESSION_INIT` package contains procedures for maintaining a configuration table of initialization parameters for the OLAP API.

This chapter contains the following topics:

Overview

The OLAP_API_SESSION_INIT package contains procedures for maintaining a configuration table of initialization parameters. When the OLAP API opens a session, it executes the ALTER SESSION commands listed in the table for any user who has the specified roles. Only the OLAP API uses this table; no other type of application executes the commands stored in it.

This functionality provides an alternative to setting these parameters in the database initialization file or the init.ora file, which would alter the environment for all users.

During installation, the table is populated with ALTER SESSION commands that have been shown to enhance the performance of the OLAP API. Unless new settings prove to be more beneficial, you do not need to make changes to the table.

The information in the table can be queried through the ALL_OLAP_ALTER_SESSION view alias, which is also described in this chapter.

Summary of OLAP_API_SESSION_INIT Subprograms

Table 8–1 OLAP_API_SESSION_INIT Subprograms

Subprogram	Description
ADD_ALTER_SESSION Procedure on page 8-3	Specifies an ALTER SESSION parameter for OLAP API users with a particular database role.
DELETE_ALTER_SESSION Procedure on page 8-5	Removes a previously defined ALTER SESSION parameter for OLAP API users with a particular database role.
CLEAN_ALTER_SESSION Procedure on page 8-6	Removes orphaned data, that is, any ALTER SESSION parameters for roles that are no longer defined in the database.

ADD_ALTER_SESSION Procedure

This procedure specifies an ALTER SESSION parameter for OLAP API users with a particular database role. It adds a row to the OLAP$ALTER_SESSION table.

Syntax

```
ADD_ALTER_SESSION (
     role_name           IN     VARCHAR2,
     session_parameter   IN     VARCHAR2);
```

Parameters

The role_name and session_parameter are added as a row in OLAP$ALTER_SESSION.

Table 8–2 ADD_ALTER_SESSION Procedure Parameters

Parameter	Description
role_name	The name of a valid role in the database. Required.
session_parameter	A parameter that can be set with a SQL ALTER SESSION command. Required.

Exceptions

Table 8–3 ADD_ALTER_SESSION Procedure Exceptions

Exception	Description
invalid_role	Role is not defined in the database.
duplicate_role	Session parameter has already been set for that role.

Examples

The following call inserts a row in OLAP$ALTER_SESSION that turns on query rewrite for users with the OLAP_DBA role.

```
call olap_api_session_init.add_alter_session(
     'OLAP_DBA', 'SET QUERY_REWRITE_ENABLED=TRUE');
```

The ALL_OLAP_ALTER_SESSION view now contains the following row:

```
ROLE            CLAUSE_TEXT
OLAP_DBA        ALTER SESSION SET QUERY_REWRITE_ENABLED=TRUE
```

DELETE_ALTER_SESSION Procedure

This procedure removes a previously defined ALTER SESSION parameter for OLAP API users with a particular database role. It deletes a row from the OLAP$ALTER_SESSION table.

Syntax

```
DELETE_ALTER_SESSION (
     role_name           IN     VARCHAR2,
     session_parameter   IN     VARCHAR2);
```

Parameters

The role_name and session_parameter together uniquely identify a row in OLAP$ALTER_SESSION.

Table 8–4 DELETE_ALTER_SESSION Procedure Parameters

Parameter	Description
role_name	The name of a valid role in the database. Required.
session_parameter	A parameter that can be set with a SQL ALTER SESSION command. Required.

Exceptions

Table 8–5 DELETE_ALTER_SESSION Procedure Exceptions

Exception	Description
invalid_role	Role is not defined in the database.
duplicate_role	Session parameter has already been set for that role.

Examples

The following call deletes a row in OLAP$ALTER_SESSION that contains a value of OLAP_DBA in the first column and QUERY_REWRITE_ENABLED=TRUE in the second column.

```
call olap_api_session_init.delete_alter_session(
     'OLAP_DBA', 'SET QUERY_REWRITE_ENABLED=TRUE');
```

CLEAN_ALTER_SESSION Procedure

This procedure removes all `ALTER SESSION` parameters for any role that is not currently defined in the database. It removes all orphaned rows in the `OLAP$ALTER_SESSION` table for those roles.

Syntax

```
CLEAN_ALTER_SESSION ();
```

Examples

The following call deletes all orphaned rows.

```
call olap_api_session_init.clean_alter_session();
```

ALL_OLAP_ALTER_SESSION View

ALL_OLAP_ALTER_SESSION is the public synonym for V$OLAP_ALTER_SESSION, which is a view for the OLAP$ALTER_SESSION table. The view and table are owned by the SYS user.

Each row of ALL_OLAP_ALTER_SESSION identifies a role and a session initialization parameter. When a user opens a session using the OLAP API, the session is initialized using the parameters for roles granted to that user. For example, if there are rows for the OLAP_DBA role and the SELECT_CATALOG_ROLE, and a user has the OLAP_DBA role, then the parameters for the OLAP_DBA role will be set, but those for the SELECT_CATALOG_ROLE will be ignored.

Table 8–6 ALL_OLAP_ALTER_SESSION Column Descriptions

Column	Datatype	NULL	Description
ROLE	VARCHAR2(30)	NOT NULL	A database role
CLAUSE_TEXT	VARCHAR2(3000)		An ALTER SESSION command

9

Creating an Analytic Workspace From Relational Tables

You can use the AW_CREATE PL/SQL package to replicate a star schema within an analytic workspace and generate relational views of the resulting workspace cube. You can query these views directly using standard SQL. You can also create OLAP Catalog metadata that maps to these views to enable access by the OLAP API.

> **See Also:** Chapter 16, "CWM2_OLAP_AW_CREATE" for the syntax of the AW_CREATE procedure calls.

This chapter contains the following topics:

Choosing to Use an Analytic Workspace

If you determine that OLAP processing would best support the needs of an application, you can use the CWM2_OLAP_AW_CREATE package to replicate your relational data warehouse within an analytic workspace.

Relational and Multidimensional Data Models

The basic data model in a relational database is a table composed of one or more columns of data. In contrast, the basic data model in an analytic workspace is a cube, in which the data is stored as one or more measures with the same dimensionality. See Chapter 4, "Designing Your Database for OLAP" for an explanation of both data models.

Advantages of OLAP

OLAP processing within an analytic workspace is optimized to support complex analytic queries. Moreover, an analytic workspace can provide an efficient means of managing summary data, which may be precalculated or calculated on the fly. See "Why OLAP?" on page 1-2 for more information.

Functional Summary

With the CWM2_OLAP_AW_CREATE package, you can accomplish the following basic tasks:

- Create an analytic workspace, and define containers within it to represent an OLAP Catalog cube. The cube must be mapped to a star schema.
- Create load definitions. These definitions specify which data to load from the relational tables and how to aggregate it within the analytic workspace.
- Use a load definition to load data from the relational tables and aggregate it within the analytic workspace.
- Build relational views of the resulting analytic workspace. The views use the OLAP_TABLE function, described in Chapter 12, to access the workspace via object technology.

Note: Currently, the source cube in the OLAP Catalog must be mapped to a star or snowflake schema, as described in "Data Warehouse Requirements" on page 5-4.

Procedure: Create the OLAP Catalog Metadata

Before you can use the CWM2_OLAP_AW_CREATE procedures, you must create a cube in the OLAP Catalog. You can use Enterprise Manager, as described in "Creating Metadata Using Oracle Enterprise Manager" on page 5-6, or you can write scripts that use the CWM2 PL/SQL packages, as described in Chapter 13, "Using the OLAP Catalog Metadata APIs".

Note: If you choose to write your own script using the CWM2 packages, be sure to create the cube for a star schema with a single fact table containing only lowest level data.

Each of the cube's hierarchies must have a solved code set to UNSOLVED LEVEL, and the join relationships between the fact table and dimension tables must be mapped with storage type indicator of LOWEST LEVEL.

For more information on setting the solved code for a hierarchy, see Chapter 20. For more information on setting the storage type indicator, see "Mapping OLAP Metadata" on page 13-4.

Procedure: Create the Analytic Workspace Cube

Once the appropriate metadata exists in the OLAP Catalog, you can create the cube within the analytic workspace. Within a script, invoke the AW_CREATE procedures as follows:

1. For each of the cube's dimensions, call the AW_DIMENSION_CREATE Procedure to define the data structures for the dimension within the analytic workspace. The first call to AW_DIMENSION_CREATE creates the analytic workspace if it does not already exist.
2. Create one or more load definitions for each dimension. Call the AW_DIM_DEFINE_LOAD Procedure to name the load definition and specify its type. You can also call the AW_DIM_FILTER_LOAD Procedure to specify a SQL WHERE clause for the query against the dimension tables.
3. Call the AW_CUBE_CREATE Procedure to define the data structures for the cube within the analytic workspace.
4. Create one or more load definitions for the cube. Call the AW_CUBE_DEFINE_LOAD Procedure to name a load definition. Call the following procedures to complete the load definition:

- To load data that meets a certain criteria, call the AW_CUBE_FILTER_LOAD Procedure to specify a SQL WHERE clause for the query against the fact table.
- To specify which of the cube's measures to load, call the AW_CUBE_MEASURE_LOAD Procedure.
- If you want to partially aggregate the cube's data within the analytic workspace, call the AW_CHOOSE_LEVEL_TUPLES Procedure to create a table of level combinations for the cube. By default, all level combinations are selected for aggregation. For partial aggregation, edit the table to deselect the appropriate levels. Then call the AW_DEFINE_AGG_PLAN Procedure to define the aggregation plan for the cube.

Note: If you do not specify partial aggregation, the cube will be fully aggregated within the analytic workspace.

---

5. Call the AW_DIMENSION_REFRESH Procedure with a given load definition to load each analytic workspace dimension.
6. Call the AW_CUBE_REFRESH Procedure with a given load definition to load the analytic workspace cube.

Procedure: Create SQL Access to the Analytic Workspace

Once you have completed the steps described in Procedure: Create the Analytic Workspace Cube, you can generate the relational views that will allow SQL applications to access the analytic workspace. These views contain calls to the `OLAP_TABLE` function. `OLAP_TABLE`, described in Chapter 12, uses object technology to present the contents of the workspace in table format.

Use the following steps to generate the views:

1. Call the AW_DIMENSION_CREATE_ACCESS Procedure to generate views of the cube's dimensions.
2. Call `the` AW_CUBE_CREATE_ACCESS Procedure to generate views of the cube's measures.

Note: In the current release, the views generated by `AW_DIMENSION_CREATE_ACCESS` and `AW_CUBE_CREATE_ACCESS` are structured in the format required by the OLAP API.

Column Structure of Dimension Views

The AW_DIMENSION_CREATE_ACCESS procedure generates a separate view for each dimension hierarchy. For example, an AW cube with the four dimensions shown in Table 9–1, would have six separate dimension views since two of the dimensions have two hierarchies.

Table 9–1 Sample Dimension Hierarchies

Dimensions	Hierarchies	Number of Views
geography	standard	2
	consolidated	
product	standard	1
channel	standard	1
time	standard	2
	ytd	

The dimension views are level-based, and they include the full lineage of every level value in every row. This type of dimension table is considered **solved**, because the fact table related to this dimension includes embedded totals for all level combinations.

Each dimension view contains the columns described in Table 9–2.

Table 9–2 Dimension View Columns

Column	Description
ET key	The embedded-total key column stores the value of the lowest populated level in the row.
Parent ET key	The parent embedded-total key column stores parent of the ET key column.
GID	The grouping ID column identifies the hierarchy level associated with each row, as described in "Grouping ID Column" on page 9-6.
Parent GID	The parent grouping ID column stores the parent of the grouping ID column.
level columns	There is a column for each level of the dimension hierarchy.
level attribute columns	There is a column for each level attribute.

Sample Dimension View

For a standard geography hierarchy with levels for TOTAL_US, REGION, and STATE, the dimension view would contain columns like the ones shown below. Level attribute columns would also be included.

```
GID PARENT_GID  ET KEY     PARENT_ET_KEY TOTAL_US  REGION      STATE
--- ----------  ------     ------------  --------  ------      -----
0   1           MA         Northeast     USA       Northeast   MA
0   1           NY         Northeast     USA       Northeast   NY
0   1           GA         Southeast     USA       Southeast   GA
0   1           CA         Southwest     USA       Southwest   CA
0   1           AZ         Southwest     USA       Southwest   AZ
1   3           Northeast  USA           USA       Northeast
1   3           Southeast  USA           USA       Southeast
1   3           Southwest  USA           USA       Southwest
3   NA          USA        NA            USA
```

Grouping ID Column

The GID identifies the hierarchy level associated with each row by assigning a zero to each non-null value and a one to each null value in the level columns. The resulting binary number is the value of the GID.

For example, a GID of 1 is assigned to a row with the following three levels.

```
TOTAL_US  REGION      STATE
--------  ------      -----
USA       Southwest
0         0           1
```

A GID of 3 is assigned to a row with the following five levels.

```
TOTAL_GEOG COUNTRY REGION     STATE  CITY
---------- ------- -------    ------ -------
World      USA     Northeast
0          0       0           1      1
```

Column Structure of Fact Views

The AW_CUBE_CREATE_ACCESS procedure generates a separate view for each dimension/hierarchy combination. For example, an analytic workspace cube with the four dimensions shown in Table 9–1, would have four separate fact views, one for each hierarchy combination show in Table 9–3.

Table 9–3 Sample Dimension/Hierarchy Combinations

Geography Dim	Product Dim	Channel Dim	Time Dim
geography/ standard	product/standard	channel/standard	time/standard
geography/ standard	product/standard	channel/standard	time/ytd
geography/ consolidated	product/standard	channel/standard	time/standard
geography/ consolidated	product/standard	channel/standard	time/ytd

The fact views are fully **solved**. They contain embedded totals for all level combinations. Each view has columns for the cube's measures, and key columns that link the fact view with its associated dimension views.

Each fact view contains the columns described in Table 9–4.

Table 9–4 Fact View Columns

Column	Description
ET key for each dimension/hierarchy	The ET key column maps to the ET key column of the associated dimension table.
GID for each dimension/hierarchy	The GID column maps to GID column of the associated dimension table.
measure columns	Columns for each of the cube's measures.
empty columns	100 empty numeric columns and 100 empty text columns. These columns may be used to store custom measures.

10

Creating Materialized Views for the OLAP API

This chapter provides information to help you create materialized views specific to the requirements of the OLAP API. It describes the kinds of materialized views you will need to create, and it presents an overview of the tools that can assist you in creating them.

See Also:

- Chapter 29, "Creating Dimension Materialized Views"
- Chapter 30, "Creating Fact Materialized Views With DBMS_ODM"
- Chapter 31, "Creating Fact Materialized Views With OLAP Summary Advisor"

This chapter includes the following topics:

Choosing a Summary Management Strategy

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 manage aggregation within analytic workspaces or you can use Oracle's query rewrite facility.

Summary Management with Analytic Workspaces

Multidimensional processing within analytic workspaces provides an efficient means of managing summary data. Summaries may be precalculated or calculated on the fly. See "The Oracle9i Integrated Relational-Multidimensional Database" on page 1-4 for more information about multidimensional processing.

You can move your warehouse data from relational tables to analytic workspaces using the `AW_CREATE` package. See Chapter 9, "Creating an Analytic Workspace From Relational Tables".

Summary Management with 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.

About Materialized Views

Materialized views store data that has been calculated from detail tables. When data in the detail tables changes, you can refresh materialized views with the new data. While a **view** only stores the query, a **materialized view** actually stores the results of a query. Thus, you will need to allocate sufficient tablespace to store the required materialized views.

The OLAP API requires a very specific set of materialized views. For query rewrite to recognize that a materialized view contains the query results, the materialized view must have been created using basically the same type of SQL commands that are generated by the OLAP API.

You should create materialized views for frequently-aggregated data that is stored at detail level in a star or snowflake schema.

Do not create materialized views for data stored in embedded-total tables or analytic workspaces. Relational tables with embedded totals contain all the summary information within the tables. Analytic workspaces provide summary management based on a native multidimensional model.

Materialized View Formats

The database provides you with several tools for generating materialized views for the OLAP API. These tools produce materialized views for dimensions and fact tables. Fact materialized views may be built with **concatenated rollup** syntax or with **grouping set** syntax.

The choices you make in establishing materialized views will be based primarily on the structure of the data in the star schema and on the query requirements of OLAP clients.

Important: You must be sure to create materialized views that are specifically for use by the OLAP API. Query rewrite will **not** map the SQL generated by the OLAP API to the materialized views generated by the `DBMS_OLAP` PL/SQL package, which is described in the *Oracle9i Data Warehousing Guide*. Do not use the `DBMS_OLAP` package for the OLAP API.

Grouping Sets

The OLAP API supports fact table materialized views that use explicit grouping set syntax. This type of materialized view uses the `GROUP BY GROUPING SETS` syntax to aggregate the data for each level combination in the summary.

Materialized views generated with grouping set syntax can support asymmetric partial summarization. A single materialized view of this type holds all the summary information for a cube.

To generate this type of materialized view, use the Oracle Data Management PL/SQL package, **`DBMS_ODM`**.

Concatenated Rollup

The OLAP API also supports fact table materialized views that use concatenated rollup syntax. This type of materialized view uses the `GROUP BY ROLLUP` syntax to aggregate the data for each level combination in the summary.

Materialized views generated with concatenated rollup syntax can support symmetric partial summarization. A single materialized view of this type holds the summary information for one hierarchy combination of a cube.

To generate this type of materialized view, use the **OLAP Summary Advisor** within Oracle Enterprise Manager.

Materialized Views and OLAP Metadata

You should create materialized views after you have defined the OLAP metadata for your star schema.

> **See Also:** Chapter 5, "Creating OLAP Catalog Metadata" for information about creating OLAP metadata.

If your OLAP metadata is visible within Enterprise Manager, you can use the OLAP Summary Advisor to create MVs in concatenated rollup form. You can also use the `DBMS_ODM` package to create MVs in grouping set form.

If your OLAP metadata is not visible within Enterprise Manager, you must use the `DBMS_ODM` package. Only grouping set style MVs are supported for this type of metadata.

Dimension Materialized Views

When creating materialized views for the OLAP API, you should create MVs for each dimension in a star schema. Dimensions may be denormalized in a single table or normalized in separate tables (snow flake schema).

The structural differences between concatenated rollup style and grouping set style apply only to materialized views for fact tables. The structure of dimension materialized views is the same whether the fact table materialized view uses concatenated rollup or grouping sets.

Creating Dimension Materialized Views

When you use OLAP Summary Advisor, dimension materialized views are automatically created along with the fact materialized views for a cube.

Alternatively, you can use the `CREATEDIMMV_GS` procedure in the `DBMS_ODM` package to create dimension materialized views.

Note: The syntax of the `CREATE MATERIALIZED VIEW` statement is the same whether generated by OLAP Summary Advisor or the `DBMS_ODM` package.

Number of Dimension Materialized Views

The dimension MV scripts produced by OLAP Summary Advisor and `DBMS_ODM` create a separate MV for each hierarchy of a dimension.

Table 10–1, " SALES_CUBE Cube" lists the dimensions and hierarchies associated with the `SALES_CUBE` cube in the Sales History (`SH`) schema.

Table 10–1 SALES_CUBE 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	

The total number of dimension materialized views required for `SALES_CUBE` is seven, the sum of the number of materialized views required for each of its dimension hierarchies.

See Also: Chapter 29, "Creating Dimension Materialized Views" for more information about creating materialized views for dimensions.

Fact Materialized Views

When creating MVs for the OLAP API, you should create materialized views for each OLAP Catalog cube that represents a star schema. The cube must be mapped to a single fact table, and the fact table may contain only lowest-level data. For more information, see "Materialized Views and OLAP Metadata" on page 10-4.

Number of Fact Materialized Views

The number of fact materialized views for a cube depends on whether you using concatenated rollup style MVs or grouping set MVs.

If you use OLAP Summary Advisor, you will generate a separate concatenated rollup style MV for each combination of hierarchies in the cube. If you use DBMS_ODM, you will generate a single grouping set style MV for the cube.

For example, the SALES_CUBE cube in the Sales History (SH) schema, described in Table 10–1, would have either one materialized view generated with grouping sets or four materialized views generated with concatenated rollup.

For SALES_CUBE, there would be a separate concatenated rollup materialized view for the each of the following dimension hierarchy combinations.

- (CHANNEL, PRODUCT, PROMOTIONS, CUSTOMERS_CUST_ROLLUP, TIMES_CAL_ROLLUP)
- (CHANNEL, PRODUCT, PROMOTIONS, CUSTOMERS_CUST_ROLLUP, TIMES_FIS_ROLLUP)
- (CHANNEL, PRODUCT, PROMOTIONS, CUSTOMERS_GEOG_ROLLUP, TIMES_CAL_ROLLUP)
- (CHANNEL, PRODUCT, PROMOTIONS, CUSTOMERS_GEOG_ROLLUP, TIMES_FIS_ROLLUP)

See Also:

- Chapter 30, "Creating Fact Materialized Views With DBMS_ODM"
- Chapter 31, "Creating Fact Materialized Views With OLAP Summary Advisor".

Choosing the Right Format for Materialized Views

Whether you choose to use grouping set or concatenated rollup for your fact materialized views will depend on the complexity of the data in your star schema and on the nature of your OLAP metadata.

For more information on the metadata requirements, see "Materialized Views and OLAP Metadata" on page 10-4.

Unless you have a very simple data model with only single-hierarchy dimensions, grouping set MVs are generally more efficient and provide greater flexibility than concatenated rollup MVs.

Query Performance

MVs generated with grouping sets provide better runtime query performance for schemas that have dimensions with multiple hierarchies. MVs generated with concatenated rollup are more efficient for schemas that have only single-hierarchy dimensions.

Build Times

If you have single-hierarchy dimensions, concatenated rollup MVs will take less time to build than grouping set MVs. If you have multiple-hierarchy dimensions, grouping set MVs generally will take less time to build.

Partial Materialization

If you want to store partially aggregated data in your materialized views, the grouping set form provides more flexibility than the concatenated rollup form. Grouping set form supports asymmetric partial materialization. Concatenated rollup form supports only symmetric partial materialization.

With grouping set form, you could store month level summaries for specific level combinations only. For example, you could summarize month data for a certain type of product within a given geographical region, without regard for the other dimension levels associated with the data. You would do this by specifying individual level combinations before generating the script for creating the MV.

With concatenated rollup form, you could store month level summaries only, but they would be aggregated over all of the dimension hierarchies associated with the cube. You could choose to limit the MV to month data by editing the script for creating the MV.

MV Size

Although a grouping set style MV may be very large, it requires significantly less tablespace than concatenated rollup style MVs. The multiple concatenated rollup style MVs for a cube store redundant data, since each hierarchy combination is stored in a separate MV. A grouping set style MV for a cube contains all hierarchy combinations within the single MV.

Lineage (Key)

With concatenated rollup form, all the dimension key columns are populated, and data may only be accessed when its full lineage is specified. With true grouping set form, dimension key columns may contain null values, and data may be accessed simply by specifying one or more levels.

Note: In the current release, all MVs, whether generated with concatenated rollup or with grouping sets, are full lineage preserving.

Part III

SQL Access Reference

Part III provides information about PL/SQL packages and procedures that either create relational views of multidimensional data or embed OLAP DML commands in their syntax.

This part contains the following chapters:

11

DBMS_AW

Using the procedures and functions in the DBMS_AW package, SQL programmers can execute OLAP single-row functions and other OLAP DML commands against analytic workspace data.

See Also:

- Oracle9i OLAP DML Reference help for the syntax of individual OLAP DML commands.
- *Oracle9i OLAP Developer's Guide to the OLAP DML* for information on analytic workspace objects.
- *PL/SQL User's Guide and Reference* for information about the DBMS_OUTPUT package.

This chapter includes the following topics:

Summary of DBMS_AW Subprograms

The following table describes the subprograms provided in DBMS_AW.

Table 11–1 DBMS_AW Subprograms

Subprogram	Description
"EXECUTE Procedure" on page 11-3	Executes one or more OLAP DML commands. Input and output is limited to 4K. Typically used in an interactive session using an analytic workspace.
"PRINTLOG Procedure" on page 11-16	Returns the session log from the last execution of the INTERP or INTERPCLOB functions.
"INTERP_SILENT Procedure" on page 11-6	Executes one or more OLAP DML commands and suppresses the output. Input is limited to 4K and output to 4G.
"INTERP Function" on page 11-8	Executes one or more OLAP DML commands. Input is limited to 4K and output to 4G. Typically used in applications when the 4K limit on output for the EXECUTE procedure is too restrictive.
"INTERPCLOB Function" on page 11-10	Executes one or more OLAP DML commands. Input and output are limited to 4G. Typically used in applications when the 4K input limit of the INTERP function is too restrictive.
"OLAP_EXPRESSION Function" on page 11-12	Returns the result set of a single-row function calculated in an analytic workspace.
"PRINTLOG Procedure" on page 11-16	Prints a session log returned by the INTERP, INTERCLOB, or GETLOG functions.

EXECUTE Procedure

The EXECUTE procedure executes one or more OLAP DML commands and directs the output to a printer buffer. It is typically used to manipulate analytic workspace data within an interactive SQL session.

When you are using SQL*Plus, you can direct the printer buffer to the screen by issuing the following command:

```
SET SERVEROUT ON
```

If you are using a different program, refer to its documentation for the equivalent setting.

Input and output is limited to 4K. For larger values, refer to the INTERP and INTERPCLOB functions in this package.

Syntax

```
DBMS_AW.EXECUTE (
    olap_commands     IN   VARCHAR2
    text              OUT  VARCHAR2);
```

Parameters

Table 11–2 EXECUTE Procedure Parameters

Parameter	Description
olap-commands	One or more OLAP DML commands separated by semicolons.
text	Output from the OLAP engine in response to the OLAP commands.

Usage Notes

Guidelines for Using Quotation Marks in OLAP DML Commands

The SQL processor evaluates the OLAP DML commands, either in whole or in part, before sending them to Oracle OLAP for processing. Follow these guidelines when formatting the OLAP DML commands in the olap-commands parameter:

- Wherever you would normally use single quote (') in an OLAP DML command, use two single quotes (''). The SQL processor strips one of the single quotes before it sends the OLAP DML command to Oracle OLAP.

- In the OLAP DML, a double quote (") indicates the beginning of a comment.

Effect of the OUTFILE Command

This procedure does not print the output of the DML commands when you have redirected the output by using the OLAP DML `OUTFILE` command.

Example

The following sample SQL*Plus session attaches an analytic workspace named `XADEMO`, creates a formula named `COST_PP` in `XADEMO`, and displays the new formula definition.

```
SQL> SET SERVEROUT ON

SQL> EXECUTE DBMS_AW.EXECUTE('AW ATTACH xademo RW; DEFINE cost_pp FORMULA
LAG(analytic_cube_f.costs, 1, time, LEVELREL time_levelrel)');

PL/SQL procedure successfully completed.

SQL> EXECUTE DBMS_AW.EXECUTE('DESCRIBE cost_pp');

DEFINE COST_PP FORMULA DECIMAL <CHANNEL GEOGRAPHY PRODUCT TIME>
EQ lag(analytic_cube_f.costs, 1, time, levelrel time.levelrel)

PL/SQL procedure successfully completed.
```

GETLOG Function

This function returns the session log from the last execution of the INTERP or INTERPCLOB functions in this package.

To print the session log returned by this function, use the DBMS_AW.PRINTLOG procedure.

Syntax

```
DBMS_AW.GETLOG()
    RETURN CLOB;
```

Returns

The session log from the latest call to INTERP or INTERPCLOB.

Example

The following example shows the session log returned by a call to INTERP, then shows the identical session log returned by GETLOG.

```
SQL> SET SERVEROUT ON SIZE 1000000
SQL> EXECUTE DBMS_AW.PRINTLOG(DBMS_AW.INTERP('AW ATTACH xademo; LISTNAMES AGGMAP'));
2 AGGMAPs
----------------------------------------
ANALYTIC_CUBE.AGGMAP.1
XADEMO_SALES_MULTIKEY_CUBE.AGGMAP.1

PL/SQL procedure successfully completed.


SQL> EXECUTE DBMS_AW.PRINTLOG(DBMS_AW.GETLOG());
2 AGGMAPs
----------------------------------------
ANALYTIC_CUBE.AGGMAP.1
XADEMO_SALES_MULTIKEY_CUBE.AGGMAP.1

PL/SQL procedure successfully completed.
```

INTERP_SILENT Procedure

The INTERP_SILENT procedure executes one or more OLAP DML commands and suppresses all output from them. It does not suppress error messages from the OLAP command interpreter.

Input to the INTERP_SILENT function is limited to 4K. If you want to display the output of the OLAP DML commands, use the EXECUTE procedure, or the INTERP or INTERPCLOB functions.

Syntax

```
DBMS_AW.INTERP_SILENT (
    olap-commands     IN VARCHAR2);
```

Parameters

Table 11–3 DBMS_AW.INTERP Function Parameters

Parameter	Description
olap-commands	One or more OLAP DML commands separated by semi-colons.

Usage Notes

Guidelines for Using Quotation Marks in OLAP DML Commands

The SQL processor evaluates the OLAP DML commands, either in whole or in part, before sending them to Oracle OLAP for processing. Follow these guidelines when formatting the OLAP DML commands in the olap-commands parameter:

- Wherever you would normally use single quote (') in an OLAP DML command, use two single quotes (''). The SQL processor strips one of the single quotes before it sends the OLAP DML command to Oracle OLAP.
- In the OLAP DML, a double quote (") indicates the beginning of a comment.

Example

The following commands show the difference in message handling between EXECUTE and INTERP_SILENT. Both commands attach the XADEMO analytic workspace in read-only mode. However, EXECUTE displays a warning message, while INTERP_SILENT does not.

```
SQL> EXECUTE DBMS_AW.EXECUTE('AW ATTACH xademo');
```

```
IMPORTANT: Analytic workspace XADEMO is read-only. Therefore, you will
not be able to use the UPDATE command to save changes to it.

PL/SQL procedure successfully completed.


SQL> EXECUTE DBMS_AW.INTERP_SILENT('AW ATTACH xademo');

PL/SQL procedure successfully completed.
```

INTERP Function

The INTERP function executes one or more OLAP DML commands and returns the session log in which the commands are executed. It is typically used in applications when the 4K limit on output for the EXECUTE procedure may be too restrictive.

Input to the INTERP function is limited to 4K. For larger input values, refer to the INTERPCLOB function of this package.

You can use the INTERP function as an argument to the PRINTLOG procedure in this package to view the session log. See the example.

Syntax

```
DBMS_AW.INTERP (
    olap-commands     IN VARCHAR2)
    RETURN CLOB;
```

Parameters

Table 11–4 DBMS_AW.INTERP Function Parameters

Parameter	Description
olap-commands	One or more OLAP DML commands separated by semi-colons.

Returns

The log file for the Oracle OLAP session in which the OLAP DML commands were executed.

Usage Notes

Guidelines for Using Quotation Marks in OLAP DML Commands

The SQL processor evaluates the OLAP DML commands, either in whole or in part, before sending them to Oracle OLAP for processing. Follow these guidelines when formatting the OLAP DML commands in the olap-commands parameter:

- Wherever you would normally use single quote (') in an OLAP DML command, use two single quotes (''). The SQL processor strips one of the single quotes before it sends the OLAP DML command to Oracle OLAP.
- In the OLAP DML, a double quote (") indicates the beginning of a comment.

Effect of the OUTFILE Command

This function does not return the output of the DML commands when you have redirected the output by using the OLAP DML `OUTFILE` command.

Example

The following sample SQL*Plus session attaches an analytic workspace named `XADEMO` and lists the members of the `PRODUCT` dimension.

```
SQL> SET SERVEROUT ON SIZE 1000000
SQL> EXECUTE DBMS_AW.PRINTLOG(DBMS_AW.INTERP('AW ATTACH cloned; REPORT product'));
PRODUCT
--------------
L1.TOTALPROD
L2.ACCDIV
L2.AUDIODIV
L2.VIDEODIV
L3.AUDIOCOMP
L3.AUDIOTAPE
     .
     .
     .
PL/SQL procedure successfully completed.
```

INTERPCLOB Function

The INTERPCLOB function executes one or more OLAP DML commands and returns the session log in which the commands are executed. It is typically used in applications when the 4K limit on input for the INTERP function may be too restrictive.

You can use the INTERPCLOB function as an argument to the PRINTLOG procedure in this package to view the session log. See the example.

Syntax

The syntax for the INTERPCLOB procedure is shown below.

```
DBMS_AW.INTERPCLOB (
    olap-commands     IN CLOB)
    RETURN CLOB;
```

Parameters

Table 11–5 DBMS_AW.INTERPCLOB Function Parameters

Parameter	Description
olap-commands	One or more OLAP DML commands separated by semi-colons.

Returns

The log for Oracle OLAP session in which the OLAP DML commands were executed.

Usage Notes

Guidelines for Using Quotation Marks in OLAP DML Commands

The SQL processor evaluates the OLAP DML commands, either in whole or in part, before sending them to Oracle OLAP for processing. Follow these guidelines when formatting the OLAP DML commands in the olap-commands parameter:

- Wherever you would normally use single quote (') in an OLAP DML command, use two single quotes (''). The SQL processor strips one of the single quotes before it sends the OLAP DML command to Oracle OLAP.
- In the OLAP DML, a double quote (") indicates the beginning of a comment.

Effect of the OUTFILE Command

This function does not return the output of the OLAP DML commands when you have redirected the output by using the OLAP DML `OUTFILE` command.

Example

The following sample SQL*Plus session creates an analytic workspace named `ELECTRONICS`, imports its contents from an EIF file stored in the `dbs` directory alias, and displays the contents of the analytic workspace.

```
SQL> SET SERVEROUT ON SIZE 1000000
SQL> EXECUTE DBMS_AW.PRINTLOG(DBMS_AW.INTERPCLOB('AW CREATE electronics; IMPORT
ALL FROM EIF FILE ''dbs/electronics.eif'' DATA DFNS; DESCRIBE'));

DEFINE GEOGRAPHY DIMENSION TEXT WIDTH 12
LD Geography Dimension Values
DEFINE PRODUCT DIMENSION TEXT WIDTH 12
LD Product Dimension Values
DEFINE TIME DIMENSION TEXT WIDTH 12
LD Time Dimension Values
DEFINE CHANNEL DIMENSION TEXT WIDTH 12
LD Channel Dimension Values
          .
          .
          .
PL/SQL procedure successfully completed.
```

OLAP_EXPRESSION Function

The OLAP_EXPRESSION function allows you to execute single-row numeric functions in the analytic workspace and thus generate custom measures in SELECT statements. In addition to calculating an expression, OLAP_EXPRESSION can be used in the WHERE and ORDER BY clauses to modify the result set of a SELECT.

Syntax

```
OLAP_EXPRESSION(
    r2c           IN  RAW(32),
    expression    IN  VARCHAR2 )
    RETURN NUMBER;
```

Parameters

Table 11–6 OLAP_EXPRESSION Function Parameters

Parameter	Description
r2c	The name of a column populated by the ROW2CELL clause of the limit map in a call to the OLAP_TABLE function.
expression	A calculation that will be performed in the analytic workspace.

Returns

An evaluation of *expression* for each row of the table object returned by the OLAP_TABLE function.

Usage Notes

You can use OLAP_EXPRESSION only with a table object returned by the OLAP_TABLE function. The returned table object must have a column populated by a ROW2CELL clause in the limit map used in the call to OLAP_TABLE. Refer to Chapter 12, "OLAP_TABLE" for more information about using this function.

Examples

View Used in These Examples

The following script was used to create a view named MEASURE_VIEW, which is used in the examples of OLAP_EXPRESSION that follow.

```
CREATE TYPE measure_row AS OBJECT (
   time                        VARCHAR2(12),
   geography                   VARCHAR2(30),
   product                     VARCHAR2(30),
   channel                     VARCHAR2(30),
   sales                       NUMBER(16),
   cost                        NUMBER(16),
   promotions                  NUMBER(16),
   quota                       NUMBER(16),
   units                       NUMBER(16),
   r2c                         RAW(32));
/

CREATE TYPE measure_table AS TABLE OF measure_row;
/

CREATE OR REPLACE VIEW measure_view AS
SELECT sales, cost, promotions, quota, units,
      time, geography, product, channel, r2c
   FROM TABLE(CAST(OLAP_TABLE(
     'xademo DURATION SESSION',
     'measure_table',
     '',
     'MEASURE sales FROM xademo_analytic_cube_f.sales
      MEASURE cost FROM xademo_analytic_cube_f.costs
      MEASURE promotions FROM xademo_analytic_cube_f.promo
      MEASURE quota FROM xademo_analytic_cube_f.quota
      MEASURE units FROM xademo_analytic_cube_f.units
      DIMENSION time FROM xademo_time WITH
        HIERARCHY xademo_time_member_parentrel
           INHIERARCHY xademo_time_member_inhier
      DIMENSION geography FROM xademo_geography WITH
         HIERARCHY xademo_geography_member_parentrel
            INHIERARCHY xademo_geography_member_inhier
      DIMENSION product FROM xademo_product WITH
         HIERARCHY xademo_product_member_parentrel
            INHIERARCHY xademo_product_member_inhier
      DIMENSION channel FROM xademo_channel WITH
         HIERARCHY xademo_channel_member_parentrel
```

```
            INHIERARCHY xademo_channel_member_inhier
   ROW2CELL r2c')
      AS measure_table))
   WHERE sales IS NOT NULL;
/
COMMIT
/
GRANT SELECT ON measure_view TO PUBLIC;
```

Time Series Function With a WHERE Clause

The following SELECT statement calculates an expression with an alias of PERIODAGO, and limits the result set to calculated values greater than 200,000. The calculation uses the LAG function to return the value of the previous time period.

```
SELECT time, cost, OLAP_EXPRESSION(r2c,
   'LAG(xademo_analytic_cube_f.costs, 1, xademo_time,
      LEVELREL xademo_time_member_levelrel)') periodago
FROM measure_view
WHERE geography = 'L1.WORLD' AND
CHANNEL = 'STANDARD_2.TOTALCHANNEL' AND
PRODUCT = 'L1.TOTALPROD' and
OLAP_EXPRESSION(r2c, 'LAG(xademo_analytic_cube_f.costs, 1, xademo_time,
   LEVELREL xademo_time_member_levelrel)') > 200000;
```

This SELECT statement produces these results.

```
TIME               COST  PERIODAGO
------------ ---------- ----------
L1.1997         1078031 2490243.07
L2.Q1.97         615399 560379.445
L2.Q2.96         649004 615398.858
L2.Q2.97         462632 649004.473
L2.Q3.96         582693 462632.064
L2.Q4.96         698166 582693.091
L3.AUG96         194498 209476.344
L3.FEB96         186762 252738.981
L3.JAN96         185755 205214.946
               .
               .
               .
```

Numeric Calculation With an ORDER BY Clause

This example subtracts costs from sales to calculate profit, and gives this expression an alias of PROFIT. The rows are ordered by geographic areas from most to least profitable.

```
SELECT geography, sales, cost, OLAP_EXPRESSION(r2c,
   'xademo_analytic_cube_f.sales - xademo_analytic_cube_f.costs') profit
FROM measure_view
WHERE
channel = 'STANDARD_2.TOTALCHANNEL' AND
product = 'L1.TOTALPROD' AND
time = 'L3.APR97'
ORDER BY OLAP_EXPRESSION(r2c,
   'xademo_analytic_cube_f.sales - xademo_analytic_cube_f.costs') DESC;
```

This SELECT statement produces these results.

```
GEOGRAPHY                          SALES       COST     PROFIT
------------------------------ ---------- ---------- ----------
L1.WORLD                         9010260     209476 8800783.17
L2.EUROPE                        3884776      95204 3789571.85
L2.AMERICAS                      2734436      55322 2679114.66
L2.ASIA                          1625379      37259 1588120.61
L3.USA                           1603043      27547 1575496.86
L2.AUSTRALIA                      765668      21692 743976.058
L3.UK                             733090      19144 713945.952
L3.CANADA                         731734      19666 712067.455
L4.NEWYORK                        684008       8020 675987.377
L3.GERMANY                        659428      12440 646988.197
L3.FRANCE                         596767      19307 577460.113
                .
                .
                .
```

PRINTLOG Procedure

This procedure sends a session log returned by the INTERP, INTERPCLOB, or GETLOG functions of this package to the print buffer, using the DBMS_OUTPUT package in PL/SQL.

When you are using SQL*Plus, you can direct the printer buffer to the screen by issuing the following command:

```
SET SERVEROUT ON SIZE 1000000
```

The SIZE clause increases the buffer from its default size of 4K.

If you are using a different program, refer to its documentation for the equivalent setting.

Syntax

The syntax for the PRINTLOG procedure is shown below.

```
DBMS_AW.PRINTLOG (
    session-log     IN CLOB);
```

Parameters

Table 11–7 DBMS_AW.PRINTLOG Procedure Parameters

Parameter	Description
session-log	The log of a session.

Example

The following example shows the session log returned by the INTERP function.

```
SQL> SET SERVEROUT ON SIZE 1000000
SQL> EXECUTE DBMS_AW.PRINTLOG(DBMS_AW.INTERP('DESCRIBE analytic_cube_f.profit'));

DEFINE ANALYTIC_CUBE.F.PROFIT FORMULA DECIMAL <CHANNEL
GEOGRAPHY PRODUCT TIME>
EQ analytic_cube.f.sales - analytic_cube.f.costs

PL/SQL procedure successfully completed.
```

12

OLAP_TABLE

This chapter describes how you can use the OLAP_TABLE function in a SQL SELECT statement to query the multidimensional data stored in an analytic workspace. This chapter contains the following topics:

Description

The OLAP_TABLE function extracts data from the LOBs in which workspace data has been stored and presents the result set in the format of a relational table. OLAP_TABLE is an implementation of the PL/SQL table functions.

The OLAP_TABLE function can be used in a SQL SELECT statement instead of, or in addition to, the names of relational tables and views. It presents fully solved data that is either stored or calculated in an analytic workspace. OLAP_TABLE accepts parameters that are passed to the OLAP engine, which selects, manipulates, and returns the data. The WHERE clause of a SELECT statement that includes a call to OLAP_TABLE only needs to identify the result set; it does not need to perform any calculations. If it does include calculations, they will be performed by the SQL engine, not the OLAP engine.

SELECT statements that use OLAP_TABLE can be used during database maintenance to create relational views, and they can be used interactively to fetch data directly into an application.

> **See Also:** *PL/SQL User's Guide and Reference* for a discussion of PL/SQL table functions.

Preliminary Steps

Most applications require the data to be presented in a specific format. You must know the requirements of your application in order to construct a call to OLAP_TABLE that returns a result set that complies with those requirements.

In addition, you need to gather information about the data containers in the analytic workspace and decide how you are going to map them to the columns of a relational view. These are the steps you might take:

1. Identify the measures that you want to make available to applications.
2. Identify the dimensions (including composite dimensions) of the measures.
3. For hierarchical dimensions, identify the objects that support the hierarchy.
4. Identify the dimension attributes, which are data containers that provide additional information about the dimensions.
5. If you plan to create OLAP catalog metadata, generate the additional data containers that are needed by the Java-based OLAP API.

Following are descriptions of these data containers.

Measures

Measures are VARIABLE, FORMULA, or RELATION containers with a numeric data type. If you are creating views for a star schema, you will experience the best performance and data retrieval if the measures represented in a single fact view have the exact same dimensions listed in the exact same order.

For example, the following variables compose the same cube and are dimensioned identically.

```
DEFINE ANALYTIC_CUBE_F.COSTS VARIABLE DECIMAL <ANALYTIC_CUBE_COMPOSITE <CHANNEL GEOGRAPHY
PRODUCT TIME>>

DEFINE ANALYTIC_CUBE_F.SALES VARIABLE DECIMAL <ANALYTIC_CUBE_COMPOSITE <CHANNEL GEOGRAPHY
PRODUCT TIME>>
```

You can combine these variables with formulas derived from them. Although formulas do not use composites, they are defined with the same dimensions in the same order as their source variables. For example, the following command creates a formula named ANALYTIC_CUBE_PROFIT, which is calculated by subtracting ANALYTIC_CUBE_F.COSTS from ANALYTIC_CUBE_F.SALES.

```
->DEFINE analytic_cube_profit FORMULA analytic_cube_f.sales - analytic_cube_f.costs
```

The resulting formula is dimensioned the same as the source variables, but without the composite.

```
->DESCRIBE analytic_cube_profit

DEFINE ANALYTIC_CUBE_PROFIT FORMULA DECIMAL <CHANNEL GEOGRAPHY PRODUCT TIME>
EQ analytic_cube_f.sales - analytic_cube_f.costs
```

You can also specify formulas within the MEASURE clause of the OLAP_TABLE function.

Dimensions

If a measure is sparse, then it is probably dimensioned by a composite or a conjoint dimension. The definition of a measure identifies its dimensions.

Hierarchies

Dimensions that contain members at all levels of a hierarchy are supported by several workspace objects that define the hierarchy: hierarchy dimensions, hierarchy relations, level dimensions, level relations, and “in hierarchy” variables.

A flat dimension (that is, one without a hierarchy, or one in which all members are at the same level of a hierarchy) does not require these supporting objects.

Hierarchy Dimensions

When a dimension has more than one hierarchy, then a hierarchy dimension is used to identify them. The members of the hierarchy dimension are the names of the hierarchies.

The following example shows the hierarchy dimension for the GEOGRAPHY dimension.

```
->DESCRIBE geography_hierlist

DEFINE GEOGRAPHY_HIERLIST DIMENSION TEXT

->REPORT W 25 geography_hierlist

GEOGRAPHY_HIERLIST
------------------
STANDARD
CONSOLIDATED
```

Hierarchy Relations

A self-relation identifies the parent of each dimension member. This type of relation is often called a parent relation. In the following example, the GEOGRAPHY dimension has a parent relation named GEOGRAPHY_PARENTREL.

GEOGRAPHY_HIERLIST also dimensions the parent relation. GEOGRAPHY has two hierarchies, STANDARD and CONSOLIDATED, which are the dimension members of GEOGRAPHY_HIERLIST.

```
->DESCRIBE geography_member_parentrel

DEFINE GEOGRAPHY_MEMBER_PARENTREL RELATION GEOGRAPHY <GEOGRAPHY
GEOGRAPHY_HIERLIST>

->LIMIT geography TO 'L4.KUALALUMPUR'
->LIMIT geography ADD ANCESTORS USING geography_member_parentrel
->REPORT W 16 DOWN geography W 20 geography_member_parentrel
```

```
                -------GEOGRAPHY_MEMBER_PARENTREL--------
                -----------GEOGRAPHY_HIERLIST------------
GEOGRAPHY            STANDARD           CONSOLIDATED
---------------- -------------------- --------------------
L4.KUALALUMPUR   L3.MALAYSIA          L6.MALAYSIA
L3.MALAYSIA      L2.ASIA              NA
L6.MALAYSIA      NA                   L5.ASIA
L2.ASIA          L1.WORLD             NA
L5.ASIA          NA                   NA
L1.WORLD         NA                   NA
```

From this example, you can see that levels L1, L2, and L3 are in the STANDARD hierarchy, and levels L5 and L6 are in the CONSOLIDATED hierarchy. Malaysia and Asia are each represented by two dimension members, one for each hierarchy.

Level Dimensions

The levels of a dimension hierarchy are defined by the members of a level dimension. This dimension has a TEXT data type so that the members can have descriptive names. For example, GEOGRAPHY_LEVELLIST is the level dimension for GEOGRAPHY.

```
->DESCRIBE geography_levellist
DEFINE GEOGRAPHY_LEVELLIST DIMENSION TEXT
```

Six levels are defined for the two GEOGRAPHY hierarchies.

```
->REPORT geography_levellist

GEOGRAPHY_LEVELLIST
-------------------
L4
L3
L2
L1
L6
L5
```

In-Hierarchy Variables

If a hierarchical dimension contains members that are excluded from a hierarchy, then a boolean variable is used to identify whether a dimension member is in the hierarchy (YES) or not in the hierarchy (NO or NA). If all the members of a dimension are included in the hierarchy (which is typically the case when there is only one

hierarchy), then this boolean dimension is not required because there is no ambiguity. However, if a dimension member is part of one hierarchy but excluded from another (which is typically the case when there are multiple hierarchies) an NA value in the hierarchy relation is ambiguous. It can mean either that the member is at the top level of the hierarchy and therefore has no parent, or that it is excluded from the hierarchy.

The following example shows an in-hierarchy variable named GEOGRAPHY_INHIERARCHY defined for the GEOGRAPHY dimension, which has two hierarchies, STANDARD and CONSOLIDATED.

```
->DESCRIBE geography_member_inhier
DEFINE GEOGRAPHY_MEMBER_INHIER VARIABLE BOOLEAN <GEOGRAPHY GEOGRAPHY_HIERLIST>

->REPORT DOWN geography W 12 geography_member_inhier

               -GEOGRAPHY_MEMBER_INHIER-
               --- GEOGRAPHY_HIERLIST---
GEOGRAPHY        STANDARD   CONSOLIDATED
-------------- ------------ ------------
L4.KUALALUMPUR          yes          yes
L3.MALAYSIA             yes           NA
L6.MALAYSIA              NA          yes
L2.ASIA                 yes           NA
L5.ASIA                  NA          yes
L1.WORLD                yes           NA
```

Grouping IDs

Grouping IDs identify the depth of a dimension member in the hierarchy. You can create a GID variable manually by using the GROUPINGID command in the OLAP DML. Grouping IDs are used by the OLAP API to improve performance.

```
->DESCRIBE geography_member_gid

DEFINE GEOGRAPHY_MEMBER_GID VARIABLE INTEGER <GEOGRAPHY GEOGRAPHY_HIERLIST>

->REPORT DOWN geography W 12 geography_member_gid
```

```
                --GEOGRAPHY_MEMBER_GID---
                ---GEOGRAPHY_HIERLIST----
GEOGRAPHY         STANDARD   CONSOLIDATED
------------- ------------ ------------
L4.KUALALUMPUR            0            0
L3.MALAYSIA               1           NA
L6.MALAYSIA              NA            1
L2.ASIA                   3           NA
L5.ASIA                  NA            3
L1.WORLD                  7           NA
```

Parent Grouping IDs

Parent grouping IDs provide the GID value of the parent of each dimension member. OLAP_TABLE calculates the parent grouping IDs from the member grouping IDs. Thus, you do *not* need to define the parent GIDs in an object in the analytic workspace. However, you *do* need to specify the PARENTGID clause so that OLAP_TABLE will generate them.

This information is used by the OLAP API to improve performance. If you specify a parent relation, then you also need to specify a parent GID.

Family Relations

A family relation is used when generating a view in rollup form, that is, a view in which a multiple-column key identifies the full parentage of each dimension value. Each column in the key contains values at one level of the dimension hierarchy. A family relation formats the information in this way in the analytic workspace.

You can create a family relation manually by defining a relation and populating it using the HIERHEIGHT command in the OLAP DML.

The following is the definition of the family relation for GEOGRAPHY.

```
->DESCRIBE geography_member_familyrel

DEFINE GEOGRAPHY_MEMBER_FAMILYREL RELATION GEOGRAPHY <GEOGRAPHY
GEOGRAPHY_LEVELLIST GEOGRAPHY_HIERLIST>

->LIMIT geography_levellist TO FIRST 4
->REPORT W 12 DOWN geography W 16 geography_member_familyrel


GEOGRAPHY.HIERLIST: STANDARD
```

```
                    --------------------GEOGRAPHY_MEMBER_FAMILYREL--------------------
                    ------------------------GEOGRAPHY_LEVELLIST-----------------------
GEOGR
APHY                      L4              L3              L2              L1
------------ ---------------- --------------- --------------- ----------------
L4.ADELAIDE  L4.ADELAIDE      L3.CENTRAL.AUST  L2.AUSTRALIA     L1.WORLD
L4.AMSTERDAM L4.AMSTERDAM     L3.NETHERLANDS   L2.EUROPE        L1.WORLD
L4.ATHENS    L4.ATHENS        L3.GREECE        L2.EUROPE        L1.WORLD
L4.BANGKOK   L4.BANGKOK       L3.THAILAND      L2.ASIA          L1.WORLD
                                 .
                                 .
                                 .
```

See Also: Oracle9i OLAP DML Reference help for syntax and examples of the `GROUPINGID` and `HIERHEIGHT` commands.

Attributes

Attributes are typically text variables or relations that provide descriptive information about dimension members, and are useful for displaying the data. Dimension members are usually very cryptic, and are more useful for uniquely identifying the data internally than for labeling the data for users in a table or graph. For this reason, dimensions often have one or more variables that provide descriptions of the dimension members.

Attributes can also provide other types of information and be other data types, like the end date and time span attributes for a time dimension. The following example shows attributes for the `TIME` dimension.

```
->LIMIT time_hierlist TO 'STANDARD'
->REPORT DOWN time time_short.description time_end_date time_time_span


LANGUAGELIST: AMERICAN_AMERICA
               ----------TIME_HIERLIST---------
               ------------STANDARD------------
               TIME_SHORT
               .DESCRIPTI  TIME_END_ TIME_TIME_
TIME           ON          DATE      SPAN
-------------- ---------- ---------- ----------
L1.1996        1996       31DEC96        366.00
L1.1997        1997       31MAY97        151.00
L2.Q1.96       Q1.96      31MAR96         91.00
```

```
L2.Q1.97        Q1.97       31MAR97          90.00
L2.Q2.96        Q2.96       30JUN96          91.00
              .
              .
              .
```

Basic Steps

There are three steps to using the OLAP_TABLE function:

1. Define an object type. Equivalent to defining a row.
2. Create a type of these objects. Equivalent to defining a table.
3. Embed a call to the OLAP_TABLE function in a SELECT statement.

Defining a Row

When you define a row, you are actually defining an abstract object type. An abstract object type is composed of attributes, which are equivalent to the columns of a table. (These attributes have no relationship to the attributes described in "Attributes" on page 12-8.) When you ultimately create a relational view, you will select its columns from these attributes. However, it is generally easier to understand the process in terms of rows and columns instead of object types and attributes.

This is the basic syntax for defining a row. The last column is defined as type RAW, and stores information used by the single-row functions in DBMS_AW. If you are not going to use those functions, then you do not need to define this column.

```
CREATE TYPE row_name AS OBJECT (
   column_first       datatype,
   column_second      datatype,
   column_last        RAW(32);
```

Example 12–1 defines a row for a product dimension table. The five VARCHAR2 columns of PRODUCT_ROW (PRODUCT, PRODUCT_LABEL, and so forth) ultimately define the available columns of a product dimension view.

Example 12–1 Creating the PRODUCT_ROW Object Type

```
CREATE TYPE product_row AS OBJECT (
   product             VARCHAR2(30),
   product_label       VARCHAR2(30),
   product_parent      VARCHAR2(30),
```

```
product_level       VARCHAR2(2),
subcategory         VARCHAR2(30),
category            VARCHAR2(15),
all_products        VARCHAR2(15)
r2c                 RAW(32));
```

Creating a Table

An abstract table type is a collection of abstract object types. The table type describes the table that will be populated by OLAP_TABLE. This is the basic syntax for creating a table type:

```
CREATE TYPE table_name AS TABLE OF row_name;
```

Example 12–2 creates a table of the PRODUCT_ROW objects that were created in Example 12–1.

Example 12–2 Creating the PRODUCT_TABLE Table Type

```
CREATE TYPE product_table AS TABLE OF product_row;
```

Using OLAP_TABLE in a SELECT Statement

A view of an analytic workspace is like any other relational view in being a saved SELECT statement. The difference is that the OLAP_TABLE function takes the place of a relational table.

The following syntax shows how you would use OLAP_TABLE to create a view:

```
CREATE OR REPLACE VIEW view_name AS
SELECT columns
   FROM TABLE(OLAP_TABLE(parameters))
WHERE conditions;
```

Where:

columns are the names of attribute columns in the logical table object that you defined. You do not need to reference all of the columns, only those that you will use as targets in the limit map of OLAP_TABLE.

conditions modify the result set from OLAP_TABLE. These operators are processed in the analytic workspace: =, !=, IN, NOT IN. Conditions that are not supported in the analytic workspace are executed in SQL on the returned result set.

Applications can also generate SELECT statements on the fly that use calls to OLAP_TABLE instead of, or in addition to, the names of relational tables. This type

of application can generate calls to OLAP_TABLE with parameters defined by the user.

OLAP_TABLE Reference

The OLAP_TABLE function extracts multidimensional data from an analytic workspace and presents it in the two-dimensional format of a relational table. It can be used wherever you would use the name of a table or view. The analytic workspace data can be stored or calculated on the fly from stored data. The result set is a table of objects that can be joined to relational tables and views, or to other tables of objects populated by OLAP_TABLE.

Syntax

```
OLAP_TABLE(
      aw_attach          IN VARCHAR2,
      table_name         IN VARCHAR2,
      olap_command       IN VARCHAR2,
      limit_map          IN VARCHAR2);
```

The OLAP_TABLE function returns the table of objects identified by *table_name*, which has been populated according to the rules defined in *limit_map*.

Parameters

Table 12–1 OLAP_TABLE Function Parameters

Parameter	Description
aw_attach	The name of the analytic workspace with the source data
table_name	The name of the table that has been defined to structure the multidimensional data in tabular form
olap_command	An OLAP DML command that will be executed before the data is fetched
limit_map	A keyword-based map that identifies the source objects in *aw_attach* and the target columns in *table_name*.

AW_ATTACH Parameter

The first parameter of the OLAP_TABLE function provides the name of the analytic workspace where the source data is stored and specifies how long the analytic workspace will be attached to your OLAP session, which opens for your first call to OLAP_TABLE. You can detach the analytic workspace either at the end of the query or at the end of the session. This is the full syntax of this parameter:

```
'[owner.]aw_name DURATION QUERY | SESSION'
```

For example:

```
'sys.xademo DURATION QUERY'
```

Specify *owner* whenever you are creating views that will be accessed by other users. Otherwise, you can omit the *owner* if you own the analytic workspace. It is required only when you are logged in under a different user name than the owner.

If you specify `SESSION`, then you can use an empty string for this parameter in subsequent calls to `OLAP_TABLE`, because the analytic workspace is already attached. If you repeat the connection string unnecessarily, it is simply ignored.

`SESSION` provides slightly better performance than `QUERY`, because the analytic workspace is attached only once instead of multiple times in the session. However, you will not see modifications made by other users in the meantime.

Table_Name Parameter

The second parameter identifies the name of the table of objects that you defined, as shown in "Creating a Table" on page 12-10. The syntax of this parameter is:

```
'table_name'
```

For example:

```
'product_table'
```

OLAP_Command Parameter

The third parameter of the `OLAP_TABLE` function is a single OLAP DML command. If you want to execute more than one command, then you must create a program in your analytic workspace and call the program in this parameter.

A common use of this parameter is to limit one or more dimensions. If you limit one of the dimensions specified in a `DIMENSION` clause, then the status of that dimension is changed only during execution of this call to `OLAP_TABLE`; it does not affect the rest of your OLAP session. However, other commands can affect your session.

The syntax of this parameter is:

```
'olap_command'
```

For example:

```
'LIMIT product TO product_member_levelrel ''L2'''
```

Another use is to execute the OLAP FETCH command in this parameter and omit the limit map.

The power and flexibility of this parameter comes from its ability to process virtually any data manipulation commands available in the OLAP DML.

Limit_Map Parameter

The fourth (and last) parameter of the OLAP_TABLE function maps workspace objects to columns in the table and identifies the role of each one. It is called a limit map because it combines with the WHERE clause of a SQL SELECT statement to issue a series of LIMIT commands to the analytic workspace. The contents of the limit map populate the table specified in the *table_name* parameter.

All or part of the limit map can be stored in a text variable in the analytic workspace. To insert the variable in the limit map, precede the name of the variable with an ampersand (&). This practice is called ampersand substitution in the OLAP DML.

The syntax of the limit map has numerous clauses, primarily for defining dimension hierarchies. Pay close attention to the presence or absence of commas, since syntax errors will prevent your limit map from being parsed.

```
'[MEASURE column FROM {measure | AW_EXPR expression}]
          .
          .
          .
 DIMENSION [column FROM] dimension
    [WITH
       [HIERARCHY [column FROM] hierarchy_relation[(hierarchy_dimension 'hierarchy')]
          [INHIERARCHY inhierarchy_variable]
          [GID column FROM gid_variable]
          [PARENTGID column FROM gid_variable]
          [FAMILYREL col1, col2, coln FROM
             {expression1, expression2, expressionn |
              family_relation USING level_dimension }
             [LABEL label_variable]]
          .
          .
          .
       ]
       [ATTRIBUTE column FROM attribute_variable]
          .
          .
          .
```

```
   ]
 [ROW2CELL column]
 [LOOP composite_dimension]
 [PREDMLCMD olap_command]
 [POSTDMLCMD olap_command]
'
```

Where:

column is the name of a column in the target table.

measure is a business measure that is stored in the analytic workspace.

dimension is a dimension in the analytic workspace

expression is a formula or qualified data reference for objects in the analytic workspace

hierarchy_relation is a self-relation in the analytic workspace that defines the hierarchies for *dimension*.

hierarchy_dimension is a dimension in the analytic workspace that contains the names of the hierarchies for *dimension*.

hierarchy is a member of *hierarchy_dimension*.

inhierarchy_variable is a Boolean variable in the analytic workspace that identifies whether a dimension member is in *hierarchy*.

gid_variable is the name of a variable in the analytic workspace that contains the grouping ID of each dimension member.

attribute_variable is the name of a variable in the analytic workspace that contains attribute values for *dimension*.

sparse_dimension is the name of a composite dimension used in the definition of *measure*.

olap_command is an OLAP DML command.

MEASURE column FROM {measure | AW_EXPR expression}

The MEASURE clause maps a variable, formula, or relation in the analytic workspace to a column in the target table.

Alternatively, the AW_EXPR keyword can map a calculation performed by the OLAP engine on one or more of these objects to a column. For example, you could specify calculations such as these:

```
analytic_cube_sales - analytic_cube_cost
```

or

```
LAGDIF(analytic_cube_sales, 1, time, LEVELREL time.lvlrel)
```

You can list any number of MEASURE clauses. This clause is optional when, for example, you wish to create a dimension view.

Refer to "Measures" on page 12-3 for additional information about measures in an analytic workspace.

DIMENSION [column FROM] dimension...

The DIMENSION clause identifies a dimension or conjoint in the analytic workspace that dimensions one or more measures, attributes, or hierarchies in the limit map. Refer to "Dimensions" on page 12-3 for additional information about dimensions in an analytic workspace.

The *column* subclause is optional when you do not want the dimension members themselves to be represented in the table. In this case, you should include a dimension attribute that can be used for data selection.

Every limit map should have at least one DIMENSION clause. If the limit map contains MEASURE clauses, then it should also contain a single DIMENSION clause for each dimension of the measures, unless a dimension is being limited to a single value. If the measures are dimensioned by a composite, then you must identify each dimension in the composite with a DIMENSION clause. For the best performance when fetching a large result set, identify the composite in a LOOP clause.

A dimension can be named in only one DIMENSION clause. Subclauses of DIMENSION identify the dimension hierarchy and attributes.

WITH...

The WITH clause introduces a HIERARCHY or ATTRIBUTE subclause. If you omit these subclauses from the limit map, then omit the WITH clause also. However, if you include one or both of these subclauses, then precede them with a single WITH clause.

HIERARCHY [column FROM] hierarchy_relation[(hierarchy_dimension 'hierarchy')]...

The HIERARCHY subclause identifies the parent self-relation in the analytic workspace that defines the hierarchies for dimension. Refer to "Hierarchies" on

page 12-3 for additional information on dimension hierarchies in an analytic workspace.

If *hierarchy_dimension* has more than one member, then you can specify the one that you want with a *(hierarchy_dimension 'hierarchy')* phrase. To include multiple hierarchies, specify a HIERARCHY subclause for each one. The *hierarchy_dimension* is limited to *hierarchy* for all workspace objects that are referenced in subsequent subclauses (that is, INHIERARCHY, GID, PARENTGID, and FAMILYREL).

The HIERARCHY subclause is optional when *dimension* does not have a hierarchy, or when the status of *dimension* has been limited to a single level of the hierarchy.

INHIERARCHY inhierarchy_variable The INHIERARCHY subclause identifies a boolean variable in the analytic workspace that identifies whether a dimension member is in hierarchy. It is required only when there are members of the dimension that are omitted from the hierarchy, which is typical when a dimension has multiple hierarchies. Refer to "In-Hierarchy Variables" on page 12-5 for additional information about in-hierarchy variables.

GID column FROM gid_variable The GID subclause maps an integer variable in the analytic workspace, which contains the grouping ID for each dimension member, to a column in the target table. It is required for Java applications that use the OLAP API. Refer to "Grouping IDs" on page 12-6 for additional information about GIDs.

PARENTGID column FROM gid_variable The PARENTGID subclause calculates the grouping IDs for the parent relation using the GID variable in the analytic workspace. The parent GIDs are not stored in a workspace object. Instead, you specify the same GID variable for the PARENTGID clause that you used in the GID clause.

The PARENTGID clause is recommended for Java applications that use the OLAP API. Refer to "Grouping IDs" on page 12-6 for additional information about GIDs.

FAMILYREL col1, col2, coln FROM {expression1, expression2, expressionn | family_relation USING level_dimension } [LABEL label_variable] The FAMILYREL subclause is used primarily to map a family relation in the analytic workspace to multiple columns in the target table. List the columns in the order of level_dimension. If you do not want a particular level included, then specify null for the target column. The resulting view is in **rollup form**, in which each level of the hierarchy is represented in a separate column, and the full parentage of each dimension member is identified within the row. Refer to "Family Relations" on page 12-7 for more information about family relations.

The FAMILYREL subclause can also be used to map a list of qualified data references (QDRs) to multiple columns. In this usage, the first QDR maps to the first column, the second QDR maps to the second column, and so forth.

The LABEL keyword identifies a text attribute that provides more meaningful names for the dimension members.

You can use multiple FAMILYREL clauses for each hierarchy.

ATTRIBUTE column FROM attribute_variable

The ATTRIBUTE clause maps a variable in the analytic workspace to a column in the target table. If attribute_variable has multiple dimensions, then values are mapped for all members of *dimension*, but only for the first member in the current status of additional dimensions. For example, if your attributes have a language dimension, then you must set the status of that dimension to a particular language. You can set the status of dimensions in a PREDMLCMD clause.

ROW2CELL column

The ROW2CELL clause populates a RAW(32) column with information needed by the single-row functions in the DBMS_AW package. Use this clause when creating a view that will be used by these functions.

LOOP sparse_dimension

The LOOP clause identifies a single named composite that dimensions one or more measures specified in the limit map. It improves performance when fetching a large result set; however, it can slow the retrieval of a small number of values.

PREDMLCMD olap_command

The PREDMLCMD specifies an OLAP DML command that is executed before the data is fetched from the analytic workspace into the target table. It can be used, for example, to execute a model or forecast whose results will be fetched into the table.

POSTDMLCMD olap_command

The POSTDMLCMD specifies an OLAP DML command that is executed after the data is fetched from the analytic workspace into the target table. It can be used, for example, to delete objects or data that were created by commands in the PREDMLCMD clause, or to restore the dimension status that was changed in a PREDMLCMD clause.

Examples

Because different applications have different requirements, several different formats are commonly used for fetching data into SQL from an analytic workspace. The examples in this chapter show how to create views using a variety of different formats.

Although these examples are shown as views, the SELECT statements can be extracted from them and used directly to fetch data from an analytic workspace into an application.

Creating a View

To create a view, use a text editor to create a PL/SQL script that defines the row, the table, and the view. Example 12–3 is a template that you can use as the starting point for the SQL scripts that you will develop for views of your analytic workspace. You can then execute the script with the @ command in SQL*Plus.

Example 12–3 Template for Creating a View

```
SET ECHO ON
SET SERVEROUT ON

DROP TYPE table_obj;
DROP TYPE row_obj;

CREATE TYPE row_obj AS OBJECT (
   column_first     datatype,
   column_next      datatype,
   column_last      datatype);
/
CREATE TYPE table_obj AS TABLE OF row_obj;
/
CREATE OR REPLACE VIEW view AS
SELECT column1, column2, columnn
   FROM TABLE(OLAP_TABLE(
      'connection',
      'table_obj',
      'olap_command',
      'limit_map'));
/
COMMIT
/
GRANT SELECT ON view TO PUBLIC;
```

Creating Views of Embedded Total Dimensions

Example 12–4 shows the PL/SQL script used to create a view of the TIME dimension STANDARD hierarchy.

Example 12–4 Script for a Dimension View

```
CREATE TYPE time_std_row AS OBJECT (
   time_id                   VARCHAR2(16),
   standard_short_label      VARCHAR2(16),
   standard_end_date         DATE,
   standard_timespan         NUMBER(6));
/

CREATE TYPE time_std_table AS TABLE OF time_std_row;
/

CREATE OR REPLACE VIEW time_std_view AS
SELECT time_id, standard_short_label, standard_end_date, standard_timespan
FROM TABLE(OLAP_TABLE('xademo DURATION SESSION', 'time_std_table',
      'LIMIT time_hierlist TO ''STANDARD''',
      'DIMENSION time_id FROM time WITH
       HIERARCHY time_member_parentrel
           INHIERARCHY time_member_inhier
       ATTRIBUTE standard_short_label FROM time_short.description
       ATTRIBUTE standard_end_date FROM time_end_date
       ATTRIBUTE standard_timespan FROM time_time_span'));
/
```

```
SQL> SELECT * FROM time_std_view;

TIME_ID  STANDARD STANDARD_ STANDARD_TIMESPAN
-------- -------- --------- -----------------
L1.1996  1996     31-DEC-96               366
L1.1997  1997     31-MAY-97               151
L2.Q1.96 Q1.96    31-MAR-96                91
L2.Q2.96 Q2.96    30-JUN-96                91
L2.Q3.96 Q3.96    30-SEP-96                92
L2.Q4.96 Q4.96    31-DEC-96                92
L2.Q1.97 Q1.97    31-MAR-97                90
L2.Q2.97 Q2.97    31-MAY-97                61
L3.JAN96 Jan96    31-JAN-96                31
L3.FEB96 Feb96    29-FEB-96                29
L3.MAR96 Mar96    31-MAR-96                31

                                    .
                                    .
                                    .
```

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 show up.

Creating Views of Embedded Total Measures

In a star schema, a separate measure view is needed with columns that can be joined to each of the dimension views. Example 12–5 shows the PL/SQL script used to create a measure view with a column populated by ROW2CELL to support custom measures. For an example of creating a custom measure, refer to "OLAP_EXPRESSION Function" on page 11-12.

Example 12–5 Script for a Measure View

```
CREATE TYPE measure_row AS OBJECT (
   time                      VARCHAR2(12),
   geography                 VARCHAR2(30),
   product                   VARCHAR2(30),
   channel                   VARCHAR2(30),
   sales                     NUMBER(16),
   cost                      NUMBER(16),
   promotions                NUMBER(16),
   quota                     NUMBER(16),
   units                     NUMBER(16),
```

```
   r2c                         RAW(32));
/

CREATE TYPE measure_table AS TABLE OF measure_row;
/

CREATE OR REPLACE VIEW measure_view AS
SELECT sales, cost, promotions, quota, units,
      time, geography, product, channel, r2c
   FROM TABLE(OLAP_TABLE(
     'xademo DURATION SESSION',
     'measure_table',
     '',
     'MEASURE sales FROM analytic_cube_f.sales
      MEASURE cost FROM analytic_cube_f.costs
      MEASURE promotions FROM analytic_cube_f.promo
      MEASURE quota FROM analytic_cube_f.quota
      MEASURE units FROM analytic_cube_f.units
      DIMENSION time FROM time WITH
        HIERARCHY time_member_parentrel
           INHIERARCHY time_member_inhier
      DIMENSION geography FROM geography WITH
         HIERARCHY geography_member_parentrel
            INHIERARCHY geography_member_inhier
      DIMENSION product FROM product WITH
         HIERARCHY product_member_parentrel
            INHIERARCHY product_member_inhier
      DIMENSION channel FROM channel WITH
         HIERARCHY channel_member_parentrel
            INHIERARCHY channel_member_inhier
   ROW2CELL r2c'))
   WHERE sales IS NOT NULL;
/

SQL> SELECT channel, sales, cost, promotions, quota, units FROM measure_view
     WHERE    product = 'L1.TOTALPROD'
     AND    geography = 'L1.WORLD'
     AND         time = 'L1.1996';


CHANNEL                          SALES       COST PROMOTIONS      QUOTA     UNITS
------------------------------ ---------- ---------- ---------- ---------- ---------
STANDARD_1.CATALOG              76843552     125398     110249      16525     25209
STANDARD_1.DIRECT               41403560    2364845     518649    5458917    118851
STANDARD_2.TOTALCHANNEL        118247112    2490243     628898    5475442    144060
```

Creating Views in Rollup Form

Rollup form uses a column for each hierarchy level to show the full parentage of each dimension member. The only difference between the syntax for rollup form and the syntax for embedded total form is the addition of a FAMILYREL clause in the definition of each dimension in the limit map.

Example 12–6 shows the PL/SQL script used to create a rollup view of the PRODUCT dimension. It shows a dimension view to highlight the differences in the syntax of the limit map from the one used for the embedded total form, as shown in Example 12–4, "Script for a Dimension View". Note that the target columns for these levels are listed in the FAMILYREL clause from base level to most aggregate, which is the order they are listed in the level list dimension. The family relation returns four columns. The most aggregate level (all products) is omitted from the view by mapping it to null.

Example 12–7 shows the alternative syntax for the FAMILYREL clause, which uses QDRs to identify exactly which columns will be mapped from the family relation.

These two limit maps generate identical views.

Example 12–6 Script for a Rollup View of Products

```
CREATE TYPE product_row AS OBJECT (
   equipment      VARCHAR2(20),
   components     VARCHAR2(20),
   divisions      VARCHAR2(20));
/

CREATE TYPE product_table AS TABLE OF product_row;
/

CREATE OR REPLACE VIEW product_view AS
SELECT equipment, components, divisions
   FROM TABLE(OLAP_TABLE('xademo DURATION QUERY', 'product_table',
   '',
   'DIMENSION product WITH
       HIERARCHY product_member_parentrel
       FAMILYREL equipment, components, divisions, null
          FROM product_member_familyrel USING product_levellist
          LABEL product_short.description
     '));


SQL>  SELECT * FROM product_view
```

```
      ORDER BY divisions, components, equipment;

EQUIPMENT            COMPONENTS           DIVISIONS
-------------------- -------------------- --------------------
Chrm Cas             Audio Tape           Accessory Div
Mtl Cassette         Audio Tape           Accessory Div
Std Cassette         Audio Tape           Accessory Div
                     Audio Tape           Accessory Div
                     .
                     .
                     .
Standard VCR         VCR                  Video Div
Stereo VCR           VCR                  Video Div
                     VCR                  Video Div
                                          Video Div
```

Example 12–7 Script Using QDRs in the FAMILYREL Clause

```
CREATE TYPE product_row AS OBJECT (
   equipment      VARCHAR2(15),
   components     VARCHAR2(15),
   divisions      VARCHAR2(15));
/

CREATE TYPE product_table AS TABLE OF product_row;
/

CREATE OR REPLACE VIEW product_view AS
SELECT equipment, components, divisions
   FROM TABLE(OLAP_TABLE('xademo DURATION QUERY', 'product_table',
   '',
   'DIMENSION product WITH
       HIERARCHY product_member_parentrel
          FAMILYREL equipment, components, divisions FROM
             product_member_familyrel(product_levellist ''L4''),
             product_member_familyrel(product_levellist ''L3''),
             product_member_familyrel(product_levellist ''L2'')
             LABEL product_short.description
     '));
/

SQL> SELECT * FROM product_view
     ORDER BY divisions, components, equipment;

EQUIPMENT       COMPONENTS      DIVISIONS
```

```
-------------- -------------- ---------------
Chrm Cas       Audio Tape     Accessory Div
Mtl Cassette   Audio Tape     Accessory Div
Std Cassette   Audio Tape     Accessory Div
               Audio Tape     Accessory Div
               .
               .
               .
Standard VCR   VCR            Video Div
Stereo VCR     VCR            Video Div
               VCR            Video Div
                              Video Div
```

Part IV

OLAP Catalog Metadata API Reference

Part IV describes the PL/SQL APIs for creating and viewing `CWM2` metadata.

This part contains the following chapters:

13

Using the OLAP Catalog Metadata APIs

The OLAP Catalog PL/SQL packages provide stored procedures for creating, dropping, and updating OLAP metadata. This chapter explains how to call these procedures from within scripts. For complete syntax descriptions, refer to the reference chapter for each package.

See Also:

- Chapter 5, "Creating OLAP Catalog Metadata" for an introduction to the OLAP Catalog
- "OLAP Metadata Model" on page 4-8 for a description of the logical entities in the OLAP Catalog

This chapter discusses the following topics:

OLAP Metadata Entities

OLAP metadata entities are: **dimensions**, **hierarchies**, **levels**, **level attributes**, **dimension attributes**, **measures**, **cubes**, and **measure folders**. A separate PL/SQL package exists for each type of entity. The package provides procedures for creating, dropping, locking, and specifying descriptions for entities of that type. For example, to create a dimension, you would call `CWM2_OLAP_DIMENSION.CREATE_DIMENSION`, to create a level, you would call `CWM2_OLAP_LEVEL.CREATE_LEVEL`, and so on.

Each entity of metadata is uniquely identified by its owner and its name.

Note: When you create an OLAP metadata entity, you are simply adding a row to an OLAP Catalog table that identifies all the entities of that type. Creating an entity does not fully define a dimension or a cube, nor does it involve any mapping to warehouse dimension tables or fact tables.

To fully construct a dimension or a cube, you must understand the hierarchical relationships between the component metadata entities.

Constructing a Dimension

Creating a dimension entity is only the first step in constructing the OLAP metadata for a dimension. Each dimension must have at least one level. More typically, it will have multiple levels, hierarchies, and attributes. Table 13–1 shows the parent-child relationships between the metadata components of a dimension.

Table 13–1 Hierarchical Relationships Between Components of a Dimension

Parent Entity	Child Entity
dimension	dimension attribute, hierarchy, level
dimension attribute	level attribute
hierarchy	level
level	level attribute

Procedure: Construct an OLAP Dimension

Generally, you will create hierarchies and dimension attributes after creating the dimension and before creating the dimension levels and level attributes. Once the levels and level attributes are defined, you can map them to columns in one or more warehouse dimension tables. The general steps are as follows:

1. Call procedures in `CWM2_OLAP_DIMENSION` to create the dimension.
2. Call procedures in `CWM2_OLAP_DIMENSION_ATTRIBUTE` to create dimension attributes.
3. Call procedures in `CWM2_OLAP_HIERARCHY` to define hierarchical relationships for the dimension's levels.
4. Call procedures in `CWM2_OLAP_LEVEL` to create levels and assign them to hierarchies.
5. Call procedures in `CWM2_OLAP_LEVEL_ATTRIBUTE` to create level attributes and assign them to dimension attributes.
6. Call procedures in `CWM2_OLAP_TABLE_MAP` to map the dimension's levels and level attributes to columns in a dimension table.

Constructing a Cube

Creating a cube entity is only the first step in constructing the OLAP metadata for a cube. Each cube must have at least one dimension and at least one measure. More typically, it will have multiple dimensions and multiple measures.

Procedure: Construct an OLAP Cube

The general steps for constructing a cube are as follows:

1. Follow the steps in "Procedure: Construct an OLAP Dimension" on page 13-3 to create the cube's dimensions.
2. Call procedures in `CWM2_OLAP_CUBE` to create the cube and identify its dimensions.
3. Call procedures in `CWM2_OLAP_MEASURE` to create the cube's measures.
4. Call procedures in `CWM2_OLAP_TABLE_MAP` to map the cube's measures to columns in a fact table and to map foreign key columns in the fact table to key columns in the dimension tables.

Mapping OLAP Metadata

OLAP metadata mapping is the process of establishing the links between logical metadata entities and the physical locations where the data is stored. Dimension levels and level attributes map to columns in dimension tables. Measures map to columns in fact tables. The mapping process also specifies the join relationships between a fact table and its associated dimension tables.

Note: The dimension tables and fact tables may be implemented as views. For example, the views you can generate using the CWM2_OLAP_AW_CREATE package may be the data source for OLAP metadata. These views project an image of relational fact tables and dimension tables over an analytic workspace, where the data actually resides.

Mapping to Columns

Each dimension level maps to one or more columns in a dimension table. All the columns of a multicolumn level must be mapped within the same table. All the levels of a dimension may be mapped to columns in the same table (a traditional star schema), or the levels may be mapped to columns in separate tables (snowflake schema).

The CWM2_OLAP_TABLE_MAP package contains the mapping procedures for CWM2 metadata. The MAP_DIMTBL_HIERLEVEL procedure maps a level of a given hierarchy to columns in a dimension table. The MAP_DIMTBL_LEVEL procedure maps a level with no hierarchical context to columns in a dimension table.

Each level attribute maps to a single column in the same table as its associated level. The MAP_DIMTBL_HIERLEVELATTR maps a level attribute of a given hierarchy to a column in a dimension table. The MAP_DIMTBL_LEVELATTR maps a level attribute with no hierarchical context to a column in a dimension table.

Each measure maps to a single column in a fact table. All the measures mapped within the same fact table must share the same dimensionality. The MAP_FACTTBL_MEASURE procedure maps a measure to a column in a fact table.

Joining Fact Tables with Dimension Tables

Once you have mapped the levels, level attributes, and measures, you can specify the mapping of logical foreign key columns in the fact table to level key columns in dimension tables.

The CWM2_OLAP_TABLE_MAP.MAP_FACTTBL_LEVELKEY procedure defines the join relationships between a cube and its dimensions. This procedure takes as input: the cube name, the fact table name, a mapping string, and a storage type indicator specifying how data is stored in the fact table.

The storage type indicator can have any of the following values:

- **LOWEST LEVEL** (Required in CWM, supported but not required in CWM2).
  A single fact table stores unsolved data for all the measures of a cube. If any of the cube's dimensions have more than one hierarchy, they must all have the same lowest level. Each foreign key column in the fact table maps to a level key column in a dimension table.
- **ET** (CWM2 only).
  Fact tables store completely solved data (with embedded totals) for specific hierarchies of the cube's dimensions. Typically, the data for each combination of hierarchies is stored in a separate fact table. Each fact table must have the same columns. Multiple hierarchies in dimensions do not have to share the same lowest level.

  An embedded total key and a grouping ID key (GID) in the fact table map to corresponding columns that identify a dimension hierarchy in a solved dimension table. The ET key identifies the lowest level value present in a row. The GID identifies the hierarchy level associated with each row. For more information, see "Grouping ID Column" on page 9-6.
- **ROLLED UP** (CWM2 only).
  Same as for ET, but with key columns in the fact table for each level of each dimension hierarchy. The presence of fully populated level keys in the fact table facilitates aggregation at runtime.

Validating OLAP Metadata

To test the validity of OLAP metadata, use the VALIDATE_CUBE and VALIDATE_DIMENSION procedures in the CWM2_OLAP_VALIDATE package. The validation process checks the structural integrity of the metadata and verifies that it is properly mapped to columns in tables or views.

Important: The validation process ensures that mapping information has been properly specified. It does not ensure that the source tables and columns still exist.

You can determine whether or not a cube is valid by checking the INVALID column of the ALL_OLAP2_CUBES view. You can determine whether or not a dimension is valid by checking the INVALID column of the ALL_OLAP2_DIMENSIONS view.

Structural Validation

Structural validation ensures that cubes and dimensions have all their required components parts.

Cubes

To be structurally valid, a cube must meet the following criteria:

- It must have at least one valid dimension.
- It must have at least one measure.

Dimensions

To be structurally valid, a dimension must meet the following criteria:

- It must have at least one level.
- It may have one or more hierarchies. Each hierarchy must have at least one level.
- It may have one or more dimension attributes. Each dimension attribute must have at least one level attribute.

Mapping Validation

Mapping validation ensures that the metadata has been properly mapped to columns in tables or views.

Cubes

To be valid, a cube's mapping must meet the following criteria:

- It must be mapped to one or more fact tables.
- All of the cube's measures must be mapped to columns in a fact table. If there are multiple fact tables, all the measures must be in each one.
- Every dimension/hierarchy combination must be mapped to one of the fact tables.

Dimensions

To be valid, a dimension's mapping must meet the following criteria:

- All levels must be mapped to columns in a dimension table.
- Level attributes must be mapped to columns in the same table as the corresponding levels.

Invoking the Procedures

When using the OLAP Catalog write API, you should be aware of logic and conventions that are common to all the CWM2 procedures.

Security Checks and Error Conditions

Each CWM2 procedure first checks the calling user's security privileges. The calling user must be the entity owner and must have the OLAP_DBA role. If the calling user does not meet the security requirements, the procedure fails with an exception. For example, if your identity is jsmith, you cannot successfully execute CWM2_OLAP_HIERARCHY.DROP_HIERARCHY for a hierarchy owned by jjones.

After verifying the security requirements, each procedure checks for the existence of the entity and of its parent entities. All procedures, except CREATE procedures, return an error if the entity does not already exist. For example, if you call CWM2_OLAP_LEVEL.SET_DESCRIPTION, and the level does not already exist, the procedure will fail. Similarly, if you call CWM2_OLAP_MEASURE.SET_DESCRIPTION and the measure exists but the parent cube does not exist, then the procedure will fail.

Case Requirements for Parameters

You can specify arguments to CWM2 procedures in lower case, upper case, or mixed case.

If the argument is a metadata entity name (for example, dimension_name) or a value that will be used in further processing by other procedures (for example, the solved_code of a hierarchy), the procedure converts the argument to upper case. For all other arguments, the case that you specify is retained.

Creating and Saving Metadata

None of the procedures that create, map, and validate OLAP metadata include a COMMIT. Your script should execute all the statements that create and map new

metadata, then validate the metadata by calling procedures in CWM2_OLAP_VALIDATE, and finally do a COMMIT to commit the new metadata to the database.

However, if the metadata is specifically for the OLAPI API, you must refresh the OLAP API Metadata Reader tables after validating the metadata. This procedure, CWM2_OLAP_METADATA_REFRESH.MR_REFRESH, *does* include a COMMIT.

Viewing OLAP Catalog Metadata

A set of views, identified by the ALL_OLAP2 prefix, presents the metadata in the OLAP Catalog. The metadata may have been created with the CWM2 PL/SQL packages or with Enterprise Manager. The ALL_OLAP2 views are automatically populated whenever changes are made to the metadata.

A second set of views, identified by the MRV_OLAP prefix, also presents OLAP Catalog metadata. However, these views are structured specifically to support fast querying by the OLAP API's Metadata Reader. These views must be explicitly refreshed whenever changes are made to the metadata.

See Also:

- Chapter 14, "Viewing OLAP Catalog Metadata" for more information on the ALL_OLAP2 views.
- Chapter 24, "CWM2_OLAP_METADATA_REFRESH" for more information on refreshing metadata tables for the OLAP API.

Example: Creating OLAP Metadata for a Dimension Table

Example 13–1 Creating Metadata for a Dimension Table

In the Sales History sample schema, PRODUCTS is a dimension table with the following columns:

Column Name	Data Type
PROD_ID	NUMBER
PROD_NAME	VARCHAR2
PROD_DESC	VARCHAR2
PROD_SUBCATEGORY	VARCHAR2
PROD_SUBCAT_DESC	VARCHAR2

Column Name	Data Type
PROD_CATEGORY	VARCHAR2
PROD_CAT_DESC	VARCHAR2
PROD_WEIGHT_CLASS	NUMBER
PROD_UNIT_OF_MEASURE	VARCHAR2
PROD_PACK_SIZE	VARCHAR2
SUPPLIER_ID	NUMBER
PROD_STATUS	VARCHAR2
PROD_LIST_PRICE	NUMBER
PROD_MIN_PRICE	NUMBER
PROD_TOTAL	VARCHAR2

The following statements, excerpted from a PL/SQL script, create a logical CWM2 dimension, PRODUCT_DIM, for the PRODUCTS dimension table.

```
---   Create the PRODUCT Dimension    ---
cwm2_olap_dimension.create_dimension('SH', 'PRODUCT_DIM', 'Product',
          'Products', 'Product Dimension', 'Product Dimension Values');

---   Create Dimension Attributes  ---
cwm2_olap_dimension_attribute.create_dimension_attribute('SH', 'PRODUCT_DIM',
          'Long Description', 'Long Descriptions',
          'Long Desc', 'Long Product Descriptions', true);
cwm2_olap_dimension_attribute.create_dimension_attribute('SH', 'PRODUCT_DIM',
          'PROD_NAME_DIM', 'Product Name',
          'Prod Name', 'Product Name');

---   Create STANDARD Hierarchy  ---
cwm2_olap_hierarchy.create_hierarchy('SH', 'PRODUCT_DIM', 'STANDARD',
         'Standard', 'Std Product', 'Standard Product Hierarchy',
         'Unsolved Level-Based');

---   Create Levels  ---
cwm2_olap_level.create_level('SH', 'PRODUCT_DIM', 'L4',
        'Product ID', 'Product Identifiers',
        'Prod Key','Product Key');
cwm2_olap_level.create_level('SH', 'PRODUCT_DIM', 'L3',
```

```
        'Product Sub-Category','Product Sub-Categories',
        'Prod Sub-Category', 'Sub-Categories of Products');
cwm2_olap_level.create_level('SH', 'PRODUCT_DIM', 'L2',
        'Product Category', 'Product Categories',
        'Prod Category', 'Categories of Products');
cwm2_olap_level.create_level('SH', 'PRODUCT_DIM', 'L1',
        'Total Product', 'Total Products',
        'Total Prod', 'Total Product');

---   Create Level Attributes  ---
cwm2_olap_level_attribute.create_level_attribute('SH', 'PRODUCT_DIM',
         'Long Description', 'L4', 'Long Description',
         'PRODUCT_LABEL', 'L4 Long Desc',
         'Long Labels for PRODUCT Identifiers', TRUE);
cwm2_olap_level_attribute.create_level_attribute('SH', 'PRODUCT_DIM',
        'Long Description', 'L3', 'Long Description',
        'SUBCATEGORY_LABEL', 'L3 Long Desc',
        'Long Labels for PRODUCT Sub-Categories', TRUE);
cwm2_olap_level_attribute.create_level_attribute('SH', 'PRODUCT_DIM',
        'Long Description', 'L2', 'Long Description',
        'CATEGORY_LABEL', 'L2 Long Desc',
        'Long Labels for PRODUCT Categories', TRUE);
cwm2_olap_level_attribute.create_level_attribute('SH', 'PRODUCT_DIM',
          'PROD_NAME_DIM', 'L4', 'PROD_NAME_LEV',
          'Product Name', 'Product Name');

---   Add levels to hierarchies  ---
cwm2_olap_level.add_level_to_hierarchy('SH', 'PRODUCT_DIM', 'STANDARD',
          'L4', 'L3');
cwm2_olap_level.add_level_to_hierarchy('SH', 'PRODUCT_DIM', 'STANDARD',
          'L3', 'L2');
cwm2_olap_level.add_level_to_hierarchy('SH', 'PRODUCT_DIM', 'STANDARD',
          'L2', 'L1');
cwm2_olap_level.add_level_to_hierarchy('SH', 'PRODUCT_DIM', 'STANDARD',
          'L1');

---   Create mappings  ---
cwm2_olap_table_map.Map_DimTbl_HierLevel('SH', 'PRODUCT_DIM',
          'STANDARD', 'L4',
          'SH', 'PRODUCTS', 'PROD_ID');
cwm2_olap_table_map.Map_DimTbl_HierLevelAttr('SH', 'PRODUCT_DIM',
          'Long Description', 'STANDARD', 'L4', 'Long Description', 'SH',
          'PRODUCTS', 'PROD_DESC');
cwm2_olap_table_map.Map_DimTbl_HierLevelAttr('SH', 'PRODUCT_DIM',
          'PROD_NAME_DIM', 'STANDARD', 'L4', 'PROD_NAME_LEV', 'SH',
```

```
          'PRODUCTS', 'PROD_NAME');
cwm2_olap_table_map.Map_DimTbl_HierLevel('SH', 'PRODUCT_DIM',
          'STANDARD', 'L3','SH', 'PRODUCTS', 'PROD_SUBCATEGORY');
cwm2_olap_table_map.Map_DimTbl_HierLevelAttr('SH', 'PRODUCT_DIM',
          'Long Description', 'STANDARD', 'L3', 'Long Description', 'SH',
          'PRODUCTS', 'PROD_SUBCAT_DESC');
cwm2_olap_table_map.Map_DimTbl_HierLevel('SH', 'PRODUCT_DIM',
          'STANDARD', 'L2','SH', 'PRODUCTS', 'PROD_CATEGORY');
cwm2_olap_table_map.Map_DimTbl_HierLevelAttr('SH', 'PRODUCT_DIM',
          'Long Description', 'STANDARD', 'L2', 'Long Description', 'SH',
          'PRODUCTS', 'PROD_CAT_DESC');
cwm2_olap_table_map.Map_DimTbl_HierLevel('SH', 'PRODUCT_DIM',
          'STANDARD', 'L1','SH', 'PRODUCTS', 'PROD_TOTAL');
```

Example: Creating OLAP Metadata for a Fact Table

In the Sales History sample schema, COSTS is a fact table with the following columns.I

Column Name	Data Type
PROD_ID	NUMBER
TIME_ID	DATE
UNIT_COST	NUMBER
UNIT_PRICE	NUMBER

The following statements create a logical CWM2 cube object, ANALYTIC_CUBE, for the COSTS fact table. The dimensions of the cube are: PRODUCT_DIM, shown in "Example: Creating OLAP Metadata for a Dimension Table" on page 13-8, and TIME_DIM, a time dimension mapped to a table TIME.

```
---   Create the ANALYTIC_CUBE Cube  ---
cwm2_olap_cube.create_cube('SH', 'ANALYTIC_CUBE', 'Analytics',
     'Analytic Cube','Unit Cost and Price Analysis');

---  Add the dimensions to the cube  ---
cwm2_olap_cube.add_dimension_to_cube('SH', 'ANALYTIC_CUBE',
         'SH', 'TIME_DIM');
cwm2_olap_cube.add_dimension_to_cube('SH', 'ANALYTIC_CUBE',
         'SH', 'PRODUCT_DIM');

---   Create the measures  ---
```

```
cwm2_olap_measure.create_measure('SH', 'ANALYTIC_CUBE', 'UNIT_COST',
          'Unit Cost','Unit Cost', 'Unit Cost');
cwm2_olap_measure.create_measure('SH', 'ANALYTIC_CUBE', 'UNIT_PRICE',
          'Unit Price','Unit Price', 'Unit Price');

---   Create the mappings  ---
cwm2_olap_table_map.Map_FactTbl_LevelKey
     ('SH', 'ANALYTIC_CUBE','SH', 'COSTS', 'LOWEST LEVEL',
      'DIM:SH.PRODUCTS/HIER:STANDARD/LVL:L4/COL:PROD_ID;
       DIM:SH.TIME/HIER:CALENDAR/LVL:L3/COL:MONTH;');
cwm2_olap_table_map.Map_FactTbl_Measure
     ('SH', 'ANALYTIC_CUBE','UNIT_COST', 'SH', 'COSTS', 'UNIT_COST',
      'DIM:SH.PRODUCTS/HIER:STANDARD/LVL:L4/COL:PROD_ID;
       DIM:SH.TIME/HIER:CALENDAR/LVL:L3/COL:MONTH;');
cwm2_olap_table_map.Map_FactTbl_Measure
     ('SH', 'ANALYTIC_CUBE','UNIT_PRICE', 'SH', 'COSTS', 'UNIT_PRICE',
      'DIM:SH.PRODUCTS/HIER:STANDARD/LVL:L4/COL:PROD_ID;
       DIM:SH.TIME/HIER:CALENDAR/LVL:L3/COL:MONTH;');
```

14

Viewing OLAP Catalog Metadata

This chapter describes the OLAP Catalog metadata views. All OLAP metadata, whether created with the `CWM2` PL/SQL packages or with Enterprise Manager, is presented in these views.

> **See Also:** Chapter 13, "Using the OLAP Catalog Metadata APIs".

> **Note:** A second set of views, called the OLAP API Metadata Reader views, presents much of the same information as the OLAP Catalog views. The Metadata Reader views are structured to facilitate fast queries by the OLAP API. See Chapter 24 for more information.

This chapter discusses the following topics:

- Access to OLAP Catalog Views
- Views of the Dimensional Model
- Views of Mapping Information

Access to OLAP Catalog Views

The OLAP Catalog read API consists of two sets of corresponding views:

- ALL_ views displaying all valid OLAP metadata accessible to the current user.
- DBA_ views displaying all OLAP metadata (both valid and invalid) in the entire database. DBA_ views are intended only for administrators.

> **Note:** The OLAP Catalog tables are owned by OLAPSYS. To create OLAP metadata in these tables, the user must have the OLAP_DBA role.

The columns of the ALL_ and DBA_ views are identical. Only the ALL_ views are listed in this chapter.

Views of the Dimensional Model

The following views show the basic dimensional model of OLAP metadata.

For more information on the logical model, see Chapter 4, "Designing Your Database for OLAP".

Table 14–1 OLAP Catalog Dimensional Model Views

View Name Synonym	Description
ALL_OLAP2_CUBES	Lists all cubes in an Oracle instance.
ALL_OLAP2_CUBE_MEASURES	Lists the measures within each cube.
ALL_OLAP2_CUBE_DIM_USES	Lists the dimensions within each cube.
ALL_OLAP2_CUBE_MEAS_DIM_USES	Shows how each measure is aggregated along each of its dimensions.
ALL_OLAP2_DIMENSIONS	Lists all OLAP dimensions in an Oracle instance.
ALL_OLAP2_DIM_HIERARCHIES	Lists the hierarchies within each dimension.
ALL_OLAP2_DIM_LEVELS	Lists the levels within each dimension.
ALL_OLAP2_DIM_ATTRIBUTES	Lists the dimension attributes within each dimension.
ALL_OLAP2_DIM_LEVEL_ATTRIBUTES	Lists the level attributes within each level.
ALL_OLAP2_DIM_ATTR_USES	Shows how level attributes are associated with each dimension attribute.
ALL_OLAP2_DIM_HIER_LEVEL_USES	Show how levels are ordered within each hierarchy.
ALL_OLAP2_CATALOGS	List all measure folders (catalogs) within the Oracle instance.
ALL_OLAP2_CATALOG_ENTITY_USES	Lists the measures within each measure folder.
ALL_OLAP2_ENTITY_DESC_USES	Lists the reserved attributes that have application-specific meanings. Examples are dimension attributes that are used for long and short descriptions and time-series calculations (end date, time span, period ago, and so on).

Views of Mapping Information

The following views show how the basic dimensional model is mapped to relational tables or views.

Table 14–2 OLAP Catalog Mapping Views

View Synonym Name	Description
ALL_OLAP2_CUBE_MEASURE_MAPS	Shows the mapping of each measure to a column.
ALL_OLAP2_DIM_LEVEL_ATTR_MAPS	Shows the mapping of each level attribute to a column.
ALL_OLAP2_LEVEL_KEY_COLUMN_USES	Shows the mapping of each level to a unique key column.
ALL_OLAP2_JOIN_KEY_COLUMN_USES	Shows the joins between two levels in a hierarchy.
ALL_OLAP2_HIER_CUSTOM_SORT	Shows the default sort order for level columns within hierarchies.
ALL_OLAP2_FACT_TABLE_GID	Shows the Grouping ID column for each hierarchy in each fact table.
ALL_OLAP2_FACT_LEVEL_USES	Shows the joins between dimension tables and fact tables in a star or snowflake schema.

ALL_OLAP2_CUBES

ALL_OLAP2_CUBES lists all cubes in an Oracle instance.

Column	Data Type	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the cube.
CUBE_NAME	VARCHAR2(30)	NOT NULL	Name of the cube.
INVALID	VARCHAR2(2)	NOT NULL	Whether or not this cube is in an invalid state. See "Validating OLAP Metadata" on page 13-5.
DISPLAY_NAME	VARCHAR2(30)		Display name for the cube.
DESCRIPTION	VARCHAR2(2000)		Description of the cube.
MV_SUMMARYCODE	VARCHAR2(2)		If this cube has an associated materialized view, the MV summary code specifies whether it is in Grouping Set (**groupingset**) or Rolled Up (**rollup**) form. See Chapter 10, "Creating Materialized Views for the OLAP API".

ALL_OLAP2_CUBE_MEASURES

ALL_OLAP2_CUBE_MEASURES lists the measures within each cube.

Column	Data Type	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the cube that contains the measure.
CUBE_NAME	VARCHAR2(30)	NOT NULL	Name of the cube that contains the measure.
MEASURE_NAME	VARCHAR2(30)	NOT NULL	Name of the measure.
DISPLAY_NAME	VARCHAR2(30)		Display name for the measure.
DESCRIPTION	VARCHAR2(2000)		Description of the measure.

ALL_OLAP2_CUBE_DIM_USES

ALL_OLAP2_CUBE_DIM_USES lists the dimensions within each cube.

A dimension may be associated more than once with the same cube, but each association is specified in a separate row, under its own unique dimension alias.

Column	Data Type	NULL	Description
CUBE_DIMENSION_USE_ID	NUMBER	NOT NULL	ID of the association between a cube and a dimension.
OWNER	VARCHAR2(30)	NOT NULL	Owner of the cube.
CUBE_NAME	VARCHAR2(30)	NOT NULL	Name of the cube.
DIMENSION_OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension.
DIMENSION_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension.
DIMENSION_ALIAS	VARCHAR2(30)		Alias of the dimension, to provide unique identity of dimension use within the cube.
DEFAULT_CALC_ HIERARCHY_NAME	VARCHAR2(30)		The default hierarchy to be used for drilling up or down within the dimension.
DEPENDENT_ON_DIM_USE_ ID	NUMBER		ID of the cube/dimension association on which this cube/dimension association depends.

ALL_OLAP2_CUBE_MEAS_DIM_USES

ALL_OLAP2_CUBE_MEAS_DIM_USES shows how each measure is aggregated along each of its dimensions. The default aggregation method is addition.

Column	Data Type	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the cube that contains this measure.
CUBE_NAME	VARCHAR2(30)	NOT NULL	Name of the cube that contain this measure.
MEASURE_NAME	VARCHAR2(30)	NOT NULL	Name of the measure.
DIMENSION_OWNER	VARCHAR2(30)	NOT NULL	Owner of a dimension associated with this measure.
DIMENSION_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension.
DIMENSION_ALIAS	VARCHAR2(30)		Alias of the dimension.
DEFAULT_AGGR_ FUNCTION_USE_ID	NUMBER		The default aggregation method used to aggregate this measure's data over this dimension. If this column is null, the aggregation method is addition.

ALL_OLAP2_DIMENSIONS

ALL_OLAP2_DIMENSIONS lists all the OLAP dimensions in the Oracle instance.

OLAP dimensions created with the CWM2 APIs have no association with database dimension objects. OLAP dimensions created in Enterprise Manager are based on database dimension objects.

Column	Data Type	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension.
DIMENSION_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension.
PLURAL_NAME	VARCHAR2(30)		Plural name for the dimension. Used for display.
DISPLAY_NAME	VARCHAR2(30)		Display name for the dimension.
DESCRIPTION	VARCHAR2(2000)		Description of the dimension.
DEFAULT_DISPLAY_HIERARCHY	VARCHAR2(30)	NOT NULL	Default display hierarchy for the dimension.
INVALID	VARCHAR2(1)	NOT NULL	Whether or not the dimension is valid. See "Validating OLAP Metadata" on page 13-5
DIMENSION_TYPE	VARCHAR2(10)		Not used.

ALL_OLAP2_DIM_HIERARCHIES

ALL_OLAP2_DIM_HIERARCHIES lists the hierarchies within each dimension.

Column	Data Type	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension.
DIMENSION_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension.
HIERARCHY_NAME	VARCHAR2(30)	NOT NULL	Name of the hierarchy.
DISPLAY_NAME	VARCHAR2(30)		Display name for the hierarchy.
DESCRIPTION	VARCHAR2(2000)		Description of the hierarchy.
SOLVED_CODE	VARCHAR2(2)	NOT NULL	The solved code may be one of the following: **UNSOLVED LEVEL-BASED,** for a hierarchy that contains no embedded totals and is stored in a level-based dimension table. **SOLVED LEVEL-BASED**, for a hierarchy that contains embedded totals, has a grouping ID, and is stored in a level-based dimension table. **SOLVED VALUE-BASED,** for a hierarchy that contains embedded totals for all level combinations and is stored in a parent-child dimension table. For information about mapping hierarchies with different solved codes, see "Joining Fact Tables with Dimension Tables" on page 13-4.

ALL_OLAP2_DIM_LEVELS

ALL_OLAP2_DIM_LEVELS lists the levels within each dimension.

Column	Data Type	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension containing this level.
DIMENSION_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension containing this level.
LEVEL_NAME	VARCHAR2(30)	NOT NULL	Name of the level.
DISPLAY_NAME	VARCHAR2(30)		Display name for the level.
DESCRIPTION	VARCHAR2(2000)		Description of the level.
LEVEL_TABLE_OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension table that contains the columns for this level.

Column	Data Type	NULL	Description
LEVEL_TABLE_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension table that contains the columns for this level.

ALL_OLAP2_DIM_ATTRIBUTES

ALL_OLAP2_DIM_ATTRIBUTES lists the dimension attributes within each dimension.

Column	Data Type	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension.
DIMENSION_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension.
ATTRIBUTE_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension attribute.
DISPLAY_NAME	VARCHAR2(30)		Display name for the dimension attribute.
DESCRIPTION	VARCHAR2(2000)		Description of the dimension attribute.
DESC_ID	NUMBER		If the attribute is reserved, its type is listed in this column. Examples of reserved dimension attributes are long and short descriptions and time-related attributes, such as end date, time span, and period ago.

ALL_OLAP2_DIM_LEVEL_ATTRIBUTES

ALL_OLAP2_DIM_LEVEL_ATTRIBUTES lists the level attributes within each level.

Column	Data Type	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension containing the level.
DIMENSION_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension containing the level.
ATTRIBUTE_NAME	VARCHAR2(30)		Name of the level attribute. If no attribute name is specified, the column name is used.
DISPLAY_NAME	VARCHAR2(30)		Display name for the level attribute.
DESCRIPTION	VARCHAR2(2000)		Description of the level attribute.
DETERMINED_BY_ LEVEL_NAME	VARCHAR2(30)	NOT NULL	Name of the level.

ALL_OLAP2_DIM_ATTR_USES

ALL_OLAP2_DIM_ATTR_USES shows how level attributes are associated with each dimension attribute.

The same level attribute can be included in more than one dimension attribute.

Column	Data Type	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension.
DIMENSION_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension.
DIM_ATTRIBUTE_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension attribute.
LEVEL_NAME	VARCHAR2(30)	NOT NULL	Name of a level within the dimension.
LVL_ATTRIBUTE_NAME	VARCHAR2(30)	NOT NULL	Name of an attribute for this level. This level attribute is included in the dimension attribute.

ALL_OLAP2_DIM_HIER_LEVEL_USES

ALL_OLAP2_DIM_HIER_LEVEL_USES shows how levels are ordered within each hierarchy.

Within separate hierarchies, the same parent level may be hierarchically related to a different child level.

Column	Data Type	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension.
DIMENSION_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension.
HIERARCHY_NAME	VARCHAR2(30)	NOT NULL	Name of the hierarchy.
PARENT_LEVEL_NAME	VARCHAR2(30)	NOT NULL	Name of the parent level.
CHILD_LEVEL_NAME	VARCHAR2(30)	NOT NULL	Name of the child level.
POSITION	NUMBER	NOT NULL	Position of this parent-child relationship within the hierarchy, with position 1 being the most detailed.

ALL_OLAP2_CATALOGS

ALL_OLAP2_CATALOGS lists all the measure folders (catalogs) within the Oracle instance.

Column	Data Type	NULL	Description
CATALOG_ID	NUMBER	NOT NULL	ID of the measure folder.
CATALOG_NAME	VARCHAR2(30)	NOT NULL	Name of the measure folder.
PARENT_CATALOG_ID	NUMBER		ID of the parent measure folder. This column is null for measure folders at the root of the measure folder tree.
DESCRIPTION	VARCHAR2(2000)		Description of the measure folder.

ALL_OLAP2_CATALOG_ENTITY_USES

ALL_OLAP2_CATALOG_ENTITY_USES lists the measures within each measure folder.

Column	Data Type	NULL	Description
CATALOG_ID	NUMBER	NOT NULL	ID of the measure folder.
ENTITY_OWNER	VARCHAR2(30)	NOT NULL	Owner of the measure's cube.
ENTITY_NAME	VARCHAR2(30)	NOT NULL	Name of the measure's cube.
CHILD_ENTITY_NAME	VARCHAR2(30)	NOT NULL	Name of the measure in the measure folder.

ALL_OLAP2_ENTITY_DESC_USES

ALL_OLAP2_ENTITY_DESC_USES lists the reserved attributes and shows whether or not dimensions are time dimensions.

Column	Data Type	NULL	Description
DESCRIPTOR_ID	NUMBER	NOT NULL	Name of the reserved attribute or dimension type. The reserved dimension attributes are listed in Table 19–1, " Reserved Dimension Attributes" on page 19-2. The reserved level attributes are listed in Table 22–1, " Reserved Level Attributes" on page 22-2.
ENTITY_OWNER	VARCHAR2(30)	NOT NULL	Owner of the metadata entity.
ENTITY_NAME	VARCHAR2(30)	NOT NULL	Name of the metadata entity.

Column	Data Type	NULL	Description
CHILD_ENTITY_NAME	VARCHAR2(30)		Name of the child entity (if applicable). A dimension attribute is a child entity of a dimension. A level attribute is a child entity of a dimension attribute.
SECONDARY_CHILD_ENTITY_NAME	VARCHAR2(30)		Name of the secondary child entity name (if applicable). A dimension attribute is a child entity of a dimension. A level attribute is a child entity of a dimension attribute. A level attribute could be the secondary child entity of a dimension.

ALL_OLAP2_CUBE_MEASURE_MAPS

ALL_OLAP2_CUBE_MEASURE_MAPS shows the mapping of each measure to a column.

Column	Data Type	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the cube.
CUBE_NAME	VARCHAR2(30)	NOT NULL	Name of the cube.
MEASURE_NAME	VARCHAR2(30)	NOT NULL	Name of the measure contained in this cube.
DIM_HIER_COMBO_ID	NUMBER	NOT NULL	ID of the association between this measure and one combination of its dimension hierarchies.
FACT_TABLE_OWNER	VARCHAR2(30)	NOT NULL	Owner of the fact table.
FACT_TABLE_NAME	VARCHAR2(30)	NOT NULL	Name of the fact table.
COLUMN_NAME	VARCHAR2(30)	NOT NULL	Name of the column in the fact table where this measure's data is stored.

ALL_OLAP2_DIM_LEVEL_ATTR_MAPS

ALL_OLAP2_DIM_LEVEL_ATTR_MAPS shows the mapping of each level attribute to a column.

The mapping of level attributes to levels is dependent on hierarchy. The same level may have different attributes when it is used in different hierarchies.

Column	Data Type	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension.
DIMENSION_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension.
HIERARCHY_NAME	VARCHAR2(30)		Name of the hierarchy containing this level.

Column	Data Type	NULL	Description
ATTRIBUTE_NAME	VARCHAR2(30)		Name of a dimension attribute grouping containing this level attribute.
LVL_ATTRIBUTE_NAME	VARCHAR2(30)	NOT NULL	Name of the level attribute, or name of the column if the level attribute name is not specified.
LEVEL_NAME	VARCHAR2(30)	NOT NULL	Name of the level.
TABLE_OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension table containing the level and level attribute.
TABLE_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension table containing the level and level attribute columns.
COLUMN_NAME	VARCHAR2(30)	NOT NULL	Name of the column containing the level attribute.
DTYPE	VARCHAR2(10)	NOT NULL	Data type of the column containing the level attribute.

ALL_OLAP2_LEVEL_KEY_COLUMN_USES

ALL_OLAP2_LEVEL_KEY_COLUMN_USES shows the mapping of each level to a unique key column.

If the level is mapped to more than one column, each column mapping is represented in a separate row in the view.

Column	Data Type	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension.
DIMENSION_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension.
HIERARCHY_NAME	VARCHAR2(30)		Name of the hierarchy that includes this level.
CHILD_LEVEL_NAME	VARCHAR2(30)	NOT NULL	Name of the level.
TABLE_OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension table.
TABLE_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension table.
COLUMN_NAME	VARCHAR2(30)	NOT NULL	Name of the column that stores CHILD_LEVEL_NAME.
POSITION	NUMBER		Position of the column within the key. Applies to multi-column keys only (where the level is mapped to more than one column).

ALL_OLAP2_JOIN_KEY_COLUMN_USES

ALL_OLAP2_JOIN_KEY_COLUMN_USES shows the joins between two levels in a hierarchy. The joins are between dimension tables in a snowflake schema, and between level columns in a star schema.

If the level is mapped to more than one column, each column mapping is represented in a separate row in the view.

Column	Data Type	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension.
DIMENSION_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension.
HIERARCHY_NAME	VARCHAR2(30)	NOT NULL	Name of the hierarchy.
CHILD_LEVEL_NAME	VARCHAR2(30)	NOT NULL	Child level in the hierarchy.
TABLE_OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension table.
TABLE_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension table.
COLUMN_NAME	VARCHAR2(30)	NOT NULL	Name of the child level column in the dimension table. In a star schema, this is the column associated with CHILD_LEVEL_NAME. In a snowflake schema, this is the parent column of CHILD_LEVEL_NAME in the same dimension table.
POSITION	NUMBER		Position of column within the key. Applies to multi-column keys only (where the level is mapped to more than one column).
JOIN_KEY_TYPE	VARCHAR2(30)	NOT NULL	The key is of type **SNOWFLAKE** if the join key is a logical foreign key. The key is of type **STAR** if the join key refers to a column within the same table.

ALL_OLAP2_HIER_CUSTOM_SORT

ALL_OLAP2_HIER_CUSTOM_SORT shows the sort order for level columns within hierarchies. Custom sorting information is optional.

Custom sorting information specifies how to sort the members of a hierarchy based on columns in the dimension table. The specific columns in the dimension tables may be the same as the key columns or may be related attribute columns.

Custom sorting can specify that the column be sorted in ascending or descending order, with nulls first or nulls last. Custom sorting can be applied at multiple levels of a dimension.

Column	Data Type	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension.
DIMENSION_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension.
HIERARCHY_NAME	VARCHAR2(30)	NOT NULL	Name of the hierarchy.
TABLE_OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension table.
TABLE_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension table.
COLUMN_NAME	VARCHAR2(30)	NOT NULL	Name of the column to be sorted.
POSITION	NUMBER	NOT NULL	Represents the position within a multi-column SORT_POSITION. In most cases, a single column represents SORT_POSITION, and the value of POSITION is 1.
SORT_POSITION	NUMBER	NOT NULL	Position within the sort order of the level to be sorted.
SORT_ORDER	VARCHAR2(4)	NOT NULL	Sort order. Can be either Ascending or Descending.
NULL_ORDER	VARCHAR2(5)	NOT NULL	Where to insert null values in the sort order. Can be either **Nulls First** or Nulls Last.

ALL_OLAP2_FACT_TABLE_GID

ALL_OLAP2_FACT_TABLE_GID shows the Grouping ID column for each hierarchy in each fact table. For more information, see "Grouping ID Column" on page 9-6.

Column	Data Type	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the cube.
CUBE_NAME	VARCHAR2(30)	NOT NULL	Name of the cube.
DIMENSION_OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension.
DIMENSION_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension
HIERARCHY_NAME	VARCHAR2(30)	NOT NULL	Name of the hierarchy.
DIM_HIER_COMBO_ID	NUMBER	NOT NULL	ID of the dimension-hierarchy association.
FACT_TABLE_OWNER	VARCHAR2(30)	NOT NULL	Owner of the fact table.
FACT_TABLE_NAME	VARCHAR2(30)	NOT NULL	Name of the fact table.
COLUMN_NAME	VARCHAR2(30)	NOT NULL	Name of the GID column.

ALL_OLAP2_FACT_LEVEL_USES

ALL_OLAP2_FACT_LEVEL_USES shows the joins between dimension tables and fact tables in a star or snowflake schema. For more information, see "Joining Fact Tables with Dimension Tables" on page 13-4.

Column	Data Type	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the cube.
CUBE_NAME	VARCHAR2(30)	NOT NULL	Name of the cube.
DIMENSION_OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension.
DIMENSION_NAME	NUMBER	NOT NULL	Name of the dimension.
DIMENSION_ALIAS	VARCHAR2(30)		Dimension alias (if applicable).
HIERARCHY_NAME		NOT NULL	Name of the hierarchy.
DIM_HIER_COMBO_ID	NUMBER	NOT NULL	ID of the dimension hierarchy combination associated with this fact table.
LEVEL_NAME	VARCHAR2(30)		Name of the level within the hierarchy where the mapping occurs.
FACT_TABLE_OWNER	VARCHAR2(30)	NOT NULL	Owner of the fact table.
FACT_TABLE_NAME	VARCHAR2(30)	NOT NULL	Name of the fact table.
COLUMN_NAME	VARCHAR2(30)	NOT NULL	Name of the foreign key column in the fact table.
POSITION	NUMBER		Position of this column within a multi-column key.
DIMENSION_KEYMAP_TYPE	VARCHAR2(30)	NOT NULL	Type of key mapping for the fact table. Values may be: **LL** (Lowest Level), when only lowest-level dimension members are stored in the key column. The fact table is unsolved. **ET** (Embedded Totals), when dimension members for all level combinations are stored in the key column. The fact table is solved (contains embedded totals for all level combinations). **RU** (Rolled Up), when dimension members for each level are stored in a separate key column (multi-column key).
FOREIGN_KEY_NAME	VARCHAR2(30)		Name of the foreign key constraint applied to the foreign key column. Constraints are not used by the CWM2 APIs.

.

15

CWM2_OLAP_AW_ACCESS

The CWM2_OLAP_AW_ACCESS package contains procedures for generating scripts that create views of analytic workspace objects. After running the scripts and creating the views, you can use standard SQL to access data stored in the analytic workspace. You can also use the views to define OLAP metadata so that OLAP API applications can access the multidimensional objects.

See Also:

- Chapter 2, "Manipulating Multidimensional Data" for a discussion of analytic workspaces and the OLAP DML.
- Chapter 3, "Developing OLAP Applications" for information about how this package fits into the process of preparing a database for use with OLAP applications.
- Chapter 9, "Creating an Analytic Workspace From Relational Tables" for information on creating SQL access for analytic workspaces created by AW_CREATE.

This chapter contains the following topics:

When to Use the AW_ACCESS Package

If your analytic workspace was created by the CWM2_AW_CREATE package, you will use procedures in that package to generate the views of the workspace. Those procedures depend on structures within the analytic workspace that are specific to AW_CREATE.

If your analytic workspace was created by some other means (OLAP Worksheet or the DBMS_AW package), you can use the CWM2_AW_ACCESS package to generate the views. CWM2_AW_ACCESS is essentially a wrapper for the OLAP_TABLE function.

See Also:

- Chapter 12, "OLAP_TABLE".
- Chapter 9, "Creating an Analytic Workspace From Relational Tables" for information on creating SQL access for analytic workspaces created by AW_CREATE.

Prerequisites

The utl_file_dir parameter must be set to a valid directory, as described in "Initialization Parameters for Oracle OLAP" on page 6-3. Otherwise, the procedures in CWM2_OLAP_AW_ACCESS will not be able to write the SQL scripts to a file.

Process Overview

These are the basic steps you need to follow to generate views of data stored in an analytic workspace. They are described more fully throughout this chapter.

1. Explore the analytic workspace and identify the objects that you want to expose in a relational view.
2. For each view, create a text file that defines the mapping between analytic workspace objects and columns in the view.

   If you intend to create OLAP Catalog metadata, then you need to generate views that form a star schema, that is, fact views and dimension views. For more information about OLAP Catalog schema requirements, refer to Chapter 4, "Designing Your Database for OLAP".
3. In PL/SQL, execute the CreateAWAccessStructures_FR procedure for each input text file.

   Tip: Create a script that executes these procedures.

4. Use a text editor to view the resulting scripts and make whatever changes you wish.
5. In PL/SQL, run the scripts.
6. If errors are triggered, do the following:
   a. Identify and fix the problems in the input files.
   b. Delete the script files.

      `CreateAWAccessStructures_FR` will not overwrite an existing output file. If you created a script to execute this procedure on each of your input files, you may want to begin that script by deleting existing output files.

   c. Regenerate the script files.
7. In PL/SQL, select data from the views to verify that they work properly. Errors at this stage are caused by problems in the definition of the workspace objects.

   If necessary, correct the errors and regenerate the views.
8. When no errors occur, commit the views to the database.
9. Change the access protection of the views with commands such as this:

```
GRANT SELECT ON VIEW electro_product_view TO PUBLIC
```

Preparing the Analytic Workspace

The `CWM2_OLAP_AW_ACCESS` package can expose various types of analytic workspace objects in relational views. You will need to gather information about these objects and decide how you are going to map them to the columns of a relational view. These are the steps you might take:

1. Identify the measures that you want to make available to applications.
2. Identify the dimensions of the measures.
3. For hierarchical dimensions, identify the objects that support the hierarchy:
   - Parent relation
   - Hierarchy dimension
   - Inhierarchy variable
4. Identify the dimension attributes, which are objects that provide additional information about the dimensions.

5. If you plan to create OLAP Catalog metadata, be sure that you have a GID needed by the OLAP API. For a description of this object, refer to Chapter 12, "OLAP_TABLE".

Specifying the Source and Target Objects

A delimited text string specifies multidimensional source objects in the analytic workspace and maps them to target columns in a relational view. You can supply this delimited text string either in files (as described in "CreateAWAccessStructures_FR Procedure" on page 15-17) or directly in the command line (as described in "CreateAWAccessStructures Procedure" on page 15-18).

Each source and target object is defined by a keyword followed by one or more values. Two colons (`:`) delimit the keywords and values. In the following example, `MEASURE` is a keyword, and `sales` and `costs` are the names of measures in the analytic workspace.

```
MEASURE::sales::costs
```

When you provide mapping information in a text file, each keyword begins a new line:

```
MEASURE::sales::costs
MEASURE COLUMNS::sales::costs
```

When you provide mapping information directly in the command line, a semicolon delimits the individual object specifications:

```
MEASURE::sales::costs;MEASURE COLUMNS::sales::costs
```

Each call to one of these procedures generates a single view. For example, to create one fact view and three dimension views, you must execute the procedure four times. If you are supplying input files for the mapping information, then you must create four files, one for each view that you want to generate.

Note: If you are creating views that will be accessed directly using SQL, then you can structure the views in whatever way is appropriate for your application.

If you will use the views to create OLAP Catalog metadata, then you must create a star schema with measure views and dimension views as described in this chapter.

Defining Dimension Views

For a star schema, you must define a dimension view for every hierarchy of every dimension of the fact view. A flat dimension, that is, one with no hierarchies, requires a single dimension view.

Since each call to one of these procedures generates a single view, you must create a separate mapping file for each one. For example, if the GEOGRAPHY dimension has two hierarchies, then you need to create two mapping files.

Table 15–1 describes the keywords that identify the source data in an analytic workspace that will be used to create a dimension view. The object naming conventions used by AW_CREATE are provided in the description of the source data. Table 15–2 describes the keywords that specify the target columns in the generated database dimension view. Enter these keywords in the same input file. Some of these keywords are required and others are optional. DIMENSION must be the first keyword. The

Table 15–1 Keywords for Defining the Source Data for a Dimension View

Keyword	Description
DIMENSION	A workspace DIMENSION, which dimensions the measures in the fact view, as described in "Dimensions" on page 12-3. This keyword must appear first. Required. AW_CREATE name: [*owner_*]*dimension*
HIERARCHY	A workspace RELATION that identifies the parent value for each dimension value in the hierarchy, as described in "Hierarchies" on page 12-3. Required for hierarchical dimensions. AW_CREATE name: [*owner_*]*dimension*_PARENTREL
IN HIERARCHY	A workspace VARIABLE with a BOOLEAN data type that identifies whether or not each value of DIMENSION is included in the hierarchy, as described in "In-Hierarchy Variables" on page 12-5. Required only when some dimension members are omitted from the hierarchy. AW_CREATE name: [*owner_*]*dimension*_INHIERARCHY

Table 15–1 (Cont.) Keywords for Defining the Source Data for a Dimension View

Keyword	Description
HIERARCHY DIMENSION	The workspace DIMENSION that contains the names of the hierarchies, as described in "Hierarchy Dimensions" on page 12-4. Required only if more than one hierarchy is defined for DIMENSION. AW_CREATE name: [*owner*_]*dimension*_HIERDIM
HIERARCHY DIMENSION VALUE	The dimension member in the HIERARCHY DIMENSION object that identifies the hierarchy. Required only if HIERARCHY DIMENSION is specified.
GID	A workspace VARIABLE with an INTEGER data type that identifies the hierarchy level of each dimension value. Use the GROUPINGID command to generate this variable, as described in "Grouping IDs" on page 12-6. Improves performance of the OLAP API. AW_CREATE name: [*owner*_]*dimension*_GID
PARENT GID	The same GID variable used with the GID keyword. Parent grouping IDs will be generated automatically from the GID variable.
ATTRIBUTES	One or more workspace VARIABLE objects that contain descriptive information about the dimension members, as described in "Attributes" on page 12-8. Optional. AW_CREATE name: [*owner*_]*dimension_attribute*
COLUMN LEVEL DIMENSION	A workspace DIMENSION whose values identify the levels of a cube dimension, and which dimensions the COLUMN LEVEL RELATION object. Required for hierarchical dimensions. AW_CREATE name: [*owner*_]*dimension*_LVLDIM
COLUMN LEVEL RELATION	A workspace RELATION with a value for each level in the hierarchy. The HIERHEIGHT command in the OLAP DML generates the values of this relation. Required for hierarchical dimensions. AW_CREATE name: [*owner*_]*dimension*_FAMILYREL

Important: When listing the keywords for the target columns, you must list `DIMENSION COLUMN`, `PARENT COLUMN`, and `GID COLUMN` in that order. All column names must comply with Oracle requirements.

Table 15–2 Keywords for Defining the Target Columns for a Dimension View

Keyword	Description
`DIMENSION COLUMN`	A valid name for the column that will represent dimension values from `DIMENSION`. Required.
`PARENT COLUMN`	A valid name for the column that will represent the parent value for each dimension value. Required for hierarchical dimensions.
`GID COLUMN`	A valid name for the column that will represent the grouping IDs from `GID`. Required for hierarchical dimensions.
`PARENT GID COLUMN`	A valid name for the column that will represent the calculated parent grouping IDs. Optional.
`DIMENSION DATATYPES`	The data types of the previously specified columns, as follows: First value: `DIMENSION COLUMN` Second value: `PARENT COLUMN` Third value: `GID COLUMN` Fourth value: `PARENT GID` Required for each defined column. For information about compatible workspace and database data types, search for the `SQL FETCH` command in the Oracle9i OLAP DML Reference help.
`LEVEL COLUMNS`	Valid names for the columns that represent level values. You must identify a column for each value in `COLUMN LEVEL DIMENSION`. For example, if the level dimension has four values, then you must define four columns. Required for hierarchical dimensions.

Table 15–2 (Cont.) Keywords for Defining the Target Columns for a Dimension View

Keyword	Description
LEVEL DATATYPES	The data types of the columns listed in LEVEL COLUMNS. The data types must correspond in number and order to the columns listed in LEVEL COLUMNS, that is, the first column will be defined with the first data type, the second column will be defined with the second data type, and so forth. Required when LEVEL COLUMNS is specified.
ATTRIBUTE COLUMNS	Valid names for the columns that represent attribute values. The columns must correspond in number and order to the variables listed in ATTRIBUTES, that is, the first column will represent the first variable, the second column will represent the second variable, and so forth. Optional.
ATTRIBUTE DATATYPES	The data type of the columns listed in ATTRIBUTE COLUMNS. The data types must correspond in number and order to the columns listed in ATTRIBUTE COLUMNS, that is, the first column will be defined with the first data type, the second column will be defined with the second data type, and so forth. Required when ATTRIBUTE COLUMNS is specified.

Defining Fact Views

You can create a single group of views for several measures if they are dimensioned identically, as described in "Measures" on page 12-3.

For the OLAP API, you need to create one view for each combination of dimension hierarchies. The views must contain columns for the measures themselves and the dimension values that qualify this data. You can copy statements from the input files for dimension views into the input files for fact views.

Create input files (or text strings) that includes the following keywords:

- All of the keywords in Table 15–3. They must appear in the order shown at the beginning, before keywords for the dimensions.
- The following keywords from Table 15–1, " Keywords for Defining the Source Data for a Dimension View" if they appear in the input file for the dimension view: DIMENSION, HIERARCHY, IN HIERARCHY, and GID. If you wish to create a denormalized view for use by SQL applications, you can include additional keywords.

- Keywords from Table 15–2, " Keywords for Defining the Target Columns for a Dimension View" that correspond to the source data keywords. The OLAP API uses the DIMENSION and GID columns in the fact views, and uses the dimension views for all other information about the dimensions. Thus, you only need to define columns for the dimension members and the GIDs.

Table 15–3 lists the keywords that map workspace measures to columns in a fact view.

Table 15–3 Additional Keywords for Defining a Fact View

Keyword	Description
MEASURE	One or more workspace VARIABLE, RELATION, or FORMULA objects that are dimensioned identically, as described in "Measures" on page 12-3. The MEASURE keyword must appear before the other keywords listed in this table.
MEASURE COLUMNS	The names for the columns in the fact view where the data from MEASURE will be represented. You can specify any valid column name. The columns correspond in number and order to the workspace objects listed in MEASURE, that is, the first measure will be mapped to the first column, the second measure to the second column, and so forth.
MEASURE DATATYPES	The data types of the columns in the fact view. The data types must correspond in number and order to the columns listed in MEASURE COLUMNS, that is, the first column will be defined with the first data type, the second column will be defined with the second data type, and so forth. For a comparison between workspace data types and database data types, search for the SQL FETCH command in the Oracle9i OLAP DML Reference help.

Example: Creating Views

This example creates fact views and dimension views for two variables, sales and costs. These variables were not created by the AW_CREATE process.

The following are the object definitions for sales and costs. Note that they are dimensioned identically.

```
DEFINE SALES VARIABLE SHORT <GEOGRAPHY PRODUCT CHANNEL TIME>
DEFINE COSTS VARIABLE SHORT <GEOGRAPHY PRODUCT CHANNEL TIME>
```

In a star schema for use with OLAP Catalog metadata, you would create dimension views for each hierarchy and fact views for each combination of dimension hierarchies.

If the hierarchies shown in Table 15–4 have been defined for these dimensions, then the following views must be generated:

- Six dimension views(2+1+1+2)
- Four fact views (2*1*1*2)

Table 15–4 Sample Dimension Hierarchies

Dimensions	Hierarchies	Required Number of Views
geography	standard	2
	consolidated	
product	standard	1
channel	standard	1
time	standard	2
	ytd	

Example: Input Files for Mapping Variables to Views

This example creates views in a star schema for use by the OLAP API.

Geography Dimension Standard Hierarchy View

These statements define the geography dimension view for the STANDARD hierarchy. A separate file is required to generate another view to support the CONSOLIDATED hierarchy, but it is not included in this example.

```
DIMENSION::geography
HIERARCHY::geography.parentrel
INHIERARCHY: geography.inhierarchy
HIERARCHY DIMENSION::geography.hierarchies
HIERARCHY DIMENSION VALUE::STANDARD
GID::geography.gid
PARENT GID::geography.gid
ATTRIBUTES::geography.longlabel::geography.shortlabel
COLUMN LEVEL DIMENSION::geography.lvldim
COLUMN LEVEL RELATION::geography.hierheight

DIMENSION COLUMN::geography
```

```
PARENT COLUMN::geog_parent
GID COLUMN::geog_gid
PARENT GID COLUMN::geogp_gid
DIMENSION DATATYPES::varchar2(16)::varchar2(16)::number(10)::number(10)
LEVEL COLUMNS::city::country::continent::world
LEVEL DATATYPES::varchar2(16)::varchar2(16)::varchar2(16)::varchar2(16)
ATTRIBUTE COLUMNS::geog_long::geog_short
ATTRIBUTE DATATYPES::varchar(32)::varchar(16)
```

Product Dimension View

The following statements define the `product` dimension view.

```
DIMENSION::product
HIERARCHY::product.parentrel
GID::product.gid
PARENT GID::product.gid
ATTRIBUTES::product.longlabel::product.shortlabel
COLUMN LEVEL DIMENSION::product.lvldim
COLUMN LEVEL RELATION::product.hierheight

DIMENSION COLUMN::product
PARENT COLUMN::prod_parent
GID COLUMN::prod_gid
PARENT GID COLUMN::prod_gid
DIMENSION DATATYPES::varchar2(16)::varchar2(16)::number(10)::number(10)
LEVEL COLUMNS::equipment::component::division::totalprod
LEVEL DATATYPES::varchar2(16)::varchar2(16)::varchar2(16)::varchar2(16)
ATTRIBUTE COLUMNS::prod_long::prod_short
ATTRIBUTE DATATYPES::varchar(32)::varchar(16)
```

Channel Dimension View

These statements define the `channel` dimension view.

```
DIMENSION::channel
HIERARCHY::channel.parentrel
GID::channel.gid
PARENT GID::channel.gid
ATTRIBUTES::channel.longlabel::channel.shortlabel
COLUMN LEVEL DIMENSION::channel.lvldim
COLUMN LEVEL RELATION::channel.hierheight

DIMENSION COLUMN::channel
PARENT COLUMN::chan_parent
GID COLUMN::chan_gid
```

```
PARENT GID COLUMN::chanp_gid
DIMENSION DATATYPES::varchar2(16)::varchar2(16)::number(10)::number(10)
LEVEL COLUMNS::outlet::totalchan
LEVEL DATATYPES::varchar2(16)::varchar2(16)
ATTRIBUTE COLUMNS::chan_long::chan_short
ATTRIBUTE DATATYPES::varchar(32)::varchar(16)
```

Time Standard Hierarchy Input File

These statements define the `time` dimension view for the `STANDARD` hierarchy. A separate file is required to generate another view to support the `YTD` hierarchy, but it is not included in this example.

```
DIMENSION::time
HIERARCHY::time.parentrel
INHIERARCHY: time.inhierarchy
HIERARCHY DIMENSION::time.hierarchies
HIERARCHY DIMENSION VALUE::STANDARD
GID::time.gid
PARENT GID::time.gid
ATTRIBUTES::time.longlabel::time.shortlabel
COLUMN LEVEL DIMENSION::time.lvldim
COLUMN LEVEL RELATION::time.hierheight

DIMENSION COLUMN::time
PARENT COLUMN::time_parent
GID COLUMN::time_gid
PARENT GID COLUMN::timep_gid
DIMENSION DATATYPES::varchar2(8)::varchar2(8)::number(10)::number(10)
LEVEL COLUMNS::month::quarter::year
LEVEL DATATYPES::varchar2(16)::varchar2(16)::varchar2(16)
ATTRIBUTE COLUMNS::time_long::time_short
ATTRIBUTE DATATYPES::varchar(32)::varchar(16)
```

Sales and Costs Fact Views

For the OLAP API, you need to create a fact view for each combination of dimension hierarchies. In addition to the fact columns, the OLAP API also needs columns for dimension members and grouping IDs.

The following statements identify two workspace measures, `sales` and `costs`, as the source objects for a fact view. The fact view will have columns for the data from `sales` and `costs`. Both of these columns will have a `NUMBER` data type with 12 significant digits and 2 decimal places. The data from `sales` will be fetched into the `sales` column, and the data from `costs` will be fetched into the `costs` column.

The following is an example of just one of the four input files needed by the `sales` and `costs` measures. The statements defining the `product` and `channel` columns are also omitted, as indicated by the ellipsis.

```
MEASURE::sales::costs
MEASURE COLUMNS::sales::costs
MEASURE DATATYPES::number(12,2)::number(12,2)

DIMENSION::geography
HIERARCHY::geography.parentrel
INHIERARCHY: geography.inhierarchy
HIERARCHY DIMENSION::geography.hierarchies
HIERARCHY DIMENSION VALUE::STANDARD
GID::geography.gid

DIMENSION COLUMN::geography
GID COLUMN::geog_gid
DIMENSION DATATYPES::varchar2(16)::number(10)
     .
     .
     .
DIMENSION::time
HIERARCHY::time.parentrel
INHIERARCHY: time.inhierarchy
HIERARCHY DIMENSION::time.hierarchies
HIERARCHY DIMENSION VALUE::STANDARD
GID::time.gid

DIMENSION COLUMN::time
GID COLUMN::time_gid
DIMENSION DATATYPES::varchar2(8)::number(10)
```

Example: Script for the Product View

This PL/SQL command uses the `/users/oracle/mapfiles/product.txt` input file shown in "Product Dimension View" on page 15-11 to generate a script named `/users/oracle/scripts/product.sql`. The resulting view will be named `electro_product_view`.

```
CALL CWM2_OLAP_AW_ACCESS.CREATEAWACCESSSTRUCTURES_FR(
   '/users/oracle/scripts/', 'product.sql', 'electro_product_',
   'scott.electronics', '/users/oracle/mapfiles/', 'product.txt');
```

Before executing the script, you may edit it.

```
--product.sql
--Generated on: 15-FEB-2002 09:16:42am

SET ECHO ON
SET LINESIZE 200
SET PAGESIZE 50
SET SERVEROUT ON

DROP TYPE electro_product_TBL;
DROP TYPE electro_product_OBJ;

CREATE TYPE electro_product_OBJ AS OBJECT (
    PRODUCT VARCHAR2(16),
    PROD_PARENT VARCHAR2(16),
    PROD_GID NUMBER(10),
    PRODP_GID NUMBER(10),
    EQUIPMENT VARCHAR2(16),
    COMPONENT VARCHAR2(16),
    DIVISION VARCHAR2(16),
    TOTALPROD VARCHAR2(16),
    PROD_LONG VARCHAR(32),
    PROD_SHORT VARCHAR(16));
/

CREATE TYPE electro_product_TBL AS TABLE OF electro_product_OBJ;
/

CREATE OR REPLACE FUNCTION electro_product_LMAP RETURN VARCHAR2 IS
--This function will return the following Limit Map:
--DIMENSION PRODUCT FROM PRODUCT
--    WITH HIERARCHY PROD_PARENT FROM PRODUCT.PARENTREL
--        GID PROD_GID FROM PRODUCT.GID
--        PARENTGID PRODP_GID FROM PRODUCT.GID
--        LEVELREL EQUIPMENT, COMPONENT, DIVISION, TOTALPROD FROM
PRODUCT.HIERHEIGHT USING PRODUCT.LVLDIM
--    ATTRIBUTE PROD_LONG FROM PRODUCT.LONGLABEL
--    ATTRIBUTE PROD_SHORT FROM PRODUCT.SHORTLABEL
vRetVal VARCHAR2(443) := '';

BEGIN
    vRetVal := vRetVal || 'DIMENSION PRODUCT FROM PRODUCT  ';
    vRetVal := vRetVal || 'WITH HIERARCHY PROD_PARENT FROM PRODUCT.PARENTREL  ';
    vRetVal := vRetVal || 'GID PROD_GID FROM PRODUCT.GID  ';
    vRetVal := vRetVal || 'PARENTGID PRODP_GID FROM PRODUCT.GID  ';
    vRetVal := vRetVal || 'LEVELREL EQUIPMENT, COMPONENT, DIVISION, TOTALPROD
```

```
FROM PRODUCT.HIERHEIGHT USING PRODUCT.LVLDIM  ';
    vRetVal := vRetVal || 'ATTRIBUTE PROD_LONG FROM PRODUCT.LONGLABEL  ';
    vRetVal := vRetVal || 'ATTRIBUTE PROD_SHORT FROM PRODUCT.SHORTLABEL';
    RETURN vRetVal;
END electro_product_LMAP;
/

SHOW ERRORS;

CREATE OR REPLACE VIEW electro_product_VIEW AS SELECT * FROM
TABLE(CAST(OLAP_TABLE('scott.electronics DURATION QUERY', 'electro_product_TBL',
'', electro_product_LMAP())AS electro_product_TBL));

--The command below should be modified to provide appropriate security to
Analytic Workspace data.
--GRANT SELECT ON electro_product_VIEW TO PUBLIC;

--End of file: product.sql
```

Example: Product View

The script shown in "Example: Script for the Product View" on page 15-13 creates a view named `ELECTRO_PRODUCT_VIEW`, which has the following definition:

```
SELECT "PRODUCT", "PROD_PARENT", "PROD_GID", "PRODP_GID" "EQUIPMENT",
     "COMPONENT","DIVISION, "TOTALPROD", "PROD_LONG", "PROD_SHORT"
     FROM TABLE(CAST (OLAP_TABLE('scott.electronics DURATION QUERY',
     'electro_product_TBL', '', electro_product_LMAP()) AS electro_product_TBL))
```

Use a command like the following to access data about products from the `electronics` analytic workspace:

```
select product, prod_long, prod_short from electro_product_view
     where prod_gid=0;

PRODUCT          PROD_LONG                       PROD_SHORT
---------------- ------------------------------- ---------------
PORTCD           Portable CD Player              Port CD
PORTST           Portable Stereo                 Port Stereo
PORTCAS          Portable Cassette               Port Cassette
TUNER            Tuner                           Tuner
                      .
                      .
                      .
METALCAS         Metal Cassette                  Mtl Cassette
```

```
STNDCAS          Standard Cassette                Std Cassette
STNDVHSVIDEO     Standard VHS Video               VHS Video
8MMVIDEO         8MM Video                        8MM Video
HI8VIDEO         Hi 8 Video                       Hi8 Video

22 rows selected.
```

Summary of CWM2_OLAP_AW_ACCESS Subprograms

Table 15–5 lists the subprograms provided in `CWM2_OLAP_AW_ACCESS`.

Table 15–5 CWM2_OLAP_AW_ACCESS

Subprogram	Description
CreateAWAccessStructures_FR Procedure	Functions the same way as `CreateAWAccessStructures` except that it accepts a file that contains the mapping information. This procedure parses the information contained in the file and passes it, along with the other parameters, to `CreateAWAccessStructures`.
CreateAWAccessStructures Procedure	Generates one or more scripts. The scripts create views that represent the multidimensional objects in an analytic workspace. The views take the place of dimension tables and measure tables when creating metadata. This procedure accepts a delimited text string on the command line for the mapping information. The mapping information identifies source objects in the analytic workspace and target columns in the database.

CreateAWAccessStructures_FR Procedure

This procedure creates a SQL script that will create the relational objects needed for SQL to access objects in the analytic workspace, such as object types and views. As input, it takes a text file that maps the workspace objects to columns of the views.

Syntax

```
CreateAWAccessStructures_FR(
   script_directory      VARCHAR2,
   script_name           VARCHAR2,
   prefix                VARCHAR2,
   aw_name               VARCHAR2,
   infile_directory      VARCHAR2
   infile_name           VARCHAR2);
```

Parameters

Table 15–6 CreateAWAccessStructures_FR Procedure Parameters

Parameter	Description
`script_directory`	An existing directory path where `script_name` will be written.
`script_name`	The file name that will be given to the generated SQL script. This procedure does not overwrite an existing file, so be sure that a file by the name you specify does not already exist in `script_directory`.
`prefix`	A prefix that will be given to the view created by executing the script. This prefix identifies the objects in a schema that relate to the analytic workspace. It can be up to 25 characters long, and must comply with the requirements for a database object name.
`aw_name`	The name of the analytic workspace where the source objects are stored.
`infile_directory`	The directory path where the `infile_name` is stored.
`infile_name`	The name of the input file that contains mapping information, as described in Example , "Specifying the Source and Target Objects".

CreateAWAccessStructures Procedure

This procedure creates a SQL script that will create the relational objects needed for SQL to access objects in the analytic workspace, such as object types and views. It takes a delimited string as input for the mapping information.

Syntax

```
CreateAWAccessStructures(
   script_directory      VARCHAR2,
   script_filename       VARCHAR2,
   prefix                VARCHAR2,
   aw_name               VARCHAR2,
   mapping_info          VARCHAR2);
```

Parameters

Table 15–7 CreateAWAccessStructures Procedure Parameters

Parameter	Description
`script_directory`	An existing directory path where `script_name` will be written.
`script_name`	The file name that will be given to the generated SQL script. This procedure does not overwrite an existing file, so be sure that a file by the name you specify does not already exist in `script_directory`.
`prefix`	A prefix that will be given to the view created by executing the script. This prefix identifies the objects in a schema that relate to the analytic workspace. It can be up to 25 characters long, and must comply with the requirements for a database object name.
`aw_name`	The name of the analytic workspace where the source objects are stored.
`mapping_info`	A delimited string that contains mapping information, as described in "Specifying the Source and Target Objects" on page 15-4.

16

CWM2_OLAP_AW_CREATE

The CWM2_OLAP_AW_CREATE package provides procedures for moving data from a relational data warehouse to an analytic workspace and generating relational views of the workspace.

See Also:

- Chapter 9, "Creating an Analytic Workspace From Relational Tables".
- Chapter 13, "Using the OLAP Catalog Metadata APIs".

This chapter discusses the following topics:

- Summary of CWM2_OLAP_AW_CREATE Subprograms

Summary of CWM2_OLAP_AW_CREATE Subprograms

Table 16–1 CWM2_OLAP_AW_CREATE Subprograms

Subprogram	Description
AW_DIMENSION_CREATE Procedure	Creates containers within an analytic workspace to hold the dimension members of an OLAP Catalog dimension.
AW_DIM_DEFINE_LOAD Procedure	Creates a load definition for a dimension.
AW_DIM_FILTER_LOAD Procedure	Specifies a SQL WHERE clause to use in the query against the dimension table.
AW_DIMENSION_REFRESH Procedure	Uses a load definition to load dimension members into an analytic workspace.
AW_DIMENSION_CREATE_ACCESS Procedure	Generates relational views of an analytic workspace dimension.
AW_CUBE_CREATE Procedure	Creates containers within an analytic workspace to hold the data of an OLAP Catalog cube.
AW_CUBE_DEFINE_LOAD Procedure	Creates a load definition for a cube.
AW_CUBE_FILTER_LOAD Procedure	Specifies a SQL WHERE clause to use in the query against the fact table.
AW_CUBE_MEASURE_LOAD Procedure	Specifies a measure to load into the analytic workspace.
AW_CHOOSE_LEVEL_TUPLES Procedure	Creates a table of level combinations for a cube.
AW_DEFINE_AGG_PLAN Procedure	Specifies how to aggregate the cube's data within the analytic workspace.
AW_CUBE_REFRESH Procedure	Uses a load definition to load data from a fact table into an analytic workspace.
AW_CUBE_CREATE_ACCESS Procedure	Generates relational views of an analytic workspace cube.

AW_DIMENSION_CREATE Procedure

This procedure creates the containers within an analytic workspace to hold an OLAP Catalog dimension.

The OLAP Catalog dimension must conform to the requirements specified in "Basic Star or Snowflake Schema" on page 5-4.

If the analytic workspace does not already exist, AW_DIMENSION_CREATE creates it.

Syntax

```
AW_DIMENSION_CREATE (
          aw_owner           IN   VARCHAR2,
          aw_name            IN   VARCHAR2,
          dimension_owner    IN   VARCHAR2,
          dimension_name     IN   VARCHAR2);
```

Parameters

Table 16–2 AW_DIMENSION_CREATE Procedure Parameters

Parameter	Description
aw_owner	Owner of the analytic workspace.
aw_name	Name of the analytic workspace.
dimension_owner	Owner of the OLAP Catalog dimension.
dimension_name	Name of the OLAP Catalog dimension.

AW_DIM_DEFINE_LOAD Procedure

This procedure creates a load definition for an OLAP Catalog dimension. The load definition specifies the dimension members to load from the dimension tables and how to store them within the analytic workspace.

The load definition is used by AW_DIMENSION_REFRESH.

The load definition can have one of the types described in Table 16–3.

Table 16–3 Load Types for Dimensions

Load Type	Description
FULL	Load all dimension members. If the dimension has already been loaded, delete all members and replace with the new ones.
FULL_ADDITIONS ONLY	Load all dimension members. If the dimension has already been loaded, keep the existing members that have not changed and add the new members.

Syntax

```
AW_DIM_DEFINE_LOAD (
          dimension_owner       IN   VARCHAR2,
          dimension_name        IN   VARCHAR2,
          load_name             IN   VARCHAR2,
          load_type             IN   VARCHAR2,
          unique_keys           IN   VARCHAR2);
```

Parameters

Table 16–4 AW_DIM_DEFINE_LOAD Procedure Parameters

Parameter	Description
`dimension_owner`	Owner of the OLAP Catalog dimension.
`dimension_name`	Name of the OLAP Catalog dimension.
`load_name`	Name of a load definition.
`load_type`	Type of data load. Specify one of the values listed in Table 16–3, "Load Types for Dimensions".
`unique_keys`	Whether or not the members of this dimension are unique across all levels. Values can be **YES** or **NO**. The default is **NO**.

AW_DIM_FILTER_LOAD Procedure

This procedure creates a SQL WHERE clause to be added to a load definition for a dimension. The WHERE clause specifies which members of a hierarchy should be loaded from the dimension table to the analytic workspace.

Syntax

```
AW_DIM_FILTER_LOAD (
          dimension_owner       IN   VARCHAR2,
          dimension_name        IN   VARCHAR2,
          load_name             IN   VARCHAR2
          dim_table_owner       IN   VARCHAR2,
          dim_table_name        IN   VARCHAR2,
          where_clause          IN   VARCHAR2);
```

Parameters

Table 16–5 AW_DIM_FILTER_LOAD Procedure Parameters

Parameter	Description
dimension_owner	Owner of the OLAP Catalog dimension.
dimension_name	Name of the OLAP Catalog dimension.
load_name	Name of a load definition.
dim_table_owner	Owner of the dimension table that underlies this OLAP Catalog dimension.
dim_table_name	Name of the dimension table that underlies this OLAP Catalog dimension.
where_clause	A SQL WHERE clause that specifies which rows to load from the dimension table.

AW_DIMENSION_REFRESH Procedure

This procedure uses a load definition to load values from dimension tables into an analytic workspace.

If the analytic workspace dimension has never been populated, AW_DIMENSION_REFRESH does a full load. Otherwise, it refreshes the dimension based on the load type described in Table 16–3 and on any filter criteria that may have been established by a call to AW_DIM_FILTER_LOAD.

Syntax

```
AW_DIMENSION_REFRESH (
          aw_owner              IN   VARCHAR2,
          aw_name               IN   VARCHAR2,
          dimension_owner       IN   VARCHAR2,
          dimension_name        IN   VARCHAR2,
          load_name             IN   VARCHAR2);
```

Parameters

Table 16–6 AW_DIMENSION_REFRESH Procedure Parameters

Parameter	Description
aw_owner	Owner of the analytic workspace.

Table 16–6 (Cont.) AW_DIMENSION_REFRESH Procedure Parameters

Parameter	Description
aw_name	Name of the analytic workspace.
dimension_owner	Owner of the OLAP Catalog dimension.
dimension_name	Name of the OLAP Catalog dimension.
load_name	Name of a load definition.

AW_DIMENSION_CREATE_ACCESS Procedure

This procedure creates a script that you can run to generate relational views of an AW dimension.

The views contain calls to the OLAP_TABLE function. OLAP_TABLE, described in Chapter 12, uses object technology to present the contents of the workspace in table format

The AW_DIMENSION_CREATE_ACCESS procedure creates a separate ET-style view for each dimension hierarchy. Each view has a column for each level and level attribute participating in the hierarchy. It also contains grouping ID columns and ET key columns as described in Table 9–2, "Dimension View Columns" on page 9-5.

> **See Also:** "Procedure: Create SQL Access to the Analytic Workspace" on page 9-4 and "Column Structure of Dimension Views" on page 9-5.

Syntax

```
AW_DIMENSION_CREATE_ACCESS (
          dimension_owner       IN   VARCHAR2,
          dimension_name        IN   VARCHAR2,
          aw_owner              IN   VARCHAR2,
          aw_name               IN   VARCHAR2,
          prefix                IN   VARCHAR2,
          access_type           IN   VARCHAR2,
          script_directory      IN   VARCHAR2,
          script_name           IN   VARCHAR2);
```

Parameters

Table 16–7 AW_DIMENSION_CREATE_ACCESS Procedure Parameters

Parameter	Description
dimension_owner	Owner of the OLAP Catalog dimension.
dimension_name	Name of the OLAP Catalog dimension.
aw_owner	Owner of the analytic workspace.
aw_name	Name of the analytic workspace.
prefix	Prefix to be applied to the name of the ADT, the name of the table of ADTs, and the name of the view. See Usage Notes.
access_type	How the view will be accessed. Examples are straight SQL, OLAP API, and Discoverer. This argument is not currently used.
script_directory	The directory that will contain the script.
script_name	The script that will generate the views.

Usage Notes

The script creates an ADT (abstract data type) that encapsulates multidimensional data in the analytic workspace, a table of the ADTs, and a view of the table. The ADT, table, and view are named according to specific rules.

The ADT name is the concatenation of the user-supplied prefix, the first five characters of the dimension owner, the first five characters of the dimension name, and the suffix `OBJ`. The following name identifies an ADT for the Product dimension owned by SH, with a prefix of `mydim`.

```
mydim_sh_produ_OBJ
```

The table of ADT name is like the name of the ADT, but with the suffix TBL. For example,

```
mydim_sh_produ_TBL
```

The view name is similarly constructed, but it contains additional information: the first five characters of the hierarchy name and a hierarchy sequence number. The hierarchy sequence number uniquely identifies each view, starting with one. The following name identifies the first view of the Product dimension owned by SH.

```
mydim_sh_produ_std_1_view
```

AW_CUBE_CREATE Procedure

This procedure creates the containers within an analytic workspace to hold an OLAP Catalog cube.

The OLAP Catalog cube must conform to the requirements specified in "Basic Star or Snowflake Schema" on page 5-4.

Syntax

```
AW_CUBE_CREATE (
          aw_owner         IN   VARCHAR2,
          aw_name          IN   VARCHAR2,
          cube_owner       IN   VARCHAR2,
          cube_name        IN   VARCHAR2);
```

Parameters

Table 16–8 AW_CUBE_CREATE Procedure Parameters

Parameter	Description
aw_owner	Owner of the analytic workspace.
aw_name	Name of the analytic workspace.
cube_owner	Owner of the OLAP Catalog cube.
cube_name	Name of the OLAP Catalog cube.

AW_CUBE_DEFINE_LOAD Procedure

This procedure creates a load definition for an OLAP Catalog cube. The load definition specifies the data to load from the fact table and how to aggregate the data within the analytic workspace.

The load definition is used by AW_CUBE_REFRESH.

Syntax

```
AW_CUBE_DEFINE_LOAD (
          cube_owner              IN   VARCHAR2,
          cube_name               IN   VARCHAR2,
          load_name               IN   VARCHAR2,
          load_type               IN   VARCHAR2);
```

Parameters

Table 16–9 AW_CUBE_DEFINE_LOAD Procedure Parameters

Parameter	Description
cube_owner	Owner of the OLAP Catalog cube.
cube_name	Name of the OLAP Catalog cube.
load_name	Name of a load definition.
load_type	Type of data load. This argument is not used in the current release.

AW_CUBE_FILTER_LOAD Procedure

This procedure creates a SQL WHERE clause to be added to a load definition for a cube. The WHERE clause specifies which rows should be loaded from the fact table to the analytic workspace.

Syntax

```
AW_CUBE_FILTER_LOAD (
          cube_owner              IN   VARCHAR2,
          cube_name               IN   VARCHAR2,
          load_name               IN   VARCHAR2
          fact_table_owner        IN   VARCHAR2,
          fact_table_name         IN   VARCHAR2,
          where_clause            IN   VARCHAR2);
```

Parameters

Table 16–10 AW_CUBE_FILTER_LOAD Procedure Parameters

Parameter	Description
cube_owner	Owner of the OLAP Catalog cube.
cube_name	Name of the OLAP Catalog cube.
load_name	Name of a load definition.
fact_table_owner	Owner of the fact table that underlies this OLAP Catalog cube.
fact_table_name	Name of the fact table that underlies this OLAP Catalog cube
where_clause	A SQL WHERE clause that specifies which rows to load from the fact table.

AW_CUBE_MEASURE_LOAD Procedure

This procedure specifies a measure to load from the fact table to the analytic workspace. The load instructions are added to load definition created by AW_CUBE_DEFINE_LOAD.

Syntax

```
AW_CUBE_MEASURE_LOAD (
          cube_owner              IN   VARCHAR2,
          cube_name               IN   VARCHAR2,
          measure_name            IN   VARCHAR2
          load_name               IN   VARCHAR2);
```

Parameters

Table 16–11 AW_CUBE_MEASURE_LOAD Procedure Parameters

Parameter	Description
cube_owner	Owner of the OLAP Catalog cube.
cube_name	Name of the OLAP Catalog cube.
measure_name	Name of one of the cube's measures.
load_name	Name of a load definition.

AW_CHOOSE_LEVEL_TUPLES Procedure

This procedure creates a table listing all the level combinations associated with a cube.

The table, SYS.OlapTabLevels, has columns for each of the cube's dimensions. The rows contain the level names for the dimensions. By default, all the levels are selected for aggregation within the analytic workspace.

If you want to specify partial aggregation, you must edit the table. Uncheck each level for which summary data should not be stored.

Once you have established the table of level tuples for a cube, you can call AW_DEFINE_AGG_PLAN to define a set of aggregation rules for a given load definition.

Syntax

```
AW_CHOOSE_LEVEL_TUPLES (
                        cube_owner  IN   VARCHAR2,
                        cube_name   IN   VARCHAR2);
```

Parameters

Table 16–12 AW_CHOOSE_LEVEL_TUPLES Procedure Parameters

Parameter	Description
cube_owner	Owner of the OLAP Catalog cube.
cube_name	Name of the OLAP Catalog cube.

AW_DEFINE_AGG_PLAN Procedure

This procedure reads a table of level combinations for a cube and defines an aggregation plan for a cube load definition.

Before calling this procedure, call AW_CHOOSE_LEVEL_TUPLES to create the table SYS.OlapTabLevels. To specify partial aggregation, you must edit this table before calling AW_DEFINE_AGG_PLAN.

Syntax

```
AW_DEFINE_AGG_PLAN (
          cube_owner     IN   VARCHAR2,
          cube_name      IN   VARCHAR2,
          load_name      IN   VARCHAR2);
```

Parameters

Table 16–13 AW_DEFINE_AGG_PLAN Procedure Parameters

Parameter	Description
cube_owner	Owner of the OLAP Catalog cube.
cube_name	Name of the OLAP Catalog cube.
load_name	Name of a load definition.

AW_CUBE_REFRESH Procedure

This procedure uses a load definition to load data from a fact table into an analytic workspace. Unless a filter criteria was established for the data load, AW_CUBE_REFRESH loads all the data from the fact table.

If an aggregation plan was established for this data load, AW_CUBE_REFRESH aggregates the data to the specified level. Otherwise, it fully aggregates the data within the analytic workspace.

Syntax

```
AW_CUBE_REFRESH (
          aw_owner          IN   VARCHAR2,
          aw_name           IN   VARCHAR2,
          cube_owner        IN   VARCHAR2,
          cube_name         IN   VARCHAR2,
          load_name         IN    VARCHAR2);
```

Parameters

Table 16–14 AW_CUBE_REFRESH Procedure Parameters

Parameter	Description
aw_owner	Owner of the analytic workspace.
aw_name	Name of the analytic workspace.
cube_owner	Owner of the cube.
cube_name	Name of the cube.
load_name	Name of the load definition.

AW_CUBE_CREATE_ACCESS Procedure

The AW_CUBE_CREATE_ACCESS procedure creates a script that you can run to generate fact views of AW cubes.

The views contain calls to the OLAP_TABLE function. OLAP_TABLE, described in Chapter 12, uses object technology to present the contents of the workspace in table format

The script created by AW_CUBE_CREATE_ACCESS generates a fully solved fact view for every dimension/hierarchy of the cube. Each view has a column for each of the cube's measures. It also contains a grouping ID column and an ET key column for each dimension to link the fact view with the associated dimension views.

> **See Also:** "Procedure: Create SQL Access to the Analytic Workspace" and Table 9–4, "Fact View Columns" on page 9-7.

Syntax

```
AW_CUBE_CREATE_ACCESS (
          cube_owner          IN   VARCHAR2,
          cube_name           IN   VARCHAR2,
          aw_owner            IN   VARCHAR2,
          aw_name             IN   VARCHAR2,
          prefix              IN   VARCHAR2,
          access_type         IN   VARCHAR2,
          script_directory    IN   VARCHAR2,
          script_name         IN   VARCHAR2);
```

Parameters

Table 16–15 AW_CUBE_CREATE_ACCESS Procedure Parameters

Parameter	Description
cube_owner	Owner of the OLAP Catalog cube.
cube_name	Name of the OLAP Catalog cube.
aw_owner	Owner of the analytic workspace.
aw_name	Name of the analytic workspace.
prefix	Prefix to be applied to the name of the ADT, the name of the table of ADTs, and the name of the view. See Usage Notes.

Table 16–15 (Cont.) AW_CUBE_CREATE_ACCESS Procedure Parameters

Parameter	Description
access_type	How the view will be accessed. Examples are straight SQL, OLAP API, and Discoverer. This argument is not currently used.
script_directory	The directory that will contain the script.
script_name	The script that will generate the views.

Usage Notes

The script creates an ADT (abstract data type) that encapsulates multidimensional data in the analytic workspace, a table of the ADTs, and a view of the table. The ADT, table, and view are named according to specific rules.

The ADT name is the concatenation of the user-supplied prefix, the first five characters of the cube owner, the first five characters of the cube name, and the suffix OBJ. The following name identifies an ADT for the Sales cube owned by SH, with a prefix of mydim.

```
mydim_sh_sales_OBJ
```

The table of ADT name is like the name of the ADT, but with the suffix TBL. For example,

```
mydim_sh_sales_TBL
```

The view name is similarly constructed, but it contains an additional sequence number that uniquely identifies the view. The following name identifies the first view of the Sales cube owned by SH, with a prefix of mydim.

```
mydim_sh_sales_1_view
```

17

CWM2_OLAP_CUBE

The CWM2_OLAP_CUBE package provides procedures for creating, dropping, and locking cubes. It also provides procedures for setting general properties of cubes.

See Also: Chapter 13, "Using the OLAP Catalog Metadata APIs"

This chapter discusses the following topics:

- Understanding Cubes
- Summary of CWM2_OLAP_CUBE Subprograms
- Example: Creating a Cube

Understanding Cubes

A cube is an OLAP metadata entity. This means that it is a logical object, identified by name and owner, within the OLAP Catalog.

A cube is a multidimensional framework to which you can assign measures. A measure represents data stored in fact tables. The fact tables may be relational tables or views. The views may reference data stored in analytic workspaces. Cubes are fully described in "OLAP Metadata Model" on page 4-8.

Use the procedures in the CWM2_OLAP_CUBE package to create, drop, and lock cubes, to associate dimensions with cubes, and to specify descriptive information for display purposes.

You must create the cube before using the CWM2_OLAP_MEASURE package to create the cube's measures.

Summary of CWM2_OLAP_CUBE Subprograms

Table 17–1 CWM2_OLAP_CUBE Subprograms

Subprogram	Description
ADD_DIMENSION_TO_CUBE Procedure on page 17-3	Adds a dimension to a cube.
CREATE_CUBE Procedure on page 17-4	Creates a cube.
DROP_CUBE Procedure on page 17-5	Drops a cube.
LOCK_CUBE Procedure on page 17-6	Locks a cube's metadata for update.
REMOVE_DIMENSION_FROM_CUBE Procedure on page 17-6	Removes a dimension from a cube.
SET_CUBE_NAME Procedure on page 17-7	Sets the name of a cube.
SET_DEFAULT_CUBE_DIM_CALC_HIER Procedure on page 17-8	Sets the default calculation hierarchy for a dimension of the cube.
SET_DESCRIPTION Procedure on page 17-9	Sets the description for a cube.

Table 17–1 (Cont.) CWM2_OLAP_CUBE Subprograms

Subprogram	Description
SET_DISPLAY_NAME Procedure on page 17-10	Sets the display name for a cube.
SET_MV_SUMMARY_CODE Procedure on page 17-11	Sets the format for materialized views associated with a cube.
SET_SHORT_DESCRIPTION Procedure on page 17-12	Sets the short description for a cube.

ADD_DIMENSION_TO_CUBE Procedure

This procedure adds a dimension to a cube.

Syntax

```
ADD_DIMENSION_TO_CUBE  (
          cube_owner          IN   VARCHAR2,
          cube_name           IN   VARCHAR2,
          dimension_owner     IN   VARCHAR2,
          dimension_name      IN   VARCHAR2);
```

Parameters

Table 17–2 ADD_DIMENSION_TO_CUBE Procedure Parameters

Parameter	Description
cube_owner	Owner of the cube.
cube_name	Name of the cube.
dimension_owner	Owner of the dimension to be added to the cube.
dimension_name	Name of the dimension to be added to the cube.

Exceptions

Table 17–3 ADD_DIMENSION_TO_CUBE Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.

Table 17–3 (Cont.) ADD_DIMENSION_TO_CUBE Procedure Exceptions

Exception	Description
cube_not_found	Cube not found.
dimension_not_found	Dimension not found.

CREATE_CUBE Procedure

This procedure creates a new cube in the OLAP Catalog.

Descriptions and display properties must also be established as part of cube creation. Once the cube has been created, you can override these properties by calling other procedures in this package.

Syntax

```
CREATE_CUBE (
          cube_owner            IN   VARCHAR2,
          cube_name             IN   VARCHAR2,
          display_name          IN   VARCHAR2,
          short_description     IN   VARCHAR2,
          description           IN   VARCHAR2);
```

Parameters

Table 17–4 CREATE_CUBE Procedure Parameters

Parameter	Description
cube_owner	Owner of the cube.
cube_name	Name of the cube.
display_name	Display name for the cube.
short_description	Short description of the cube.
description	Description of the cube.

Exceptions

Table 17–5 CREATE_CUBE Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
cube_already_exists	Cube already exists.

DROP_CUBE Procedure

This procedure drops a cube from the OLAP 2 Catalog.

> **Note:** When a cube is dropped, its associated measures are also dropped. However, the cube's dimensions are not dropped. They might be mapped within the context of a different cube.

Syntax

```
DROP_CUBE (
         cube_owner     IN   VARCHAR2,
         cube_name      IN   VARCHAR2);
```

Parameters

Table 17–6 DROP_CUBE Procedure Parameters

Parameter	Description
cube_owner	Owner of the cube.
cube_name	Name of the cube.

Exceptions

Table 17–7 DROP_CUBE Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
cube_not_found	Cube not found.

LOCK_CUBE Procedure

This procedure locks the cube's metadata for update by acquiring a database lock on the row that identifies the cube in the CWM2 model table.

Syntax

```
LOCK_CUBE (
          cube_owner        IN   VARCHAR2,
          cube_name         IN   VARCHAR2.
          wait_for_lock     IN   BOOLEAN DEFAULT FALSE);
```

Parameters

Table 17–8 LOCK_CUBE Procedure Parameters

Parameter	Description
cube_owner	Owner of the cube.
cube_name	Name of the cube.
wait_for_lock	(Optional) Whether or not to wait for the cube to be available when it is already locked by another user. If you do not specify a value for this parameter, the procedure does not wait to acquire the lock.

Exceptions

Table 17–9 LOCK_CUBE Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
cube_not_found	Cube not found.

REMOVE_DIMENSION_FROM_CUBE Procedure

This procedure removes a dimension from a cube.

Syntax

```
REMOVE_DIMENSION_FROM_CUBE  (
          cube_owner          IN   VARCHAR2,
          cube_name           IN   VARCHAR2,
          dimension_owner     IN   VARCHAR2,
          dimension_name      IN   VARCHAR2);
```

Parameters

Table 17–10 REMOVE_DIMENSION_FROM_CUBE Procedure Parameters

Parameter	Description
cube_owner	Owner of the cube.
cube_name	Name of the cube.
dimension_owner	Owner of the dimension to be removed from the cube.
dimension_name	Name of the dimension to be removed from the cube.

Exceptions

Table 17–11 REMOVE_DIMENSION_FROM_CUBE Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
cube_not_found	Cube not found.
dimension_not_found	Dimension not found.

SET_CUBE_NAME Procedure

This procedure sets the name for a cube.

Syntax

```
SET_CUBE_NAME (
          cube_owner        IN   VARCHAR2,
          cube_name         IN   VARCHAR2,
          set_cube_name     IN   VARCHAR2);
```

Parameters

Table 17–12 SET_CUBE_NAME Procedure Parameters

Parameter	Description
cube_owner	Owner of the cube.
cube_name	Original name of the cube.
set_cube_name	New name for the cube.

Exceptions

Table 17–13 SET_CUBE_NAME Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
cube_not_found	Cube not found.

SET_DEFAULT_CUBE_DIM_CALC_HIER Procedure

This procedure sets the default calculation hierarchy for a dimension of this cube.

Syntax

```
SET_DEFAULT_CUBE_DIM_CALC_HIER (
          cube_owner       IN   VARCHAR2,
          cube_name        IN   VARCHAR2,
          dimension_owner  IN   VARCHAR2,
          dimension_name   IN   VARCHAR2,
          hierarchy_name   IN   VARCHAR2);
```

Parameters

Table 17–14 SET_DEFAULT_CUBE_DIM_CALC_HIER Procedure Parameters

Parameter	Description
cube_owner	Owner of the cube.
cube_owner	Name of the cube.
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
hierarchy_name	Name of the hierarchy to be used by default for this dimension.

Exceptions

Table 17–15 SET_DEFAULT_CUBE_DIM_CALC_HIER Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
cube_not_found	Cube not found.

SET_DESCRIPTION Procedure

This procedure sets the description for a cube.

Syntax

```
SET_DESCRIPTION (
          cube_owner     IN   VARCHAR2,
          cube_name      IN   VARCHAR2,
          description    IN   VARCHAR2);
```

Parameters

Table 17–16 SET_DESCRIPTION Procedure Parameters

Parameter	Description
cube_owner	Owner of the cube.
cube_name	Name of the cube.
description	Description of the cube.

Exceptions

Table 17–17 SET_DESCRIPTION Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
cube_not_found	Cube not found.

SET_DISPLAY_NAME Procedure

This procedure sets the display name for a cube.

Syntax

```
SET_DISPLAY_NAME (
          cube_owner     IN   VARCHAR2,
          cube_name      IN   VARCHAR2,
          display_name   IN   VARCHAR2);
```

Parameters

Table 17–18 SET_DISPLAY_NAME Procedure Parameters

Parameter	Description
cube_owner	Owner of the cube.
cube_name	Name of the cube.
display_name	Display name for the cube.

Exceptions

Table 17–19 SET_DISPLAY_NAME Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
cube_not_found	Cube not found.

SET_MV_SUMMARY_CODE Procedure

This procedure specifies the form of materialized views for this cube. Materialized views may be in Grouping Set (groupingset) or Rolled Up (rollup) form.

In a materialized view in Rolled Up form, all the dimension key columns are populated, and data may only be accessed when its full lineage is specified.

In a materialized view in Grouping Set form, dimension key columns may contain null values, and data may be accessed simply by specifying one or more levels.

Syntax

```
SET_MV_SUMMARY_CODE (
          cube_owner            IN   VARCHAR2,
          cube_name             IN   VARCHAR2,
          summary_code          IN   VARCHAR2);
```

Parameters

Table 17–20 SET_MV_SUMMARY_CODE Procedure Parameters

Parameter	Description
cube_owner	Owner of the cube.
cube_name	Name of the cube.
summary_code	One of the following case-insensitive values: ■ rollup, for Rolled Up form. ■ groupingset, for Grouping Set form.

Exceptions

Table 17–21 SET_MV_SUMMARY_CODE Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
cube_not_found	Cube not found.

SET_SHORT_DESCRIPTION Procedure

This procedure sets the short description for a cube.

Syntax

```
SET_DESCRIPTION (
          cube_owner            IN   VARCHAR2,
          cube_name             IN   VARCHAR2,
          short_description     IN   VARCHAR2);
```

Parameters

Table 17–22 SET_SHORT_DESCRIPTION Procedure Parameters

Parameter	Description
cube_owner	Owner of the cube.
cube_name	Name of the cube.
short_description	Short description of the cube.

Exceptions

Table 17–23 SET_SHORT_DESCRIPTION Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
cube_not_found	Cube not found.

Example: Creating a Cube

The following statements drop the cube SALES_CUBE, recreate it, and add the dimensions TIME_DIM, GEOG_DIM, and PRODUCT_DIM.

Dropping the cube removes the cube entity, along with its measures, from the OLAP Catalog. However, dropping the cube does not cause the cube's dimensions to be dropped.

```
execute cwm2_olap_cube.drop_cube('JSMITH', 'SALES_CUBE');
execute cwm2_olap_cube.create_cube
     ('JSMITH', 'SALES_CUBE', 'Sales', 'Sales Cube',
      'Sales dimensioned over geography, product, and time' );
execute cwm2_olap_cube.add_dimension_to_cube
     ('JSMITH', 'SALES_CUBE', 'JSMITH', 'TIME_DIM');
execute cwm2_olap_cube.add_dimension_to_cube
     ('JSMITH', 'SALES_CUBE', 'JSMITH', 'GEOG_DIM');
execute cwm2_olap_cube.add_dimension_to_cube
     ('JSMITH', 'SALES_CUBE', 'JSMITH', 'PRODUCT_DIM');
```

18

CWM2_OLAP_DIMENSION

The `CWM2_OLAP_DIMENSION` package provides procedures for creating, dropping, and locking dimensions. It also provides procedures for setting general dimension properties.

See Also: Chapter 13, "Using the OLAP Catalog Metadata APIs".

This chapter discusses the following topics:

- Understanding Dimensions
- Summary of CWM2_OLAP_DIMENSION Subprograms
- Example: Creating a CWM2 Dimension

Understanding Dimensions

A dimension is an OLAP metadata entity. This means that it is a logical object, identified by name and owner, within the OLAP Catalog. Logical OLAP dimensions are fully described in "OLAP Metadata Model" on page 4-8.

> **Note:** Dimensions in CWM2 map directly to columns in dimension tables and have no relationship to Oracle database dimension objects.

Use the procedures in the CWM2_OLAP_DIMENSION package to create, drop, and lock CWM2 dimension entities and to specify descriptive information for display purposes. To fully define a CWM2 dimension, follow the steps listed in "Constructing a Dimension" on page 13-2.

Summary of CWM2_OLAP_DIMENSION Subprograms

Table 18–1 CWM2_OLAP_DIMENSION Subprograms

Subprogram	Description
CREATE_DIMENSION Procedure on page 18-3	Creates a dimension.
DROP_DIMENSION Procedure on page 18-4	Drops a dimension.
LOCK_DIMENSION Procedure on page 18-5	Locks the dimension metadata for update.
SET_DEFAULT_DISPLAY_HIERARCHY Procedure on page 18-6	Sets the default hierarchy for a dimension.
SET_DESCRIPTION Procedure on page 18-7	Sets the description for a dimension.
SET_DIMENSION_NAME Procedure on page 18-7	Sets the name of a dimension.
SET_DISPLAY_NAME Procedure on page 18-8	Sets the display name for a dimension.
SET_PLURAL_NAME Procedure on page 18-9	Sets the plural name for a dimension.

Table 18–1 (Cont.) CWM2_OLAP_DIMENSION Subprograms

Subprogram	Description
SET_SHORT_DESCRIPTION Procedure on page 18-10	Sets the short description for a dimension.

CREATE_DIMENSION Procedure

This procedure creates a new dimension entity in the OLAP Catalog.

By default the new dimension is a normal dimension, but you can specify the value TIME for the dimension_type parameter to create a time dimension.

Descriptions and display properties must also be established as part of dimension creation. Once the dimension has been created, you can override these properties by calling other procedures in this package.

Syntax

```
CREATE_DIMENSION (
          dimension_owner       IN   VARCHAR2,
          dimension_name        IN   VARCHAR2,
          display_name          IN   VARCHAR2,
          plural_name           IN   VARCHAR2,
          short_description     IN   VARCHAR2,
          description           IN   VARCHAR2,
          dimension_type        IN   VARCHAR2 DEFAULT NULL);
```

Parameters

Table 18–2 CREATE_DIMENSION Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
display_name	Display name for the dimension.
plural_name	Plural name for the dimension.
short_description	Short description of the dimension.
description	Description of the dimension.

Table 18–2 (Cont.) CREATE_DIMENSION Procedure Parameters

Parameter	Description
dimension_type	(Optional) Type of the dimension. Specify the value TIME to create a time dimension. If you do not specify this parameter, the dimension is created as a normal dimension.

Exceptions

Table 18–3 CREATE_DIMENSION Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
dimension_already_exists	Cannot create dimension. Dimension already exists.

DROP_DIMENSION Procedure

This procedure drops a dimension entity from the OLAP Catalog. All related levels, hierarchies, and dimension attributes are also dropped.

Syntax

```
DROP_DIMENSION (
        dimension_owner     IN   VARCHAR2,
        dimension_name      IN   VARCHAR2);
```

Parameters

Table 18–4 DROP_DIMENSION Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.

Exceptions

Table 18–5 DROP_DIMENSION Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
dimension_not_found	Dimension not found.

LOCK_DIMENSION Procedure

This procedure locks the dimension metadata for update by acquiring a database lock on the row that identifies the dimension in the CWM2 model table.

Syntax

```
LOCK_DIMENSION (
          dimension_owner     IN   VARCHAR2,
          dimension_name      IN   VARCHAR2.
          wait_for_lock       IN   BOOLEAN DEFAULT FALSE);
```

Parameters

Table 18–6 LOCK_DIMENSION Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
wait_for_lock	(Optional) Whether or not to wait for the dimension to be available when it is already locked by another user. If you do not specify a value for this parameter, the procedure does not wait to acquire the lock.

Exceptions

Table 18–7 LOCK_DIMENSION Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.

Table 18–7 (Cont.) LOCK_DIMENSION Procedure Exceptions

Exception	Description
dimension_not_found	Dimension not found.

SET_DEFAULT_DISPLAY_HIERARCHY Procedure

This procedure sets the default hierarchy to be used for display purposes.

Syntax

```
SET_DEFAULT_DISPLAY_HIERARCHY  (
          dimension_owner     IN   VARCHAR2,
          dimension_name      IN   VARCHAR2,
          hierarchy_name      IN   VARCHAR2);
```

Parameters

Table 18–8 SET_DEFAULT_DISPLAY_HIERARCHY Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
hierarchy_name	Name of one of the dimension's hierarchies.

Exceptions

Table 18–9 SET_DEFAULT_DISPLAY_HIERARCHY Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
dimension_not_found	Dimension not found.
hierarchy_not_found	This hierarchy not found for this dimension.

SET_DESCRIPTION Procedure

This procedure sets the description for a dimension.

Syntax

```
SET_DESCRIPTION (
          dimension_owner     IN   VARCHAR2,
          dimension_name      IN   VARCHAR2,
          description         IN   VARCHAR2);
```

Parameters

Table 18–10 SET_DESCRIPTION Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
description	Description of the dimension.

Exceptions

Table 18–11 SET_DESCRIPTION Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
dimension_not_found	Dimension not found.

SET_DIMENSION_NAME Procedure

This procedure sets the name for a dimension.

Syntax

```
SET_DIMENSION_NAME (
          dimension_owner        IN   VARCHAR2,
          dimension_name         IN   VARCHAR2,
          set_dimension_name     IN   VARCHAR2);
```

Parameters

Table 18–12 SET_DIMENSION_NAME Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Original name of the dimension.
set_dimension_name	New name for the dimension.

Exceptions

Table 18–13 SET_DIMENSION_NAME Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
dimension_not_found	Dimension not found.

SET_DISPLAY_NAME Procedure

This procedure sets the display name for a dimension.

Syntax

```
SET_DISPLAY_NAME (
          dimension_owner     IN   VARCHAR2,
          dimension_name      IN   VARCHAR2,
          display_name        IN   VARCHAR2);
```

Parameters

Table 18–14 SET_DISPLAY_NAME Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
display_name	Display name for the dimension.

Exceptions

Table 18–15 SET_DISPLAY_NAME Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
dimension_not_found	Dimension not found.

SET_PLURAL_NAME Procedure

This procedure sets the plural name of a dimension.

Syntax

```
SET_PLURAL_NAME  (
          dimension_owner     IN   VARCHAR2,
          dimension_name      IN   VARCHAR2,
          plural_name         IN   VARCHAR2);
```

Parameters

Table 18–16 SET_PLURAL_NAME Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
plural_name	Plural name for the dimension.

Exceptions

Table 18–17 SET_PLURAL_NAME Procedure Exceptions

Exception	Description
no_access_privileges	User does not have privileges to edit the dimension. User must be the owner or OLAP_DBA.
dimension_not_found	Dimension not found.

SET_SHORT_DESCRIPTION Procedure

This procedure sets the short description for a dimension.

Syntax

```
SET_SHORT_DESCRIPTION (
          dimension_owner       IN   VARCHAR2,
          dimension_name        IN   VARCHAR2,
          short_description     IN   VARCHAR2);
```

Parameters

Table 18–18 SET_SHORT_DESCRIPTION Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
short_description	Short description of the dimension.

Exceptions

Table 18–19 SET_SHORT_DESCRIPTION Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
dimension_not_found	Dimension not found.

Example: Creating a CWM2 Dimension

The following statement creates a CWM2 dimension entity, PRODUCT_DIM, in the JSMITH schema. The display name is Product, and the plural name is Products. The short description is Prod, and the description is Product.

```
execute cwm2_olap_dimension.create_dimension
     ('JSMITH', 'PRODUCT_DIM', 'Product', 'Products', 'Prod', 'Product');
```

The following statements change the short description to Product and the long description to Product Dimension.

```
execute cwm2_olap_dimension.set_short_description
     ('JSMITH', 'PRODUCT_DIM', 'Product');
execute cwm2_olap_dimension.set_description
     ('JSMITH', 'PRODUCT_DIM', 'Product Dimension');
```

19

CWM2_OLAP_DIMENSION_ATTRIBUTE

The `CWM2_OLAP_DIMENSION_ATTRIBUTE` package provides procedures for creating, dropping, and locking dimension attributes. It also provides procedures for setting general properties of dimension attributes.

See Also: Chapter 13, "Using the OLAP Catalog Metadata APIs".

This chapter discusses the following topics:

- Understanding Dimension Attributes
- Summary of CWM2_OLAP_DIMENSION_ATTRIBUTE Subprograms
- Example: Creating a Dimension Attribute

Understanding Dimension Attributes

A dimension attribute is an OLAP metadata entity. This means that it is a logical object, identified by name and owner, within the OLAP Catalog.

Dimension attributes define sets of level attributes for a dimension. Dimension attributes may include level attributes for some or all of the dimension's levels. For time dimensions, the dimension attributes `end date` and `time span` must be defined for all levels. Dimension attributes are fully described in "OLAP Metadata Model" on page 4-8.

Use the procedures in the `CWM2_OLAP_DIMENSION_ATTRIBUTE` package to create, drop, and lock dimension attributes and to specify descriptive information for display purposes.

Several dimension attribute names are reserved, because they have special significance within `CWM2`. The level attributes comprising a reserved dimension attribute will be mapped to columns containing specific information. The reserved dimension attributes are listed in Table 19–1.

Table 19–1 Reserved Dimension Attributes

Dimension Attribute	Description
`Long Description`	A long description of the dimension member.
`Short Description`	A short description of the dimension member.
`End Date`	For a time dimension, the last date in a time period. (Required)
`Time Span`	For a time dimension, the number of days in a time period. (Required)
`Prior Period`	For a time dimension, the time period before this time period.
`Year Ago Period`	For a time dimension, the period a year before this time period.
`ET Key`	For an embedded total dimension, the embedded total key, which identifies the dimension member at the lowest level in a row of the dimension table.
`Parent ET Key`	For an embedded total dimension, the dimension member that is the parent of the ET key.
`Grouping ID`	For an embedded total dimension, the grouping ID (GID), which identifies the hierarchical level for a row of the dimension table.
`Parent Grouping ID`	For an embedded total dimension, the dimension member that is the parent of the grouping ID.

The parent dimension must already exist before you can create dimension attributes for it. To fully define a dimension, follow the steps listed in "Constructing a Dimension" on page 13-2.

Summary of CWM2_OLAP_DIMENSION_ATTRIBUTE Subprograms

Table 19–2 CWM2_OLAP_DIMENSION_ATTRIBUTE Subprograms

Subprogram	Description
CREATE_DIMENSION_ATTRIBUTE Procedure on page 19-3	Creates a dimension attribute.
DROP_DIMENSION_ATTRIBUTE Procedure on page 19-5	Drops a dimension attribute.
LOCK_DIMENSION_ATTRIBUTE Procedure on page 19-6	Locks the dimension attribute for update.
SET_DESCRIPTION Procedure on page 19-7	Sets the description for a dimension attribute.
SET_DIMENSION_ATTRIBUTE_NAME Procedure on page 19-8	Sets the name of a dimension attribute.
SET_DISPLAY_NAME Procedure on page 19-9	Sets the display name for a dimension attribute.
SET_SHORT_DESCRIPTION Procedure on page 19-10	Sets the short description for a dimension attribute.

CREATE_DIMENSION_ATTRIBUTE Procedure

This procedure creates a new dimension attribute.

If the dimension attribute name should be reserved for mapping specific groups of level attributes, you can set the RESERVED_DIMENSION_ATTRIBUTE argument to TRUE. For more information, see Table 19–1, " Reserved Dimension Attributes".

Descriptions and display properties must also be established as part of dimension attribute creation. Once the dimension attribute has been created, you can override these properties by calling other procedures in this package.

Syntax

```
CREATE_DIMENSION_ATTRIBUTE (
          dimension_owner               IN   VARCHAR2,
          dimension_name                IN   VARCHAR2,
          dimension_attribute_name      IN   VARCHAR2,
          display_name                  IN   VARCHAR2,
          short_description             IN   VARCHAR2,
          description                   IN   VARCHAR2,
          reserved_dimension_attribute  IN   BOOLEAN DEFAULT FALSE);
```

Parameters

Table 19–3 CREATE_DIMENSION_ATTRIBUTE Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
dimension_attribute_name	Name of the dimension attribute.
display_name	Display name for the dimension attribute.
short_description	Short description of the dimension attribute.
description	Description of the dimension attribute.
reserved_dimension_attribute	Whether or not this is a reserved dimension attribute. By default, the dimension attribute is not reserved. The reserved dimension attributes are described in Table 19–1, " Reserved Dimension Attributes".

Exceptions

Table 19–4 CREATE_DIMENSION_ATTRIBUTE Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
dimension_attribute_already_exists	Dimension attribute already exists.

DROP_DIMENSION_ATTRIBUTE Procedure

This procedure drops a dimension attribute.

Syntax

```
DROP_DIMENSION_ATTRIBUTE (
        dimension_owner               IN   VARCHAR2,
        dimension_name                IN   VARCHAR2,
        dimension_attribute_name      IN   VARCHAR2);
```

Parameters

Table 19–5 DROP_DIMENSION_ATTRIBUTE Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
dimension_attribute_name	Name of the dimension attribute.

Exceptions

Table 19–6 DROP_DIMENSION_ATTRIBUTE Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
dimension_not_found	Parent dimension not found.
dimension_attribute_not_found	Dimension attribute not found.

LOCK_DIMENSION_ATTRIBUTE Procedure

This procedure locks the dimension attribute for update by acquiring a database lock on the row that identifies the dimension attribute in the CWM2 model table.

Syntax

```
LOCK_DIMENSION_ATTRIBUTE (
          dimension_owner              IN   VARCHAR2,
          dimension_name               IN   VARCHAR2,
          dimension_attribute_name     IN   VARCHAR2,
          wait_for_lock                IN   BOOLEAN DEFAULT FALSE);
```

Parameters

Table 19–7 LOCK_DIMENSION_ATTRIBUTE Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
dimension_attribute_name	Name of the dimension attribute.
wait_for_lock	(Optional) Whether or not to wait for the dimension attribute to be available when it is already locked by another user. If you do not specify a value for this parameter, the procedure does not wait to acquire the lock.

Exceptions

Table 19–8 LOCK_DIMENSION_ATTRIBUTE Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
dimension_not_found	Parent dimension not found.
dimension_attribute_not_found	Dimension attribute not found.

SET_DESCRIPTION Procedure

This procedure sets the description for a dimension attribute.

Syntax

```
SET_DESCRIPTION (
          dimension_owner              IN   VARCHAR2,
          dimension_name               IN   VARCHAR2,
          dimension_attribute_name     IN   VARCHAR2,
          description                  IN   VARCHAR2);
```

Parameters

Table 19–9 SET_DESCRIPTION Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
dimension_attribute_name	Name of the dimension attribute.
description	Description of the dimension attribute.

Exceptions

Table 19–10 SET_DESCRIPTION Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
dimension_not_found	Parent dimension not found.
dimension_attribute_not_found	Dimension attribute not found.

SET_DIMENSION_ATTRIBUTE_NAME Procedure

This procedure sets the name for a dimension attribute.

Syntax

```
SET_DIMENSION_ATTRIBUTE_NAME (
          dimension_owner                 IN   VARCHAR2,
          dimension_name                  IN   VARCHAR2,
          dimension_attribute_name        IN   VARCHAR2,
          set_dimension_attribute_name    IN   VARCHAR2,
          reserved_dimension_attribute    IN   BOOLEAN DEFAULT FALSE);
```

Parameters

Table 19–11 SET_DIMENSION__ATTRIBUTE_NAME Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
dimension_ attribute_name	Original name for the dimension attribute.
set_dimension_ attribute_name	New name for the dimension attribute.
reserved_ dimension_ attribute	Whether or not this is a reserved dimension attribute. By default, the dimension attribute is not reserved.

Exceptions

Table 19–12 SET_DIMENSION_ATTRIBUTE_NAME Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
dimension_not_found	Parent dimension not found.
dimension_attribute_ not_found	Dimension attribute not found.

SET_DISPLAY_NAME Procedure

This procedure sets the display name for a dimension attribute.

Syntax

```
SET_DISPLAY_NAME (
          dimension_owner              IN   VARCHAR2,
          dimension_name               IN   VARCHAR2,
          dimension_attribute_name     IN   VARCHAR2,
          display_name                 IN   VARCHAR2);
```

Parameters

Table 19–13 SET_DISPLAY_NAME Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
dimension_ attribute_name	Name of the dimension attribute.
display_name	Display name for the dimension attribute.

Exceptions

Table 19–14 SET_DISPLAY_NAME Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
dimension_not_found	Parent dimension not found.
dimension_attribute_ not_found	Dimension attribute not found.

SET_SHORT_DESCRIPTION Procedure

This procedure sets the short description for a dimension attribute.

Syntax

```
SET_SHORT_DESCRIPTION (
          dimension_owner             IN   VARCHAR2,
          dimension_name              IN   VARCHAR2,
          dimension_attribute_name    IN   VARCHAR2,
          short_description           IN   VARCHAR2);
```

Parameters

Table 19–15 SET_SHORT_DESCRIPTION Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
dimension_attribute_name	Name of the dimension attribute.
short_description	Short description of the dimension attribute.

Exceptions

Table 19–16 SET_SHORT_DESCRIPTION Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
dimension_not_found	Parent dimension not found.
dimension_attribute_not_found	Dimension attribute not found.

Example: Creating a Dimension Attribute

The following statement creates a dimension attribute, PRODUCT_DIM_BRAND, for the PRODUCT_DIM dimension in the JSMITH schema. The display name is Brand.

The short description is `Brand Name`, and the description is `Product Brand Name`.

```
execute cwm2_olap_dimension_attribute.create_dimension_attribute
     ('JSMITH', 'PRODUCT_DIM', 'PRODUCT_DIM_BRAND',
      'Brand', 'Brand Name', 'Product Brand Name');
```

The following statement creates a dimension attribute, `'Short Description'`, for the `PRODUCT_DIM` dimension in the `JSMITH` schema. `Short Description` is a reserved dimension attribute.

```
execute cwm2_olap_dimension_attribute.create_dimension_attribute
     ('JSMITH', 'PRODUCT_DIM', 'Short Description',
      'Short Product Names', 'Short Desc Product',
      'Short Name of Products', TRUE);
```

20

CWM2_OLAP_HIERARCHY

The CWM2_OLAP_HIERARCHY package provides procedures for creating, dropping, and locking hierarchies. It also provides procedures for setting general hierarchy properties.

See Also: Chapter 13, "Using the OLAP Catalog Metadata APIs".

This chapter discusses the following topics:

- Understanding Hierarchies
- Summary of CWM2_OLAP_HIERARCHY Subprograms
- Example: Creating a Hierarchy

Understanding Hierarchies

A hierarchy is an OLAP metadata entity. This means that it is a logical object, identified by name and owner, within the OLAP Catalog.

Hierarchies define parent-child relationships between sets of levels in a dimension. There can be multiple hierarchies associated with a single dimension, and the same level can be used in multiple hierarchies. Hierarchies are fully described in "OLAP Metadata Model" on page 4-8.

Use the procedures in the `CWM2_OLAP_HIERARCHY` package to create, drop, and lock hierarchies and to specify descriptive information for display purposes.

The parent dimension must already exist in the OLAP Catalog before you can create hierarchies for it.

Summary of CWM2_OLAP_HIERARCHY Subprograms

Table 20–1 CWM2_OLAP_HIERARCHY Subprograms

Subprogram	Description
`CREATE_HIERARCHY Procedure on page 20-2`	Creates a hierarchy.
`DROP_HIERARCHY Procedure on page 20-4`	Drops a hierarchy.
`LOCK_HIERARCHY Procedure on page 20-5`	Locks the hierarchy for update.
`SET_DESCRIPTION Procedure on page 20-6`	Sets the description for a hierarchy.
`SET_DISPLAY_NAME Procedure on page 20-7`	Sets the display name for a hierarchy.
`SET_HIERARCHY_NAME Procedure on page 20-8`	Sets the name of a hierarchy.
`SET_SHORT_DESCRIPTION Procedure on page 20-9`	Sets the short description for a hierarchy.
`SET_SOLVED_CODE Procedure on page 20-10`	Sets the solved code for a hierarchy.

CREATE_HIERARCHY Procedure

This procedure creates a new hierarchy in the OLAP Catalog.

You must specify descriptions and display properties as part of hierarchy creation. Once the hierarchy has been created, you can override these properties by calling other procedures in the CWM2_OLAP_HIERARCHY package.

Syntax

```
CREATE_HIERARCHY (
          dimension_owner       IN   VARCHAR2,
          dimension_name        IN   VARCHAR2,
          hierarchy_name        IN   VARCHAR2,
          display_name          IN   VARCHAR2,
          short_description     IN   VARCHAR2,
          description           IN   VARCHAR2,
          solved_code           IN   VARCHAR2);
```

Parameters

Table 20–2 CREATE_HIERARCHY Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
hierarchy_name	Name of the hierarchy.
display_name	Display name for the hierarchy.
short_description	Short description of the hierarchy.
description	Description of the hierarchy.

Table 20–2 (Cont.) CREATE_HIERARCHY Procedure Parameters

Parameter	Description
solved_code	Specifies whether or not the hierarchy includes embedded totals and whether it is mapped to a level-based dimension table or a parent-child dimension table. For information about mapping hierarchies with different solved codes, see "Joining Fact Tables with Dimension Tables" on page 13-4. Values for this parameter are: ■ UNSOLVED LEVEL-BASED, for a hierarchy that contains no embedded totals and is stored in a level-based dimension table ■ SOLVED LEVEL-BASED, for a hierarchy that contains embedded totals, has a grouping ID, and is stored in a level-based dimension table ■ SOLVED VALUE-BASED, for a hierarchy that contains embedded totals and is stored in a parent-child dimension table

Exceptions

Table 20–3 CREATE_HIERARCHY Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the dimension owner and have the OLAP_DBA role.
dimension_not_found	Parent dimension not found.
hierarchy_already_exists	This hierarchy already exists for this dimension.

DROP_HIERARCHY Procedure

This procedure drops a hierarchy from the OLAP Catalog.

Syntax

```
DROP_HIERARCHY (
         dimension_owner     IN   VARCHAR2,
         dimension_name      IN   VARCHAR2,
         hierarchy_name      IN   VARCHAR2);
```

Parameters

Table 20–4 DROP_HIERARCHY Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
hierarchy_name	Name of the hierarchy.

Exceptions

Table 20–5 DROP_HIERARCHY Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the dimension owner and have the OLAP_DBA role.
dimension_not_found	Parent dimension not found.
hierarchy_not_found	Hierarchy not found.

LOCK_HIERARCHY Procedure

This procedure locks the hierarchy metadata for update by acquiring a database lock on the row that identifies the hierarchy in the CWM2 model table.

Syntax

```
LOCK_HIERARCHY (
          dimension_owner     IN   VARCHAR2,
          dimension_name      IN   VARCHAR2,
          hierarchy_name      IN   VARCHAR2,
          wait_for_lock       IN   BOOLEAN DEFAULT FALSE);
```

Parameters

Table 20–6 LOCK_HIERARCHY Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
hierarchy_name	Name of the hierarchy.
wait_for_lock	(Optional) Whether or not to wait for the hierarchy to be available when it is already locked by another user. If you do not specify a value for this parameter, the procedure does not wait to acquire the lock.

Exceptions

Table 20–7 LOCK_HIERARCHY Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the dimension owner and have the OLAP_DBA role.
dimension_not_found	Parent dimension not found.
hierarchy_not_found	Hierarchy not found.

SET_DESCRIPTION Procedure

This procedure sets the description for a hierarchy.

Syntax

```
SET_DESCRIPTION (
          dimension_owner    IN   VARCHAR2,
          dimension_name     IN   VARCHAR2,
          hierarchy_name     IN   VARCHAR2,
          description        IN   VARCHAR2);
```

Parameters

Table 20–8 SET_DESCRIPTION Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
hierarchy_name	Name of the hierarchy.
description	Description of the hierarchy.

Exceptions

Table 20–9 SET_DESCRIPTION Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the dimension owner and have the OLAP_DBA role.
dimension_not_found	Parent dimension not found.
hierarchy_not_found	Hierarchy not found.

SET_DISPLAY_NAME Procedure

This procedure sets the display name for a dimension.

Syntax

```
SET_DISPLAY_NAME (
          dimension_owner     IN   VARCHAR2,
          dimension_name      IN   VARCHAR2,
          hierarchy_name      IN   VARCHAR2,
          display_name        IN   VARCHAR2);
```

Parameters

Table 20–10 SET_DISPLAY_NAME Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
hierarchy_name	Name of the hierarchy.
display_name	Display name for the hierarchy.

Exceptions

Table 20–11 SET_DISPLAY_NAME Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the dimension owner and have the OLAP_DBA role.
dimension_not_found	Parent dimension not found.
hierarchy_not_found	Hierarchy not found.

SET_HIERARCHY_NAME Procedure

This procedure sets the name for a hierarchy.

Syntax

```
SET_HIERARCHY_NAME (
          dimension_owner        IN   VARCHAR2,
          dimension_name         IN   VARCHAR2,
          hierarchy_name         IN   VARCHAR2,
          set_hierarchy_name     IN   VARCHAR2);
```

Parameters

Table 20–12 SET_HIERARCHY_NAME Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
hierarchy_name	Original name for the hierarchy.
set_hierarchy_name	New name for the hierarchy.

Exceptions

Table 20–13 SET_HIERARCHY_NAME Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the dimension owner and have the OLAP_DBA role.
dimension_not_found	Parent dimension not found.
hierarchy_not_found	Hierarchy not found.

SET_SHORT_DESCRIPTION Procedure

This procedure sets the short description for a hierarchy.

Syntax

```
SET_SHORT_DESCRIPTION (
          dimension_owner        IN   VARCHAR2,
          dimension_name         IN   VARCHAR2,
          hierarchy_name         IN   VARCHAR2,
          short_description      IN   VARCHAR2);
```

Parameters

Table 20–14 SET_SHORT_DESCRIPTION Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
hierarchy_name	Name of the hierarchy.
short_description	Short description of the hierarchy.

Exceptions

Table 20–15 SET_SHORT_DESCRIPTION Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the dimension owner and have the OLAP_DBA role.
dimension_not_found	Parent dimension not found.
hierarchy_not_found	Hierarchy not found.

SET_SOLVED_CODE Procedure

This procedure sets the solved code for a hierarchy. The solved code specifies whether or not the data dimensioned by this hierarchy includes embedded totals and whether it is mapped to a level-based dimension table or a parent-child dimension table. If mapped to a parent-child dimension table, it cannot be accessed by the OLAP API.

For more information on mapping solved and unsolved data, see "Joining Fact Tables with Dimension Tables" on page 13-4.

Syntax

```
SET_SOLVED_CODE (
          dimension_owner     IN   VARCHAR2,
          dimension_name      IN   VARCHAR2,
          hierarchy_name      IN   VARCHAR2,
          solved_code         IN   VARCHAR2);
```

Parameters

Table 20–16 SET_SOLVED_CODE Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
hierarchy_name	Name of the hierarchy.
solved_code	One of the following values: ■ UNSOLVED LEVEL, for a hierarchy that contains no embedded totals and is stored in a level-based dimension table ■ SOLVED LEVEL, for a hierarchy that contains embedded totals, has a grouping ID, and is stored in a level-based dimension table ■ SOLVED VALUE, for a hierarchy that contains embedded totals and is stored in a parent-child dimension table. This type of hierarchy cannot be accessed by the OLAP API.

Exceptions

Table 20–17 SET_SOLVED_CODE Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the dimension owner and have the OLAP_DBA role.
dimension_not_found	Parent dimension not found.
hierarchy_not_found	Hierarchy not found.

Example: Creating a Hierarchy

The following statement creates a dimension hierarchy PRODUCT_DIM_ROLLUP, for the PRODUCT_DIM dimension in the JSMITH schema. The display name is

`Standard.` The short description is `Std Product`, and the description is `Standard Product Hierarchy`. The solved code is `SOLVED LEVEL-BASED`, meaning that this hierarchy will be mapped to an embedded total dimension table, and that the fact table associated with this dimension hierarchy will store fully solved data.

```
execute cwm2_olap_hierarchy.create_hierarchy
     ('JSMITH', 'PRODUCT_DIM', 'PRODUCT_DIM_ROLLUP',
      'Standard', 'Std Product', 'Standard Product Hierarchy',
      'SOLVED LEVEL-BASED');
```

21

CWM2_OLAP_LEVEL

The CWM2_OLAP_LEVEL package provides procedures for creating, dropping, and locking levels, and for adding levels to hierarchies. It also provides procedures for setting the general properties of levels.

See Also: Chapter 13, "Using the OLAP Catalog Metadata APIs".

This chapter discusses the following topics:

- Understanding Levels
- Summary of CWM2_OLAP_LEVEL Subprograms
- Example: Creating a Level

Understanding Levels

A level is an OLAP metadata entity. This means that it is a logical object, identified by name and owner, within the OLAP Catalog.

Dimension members are organized in levels that map to columns in dimension tables or views. Levels are typically organized in hierarchies. Every dimension must have at least one level. Levels are fully described in "OLAP Metadata Model" on page 4-8

Use the procedures in the CWM2_OLAP_LEVEL package to create, drop, and lock levels, to assign levels to hierarchies, and to specify descriptive information for display purposes.

The parent dimension and the parent hierarchy must already exist in the OLAP Catalog before you can create a level.

Summary of CWM2_OLAP_LEVEL Subprograms

Table 21–1 CWM2_OLAP_LEVEL Subprograms

Subprogram	Description
ADD_LEVEL_TO_HIERARCHY Procedure on page 21-3	Adds a level to a hierarchy.
CREATE_LEVEL Procedure on page 21-4	Creates a level.
DROP_LEVEL Procedure on page 21-5	Drops a level.
LOCK_LEVEL Procedure on page 21-6	Locks the level metadata for update.
REMOVE_LEVEL_FROM_ HIERARCHY Procedure on page 21-7	Removes a level from a hierarchy.
SET_DESCRIPTION Procedure on page 21-8	Sets the description for a level.
SET_DISPLAY_NAME Procedure on page 21-9	Sets the display name for a level.
SET_LEVEL_NAME Procedure on page 21-9	Sets the name of a level.

Table 21–1 (Cont.) CWM2_OLAP_LEVEL Subprograms

Subprogram	Description
SET_PLURAL_NAME Procedure on page 21-10	Sets the plural name for a level.
SET_SHORT_DESCRIPTION Procedure on page 21-11	Sets the short description for a level.

ADD_LEVEL_TO_HIERARCHY Procedure

This procedure adds a level to a hierarchy.

Syntax

```
ADD_LEVEL_TO_HIERARCHY (
          dimension_owner     IN   VARCHAR2,
          dimension_name      IN   VARCHAR2,
          hierarchy_name      IN   VARCHAR2,
          level_name          IN   VARCHAR2,
          parent_level_name   IN   VARCHAR2  DEFAULT);
```

Parameters

Table 21–2 ADD_LEVEL_TO_HIERARCHY Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
hierarchy_name	Name of the hierarchy.
level_name	Name of the level to add to the hierarchy.
parent_level_name	Name of the level's parent in the hierarchy. If you do not specify a parent, then the added level is the root of the hierarchy.

Exceptions

Table 21–3 ADD_LEVEL_TO_HIERARCHY Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
dimension_not_found	Parent dimension not found.
hierarchy_not_found	Hierarchy not found.
level_not_found	Level not found.

CREATE_LEVEL Procedure

This procedure creates a new level in the OLAP Catalog.

You must specify descriptions and display properties as part of level creation. Once the level has been created, you can override these properties by calling other procedures in the CWM2_OLAP_LEVEL package.

Syntax

```
CREATE_LEVEL (
          dimension_owner       IN   VARCHAR2,
          dimension_name        IN   VARCHAR2,
          level_name            IN   VARCHAR2,
          display_name          IN   VARCHAR2,
          plural_name           IN   VARCHAR2,
          short_description     IN   VARCHAR2,
          description           IN   VARCHAR2);
```

Parameters

Table 21–4 CREATE_LEVEL Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
level_name	Name of the level.
display_name	Display name for the level.
plural_name	Plural name for the level.

Table 21–4 (Cont.) CREATE_LEVEL Procedure Parameters

Parameter	Description
short_description	Short description of the level.
description	Description of the level.

Exceptions

Table 21–5 CREATE_LEVEL Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
dimension_not_found	Parent dimension not found.
level_already_exists	This level already exists for this dimension.

DROP_LEVEL Procedure

This procedure drops a level from the OLAP Catalog. All related level attributes are also dropped.

Syntax

```
DROP_LEVEL (
         dimension_owner     IN   VARCHAR2,
         dimension_name      IN   VARCHAR2,
         level_name          IN   VARCHAR2);
```

Parameters

Table 21–6 DROP_LEVEL Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
level_name	Name of the level.

Exceptions

Table 21–7 DROP_LEVEL Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
dimension_not_found	Parent dimension not found.
level_not_found	Level not found.

LOCK_LEVEL Procedure

This procedure locks the level metadata for update by acquiring a database lock on the row that identifies the level in the CWM2 model table.

Syntax

```
LOCK_LEVEL (
           dimension_owner     IN   VARCHAR2,
           dimension_name      IN   VARCHAR2,
           level_name          IN   VARCHAR2,
           wait_for_lock       IN   BOOLEAN DEFAULT FALSE);
```

Parameters

Table 21–8 LOCK_LEVEL Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
level_name	Name of the level.
wait_for_lock	(Optional) Whether or not to wait for the level to be available when it is already locked by another user. If you do not specify a value for this parameter, the procedure does not wait to acquire the lock.

Exceptions

Table 21–9 LOCK_LEVEL Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
dimension_not_found	Parent dimension not found.
level_not_found	Level not found.

REMOVE_LEVEL_FROM_HIERARCHY Procedure

This procedure removes a level from a hierarchy.

Syntax

```
REMOVE_LEVEL_FROM_HIERARCHY (
          dimension_owner     IN   VARCHAR2,
          dimension_name      IN   VARCHAR2,
          hierarchy_name      IN   VARCHAR2,
          level_name          IN   VARCHAR2);
```

Parameters

Table 21–10 REMOVE_LEVEL_FROM_HIERARCHY Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
hierarchy_name	Name of the hierarchy.
level_name	Name of the level to remove from the hierarchy.

Exceptions

Table 21–11 REMOVE_LEVEL_FROM_HIERARCHY Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.

Table 21–11 (Cont.) REMOVE_LEVEL_FROM_HIERARCHY Procedure Exceptions

Exception	Description
dimension_not_found	Parent dimension not found.
hierarchy_not_found	Hierarchy not found.
child_level_not_found	Child level not found.

SET_DESCRIPTION Procedure

This procedure sets the description for a level.

Syntax

```
SET_DESCRIPTION (
          dimension_owner     IN   VARCHAR2,
          dimension_name      IN   VARCHAR2,
          level_name          IN   VARCHAR2,
          description         IN   VARCHAR2);
```

Parameters

Table 21–12 SET_DESCRIPTION Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
level_name	Name of the level.
description	Description of the level.

Exceptions

Table 21–13 SET_DESCRIPTION Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
dimension_not_found	Parent dimension not found.
level_not_found	Level not found.

SET_DISPLAY_NAME Procedure

This procedure sets the display name for a level.

Syntax

```
SET_DISPLAY_NAME (
          dimension_owner     IN   VARCHAR2,
          dimension_name      IN   VARCHAR2,
          level_name          IN   VARCHAR2,
          display_name        IN   VARCHAR2);
```

Parameters

Table 21–14 SET_DISPLAY_NAME Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
level_name	Name of the level.
display_name	Display name for the level.

Exceptions

Table 21–15 SET_DISPLAY_NAME Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
dimension_not_found	Parent dimension not found.
level_not_found	Level not found.

SET_LEVEL_NAME Procedure

This procedure sets the name for a level.

Syntax

```
SET_LEVEL_NAME (
          dimension_owner   IN   VARCHAR2,
          dimension_name    IN   VARCHAR2,
          level_name        IN   VARCHAR2,
          set_level_name    IN   VARCHAR2);
```

Parameters

Table 21–16 SET_LEVEL_NAME Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
level_name	Original name for the level.
set_level_name	New name for the level.

Exceptions

Table 21–17 SET_LEVEL_NAME Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
dimension_not_found	Parent dimension not found.
level_not_found	Level not found.

SET_PLURAL_NAME Procedure

This procedure sets the plural name of a level.

Syntax

```
SET_PLURAL_NAME  (
          dimension_owner     IN   VARCHAR2,
          dimension_name      IN   VARCHAR2,
          level_name          IN   VARCHAR2,
          plural_name         IN   VARCHAR2);
```

Parameters

Table 21–18 SET_PLURAL_NAME Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
level_name	Name of the level.
plural_name	Plural name for the level.

Exceptions

Table 21–19 SET_PLURAL_NAME Procedure Exceptions

Exception	Description
no_access_privileges	User does not have privileges to edit the dimension. User must be the owner or OLAP_DBA.
dimension_not_found	Parent dimension not found.
level_not_found	Level not found.

SET_SHORT_DESCRIPTION Procedure

This procedure sets the short description for a level.

Syntax

```
SET_SHORT_DESCRIPTION (
          dimension_owner       IN   VARCHAR2,
          dimension_name        IN   VARCHAR2,
          level_name            IN   VARCHAR2,
          short_description     IN   VARCHAR2);
```

Parameters

Table 21–20 SET_SHORT_DESCRIPTION Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
level_name	Name of the level.
short_description	Short description of the level.

Exceptions

Table 21–21 SET_SHORT_DESCRIPTION Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
dimension_not_found	Parent dimension not found.
level_not_found	Level not found.

Example: Creating a Level

The following statements create four levels for the PRODUCT_DIM dimension and assign them to the PRODUCT_DIM_ROLLUP hierarchy.

```
execute cwm2_olap_level.create_level
     ('JSMITH', 'PRODUCT_DIM', 'TOTALPROD_LVL',
      'Total Product', 'All Products', 'Total',
      'Equipment and Parts of standard product hierarchy');
execute cwm2_olap_level.create_level
     ('JSMITH', 'PRODUCT_DIM', 'PROD_CATEGORY_LVL',
      'Product Category', 'Product Categories', 'Category',
      'Categories of standard product hierarchy');
```

```
execute cwm2_olap_level.create_level
     ('JSMITH', 'PRODUCT_DIM', 'PROD_SUBCATEGORY_LVL',
      'Product Sub-Category', 'Product Sub-Categories', 'Sub-Category',
      'Sub-Categories of standard product hierarchy');
execute cwm2_olap_level.create_level
     ('JSMITH', 'PRODUCT_DIM', 'PRODUCT_LVL',
      'Product', 'Products', 'Product',
      'Individual products of standard product hierarchy');

execute cwm2_olap_level.add_level_to_hierarchy
     ('JSMITH', 'PRODUCT_DIM', 'PRODUCT_DIM_ROLLUP',
      'PRODUCT_LVL', 'PROD_SUBCATEGORY_LVL');
execute cwm2_olap_level.add_level_to_hierarchy
     ('JSMITH', 'PRODUCT_DIM', 'PRODUCT_DIM_ROLLUP',
      'PROD_SUBCATEGORY_LVL', 'PROD_CATEGORY_LVL');
execute cwm2_olap_level.add_level_to_hierarchy
     ('JSMITH', 'PRODUCT_DIM', 'PRODUCT_DIM_ROLLUP',
      'PROD_CATEGORY_LVL', 'TOTALPROD_LVL');
execute cwm2_olap_level.add_level_to_hierarchy
     ('JSMITH', 'PRODUCT_DIM', 'PRODUCT_DIM_ROLLUP', 'TOTALPROD_LVL');
```

22

CWM2_OLAP_LEVEL_ATTRIBUTE

The CWM2_OLAP_LEVEL_ATTRIBUTE package provides a procedure for creating level attributes and associating them with levels and dimension attributes. It also provides procedures for dropping, locking, and setting the general properties of level attributes.

See Also: Chapter 13, "Using the OLAP Catalog Metadata APIs".

This chapter discusses the following topics:

- Understanding Level Attributes
- Summary of CWM2_OLAP_LEVEL_ATTRIBUTE Subprograms
- Example: Creating a Level Attribute

Understanding Level Attributes

A level attribute is an OLAP metadata entity. This means that it is a logical object, identified by name and owner, within the OLAP Catalog.

A level attribute is a child entity of a level and a dimension attribute. A level attribute stores descriptive information about its related level. For example, a level containing product identifiers might have an associated level attribute that contains color information for each product.

Each level attribute maps to a column in a dimension table. The level attribute column must be in the same table as the column (or columns) for its associated level. Level attributes are fully described in "OLAP Metadata Model" on page 4-8.

Use the procedures in the `CWM2_OLAP_LEVEL_ATTRIBUTE` package to create, drop, and lock level attributes, to assign level attributes to levels and dimension attributes, and to specify descriptive information for display purposes.

Several level attribute names are reserved, because they have special significance within `CWM2`. Reserved level attributes are associated with reserved dimension attributes of the same name. Reserved level attributes will be mapped to columns containing specific information. The reserved level attributes are listed in Table 22–1.

Table 22–1 Reserved Level Attributes

Dimension Attribute	Description
`Long Description`	A long description of the dimension member.
`Short Description`	A short description of the dimension member.
`End Date`	For a time dimension, the last date in a time period. (Required)
`Time Span`	For a time dimension, the number of days in a time period. (Required)
`Prior Period`	For a time dimension, the time period before this time period.
`Year Ago Period`	For a time dimension, the period a year before this time period.
`ET Key`	For an embedded total dimension, the embedded total key, which identifies the dimension member at the lowest level in a row of the dimension table.
`Parent ET Key`	For an embedded total dimension, the dimension member that is the parent of the ET key.

Table 22–1 (Cont.) Reserved Level Attributes

Dimension Attribute	Description
`Grouping ID`	For an embedded total dimension, the grouping ID (GID), which identifies the hierarchical level for a row of the dimension table.
`Parent Grouping ID`	For an embedded total dimension, the dimension member that is the parent of the grouping ID.

The parent dimension, parent level, and parent dimension attribute must already exist in the OLAP Catalog before you can create a level attribute.

Summary of CWM2_OLAP_LEVEL_ATTRIBUTE Subprograms

Table 22–2 CWM2_OLAP_LEVEL_ATTRIBUTE Subprograms

Subprogram	Description
`CREATE_LEVEL_ATTRIBUTE on page 22-3`	Creates a level attribute.
`DROP_LEVEL_ATTRIBUTE Procedure on page 22-5`	Drops a level attribute.
`LOCK_LEVEL_ATTRIBUTE Procedure on page 22-6`	Locks the level attribute metadata for update.
`SET_DESCRIPTION Procedure on page 22-8`	Sets the description for a level attribute.
`SET_DISPLAY_NAME Procedure on page 22-9`	Sets the display name for a level attribute.
`SET_LEVEL_ATTRIBUTE_NAME Procedure on page 22-10`	Sets the name of a level attribute.
`SET_SHORT_DESCRIPTION Procedure on page 22-12`	Sets the short description for a level attribute.

CREATE_LEVEL_ATTRIBUTE

This procedure creates a new level attribute in the OLAP Catalog and associates the level attribute with a level and with a dimension attribute.

If the level attribute name should be reserved for a specific level and dimension attribute combination, you can set the RESERVED_LEVEL_ATTRIBUTE argument to TRUE. For more information, see Table 22–1, " Reserved Level Attributes".

You must specify descriptions and display properties as part of level attribute creation. Once the level attribute has been created, you can override these properties by calling other procedures in the CWM2_OLAP_LEVEL_ATTRIBUTE package.

Syntax

```
CREATE_LEVEL_ATTRIBUTE (
          dimension_owner            IN   VARCHAR2,
          dimension_name             IN   VARCHAR2,
          dimension_attribute_name   IN   VARCHAR2,
          level_name                 IN   VARCHAR2,
          level_attribute_name       IN   VARCHAR2,
          display_name               IN   VARCHAR2,
          short_description          IN   VARCHAR2,
          description                IN   VARCHAR2,
          reserved_level_attribute   IN   BOOLEAN  FALSE);
```

Parameters

Table 22–3 CREATE_LEVEL_ATTRIBUTE Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
dimension_attribute_name	Name of the dimension attribute that includes this level attribute.
level_name	Name of the level.
level_attribute_name	Name of the level attribute.
display_name	Display name for the level attribute.
short_description	Short description of the level attribute.
description	Description of the level attribute.

Table 22–3 (Cont.) CREATE_LEVEL_ATTRIBUTE Procedure Parameters

Parameter	Description
reserved_level_attribute	Whether or not this level attribute is reserved. By default, the level attribute is not reserved. The reserved level attributes are as follows. Long Description Short Description End Date Time Span Prior Period Year Ago Period ET Key Parent ET Key Grouping ID Parent Grouping ID

Exceptions

Table 22–4 CREATE_LEVEL_ATTRIBUTE Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
dimension_not_found	Parent dimension not found.
dimension_attribute_not_found	Parent dimension attribute not found.
level_not_found	Parent level not found.
level_attribute_already_exists	This level attribute already exists for this level.

DROP_LEVEL_ATTRIBUTE Procedure

This procedure drops a level attribute from the OLAP Catalog.

Syntax

```
DROP_LEVEL_ATTRIBUTE (
         dimension_owner            IN   VARCHAR2,
         dimension_name             IN   VARCHAR2,
         dimension_attribute_name   IN   VARCHAR2,
         level_name                 IN   VARCHAR2,
         level_attribute_name       IN   VARCHAR2);
```

Parameters

Table 22–5 DROP_LEVEL_ATTRIBUTE Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
dimension_attribute_name	Name of the dimension attribute.
level_name	Name of the level.
level_attribute_name	Name of the level attribute.

Exceptions

Table 22–6 DROP_LEVEL_ATTRIBUTE Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
dimension_not_found	Parent dimension not found.
dimension_attribute_not_found	Parent dimension attribute not found.
level_not_found	Parent level not found.
level_attribute_not_found	Level attribute not found.

LOCK_LEVEL_ATTRIBUTE Procedure

This procedure locks the level attribute metadata for update by acquiring a database lock on the row that identifies the level attribute in the CWM2 model table.

Syntax

```
LOCK_LEVEL_ATTRIBUTE (
          dimension_owner            IN   VARCHAR2,
          dimension_name             IN   VARCHAR2,
          dimension_attribute_name   IN   VARCHAR2,
          level_name                 IN   VARCHAR2,
          level_attribute_name       IN   VARCHAR2,
          wait_for_lock              IN   BOOLEAN DEFAULT FALSE);
```

Parameters

Table 22–7 LOCK_LEVEL_ATTRIBUTE Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
dimension_attribute_name	Name of the dimension attribute.
level_name	Name of the level.
level_attribute_name	Name of the level attribute.
wait_for_lock	(Optional) Whether or not to wait for the level attribute to be available when it is already locked by another user. If you do not specify a value for this parameter, the procedure does not wait to acquire the lock.

Exceptions

Table 22–8 LOCK_LEVEL_ATTRIBUTE Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
dimension_not_found	Parent dimension not found.
dimension_attribute_not_found	Parent dimension attribute not found.
level_not_found	Parent level not found.

Table 22–8 (Cont.) LOCK_LEVEL_ATTRIBUTE Procedure Exceptions

Exception	Description
level_attribute_not_found	Level attribute not found.

SET_DESCRIPTION Procedure

This procedure sets the description for a level attribute.

Syntax

```
SET_DESCRIPTION (
          dimension_owner            IN   VARCHAR2,
          dimension_name             IN   VARCHAR2,
          dimension_attribute_name   IN   VARCHAR2,
          level_name                 IN   VARCHAR2,
          level_attribute_name       IN   VARCHAR2,
          description                IN   VARCHAR2);
```

Parameters

Table 22–9 SET_DESCRIPTION Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
dimension_attribute_name	Name of the dimension attribute.
level_name	Name of the level.
level_attribute_name	Name of the level attribute.
description	Description of the level attribute.

Exceptions

Table 22–10 SET_DESCRIPTION Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.

Table 22–10 (Cont.) SET_DESCRIPTION Procedure Exceptions

Exception	Description
dimension_not_found	Parent dimension not found.
dimension_attribute_not_found	Parent dimension attribute not found.
level_not_found	Parent level not found.
level_attribute_not_found	Level attribute not found.

SET_DISPLAY_NAME Procedure

This procedure sets the display name for a level attribute.

Syntax

```
SET_DISPLAY_NAME (
          dimension_owner            IN   VARCHAR2,
          dimension_name             IN   VARCHAR2,
          dimension_attribute_name   IN   VARCHAR2,
          level_name                 IN   VARCHAR2,
          level_attribute_name       IN   VARCHAR2,
          display_name               IN   VARCHAR2);
```

Parameters

Table 22–11 SET_DISPLAY_NAME Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
dimension_attribute_name	Name of the dimension attribute.
level_name	Name of the level.
level_attribute_name	Name of the level attribute.
display_name	Display name for the level attribute.

Exceptions

Table 22–12 SET_DISPLAY_NAME Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
dimension_not_found	Parent dimension not found.
dimension_attribute_not_found	Parent dimension attribute not found.
level_not_found	Parent level not found.
level_attribute_not_found	Level attribute not found.

SET_LEVEL_ATTRIBUTE_NAME Procedure

This procedure sets the name for a level attribute.

For information on reserved level attribute names, see Table 22–1, " Reserved Level Attributes".

Syntax

```
SET_LEVEL_ATTRIBUTE_NAME (
          dimension_owner             IN   VARCHAR2,
          dimension_name              IN   VARCHAR2,
          dimension_attribute_name    IN   VARCHAR2,
          level_name                  IN   VARCHAR2,
          level_attribute_name        IN   VARCHAR2,
          set_level_attribute_name    IN   VARCHAR2,
          reserved_level_attribute    IN   BOOLEAN DEFAULT FALSE);
```

Parameters

Table 22–13 SET_LEVEL_ATTRIBUTE_NAME Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
dimension_attribute_name	Name of the dimension attribute.

Table 22–13 (Cont.) SET_LEVEL_ATTRIBUTE_NAME Procedure Parameters

Parameter	Description
level_name	Name for the level.
level_attribute_name	Original name for the level attribute.
set_level_attribute_name	New name for the level attribute.
reserved_level_attribute	Whether or not this level attribute is reserved. By default, the level attribute is not reserved.

Exceptions

Table 22–14 SET_LEVEL_ATTRIBUTE_NAME Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
dimension_not_found	Parent dimension not found.
dimension_attribute_not_found	Parent dimension attribute not found.
level_not_found	Parent level not found.
level_attribute_not_found	Level attribute not found.

SET_SHORT_DESCRIPTION Procedure

This procedure sets the short description for a level attribute.

Syntax

```
SET_SHORT_DESCRIPTION (
          dimension_owner            IN   VARCHAR2,
          dimension_name             IN   VARCHAR2,
          dimension_attribute_name   IN   VARCHAR2,
          level_name                 IN   VARCHAR2,
          level_attribute_name       IN   VARCHAR2,
          short_description          IN   VARCHAR2);
```

Parameters

Table 22–15 SET_SHORT_DESCRIPTION Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
dimension_attribute_name	Name of the dimension attribute.
level_name	Name of the level.
level_attribute_name	Name of the level attribute.
short_description	Short description of the level attribute.

Exceptions

Table 22–16 SET_SHORT_DESCRIPTION Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
dimension_not_found	Parent dimension not found.
dimension_attribute_not_found	Parent dimension attribute not found.
level_not_found	Parent level not found.
level_attribute_not_found	Level attribute not found.

Example: Creating a Level Attribute

The following statements create a color attribute for the lowest level and long descriptions for all four levels of the PRODUCT_DIM dimension.

```
execute cwm2_olap_level_attribute.create_level_attribute
     ('JSMITH', 'PRODUCT_DIM', 'Product Color', 'PRODUCT_LVL', 'Product Color',
      'PROD_STD_COLOR', 'Prod Color', 'Product Color');

execute cwm2_olap_level_attribute.create_level_attribute
     ('JSMITH', 'PRODUCT_DIM', 'Long Description', 'PRODUCT_LVL',
      'Long Description','PRODUCT_STD_LLABEL', 'Product',
      'Long Labels for individual products of the PRODUCT hierarchy', TRUE);

execute cwm2_olap_level_attribute.create_level_attribute
     ('JSMITH', 'PRODUCT_DIM', 'Long Description', 'PROD_SUBCATEGORY_LVL',
      'Long Description', 'PROD_STD_LLABEL', 'Product Sub Category',
      'Long Labels for subcategories of the PRODUCT hierarchy', TRUE);

execute cwm2_olap_level_attribute.create_level_attribute
     ('JSMITH', 'PRODUCT_DIM', 'Long Description', 'PROD_CATEGORY_LVL',
      'Long Description', 'PROD_STD_LLABEL', 'Product Category',
      'Long Labels for categories of the PRODUCT hierarchy', TRUE);

execute cwm2_olap_level_attribute.create_level_attribute
     ('JSMITH', 'PRODUCT_DIM', 'Long Description', 'TOTALPROD_LVL',
      'Long Description', 'PROD_STD_LLABEL', 'Total Product',
      'Long Labels for total of the PRODUCT hierarchy', TRUE);
```

23

CWM2_OLAP_MEASURE

The CWM2_OLAP_MEASURE package provides procedures for creating, dropping, and locking measures. It also provides procedures for setting general properties of measures.

See Also: Chapter 13, "Using the OLAP Catalog Metadata APIs".

This chapter discusses the following topics:

- Understanding Measures
- Summary of CWM2_OLAP_MEASURE Subprograms
- Example: Creating a Measure

Understanding Measures

A measure is an OLAP metadata entity. This means that it is a logical object, identified by name and owner, within the OLAP Catalog.

Measures represent data stored in fact tables. The fact tables may be relational tables or views. The views may reference data stored in analytic workspaces.

Measures exist within the context of cubes, which fully specify the dimensionality of the measures' data. Measures are fully described in "OLAP Metadata Model" on page 4-8.

Use the procedures in the `CWM2_OLAP_MEASURE` package to create, drop, and lock measures, to associate a measure with a cube, and to specify descriptive information for display purposes.

The parent cube must already exist in the OLAP Catalog before you can create a measure.

Summary of CWM2_OLAP_MEASURE Subprograms

Table 23–1 CWM2_OLAP_MEASURE Subprograms

Subprogram	Description
`CREATE_MEASURE Procedure on page 23-3`	Creates a measure.
`DROP_MEASURE Procedure on page 23-4`	Drops a measure.
`LOCK_MEASURE Procedure on page 23-4`	Locks a measure's metadata for update.
`SET_DESCRIPTION Procedure on page 23-5`	Sets the description for a measure.
`SET_DISPLAY_NAME Procedure on page 23-6`	Sets the display name for a measure.
`SET_MEASURE_NAME Procedure on page 23-7`	Sets the name of a measure.
`SET_SHORT_DESCRIPTION Procedure on page 23-8`	Sets the short description for a measure.

CREATE_MEASURE Procedure

This procedure creates a new measure in the OLAP Catalog.

A measure can only be created in the context of a cube. The cube must already exist before you create the measure.

Descriptions and display properties must also be established as part of measure creation. Once the measure has been created, you can override these properties by calling other procedures in this package.

Syntax

```
CREATE_MEASURE (
          cube_owner            IN   VARCHAR2,
          cube_name             IN   VARCHAR2,
          measure_name          IN   VARCHAR2,
          display_name          IN   VARCHAR2,
          short_description     IN   VARCHAR2,
          description           IN   VARCHAR2);
```

Parameters

Table 23–2 CREATE_MEASURE Procedure Parameters

Parameter	Description
cube_owner	Owner of the cube.
cube_name	Name of the cube.
measure_name	Name of the measure.
display_name	Display name for the measure.
short_description	Short description of the measure.
description	Description of the measure.

Exceptions

Table 23–3 CREATE_MEASURE Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
cube_not_found	Parent cube not found.

Table 23–3 (Cont.) CREATE_MEASURE Procedure Exceptions

Exception	Description
measure_already_exists	This measure already exists for this cube.

DROP_MEASURE Procedure

This procedure drops a measure from a cube.

Syntax

```
DROP_MEASURE (
        cube_owner      IN   VARCHAR2,
        cube_name        IN   VARCHAR2,
        measure_name    IN   VARCHAR2);
```

Parameters

Table 23–4 DROP_MEASURE Procedure Parameters

Parameter	Description
cube_owner	Owner of the cube.
cube_name	Name of the cube.
measure_name	Name of the measure to be dropped from the cube.

Exceptions

Table 23–5 DROP_MEASURE Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
cube_not_found	Parent cube not found.
measure_not_found	Measure not found.

LOCK_MEASURE Procedure

This procedure locks the measure's metadata for update by acquiring a database lock on the row that identifies the measure in the CWM2 model table.

Syntax

```
LOCK_MEASURE (
           cube_owner         IN   VARCHAR2,
           cube_name          IN   VARCHAR2.
           measure_name        IN   VARCHAR2,
           wait_for_lock      IN   BOOLEAN DEFAULT FALSE);
```

Parameters

Table 23–6 LOCK_MEASURE Procedure Parameters

Parameter	Description
cube_owner	Owner of the cube.
cube_name	Name of the cube.
measure_name	Name of the measure to be locked.
wait_for_lock	(Optional) Whether or not to wait for the measure to be available when it is already locked by another user. If you do not specify a value for this parameter, the procedure does not wait to acquire the lock.

Exceptions

Table 23–7 LOCK_MEASURE Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
cube_not_found	Parent cube not found.
measure_not_found	Measure not found.

SET_DESCRIPTION Procedure

This procedure sets the description for a measure.

Syntax

```
SET_DESCRIPTION (
          cube_owner     IN   VARCHAR2,
          cube_name      IN   VARCHAR2,
          measure_name   IN   VARCHAR2,
          description    IN   VARCHAR2);
```

Parameters

Table 23–8 SET_DESCRIPTION Procedure Parameters

Parameter	Description
cube_owner	Owner of the cube.
cube_name	Name of the cube.
measure_name	Name of the measure.
description	Description of the measure.

Exceptions

Table 23–9 SET_DESCRIPTION Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
cube_not_found	Parent cube not found.
measure_not_found	Measure not found.

SET_DISPLAY_NAME Procedure

This procedure sets the display name for a measure.

Syntax

```
SET_DISPLAY_NAME (
          cube_owner     IN   VARCHAR2,
          cube_name      IN   VARCHAR2,
          measure_name   IN   VARCHAR2,
          display_name   IN   VARCHAR2);
```

Parameters

Table 23–10 SET_DISPLAY_NAME Procedure Parameters

Parameter	Description
cube_owner	Owner of the cube.
cube_name	Name of the cube.
measure_name	Name of the measure.
display_name	Display name for the measure.

Exceptions

Table 23–11 SET_DISPLAY_NAME Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
cube_not_found	Parent cube not found.
measure_not_found	Measure not found.

SET_MEASURE_NAME Procedure

This procedure sets the name for a measure.

Syntax

```
SET_MEASURE_NAME (
          cube_owner        IN   VARCHAR2,
          cube_name         IN   VARCHAR2,
          measure_name      IN   VARCHAR2,
          set_cube_name     IN   VARCHAR2);
```

Parameters

Table 23–12 SET_MEASURE_NAME Procedure Parameters

Parameter	Description
cube_owner	Owner of the cube.
cube_name	Name of the cube.
measure_name	Original name of the measure.
set_cube_name	New name for the measure.

Exceptions

Table 23–13 SET_MEASURE_NAME Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
cube_not_found	Parent cube not found.
measure_not_found	Measure not found.

SET_SHORT_DESCRIPTION Procedure

This procedure sets the short description for a cube.

Syntax

```
SET_DESCRIPTION (
          cube_owner            IN   VARCHAR2,
          cube_name             IN   VARCHAR2,
          measure_name          IN   VARCHAR2,
          short_description     IN   VARCHAR2);
```

Parameters

Table 23–14 SET_SHORT_DESCRIPTION Procedure Parameters

Parameter	Description
cube_owner	Owner of the cube.
cube_name	Name of the cube.
measure_name	Name of the measure.
short_description	Short description of the measure.

Exceptions

Table 23–15 SET_SHORT_DESCRIPTION Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
cube_not_found	Parent cube not found.
measure_not_found	Measure not found.

Example: Creating a Measure

The following statements create the SALES_AMOUNT and SALES_QUANTITY measures for the SALES_CUBE cube.

```
execute cwm2_olap_measure.create_measure
     ('JSMITH', 'SALES_CUBE', 'SALES_AMOUNT', 'Sales Amount',
      '$ Sales', 'Dollar Sales');
execute cwm2_olap_measure.create_measure
     ('JSMITH', 'SALES_CUBE', 'SALES_QUANTITY', 'Sales Quantity',
      'Sales Quantity', 'Quantity of Items Sold');
```

24

CWM2_OLAP_METADATA_REFRESH

The CWM2_OLAP_METADATA_REFRESH package provides a procedure that refreshes a set of metadata tables for the OLAP API.

This chapter discusses the following topics:

The OLAP API Metadata Reader Views

The Metadata Reader views present a read API to the OLAP Catalog. These views are structured to facilitate queries by the OLAP API Metadata Reader.

The Metadata Reader views, unlike the OLAP Catalog views described in Chapter 14, "Viewing OLAP Catalog Metadata", are not automatically refreshed when changes are made to the metadata.

The Metadata Reader views have public synonyms with the prefix MRV_OLAP. The union views have public synonyms with the prefix ALL_OLAP2. More than half of the MRV_OLAP views have the same name and column structure as the corresponding ALL_OLAP2 views.

The MRV_OLAP views and the ALL_OLAP2 views rely on separate sets of metadata tables. Whereas the tables that underlie the ALL_OLAP views are populated automatically, the tables that underlie the MRV_OLAP views must be explicitly refreshed whenever changes are made to the metadata.

Summary of CWM2_OLAP_METADATA_REFRESH Subprograms

Table 24–1 CWM2_OLAP_METADATA_REFRESH Subprograms

Subprogram	Description
MR_REFRESH Procedure	Refreshes the metadata tables used by the OLAP API.

MR_REFRESH Procedure

This procedure refreshes the metadata tables that underlie the MRV_OLAP views, as described in "The OLAP API Metadata Reader Views" on page 24-2.

Execute MR_REFRESH as the final statement in any script that creates, drops, or updates OLAP Catalog metadata for the OLAP API.

The MR_REFRESH procedure includes a COMMIT. The updates to the metadata tables are saved permanently in the database.

Syntax

```
MR_REFRESH;
```

25

CWM2_OLAP_PC_TRANSFORM

The CWM2_OLAP_PC_TRANSFORM package contains a procedure for generating a SQL script that creates a solved, level-based dimension table from a parent-child dimension table. .

After running the script and creating the new table, you can define OLAP metadata so that OLAP API applications can access the dimension.

See Also:

- Chapter 4, "Designing Your Database for OLAP" for information about types of data warehouse tables supported by OLAP Catalog metadata.
- Chapter 20, "CWM2_OLAP_HIERARCHY" for information about creating OLAP Catalog metadata for dimension hierarchies.

This chapter discusses the following topics:

Prerequisites

Before running the CWM2_OLAP_PC_TRANSFORM.CREATE_SCRIPT procedure, ensure that the RDBMS is enabled to write to a file. The utl_file_dir parameter must be set to a valid directory, as described in "Initialization Parameters for Oracle OLAP" on page 6-3.

A parent-child dimension table must exist and be accessible to the CWM2_OLAP_PC_TRANSFORM.CREATE_SCRIPT procedure.

Parent-Child Dimensions

A **parent-child dimension table** is one in which the hierarchical relationships are defined by a parent column and a child column. Since the hierarchy is defined by the relationship between the *values* within two columns, a parent-child dimension is sometimes referred to as having a **value-based hierarchy**.

Figure 25–1, "Sample Parent-Child Dimension Table Columns" illustrates the relationships between the values in the child and parent columns. A description column, which is an attribute of the child, is also included.

Figure 25–1 Sample Parent-Child Dimension Table Columns

CHILD	PARENT	DESCRIPTION
World		World
USA	World	United States of America
Northeast	USA	North East Region
Southeast	USA	South East Region
MA	Northeast	Massachusetts
Boston	MA	Boston, MA
Burlington	MA	Burlington, MA
NY	Northeast	New York State
New York City	NY	New York, NY
GA	Southeast	Georgia
Atlanta	GA	Atlanta,GA
Canada	World	Canada

If you choose to create OLAP Catalog metadata to represent a parent-child dimension, set the solved_code for the hierarchy to SOLVED,VALUE_BASED, as described in Chapter 20, "CWM2_OLAP_HIERARCHY".

Note: You can create OLAP Catalog metadata to represent value-based hierarchies, but this type of hierarchy is not accessible to applications that use the OLAP API.

Solved, Level-Based Dimensions

The script generated by `OLAP_PC_TRANSFORM.CREATE_SCRIPT` creates a table that stores the values from the parent-child table in levels.

The resulting level-based dimension table includes the full lineage of every level value in every row. This type of dimension table is **solved**, because the fact table related to this dimension includes embedded totals for all level combinations.

If you want to enable parent-child dimension tables for access by the OLAP API, you must convert them to solved, level-based dimension tables. The OLAP API requires that dimensions have levels and that they include a GID (Grouping ID) column and an Embedded Total (ET) key column. GIDs and ET key columns are described in Example: Creating a Solved, Level-Based Dimension Table.

Figure 25–2, "Sample Solved, Level-Based Dimension Table Columns" illustrates how the parent-child relationships in Figure 25–1 would be represented as solved levels.

Figure 25–2 Sample Solved, Level-Based Dimension Table Columns

TOT_GEOG	COUNTRY	REGION	STATE	CITY	DESCRIPTION
World	USA	Northeast	MA	Boston	Boston, MA
World	USA	Northeast	MA	Burlington	Burlington, MA
World	USA	Northeast	NY	New York City	New York, NY
World	USA	Southeast	GA	Atlanta	Atlanta, GA
World	USA	Northeast	MA		Massachusetts
World	USA	Northeast	NY		New York State
World	USA	Southeast	GA		Georgia
World	USA	Northeast			North East Region
World	USA	Southeast			South East Region
World	USA				United States of America
World	Canada				Canada
World					World

When creating OLAP Catalog metadata to represent a solved, level-based dimension hierarchy, specify a `solved_code` of `SOLVED,LEVEL_BASED`, as described in Chapter 20, "CWM2_OLAP_HIERARCHY".

Example: Creating a Solved, Level-Based Dimension Table

Assuming a parent-child dimension table with the PARENT and CHILD columns shown in Figure 25–1, you could use a command like the following to represent these columns in a solved, level-based dimension table.

```
execute cwm2_olap_pc_transform.create_script
     ('/dat1/scripts/myscripts' ,
     'jsmith' ,
     'input_tbl' ,
     'PARENT' ,
     'CHILD' ,
     'output_tbl' ,
     'jsmith_data');
```

This statement creates a script in the directory `/dat1/scripts/myscripts`. The script will convert the parent-child table `input_tbl` to the solved, level-based table `output_tbl`. Both tables are in the `jsmith_data` tablespace of the `jsmith` schema.

You can run the resulting script with the following command.

```
@create_output_tbl
```

You can view the resulting table with the following command.

```
select * from output_tbl_view
```

The resulting table would look like this.

```
GID SHORT_DESC    LONG_DESC     CHILD1 CHILD2 CHILD3    CHILD4 CHILD5
--- -----------   ------------  ------ -----  --------  ------ -------
0   Boston        Boston        World  USA    Northeast  MA    Boston
0   Burlington    Burlington    World  USA    Northeast  MA    Burlington
0   New York City New York City World  USA    Northeast  NY    New York City
0   Atlanta       Atlanta       World  USA    Southeast  GA    Atlanta
1   MA            MA            World  USA    Northeast  MA
1   NY            MA            World  USA    Northeast  NY
1   GA            GA            World  USA    Southeast  GA
3   Northeast     Northeast     World  USA    Northeast
3   Southeast     Southeast     World  USA    Southeast
7   USA           USA           World  USA
7   Canada        Canada        World  Canada
15  World         World         World
```

Grouping ID Column

The script automatically creates a GID column, as required by the OLAP API. The GID identifies the hierarchy level associated with each row by assigning a zero to each non-null value and a one to each null value in the level columns. The resulting binary number is the value of the GID. For example, a GID of 3 is assigned to the row with the level values World, USA, Northeast, since the three highest levels are assigned zeros and the two lowest levels are assigned ones.

```
CHILD1 CHILD2 CHILD3    CHILD4 CHILD5
------ -----  --------  ------ -------
World  USA    Northeast
0      0      0          1      1
```

Embedded Total Key Column

The script automatically generates columns for long description and short description. If you have columns in the input table that contain this information, you can specify them as parameters to the `CREATE_SCRIPT` procedure.

If you do not specify a column for the short description, the script creates the column and populates it with the lowest-level child value represented in each row. If you do not specify a column for the long description, the script simply replicates the short description.

The ET key column required by the OLAP API is the short description column that is created by default.

Summary of CWM2_OLAP_PC_TRANSFORM Subprograms

Table 25–1 CWM2_OLAP_PC_TRANSFORM

Subprogram	Description
`CREATE_SCRIPT Procedure` on page 25-5	Generates a script that converts a parent-child table to an embedded-total table.

CREATE_SCRIPT Procedure

This procedure generates a script that converts a parent-child dimension table to an embedded-total dimension table.

Syntax

```
CREATE_SCRIPT (
          directory          IN   VARCHAR2,
          schema             IN   VARCHAR2,
          pc_table           IN   VARCHAR2,
          pc_parent          IN   VARCHAR2,
          pc_child           IN   VARCHAR2,
          slb_table          IN   VARCHAR2,
          slb_tablespace     IN   VARCHAR2,
          pc_root            IN   VARCHAR2   DEFAULT  NULL,
          number_of_levels   IN   NUMBER     DEFAULT  NULL,
          level_names        IN   VARCHAR2   DEFAULT  NULL,
          short_description  IN   VARCHAR2   DEFAULT  NULL,
          long_description   IN   VARCHAR2   DEFAULT  NULL,
          attribute_names     IN   VARCHAR2   DEFAULT  NULL);
```

Parameters

Table 25–2 CREATE_SCRIPT Procedure Parameters

Parameter	Description
`directory`	Full path of the directory that will contain the generated script.
`schema`	Schema containing the parent-child table. This schema will also contain the solved, level-based table.
`pc_table`	Name of the parent-child table.
`pc_parent`	Name of the column in `pc_table` that contains the parent values .
`pc_child`	Name of the column in `pc_table` that contains the child values.
`slb_table`	Name of the solved, level-based table that will be created.
`slb_tablespace`	Name of the tablespace where the solved, level-based table will be created.
`pc_root`	One of the following: `null` - Root of the parent-child hierarchy is identified by `null` in the parent column. (default) *`condition`* - Root of the parent-child hierarchy is a condition, for example: `'long_des = "All Countries"'`

Table 25–2 (Cont.) CREATE_SCRIPT Procedure Parameters

Parameter	Description
number_of_levels	One of the following: null - The number of levels in the solved, level-based table will be all the levels of the hierarchy in the parent-child table. (default) *number* - The number of levels to be created in the solved, level-based table.
level_names	One of the following: null - The column names in the solved, level-based table will be the source child column name concatenated with the level number. (default) *list* - A comma-separated list of column names for the solved, level-based table.
short_description	One of the following: null - There is no short description in the parent-child table. The highest level non-null child value in each row of the solved, level-based table will be used as the short description. This constitutes the ET key column (default) *column name* - Name of the column in the parent-child table that contains the short description. This column will be copied from the parent-child table to the solved, level-based table.
long_description	One of the following: null - There is no long description in the parent-child table. The short description will be used. (default) *column name* - Name of the column in the parent-child table that contains the long description. This column will be copied from the parent-child table to the solved, level-based table.
attribute_names	One of the following: null - There are no attributes in the parent-child table. (default) *list* - A comma-separated list of attribute columns in the parent-child table. These columns will be copied from the parent-child table to the solved, level-based table

Usage Notes

1. If a table with the same name as the solved, level-based table already exists, the script will delete it.

2. You can reduce the time required to generate the script by specifying the number of levels in the `number_of_levels` parameter. If you do not specify a value for this parameter, the `CREATE_SCRIPT` procedure calculates all the levels from the parent-child table.
3. To define additional characteristics of the solved, level-based table, you can modify the generated script file before executing it.

26

CWM2_OLAP_TABLE_MAP

The CWM2_OLAP_TABLE_MAP package provides procedures for mapping OLAP metadata entities to columns in your data warehouse dimension tables and fact tables.

See Also: Chapter 13, "Using the OLAP Catalog Metadata APIs"

This chapter discusses the following topics:

- Understanding OLAP Metadata Mapping
- Summary of CWM2_OLAP_TABLE_MAP Subprograms
- Example: Mapping a Dimension
- Example: Mapping a Cube

Understanding OLAP Metadata Mapping

The CWM2_OLAP_TABLE_MAP package provides procedures for linking OLAP metadata entities to columns in fact tables and dimension tables and for establishing the join relationships between a fact table and its associated dimension tables.

Dimension levels and level attributes are mapped to columns in dimension tables. Typically, they are mapped by hierarchy. Measures are mapped to columns in fact tables.

The join relationship between the fact table and dimension tables may specified for unsolved data stored in a single fact table, for solved data stored in one fact table per hierarchy combination, or for solved data stored in a single fact table.

See Also: "Mapping OLAP Metadata" on page 13-4.

Summary of CWM2_OLAP_TABLE_MAP Subprograms

Table 26–1 CWM2_OLAP_TABLE_MAP

Subprogram	Description
MAP_DIMTBL_HIERLEVELATTR Procedure on page 26-3	Maps a hierarchical level attribute to a column in a dimension table.
MAP_DIMTBL_HIERLEVEL Procedure on page 26-5	Maps a hierarchical level to one or more columns in a dimension table.
MAP_DIMTBL_HIERSORTKEY Procedure on page 26-6	Sorts the members of a hierarchy within a column of a dimension table.
MAP_DIMTBL_LEVELATTR Procedure on page 26-7	Maps a non-hierarchical level attribute to a column in a dimension table
MAP_DIMTBL_LEVEL Procedure on page 26-9	Maps a non-hierarchical level to one or more columns in a dimension table.
MAP_FACTTBL_LEVELKEY Procedure on page 26-10	Maps the dimensions of a cube to a fact table.
MAP_FACTTBL_MEASURE Procedure on page 26-12	Maps a measure to a column in a fact table.
REMOVEMAP_DIMTBL_ HIERLEVELATTR Procedure on page 26-13	Removes the mapping of a hierarchical level attribute from a column in a dimension table.

Table 26–1 (Cont.) CWM2_OLAP_TABLE_MAP

Subprogram	Description
REMOVEMAP_DIMTBL_HIERLEVEL Procedure on page 26-15	Removes the mapping of a hierarchical level from one or more columns in a dimension table.
REMOVEMAP_DIMTBL_HIERSORTKEY Procedure on page 26-16	Removes custom sorting criteria associated with columns in a dimension table.
REMOVEMAP_DIMTBL_LEVELATTR Procedure on page 26-17	Removes the mapping of a non-hierarchical level attribute from a column in a dimension table.
REMOVEMAP_DIMTBL_LEVEL Procedure on page 26-18	Removes the mapping of a non-hierarchical level from one or more columns in a dimension table.
REMOVEMAP_FACTTBL_LEVELKEY Procedure on page 26-19	Removes the mapping of a cube's dimensions from a fact table.
REMOVEMAP_FACTTBL_MEASURE Procedure on page 26-20	Removes the mapping of a measure from a column in a fact table.

MAP_DIMTBL_HIERLEVELATTR Procedure

This procedure maps a level attribute to a column in a dimension table.

The attribute being mapped is associated with a level in the context of a hierarchy.

Syntax

```
MAP_DIMTBL_HIERLEVELATTR (
          dimension_owner            IN   VARCHAR2,
          dimension_name             IN   VARCHAR2,
          dimension_attribute_name    IN   VARCHAR2,
          hierarchy_name             IN   VARCHAR2,
          level_name                 IN   VARCHAR2,
          level_attribute_name        IN   VARCHAR2,
          table_owner                IN   VARCHAR2,
          table_name                 IN   VARCHAR2,
          attrcol                     IN   VARCHAR2);
```

Parameters

Table 26–2 MAP_DIMTBL_HIERLEVELATTR Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
dimension_attribute_name	Name of the dimension attribute.
hierarchy_name	Name of the hierarchy.
level_name	Name of the level.
level_attribute_name	Name of the level attribute associated with this level.
table_owner	Owner of the dimension table.
table_name	Name of the dimension table.
attrcol	Column in the dimension table to which this level attribute should be mapped.

Exceptions

Table 26–3 MAP_DIMTBL_HIERLEVELATTR Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the dimension owner and have the OLAP_DBA role.
dimension_not_found	Dimension not found.

Table 26–3 (Cont.) MAP_DIMTBL_HIERLEVELATTR Procedure Exceptions

Exception	Description
hierarchy_not_found	Hierarchy not found.
level_not_found	Level not found.
attribute_not_found	Level attribute not found.
table_not_found	Dimension table not found.
column_not_found	Dimension table column not found.

MAP_DIMTBL_HIERLEVEL Procedure

This procedure maps a level to one or more columns in a dimension table.

The level being mapped is identified within the context of a hierarchy.

Syntax

```
MAP_DIMTBL_HIERLEVEL (
         dimension_owner     IN   VARCHAR2,
         dimension_name      IN   VARCHAR2,
         hierarchy_name      IN   VARCHAR2,
         level_name          IN   VARCHAR2,
         table_owner          IN   VARCHAR2,
         table_name          IN   VARCHAR2,
         keycol               IN   VARCHAR2,
         parentcol           IN   VARCHAR2 DEFAULT NULL);
```

Parameters

Table 26–4 MAP_DIMTBL_HIERLEVEL Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
hierarchy_name	Name of the hierarchy.
level_name	Name of the level.
table_owner	Owner of the dimension table.
table_name	Name of the dimension table.

Table 26–4 (Cont.) MAP_DIMTBL_HIERLEVEL Procedure Parameters

Parameter	Description
keycol	Column in the dimension table to which this level should be mapped. This column will be the key for this level column in the fact table. If the level is stored in more than one column, separate the column names with commas. These columns will be the multicolumn key for these level columns in the fact table.
parentcol	Column that stores the parent level in the hierarchy. If you do not specify this parameter, the level is the root of the hierarchy.

Exceptions

Table 26–5 MAP_DIMTBL_HIERLEVEL Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
dimension_not_found	Dimension not found.
hierarchy_not_found	Hierarchy not found.
level_not_found	Level not found.
table_not_found	Dimension table not found.
column_not_found	Dimension table column not found.

MAP_DIMTBL_HIERSORTKEY Procedure

This procedure specifies how to sort the members of a hierarchy within a column of a dimension table. The column may be the key column or it may be a related attribute column. Custom sorting can specify that the column be sorted in ascending or descending order, with nulls first or nulls last.

Custom sorting information is optional and can be applied at multiple levels of a dimension.

Syntax

```
MAP_DIMTBL_HIERSORTKEY (
         dimension_owner     IN   VARCHAR2,
         dimension_name      IN   VARCHAR2,
         hierarchy_name      IN   VARCHAR2,
         sortcol             IN   VARCHAR2);
```

Parameters

Table 26–6 MAP_DIMTBL_HIERSORTKEY Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
hierarchy_name	Name of the hierarchy.
sortcol	A string specifying how to sort the values stored in a given column of a dimension table. The string specifies the table name, the column name, whether to sort in ascending or descending order, and whether to place nulls first or last. The string should be enclosed in single quotes, and it should be in the following form. TBL:*tableowner.tablename*/COL:*columnname* /ORD:ASC|DSC/NULL:FIRST|LAST

Exceptions

Table 26–7 MAP_DIMTBL_HIERSORTKEY Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
dimension_not_found	Dimension not found.
hierarchy_not_found	Hierarchy not found.

MAP_DIMTBL_LEVELATTR Procedure

This procedure maps a level attribute to a column in a dimension table.

The attribute being mapped is associated with a level that has no hierarchical context. Typically, this level is the only level defined for this dimension.

Syntax

```
MAP_DIMTBL_LEVELATTR (
          dimension_owner              IN   VARCHAR2,
          dimension_name               IN   VARCHAR2,
          dimension_attribute_name     IN   VARCHAR2,
          level_name                   IN   VARCHAR2,
          level_attribute_name          IN   VARCHAR2,
          table_owner                  IN   VARCHAR2,
          table_name                   IN   VARCHAR2,
          attrcol                       IN   VARCHAR2);
```

Parameters

Table 26–8 MAP_DIMTBL_LEVELATTR Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
dimension_attribute_name	Name of the dimension attribute.
level_name	Name of the level.
level_attribute_name	Name of the level attribute associated with this level.
table_owner	Owner of the dimension table.
table_name	Name of the dimension table.
attrcol	Column in the dimension table to which this level attribute should be mapped.

Exceptions

Table 26–9 MAP_DIMTBL_LEVELATTR Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the dimension owner and have the OLAP_DBA role.
dimension_not_found	Dimension not found.
level_not_found	Level not found.
attribute_not_found	Level attribute not found.

Table 26–9 (Cont.) MAP_DIMTBL_LEVELATTR Procedure Exceptions

Exception	Description
table_not_found	Dimension table not found
column_not_found	Dimension table column not found.

MAP_DIMTBL_LEVEL Procedure

This procedure maps a level to one or more columns in a dimension table.

The level being mapped has no hierarchical context. Typically, this level is the only level defined for this dimension.

Syntax

```
MAP_DIMTBL_LEVEL (
         dimension_owner     IN   VARCHAR2,
         dimension_name      IN   VARCHAR2,
         level_name          IN   VARCHAR2,
         table_owner          IN   VARCHAR2,
         table_name          IN   VARCHAR2,
         keycol               IN   VARCHAR2);
```

Parameters

Table 26–10 MAP_DIMTBL_LEVEL Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
level_name	Name of the level.
table_owner	Owner of the dimension table.
table_name	Name of the dimension table.
keycol	Column in the dimension table to which this level should be mapped. This column will be the key for this level column in the fact table. If the level is stored in more than one column, separate the column names with commas. These columns will be the multicolumn key for these level columns in the fact table.

Exceptions

Table 26–11 MAP_DIMTBL_LEVEL Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
dimension_not_found	Dimension not found.
level_not_found	Level not found.
table_not_found	Dimension table not found.
column_not_found	Dimension table column not found.

MAP_FACTTBL_LEVELKEY Procedure

This procedure creates the join relationships between a fact table and a set of dimension tables. A join must be specified for each of the dimensions of the cube. Each dimension is joined in the context of one of its hierarchies.

For example, if you had a cube with three dimensions, and each dimension had only one hierarchy, you could fully map the cube with one call to MAP_FACTTBL_LEVELKEY.

However, if you had a cube with three dimensions, but two of the dimensions each had two hierarchies, you would need to call MAP_FACTTBL_LEVELKEY four times to fully map the cube. For dimensions Dim1, Dim2, and Dim3, where Dim1 and Dim3 each have two hierarchies, you would specify the following mapping strings in each call to MAP_FACTTBL_LEVELKEY, as shown below.

```
Dim1_Hier1, Dim2_Hier, Dim3_Hier1
Dim1_Hier1, Dim2_Hier, Dim3_Hier2
Dim1_Hier2, Dim2_Hier, Dim3_Hier1
Dim1_Hier2, Dim2_Hier, Dim3_Hier2
```

Typically the data for each hierarchy combination would be stored in a separate fact table.

For more information, see"Joining Fact Tables with Dimension Tables" on page 13-4.

Syntax

```
MAP_FACTTBL_LEVELKEY (
          cube_owner          IN   VARCHAR2,
          cube_name           IN   VARCHAR2,
          facttable_owner     IN   VARCHAR2,
          facttable_name      IN   VARCHAR2,
          storetype            IN   VARCHAR2,
          dimkeymap            IN   VARCHAR2,
          dimktype            IN   VARCHAR2 DEFAULT NULL);
```

Parameters

Table 26–12 MAP_FACTTBL_LEVELKEY Procedure Parameters

Parameter	Description
cube_owner	Owner of the cube.
cube_name	Name of the cube.
facttable_owner	Owner of the fact table.
facttable_name	Name of the fact table.
storetype	One of the following: LOWEST LEVEL, for a fact table that stores only lowest level data ET, for a fact table that stores embedded totals in addition to lowest level data ROLLED UP, for an embedded total fact table with key columns for all levels
dimkeymap	A string specifying the mapping for each dimension of the data in the fact table. For each dimension you must specify a hierarchy and the lowest level to be mapped within that hierarchy. Enclose the string in single quotes, and separate each dimension specification with a semicolon. Each dimension specification must be in the following form: DIM:*dimname*/HIER:*hiername*/GID:*columnname*/LVL:*levelname*/COL:*columnname*; This string must also be specified as an argument to the MAP_FACTTBL_MEASURE procedure.
dimktype	This parameter is not currently used.

Exceptions

Table 26–13 MAP_FACTTBL_LEVELKEY Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the dimension owner and have the OLAP_DBA role.
cube_not_found	Cube not found.
fact_table_not_found	Fact table not found.

MAP_FACTTBL_MEASURE Procedure

This procedure maps a measure to a column in a fact table.

Syntax

```
MAP_FACTTBL_MEASURE (
          cube_owner         IN   VARCHAR2,
          cube_name          IN   VARCHAR2,
          measure_name        IN   VARCHAR2,
          facttable_owner    IN   VARCHAR2,
          facttable_name     IN   VARCHAR2,
          column_name         IN   VARCHAR2,
          dimkeymap          IN   VARCHAR2);
```

Parameters

Table 26–14 MAP_FACTTBL_MEASURE Procedure Parameters

Parameter	Description
cube_owner	Owner of the cube.
cube_name	Name of the cube.
measure_name	Name of the measure to be mapped.
facttable_owner	Owner of the fact table.
facttable_name	Name of the fact table.
column_name	Column in the fact table to which the measure will be mapped.

Table 26–14 (Cont.) MAP_FACTTBL_MEASURE Procedure Parameters

Parameter	Description
dimkeymap	A string specifying the mapping for each of the measure's dimensions. For each dimension you must specify a hierarchy and the lowest level to be mapped within that hierarchy. Enclose the string in single quotes, and separate each dimension specification with a semicolon. Each dimension specification must be in the following form: DIM:*dimname*/HIER:*hiername*/GID:*columnname*/LVL:*levelname*/COL:*columnname*; This string must also be specified as an argument to the MAP_FACTTBL_HIERLEVELKEY procedure.

Exceptions

Table 26–15 MAP_FACTTBL_MEASURE Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the dimension owner and have the OLAP_DBA role.
cube_not_found	Cube not found.
fact_table_not_found	Fact table not found.
measure_not_found	Measure not found.
column_not_found	Fact table column not found.

REMOVEMAP_DIMTBL_HIERLEVELATTR Procedure

This procedure removes the relationship between a level attribute and a column in a dimension table. The attribute is identified by the hierarchy that contains its associated level.

Upon successful completion of this procedure, the level attribute is a purely logical metadata entity. It has no data associated with it.

Syntax

```
REMOVEMAP_DIMTBL_HIERLEVELATTR (
          dimension_owner            IN   VARCHAR2,
          dimension_name             IN   VARCHAR2,
          dimension_attribute_name    IN   VARCHAR2,
          hierarchy_name             IN   VARCHAR2,
          level_name                 IN   VARCHAR2,
          level_attribute_name        IN   VARCHAR2);
```

Parameters

Table 26–16 REMOVEMAP_DIMTBL_HIERLEVELATTR Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
dimension_attribute_name	Name of the dimension attribute.
hierarchy_name	Name of the hierarchy.
level_name	Name of the level.
level_attribute_name	Name of the level attribute associated with this level.

Exceptions

Table 26–17 REMOVEMAP_DIMTBL_HIERLEVELATTR Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the dimension owner and have the OLAP_DBA role.
dimension_not_found	Dimension not found.
hierarchy_not_found	Hierarchy not found.
level_not_found	Level not found.
attribute_not_found	Level attribute not found.

REMOVEMAP_DIMTBL_HIERLEVEL Procedure

This procedure removes the relationship between a level of a hierarchy and one or more columns in a dimension table.

Upon successful completion of this procedure, the level is a purely logical metadata entity. It has no data associated with it.

Syntax

```
REMOVEMAP_DIMTBL_HIERLEVEL (
        dimension_owner     IN   VARCHAR2,
        dimension_name      IN   VARCHAR2,
        hierarchy_name      IN   VARCHAR2,
        level_name          IN   VARCHAR2);
```

Parameters

Table 26–18 REMOVEMAP_DIMTBL_HIERLEVEL Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
hierarchy_name	Name of the hierarchy.
level_name	Name of the level.

Exceptions

Table 26–19 REMOVEMAP_DIMTBL_HIERLEVEL Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
dimension_not_found	Dimension not found.
hierarchy_not_found	Hierarchy not found.
level_not_found	Level not found.

REMOVEMAP_DIMTBL_HIERSORTKEY Procedure

This procedure removes custom sorting criteria associated with columns in a dimension table.

Syntax

```
REMOVEMAP_DIMTBL_HIERSORTKEY (
         dimension_owner     IN   VARCHAR2,
         dimension_name      IN   VARCHAR2,
         hierarchy_name      IN   VARCHAR2);
```

Parameters

Table 26–20 REMOVEMAP_DIMTBL_HIERSORTKEY Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
hierarchy_name	Name of the hierarchy.

Exceptions

Table 26–21 REMOVEMAP_DIMTBL_HIERSORTKEY Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
dimension_not_found	Dimension not found.
hierarchy_not_found	Hierarchy not found.

REMOVEMAP_DIMTBL_LEVELATTR Procedure

This procedure removes the relationship between a level attribute and a column in a dimension table.

Upon successful completion of this procedure, the level attribute is a purely logical metadata entity. It has no data associated with it.

Syntax

```
REMOVEMAP_DIMTBL_LEVELATTR (
          dimension_owner             IN   VARCHAR2,
          dimension_name              IN   VARCHAR2,
          dimension_attribute_name    IN   VARCHAR2,
          level_name                  IN   VARCHAR2,
          level_attribute_name         IN   VARCHAR2);
```

Parameters

Table 26–22 REMOVEMAP_DIMTBL_LEVELATTR Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
dimension_attribute_name	Name of the dimension attribute.
level_name	Name of the level.
level_attribute_name	Name of the level attribute associated with this level.

Exceptions

Table 26–23 REMOVEMAP_DIMTBL_LEVELATTR Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the dimension owner and have the OLAP_DBA role.
dimension_not_found	Dimension not found.
level_not_found	Level not found.
attribute_not_found	Level attribute not found.

REMOVEMAP_DIMTBL_LEVEL Procedure

This procedure removes the relationship between a level and one or more columns in a dimension table.

Upon successful completion of this procedure, the level is a purely logical metadata entity. It has no data associated with it.

Syntax

```
REMOVEMAP_DIMTBL_LEVEL (
         dimension_owner     IN   VARCHAR2,
         dimension_name      IN   VARCHAR2,
         level_name          IN   VARCHAR2);
```

Parameters

Table 26–24 REMOVEMAP_DIMTBL_LEVEL Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
level_name	Name of the level.

Exceptions

Table 26–25 REMOVEMAP_DIMTBL_LEVEL Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the owner and have the OLAP_DBA role.
dimension_not_found	Dimension not found.
level_not_found	Level not found.

REMOVEMAP_FACTTBL_LEVELKEY Procedure

This procedure removes the relationship between the key columns in a fact table and the level columns of a dimension hierarchy in a dimension table.

Syntax

```
REMOVEMAP_FACTTBL_LEVELKEY (
          cube_owner        IN   VARCHAR2,
          cube_name         IN   VARCHAR2,
          facttable_owner   IN   VARCHAR2,
          facttable_name    IN   VARCHAR2 DEFAULT );
```

Parameters

Table 26–26 REMOVEMAP_FACTTBL_LEVELKEY Procedure Parameters

Parameter	Description
cube_owner	Owner of the cube.
cube_name	Name of the cube.
facttable_owner	Owner of the fact table.
facttable_name	Name of the fact table.

Exceptions

Table 26–27 REMOVEMAP_FACTTBL_LEVELKEY Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the dimension owner and have the OLAP_DBA role.
cube_not_found	Cube not found.
fact_table_not_found	Fact table not found.

REMOVEMAP_FACTTBL_MEASURE Procedure

This procedure removes the relationship between a measure column in a fact table and a logical measure associated with a cube.

Upon successful completion of this procedure, the measure is a purely logical metadata entity. It has no data associated with it.

Syntax

```
REMOVEMAP_FACTTBL_MEASURE (
          cube_owner        IN   VARCHAR2,
          cube_name         IN   VARCHAR2,
          measure_name       IN   VARCHAR2,
          facttable_owner   IN   VARCHAR2,
          facttable_name    IN   VARCHAR2,
          column_name        IN   VARCHAR2,
          dimkeymap         IN   VARCHAR2);
```

Parameters

Table 26–28 REMOVEMAP_FACTTBL_MEASURE Procedure Parameters

Parameter	Description
cube_owner	Owner of the cube.
cube_name	Name of the cube.
measure_name	Name of the measure.
facttable_owner	Owner of the fact table.
facttable_name	Name of the fact table.
column_name	Column in the fact table to which the measure is mapped.
dimkeymap	A string specifying the mapping for each of the measure's dimensions. For each dimension you must specify a hierarchy and the lowest level to be mapped within that hierarchy. Enclose the string in single quotes, and separate each dimension specification with a semicolon. Each dimension specification must be in the following form: DIM:*dimname*/HIER:*hiername*/GID:*columnname*/LVL:*levelname* This string must also be specified as an argument to the MAP_FACTTBL_HIERLEVELKEY procedure.

Exceptions

Table 26–29 REMOVEMAP_FACTTBL_MEASURE Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the dimension owner and have the OLAP_DBA role.
cube_not_found	Cube not found.
fact_table_not_found	Fact table not found
measure_not_found	Measure not found.
column_not_found	Fact table column not found.

Example: Mapping a Dimension

The following statements map the four levels of the STANDARD hierarchy in the XADEMO.PRODUCT_AW dimension to columns in the XADEMO_AW_VIEW_PRODUCT dimension table. A long description attribute is mapped for each level.

```
execute cwm2_olap_table_map.Map_DimTbl_HierLevel
    ('XADEMO', 'PRODUCT_AW', 'STANDARD', 'L4',
     'XADEMO', 'XADEMO_AW_VIEW_PRODUCT', 'L4', 'L3');
execute cwm2_olap_table_map.Map_DimTbl_HierLevelAttr
    ('XADEMO', 'PRODUCT_AW', 'Long Description', 'STANDARD', 'L4',
     'Long Description', 'XADEMO', 'XADEMO_AW_VIEW_PRODUCT', 'PROD_STD_LLABEL');

execute cwm2_olap_table_map.Map_DimTbl_HierLevel
    ('XADEMO', 'PRODUCT_AW', 'STANDARD', 'L3',
     'XADEMO', 'XADEMO_AW_VIEW_PRODUCT', 'L3', 'L2');
execute cwm2_olap_table_map.Map_DimTbl_HierLevelAttr
    ('XADEMO', 'PRODUCT_AW', 'Long Description', 'STANDARD', 'L3',
     'Long Description', 'XADEMO', 'XADEMO_AW_VIEW_PRODUCT', 'PROD_STD_LLABEL');

execute cwm2_olap_table_map.Map_DimTbl_HierLevel
    ('XADEMO', 'PRODUCT_AW', 'STANDARD', 'L2',
     'XADEMO', 'XADEMO_AW_VIEW_PRODUCT', 'L2', 'L1');
execute cwm2_olap_table_map.Map_DimTbl_HierLevelAttr
    ('XADEMO', 'PRODUCT_AW', 'Long Description', 'STANDARD', 'L2',
     'Long Description', 'XADEMO', 'XADEMO_AW_VIEW_PRODUCT', 'PROD_STD_LLABEL');

execute cwm2_olap_table_map.Map_DimTbl_HierLevel
    ('XADEMO', 'PRODUCT_AW', 'STANDARD', 'L1',
     'XADEMO', 'XADEMO_AW_VIEW_PRODUCT', 'L1', null);
```

```
execute cwm2_olap_table_map.Map_DimTbl_HierLevelAttr
    ('XADEMO', 'PRODUCT_AW', 'Long Description', 'STANDARD', 'L1',
     'Long Description', 'XADEMO', 'XADEMO_AW_VIEW_PRODUCT', 'PROD_STD_LLABEL');
```

Example: Mapping a Cube

The following statement maps the dimension join keys for a cube named ANALYTIC_CUBE_AW in the XADEMO schema. Join key relationships are specified for four dimension/hierarchy combinations:

```
PRODUCT_AW/STANDARD
CHANNEL_AW/STANDARD
TIME_AW/YTD
GEOGRAPHY_AW/CONSOLIDATED.
```

The fact table is called XADEMO_AW_SALES_VIEW_4. It stores lowest level data and embedded totals for all level combinations.

```
execute cwm2_olap_table_map.Map_FactTbl_LevelKey
        ('XADEMO', 'ANALYTIC_CUBE_AW','XADEMO', 'XADEMO_AW_SALES_VIEW_4', 'ET',
         'DIM:XADEMO.PRODUCT_AW/HIER:STANDARD/GID:PRODUCT_GID/LVL:L4/COL:PRODUCT_ET;
          DIM:XADEMO.CHANNEL_AW/HIER:STANDARD/GID:CHANNEL_GID/LVL:STANDARD_1/COL:CHANNEL_ET;
          DIM:XADEMO.TIME_AW/HIER:YTD/GID:TIME_YTD_GID/LVL:L3/COL:TIME_YTD_ET;
          DIM:XADEMO.GEOGRAPHY_AW/HIER:CONSOLIDATED/GID:GEOG_CONS_GID/LVL:L4/COL:GEOG_CONS_ET;');
```

The following statement maps the F.SALES_AW measure to the SALES column in the fact table.

```
execute cwm2_olap_table_map.Map_FactTbl_Measure
        ('XADEMO', 'ANALYTIC_CUBE_AW', 'F.SALES_AW',
         'XADEMO', 'XADEMO_AW_SALES_VIEW_4', 'SALES',
         'DIM:XADEMO.PRODUCT_AW/HIER:STANDARD/LVL:L4/COL:PRODUCT_ET;
          DIM:XADEMO.CHANNEL_AW/HIER:STANDARD/LVL:STANDARD_1/COL:CHANNEL_ET;
          DIM:XADEMO.TIME_AW/HIER:YTD/LVL:L3/COL:TIME_YTD_ET;
          DIM:XADEMO.GEOGRAPHY_AW/HIER:CONSOLIDATED/LVL:L4/COL:GEOG_CONS_ET;');
```

27

CWM2_OLAP_VALIDATE

The. `CWM2_OLAP_VALIDATE` package provides procedures for validating OLAP metadata.

See Also: Chapter 13, "Using the OLAP Catalog Metadata APIs".

This chapter discusses the following topics:

- Summary of CWM2_OLAP_VALIDATE Subprograms

Summary of CWM2_OLAP_VALIDATE Subprograms

Table 27–1 CWM2_OLAP_VALIDATE

Subprogram	Description
VALIDATE_DIMENSION Procedure on page 27-2	Validates a dimension.
VALIDATE_CUBE Procedure on page 27-2	Validates a cube.

VALIDATE_DIMENSION Procedure

This procedure validates an OLAP Catalog dimension.

Syntax

```
VALIDATE_DIMENSION (
         dimension_owner    IN   VARCHAR2,
         dimension_name      IN   VARCHAR2);
```

Parameters

Table 27–2 VALIDATE_DIMENSION Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.

Exceptions

Table 27–3 VALIDATE_DIMENSION Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the dimension owner and have the OLAP_DBA role.
dimension_not_found	Dimension not found.

VALIDATE_CUBE Procedure

This procedure validates a cube.

The validity status of a cube is displayed in the view ALL_OLAP2_CUBES, described on page 14-5.

Syntax

```
VALIDATE_CUBE (
          cube_owner      IN   VARCHAR2,
          cube_name       IN   VARCHAR2);
```

Parameters

Table 27–4 VALIDATE_CUBE Procedure Parameters

Parameter	Description
cube_owner	Owner of the cube.
cube_name	Name of the cube.

Exceptions

Table 27–5 VALIDATE_CUBE Procedure Exceptions

Exception	Description
no_access_privileges	User does not have the necessary privileges. User must be the cube owner and have the OLAP_DBA role.
cube_not_found	Cube not found.

28

CWM_CLASSIFY

The `CWM_CLASSIFY` package implements the OLAP metadata classification system, used to manage measure folders (catalogs) and classify various OLAP metadata entities. It provides procedures for creating measure folders and populating them with measures.

> **Note:** The term **catalog**, when used in the context of the classification system, refers to a measure folder. It should not be confused with the term **OLAP Catalog**, which refers to the collection of tables that implement the OLAP metadata model.

See Also: Chapter 13, "Using the OLAP Catalog Metadata APIs".

This chapter discusses the following topics:

- Understanding the OLAP Classification System
- Summary of CWM_CLASSIFY Subprograms
- Example: Creating a Measure Folder

Understanding the OLAP Classification System

The CWM_CLASSIFY package, implementing the OLAP classification system, is used primarily to manipulate OLAP measure folders.

The CWM_CLASSIFY package is part of CWM, the metadata repository that underlies the OLAP Management feature of Oracle Enterprise Manager. However, the classification system is also used by CWM2, the new metadata repository that is available via the PL/SQL packages whose names start with CWM2_OLAP.

Note: Although the CWM_CLASSIFY package manages measure folders for both metadata management systems, the measures stored within measure folders are specific to either CWM and CWM2. Measures created by Enterprise Manager cannot be accessed by CWM2 procedures, and measures created by CWM2 procedures are not visible within Enterprise Manager.

Summary of CWM_CLASSIFY Subprograms

Table 28–1 CWM_CLASSIFY Subprograms

Subprogram	Description
ADD_CATALOG_ENTITY Procedure on page 28-4	Adds a measure to a measure folder (catalog).
ADD_DESCRIPTOR_ENTITY_TYPE Procedure on page 28-5	Adds a descriptor type to an entity type.
ADD_ENTITY_DESCRIPTOR_USE Procedure on page 28-6	Attaches a descriptor to an entity.
CREATE_CATALOG Function on page 28-7	Creates a measure folder (catalog).
CREATE_DESCRIPTOR Function on page 28-8	Creates a descriptor.
CREATE_DESCRIPTOR_TYPE Procedure on page 28-9	Creates a descriptor type.
DROP_CATALOG Procedure on page 28-10	Drops a measure folder (catalog).
DROP_DESCRIPTOR Procedure on page 28-10	Drops a descriptor.
DROP_DESCRIPTOR_TYPE Procedure on page 28-11	Drops a descriptor type.
LOCK_CATALOG Procedure on page 28-12	Locks a measure folder's metadata for update.
REMOVE_CATALOG_ENTITY Procedure on page 28-12	Removes a measure from a measure folder (catalog).
REMOVE_DESCRIPTOR_ENTITY_TYPE Procedure on page 28-13	Removes a descriptor type from an entity type.
REMOVE_ENTITY_DESCRIPTOR_USE ProcedureExceptions on page 28-14	Removes a descriptor from an entity.
SET_CATALOG_DESCRIPTION Procedure on page 28-16	Sets the description for a measure folder (catalog).
SET_CATALOG_PARENT Procedure on page 28-16	Sets the parent folder for a measure folder (catalog).

ADD_CATALOG_ENTITY Procedure

This procedure adds a measure or a cube to a measure folder.

Syntax

```
ADD_CATALOG_ENTITY (
   catalog_id           IN   NUMBER,
   entity_owner         IN   VARCHAR2,
   entity_name          IN   VARCHAR2,
   child_entity_name    IN   VARCHAR2);
```

Parameters

Table 28–2 ADD_CATALOG_ENTITY Procedure Parameters

Parameter	Description
catalog_id	Name of the measure folder.
entity_owner	Owner of the cube to be added to the measure folder.
entity_name	Name of the cube to be added to the measure folder.
child_entity_name	Name of a measure. If this parameter is specified, the procedure adds this individual measure to the measure folder, instead of adding all of the cube's measures. If this parameter is NULL, the procedure adds all of the cube's measures. The default is NULL.

Exceptions

Table 28–3 ADD_CATALOG_ENTITY Procedure Exceptions

Exception	Description
element_already_exists	This cube is already added to this measure folder.
element_not_found	Cube or measure not found.
catalog_not_found	Measure folder not found.

ADD_DESCRIPTOR_ENTITY_TYPE Procedure

This procedure adds a descriptor type to a metadata entity type.

This procedure is only available to DBAs.

The following pairs of entity types and descriptor types are predefined in the OLAP Catalog.

Entity Type	Descriptor Type
Dimension	Dimension Type
Dimension	Dimension Primary Display Sort Order
Dimension	Dimension Secondary Display Sort Order
Dimension Attribute	Dimension Attribute Descriptor
Dimension Attribute	Time Dimension Attribute Type
Level Attribute	Dimension Attribute Descriptor
Level Attribute	Time Dimension Attribute Type
Level	Total Level
Level	Time Dimension Level Type
Parameter	Parameter Source Type

Syntax

```
ADD_DESCRIPTOR_ENTITY_TYPE (
          descriptor_type     IN   VARCHAR2,
          entity_type         IN   VARCHAR2);
```

Parameters

Table 28–4 ADD_DESCRIPTOR_ENTITY_TYPE Procedure Parameters

Parameter	Description
descriptor_type	Name of the descriptor type. Examples might be dimension type, or attribute type.
entity_type	One of the following types of entities: DIMENSION, CUBE, MEASURE, LEVEL, ATTRIBUTE, HIERARCHY, PARAMETER

ADD_ENTITY_DESCRIPTOR_USE Procedure

This procedure assigns a descriptor to an OLAP metadata entity. An entity may have multiple descriptors.

This procedure is only available to DBAs.

Syntax

```
ADD_ENTITY_DESCRIPTOR_USE (
   descriptor_id                IN   NUMBER,
   entity_type                  IN   VARCHAR2,
   entity_owner                 IN   VARCHAR2,
   entity_name                  IN   VARCHAR2,
   child_entity_name            IN   VARCHAR2,
   secondary_child_entity_name  IN   VARCHAR2);
```

Parameters

Table 28–5 ADD_ENTITY_DESCRIPTOR_USE Procedure Parameters

Parameter	Description
descriptor_id	Identifier of the descriptor.
entity_type	One of the following types of entities: DIMENSION, CUBE, MEASURE, LEVEL, ATTRIBUTE, HIERARCHY, PARAMETER
entity_owner	Owner of the entity.
entity_name	Name of the parent entity. If there is no child entity, this is the name of the entity to which the descriptor should be applied.
child_entity_name	If the entity is a child of entity_name, name of the child entity. If the entity is not a child of another entity, this parameter is NULL. When this parameter is specified and there is no secondary child entity, this is the name of the entity to which the descriptor should be applied. Levels, hierarchies, and dimension attributes are children of dimensions. Measures are children of cubes.
secondary_child_entity_name	Used for specifying level attributes, which are children of levels. If the entity is not a level attribute, this parameter is NULL.

Exceptions

Table 28–6 ADD_ENTITY_DESCRIPTOR_USE Procedure Exceptions

Exception	Description
entity_not_found	Entity not found.
descriptor_undefined	Descriptor value not found in ALL$OLAP_DESCRIPTORS lookup view.

CREATE_CATALOG Function

This function creates a measure folder and returns a unique identifier (NUMBER) for the measure folder.

This identifier may be used to create subfolders of this measure folder.

Syntax

```
CREATE_CATALOG (
   catalog_name            IN   VARCHAR2,
   catalog_description     IN   NUMBER,
   parent_catalog_id       IN   NUMBER);
```

Parameters

Table 28–7 CREATE_CATALOG Function Parameters

Parameter	Description
catalog_name	Name of the measure folder.
catalog_description	Description of the measure folder.
parent_catalog_id	Identifier of the parent measure folder. By default, this parameter is NULL, meaning that the new measure folder is at the root level in the hierarchy.

Exceptions

Table 28–8 CREATE_CATALOG Function Exceptions

Exception	Description
parent_catalog_not_found	Parent measure folder not found.

Table 28–8 (Cont.) CREATE_CATALOG Function Exceptions

Exception	Description
`catalog_already_exists`	A measure folder with this name already exists.
`invalid_name`	Measure folder name may not be empty or null.

CREATE_DESCRIPTOR Function

This function creates a descriptor and returns a unique identifier (`NUMBER`) for the new descriptor.

For each descriptor type, multiple descriptors may be defined. These descriptors are used as a domain to descriptor usages.

This procedure is only available to DBAs.

Syntax

```
CREATE_DESCRIPTOR (
   descriptor_type       IN   VARCHAR2,
   descriptor_value      IN   VARCHAR2,
   description           IN   VARCHAR2);
```

Parameters

Table 28–9 CREATE_DESCRIPTOR Function Parameters

Parameter	Description
`descriptor_type`	Name of the descriptor type. Examples might be `dimension type`, or `attribute type`.
`descriptor_value`	The value for the descriptor. For example, `long description` and `short description` are descriptors of type `attribute type`.
`description`	Description of the descriptor.

Exceptions

Table 28–10 CREATE_DESCRIPTOR Function Exceptions

Exception	Description
`descriptor_type_not_found`	Descriptor type must be created first using the `CREATE_DESCRIPTOR_TYPE` procedure.
`descriptor_already_exists`	This descriptor value already exists for this descriptor type.
`no-access-privileges`	Must have `OLAP_DBA` role.

CREATE_DESCRIPTOR_TYPE Procedure

This procedure creates a descriptor type.

A descriptor type serves as a domain for descriptors, which describe OLAP metadata entities. The descriptor type also specifies the metadata entities to which its descriptors may apply.

This procedure is only available to DBAs.

Syntax

```
CREATE_DESCRIPTOR_TYPE (
      descriptor_type          IN   VARCHAR2);
```

Parameters

Table 28–11 CREATE_DESCRIPTOR_TYPE Procedure Parameters

Parameter	Description
`descriptor_type`	Name of the descriptor type. Examples might be `dimension type`, or `attribute type`.

Exceptions

Table 28–12 CREATE_DESCRIPTOR_TYPE Procedure Exceptions

Exception	Description
`descriptor_type_already_exists`	A descriptor type with this name already exists.
`entity_type_not_allowed`	Entity type is not one of the supported types.

DROP_CATALOG Procedure

This procedure deletes a measure folder. By default, you must delete subfolders before deleting a measure folder. However, if you set the `cascade` parameter, all subfolders are deleted along with the measure folder.

Syntax

```
DROP_CATALOG (
   catalog_id        IN   NUMBER,
   cascade           IN   VARCHAR2);
```

Parameters

Table 28–13 DROP_CATALOG Procedure Parameters

Parameter	Description
`catalog_id`	Identifier of the measure folder.
`cascade`	Whether or not the subfolders should be deleted with the measure folder. Values may be `Y` or `N`. `Y` means that subfolders will be deleted. `N` means that subfolders will not be deleted, and if there are subfolders the measure folder will not be deleted. The default is `N`.

Exceptions

Table 28–14 DROP_CATALOG Procedure Exceptions

Exception	Description
`catalog_has_sub_catalogs`	You must drop the subfolders before deleting the measure folder.
`catalog_not_found`	Measure folder not found.

DROP_DESCRIPTOR Procedure

This procedure drops a descriptor.

Syntax

```
DROP_DESCRIPTOR (
      descriptor_id      IN   NUMBER);
```

Parameters

Table 28–15 DROP_DESCRIPTOR Procedure Parameters

Parameter	Description
descriptor_id	Descriptor identifier

Exceptions

Table 28–16 DROP_DESCRIPTOR Procedure Exceptions

Exception	Description
descriptor_not_found	Descriptor not found.
no_access_privileges	Must have the OLAP_DBA role.

DROP_DESCRIPTOR_TYPE Procedure

This procedure drops a descriptor type.

A descriptor type serves as a domain for descriptors, which describe OLAP metadata entities. The descriptor type also specifies the metadata entities to which its descriptors may apply.

This procedure is granted only to DBA.

Syntax

```
DROP_DESCRIPTOR_TYPE (
      descriptor_type          IN   VARCHAR2);
```

Parameters

Table 28–17 DROP_DESCRIPTOR_TYPE Procedure Parameters

Parameter	Description
descriptor_type	Name of the descriptor type.

Exceptions

Table 28–18 DROP_DESCRIPTOR_TYPE Procedure Exceptions

Exception	Description
descriptor_type_not_found	Descriptor type not found.

LOCK_CATALOG Procedure

This procedure locks the measure folder metadata for update. A database lock is acquired on the row for the measure folder in the CWM model table.

Syntax

```
LOCK_CATALOG (
   catalog_id           IN   NUMBER,
   wait_for_lock        IN   BOOLEAN);
```

Parameters

Table 28–19 LOCK_CATALOG Procedure Parameters

Parameter	Description
catalog_id	Identifier of the measure folder.
wait_for_lock	When true, wait for lock to released if it has already been acquired by another user. The default is false.

Exceptions

Table 28–20 LOCK_CATALOG Procedure Exceptions

Exception	Description
catalog_not_found	Measure folder not found.
failed_to_gain_lock	Failed to acquire lock.
no_access_privileges	User does not have privileges to edit the measure folder. User must be the owner or OLAP_DBA.

REMOVE_CATALOG_ENTITY Procedure

This procedure removes a cube or a measure from a measure folder.

Syntax

```
REMOVE_CATALOG_ENTITY (
   catalog_id           IN   NUMBER,
   entity_owner         IN   VARCHAR2,
   entity_name          IN   VARCHAR2,
   child_entity_name    IN   VARCHAR2);
```

Parameters

Table 28–21 REMOVE_CATALOG_ENTITY Procedure Parameters

Parameter	Description
catalog_id	Identifier of the measure folder.
entity_owner	Owner of the cube to be removed from the measure folder.
entity_name	Name of the cube to be removed from the measure folder.
child_entity_name	Name of a measure. If this parameter is specified, the procedure removes this individual measure from the measure folder, instead of removing all of the cube's measures. If this parameter is NULL, the procedure removes all of the cube's measures. The default is NULL.

Exceptions

Table 28–22 REMOVE_CATALOG_ENTITY Procedure Exceptions

Exception	Description
element_not_found	Entity not found.
catalog_not_found	Measure folder not found.

REMOVE_DESCRIPTOR_ENTITY_TYPE Procedure

This procedure removes a descriptor type from an entity type.

Syntax

```
REMOVE_DESCRIPTOR_ENTITY_TYPE (
          descriptor_type     IN   VARCHAR2,
          entity_type         IN   VARCHAR2);
```

Parameters

Table 28–23 REMOVE_DESCRIPTOR_ENTITY_TYPE Procedure Parameters

Parameter	Description
`descriptor_type`	Name of the descriptor type. Examples might be `dimension type`, or `attribute type`.
`entity_type`	One of the following types of entities: `DIMENSION`, `CUBE`, `MEASURE`, `LEVEL`, `ATTRIBUTE`, `HIERARCHY`, `PARAMETER`

Exceptions

Table 28–24 REMOVE_DESCRIPTOR_ENTITY_TYPE Procedure Exceptions

Exception	Description
`entity_not_found`	Entity not found.

REMOVE_ENTITY_DESCRIPTOR_USE Procedure

This procedure removes a descriptor from an OLAP metadata entity.

Syntax

```
REMOVE_ENTITY_DESCRIPTOR_USE (
   descriptor_id                  IN   NUMBER,
   entity_type                    IN   VARCHAR2,
   entity_owner                   IN   VARCHAR2,
   entity_name                    IN   VARCHAR2,
   child_entity_name              IN   VARCHAR2,
   secondary_child_entity_name    IN   VARCHAR2);
```

Parameters

Table 28–25 REMOVE_ENTITY_DESCRIPTOR_USE Procedure Parameters

Parameter	Description
`descriptor_id`	Identifier of the descriptor.
`entity_type`	One of the following types of entities: `DIMENSION, CUBE, MEASURE, LEVEL, ATTRIBUTE, HIERARCHY, PARAMETER`
`entity_owner`	Owner of the entity.
`entity_name`	Name of the parent entity. If there is no child entity, this is the name of the entity from which the descriptor should be removed.
`child_entity_name`	If the entity is a child of `entity_name`, name of the child entity. If the entity is not a child of another entity, this parameter is `NULL`. When this parameter is specified and there is no secondary child entity, this is the name of the entity from which the descriptor should be removed. Levels, hierarchies, and dimension attributes are children of dimensions. Measures are children of cubes.
`secondary_child_entity_name`	Used for specifying level attributes, which are children of levels. If the entity is not a level attribute, this parameter is `NULL`.

Exceptions

Table 28–26 REMOVE_ENTITY_DESCRIPTOR_USE Procedure Exceptions

Exception	Description
`entity_not_found`	Entity not found.

SET_CATALOG_DESCRIPTION Procedure

This procedure sets the description of a measure folder.

Syntax

```
SET_CATALOG_DESCRIPTION (
   catalog_id              IN   NUMBER,
   catalog_description     IN   VARCHAR2);
```

Parameters

Table 28–27 SET_CATALOG_DESCRIPTION Procedure Parameters

Parameter	Description
catalog_id	Identifier of the measure folder.
catalog_description	Description of the measure folder.

Exceptions

Table 28–28 SET_CATALOG_DESCRIPTION Procedure Exceptions

Exception	Description
catalog_not_found	Measure folder not found.

SET_CATALOG_PARENT Procedure

This procedure changes the parent folder of an existing measure folder.

Syntax

```
SET_CATALOG_PARENT (
   catalog_id              IN   NUMBER,
   parent_catalog_id       IN   NUMBER);
```

Parameters

Table 28–29 SET_CATALOG_PARENT Procedure Parameters

Parameter	Description
catalog_id	Identifier of the measure folder.

Table 28–29 (Cont.) SET_CATALOG_PARENT Procedure Parameters

Parameter	Description
parent_catalog_id	Identifier of the parent measure folder.

Exceptions

Table 28–30 SET_CATALOG_PARENT Procedure Exceptions

Exception	Description
parent_catalog_not_found	Parent measure folder not found.
catalog_not_found	Measure folder not found.
circular_dependency	Cannot add the measure folder at this position in the hierarchy. The parent is already a child of the measure folder.

Example: Creating a Measure Folder

The following statements create a measure folder called PHARMACEUTICALS and add the measure SALES_AMOUNT from SALES_CUBE to it. The measure folder is at the root level.

```
execute folder_ID = cwm_classify.create_catalog
     ('PHARMACEUTICALS', 'Pharmaceutical Sales and Planning');
execute cwm_classify.add_catalog_entity
     ('folder_ID', 'JSMITH', 'SALES_CUBE', 'SALES_AMOUNT');
```

Part V

OLAP API Materialized View Reference

Part V explains how to create materialized views for queries for aggregate data from the OLAP API.

This part contains the following chapters:

29

Creating Dimension Materialized Views

This chapter explains how to create dimension materialized views for the OLAP API.

> **See Also:** Chapter 10, "Creating Materialized Views for the OLAP API".

This chapter contains the following topics:

Creating Materialized Views for Dimensions

You can use OLAP Summary Advisor or the DBMS_ODM PL/SQL package to create dimension materialized views. When you use OLAP Summary Advisor, the dimension materialized views are automatically created along with the fact materialized views for a CWM cube. When you use the DBMS_ODM package, you must call the CREATEDIMMV_GS procedure to create dimension materialized views.

The syntax of the CREATE MATERIALIZED VIEW statement is the same whether generated by OLAP Summary Advisor or the DBMS_ODM package.

See Also:

- "Dimension Materialized Views" on page 10-4.
- "Using the DBMS_ODM Package" on page 30-2.
- "Using the OLAP Summary Advisor Wizard" on page 31-2.

Statistics and Bitmap Indexes

The scripts for creating dimension materialized views, whether generated by OLAP Summary Advisor or DBMS_ODM, include syntax for gathering statistics and creating bitmap indexes.

Statistics

Statistics are required by the optimizer in order to maximize query performance at runtime.

The following SQL statements analyze a materialized view and generate the needed information.

```
ANALYZE TABLE mv_name COMPUTE 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 ;
```

For more information about the ANALYZE TABLE statement, refer to the *Oracle9i SQL Reference*. For more information about the DBMS_STATS package, refer to the *Oracle9i Supplied PL/SQL Packages and Types Reference*.

Bitmap Indexes

Bitmap indexes optimize the performance of materialized views at runtime. Dimension materialized views for the OLAP API include bitmap indexes for all columns that contain dimension values.

The following SQL statements create bitmap indexes.

```
CREATE BITMAP INDEX index_name ON mv_name (mv_colname )
TABLESPACE tblspace_name
PCTFREE 0
COMPUTE STATISTICS
LOCAL
NOLOGGING;
```

The CREATE Statement for a Dimension Materialized View

The following example shows the basic structure of the SQL statements generated by OLAP Summary Advisor or DBMS_ODM to create a dimension materialized view for the OLAP API.

The SELECT statement contains a COUNT(*) function, a GROUPING_ID function, MAX aggregate functions, and a ROLLUP function. The following example shows the basic syntax.

```
CREATE MATERIALIZED VIEW mv_name
PARTITION BY RANGE (gid)
   (partition values less than(1) ,
    partition values less than(3) ,
     .
     .
     .
    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_cols) gid,
   MAX(attribute_col1)
   .
   .
   MAX(attribute_coln)
   level_cols
```

```
FROM
   dimension_table
GROUP BY level1, ROLLUP(level2, ..., leveln)
```

where:

mv_name is the name of the materialized view. The name is derived from the names of the dimension table and the hierarchy.

level_cols are the names of columns in the dimension table that contain data for the levels of the hierarchy, beginning with the most aggregate (*level1*) and ending with the least aggregate (*leveln*).

attribute_col is the name of a column defined as an attribute. All columns defined as attributes should be listed in a MAX function.

dimension_table is the name of the dimension table whose columns are being aggregated to create the materialized view.

level1 is the highest level of aggregation. Note that *level1* is excluded from the ROLLUP list.

leveln is the lowest level of aggregation or "leaf node", which is also the key column.

Sample Script for the TIMES_DIM Dimension

The following sample script creates materialized views for the TIMES_DIM dimension in the SH schema. This script could result from running OLAP Summary Advisor or from invoking the DBMS_ODM.CREATEDIMMV_GS procedure.

The script creates two materialized views: one for the CAL_ROLLUP hierarchy, and one for the FIS_ROLLUP hierarchy

```
CREATE materialized view  TIMES_CAL_R_OLAP
partition by range (gid) (
partition values less than(1),
partition values less than(3),
partition values less than(7),
partition values less than(MAXVALUE))
TABLESPACE SH_DATABUILD IMMEDIATE
USING NO INDEX
REFRESH FORCE
ENABLE QUERY REWRITE
AS
SELECT
   COUNT(*) COUNT_STAR,
   GROUPING_ID( TIMES.CALENDAR_YEAR, TIMES.CALENDAR_QUARTER_DESC,
```

```
                TIMES.CALENDAR_MONTH_DESC,  TIMES.TIME_ID) gid,
   max(TIMES.CALENDAR_YEAR) CALENDAR_YEAR_AR,
   max(TIMES.END_OF_CAL_YEAR) END_OF_CAL_YEAR_AR,
   max(TIMES.DAYS_IN_CAL_YEAR) DAYS_IN_CAL_YEAR_AR,
   max(TIMES.CALENDAR_QUARTER_DESC) CALENDAR_QUARTER_DESC_AR,
   max(TIMES.END_OF_CAL_QUARTER) END_OF_CAL_QUARTER_AR,
   max(TIMES.DAYS_IN_CAL_QUARTER) DAYS_IN_CAL_QUARTER_AR,
   max(TIMES.CALENDAR_QUARTER_NUMBER) CALENDAR_QUARTER_NUMBER_AR,
   max(TIMES.CALENDAR_MONTH_DESC) CALENDAR_MONTH_DESC_AR,
   max(TIMES.END_OF_CAL_MONTH) END_OF_CAL_MONTH_AR,
   max(TIMES.DAYS_IN_CAL_MONTH) DAYS_IN_CAL_MONTH_AR,
   max(TIMES.CALENDAR_MONTH_NAME) CALENDAR_MONTH_NAME_AR,
   max(TIMES.CALENDAR_MONTH_NUMBER) CALENDAR_MONTH_NUMBER_AR,
   max(TIMES.DAY_NUMBER_IN_WEEK) DAY_NUMBER_IN_WEEK_AR,
   max(TIMES.CALENDAR_WEEK_NUMBER) CALENDAR_WEEK_NUMBER_AR,
   max(TIMES.DAY_NUMBER_IN_MONTH) DAY_NUMBER_IN_MONTH_AR,
   max(TIMES.DAY_NAME) DAY_NAME_AR,
   TIMES.CALENDAR_YEAR CALENDAR_YEAR,
   TIMES.CALENDAR_QUARTER_DESC CALENDAR_QUARTER_DESC,
   TIMES.CALENDAR_MONTH_DESC CALENDAR_MONTH_DESC,
   TIMES.TIME_ID TIME_ID
FROM
   SH.TIMES  TIMES
GROUP BY
   TIMES.CALENDAR_YEAR ,
   ROLLUP(TIMES.CALENDAR_QUARTER_DESC,TIMES.CALENDAR_MONTH_DESC,TIMES.TIME_ID):

execute dbms_stats.gather_table_stats ('SH', 'TIMES_CAL_R_OLAP', degree=>
   dbms_stats.default_degree,method_opt=>'for all columns size skewonly') ;
ALTER TABLE TIMES_CAL_R_OLAP MINIMIZE RECORDS_PER_BLOCK ;

CREATE BITMAP INDEX MV_CALENDAR_QUARTER_DESCCA_BI2 ON TIMES_CAL_R_OLAP
   (CALENDAR_QUARTER_DESC)
TABLESPACE SH_IDX
PCTFREE 0
COMPUTE STATISTICS
LOCAL
NOLOGGING;

CREATE BITMAP INDEX MV_CALENDAR_MONTH_DESCCA_BI3 ON TIMES_CAL_R_OLAP
   (CALENDAR_MONTH_DESC)
TABLESPACE SH_IDX
PCTFREE 0
```

```
COMPUTE STATISTICS
LOCAL
NOLOGGING;

CREATE BITMAP INDEX MV_TIME_IDCA_BI4 ON TIMES_CAL_R_OLAP
   (TIME_ID)
TABLESPACE SH_IDX
PCTFREE 0
COMPUTE STATISTICS
LOCAL
NOLOGGING;

CREATE BITMAP INDEX MV_GID_CA_BI_4 ON TIMES_CAL_R_OLAP
   (gid)
TABLESPACE SH_IDX
PCTFREE 0
COMPUTE STATISTICS
LOCAL
NOLOGGING;

CREATE BITMAP INDEX MV_TIMES_CAL_R_OLAP_PREL_FI ON TIMES_CAL_R_OLAP
   ( (CASE GID
          WHEN(7) THEN NULL
          WHEN(3) THEN TO_CHAR( CALENDAR_YEAR)
          WHEN(1) THEN TO_CHAR( CALENDAR_QUARTER_DESC)
          ELSE TO_CHAR( CALENDAR_MONTH_DESC) END) )
TABLESPACE SH_IDX
PCTFREE 0
COMPUTE STATISTICS
LOCAL
NOLOGGING;

CREATE BITMAP INDEX MV_TIMES_CAL_R_OLAP_ET_FI ON TIMES_CAL_R_OLAP
   ( (CASE GID
          WHEN(7) THEN TO_CHAR( CALENDAR_YEAR)
          WHEN(3) THEN TO_CHAR( CALENDAR_QUARTER_DESC)
          WHEN(1) THEN TO_CHAR( CALENDAR_MONTH_DESC)
          ELSE TO_CHAR( TIME_ID) END) )
TABLESPACE SH_IDX
PCTFREE 0
COMPUTE STATISTICS
LOCAL
NOLOGGING;
```

```
execute dbms_stats.gather_table_stats('SH', 'TIMES_CAL_R_OLAP',
   degree=>dbms_stats.default_degree, estimate_percent=>
   dbms_stats.auto_sample_size, method_opt=>'for all hidden columns size 254') ;

create materialized view  TIMES_FIS_R_OLAP
partition by range (gid) (
partition values less than(1),
partition values less than(3),
partition values less than(7),
partition values less than(15),
partition values less than(MAXVALUE))
TABLESPACE SH_DATA
BUILD IMMEDIATE
USING NO INDEX
REFRESH FORCE
ENABLE QUERY REWRITE
AS
SELECT
   COUNT(*) COUNT_STAR,
   GROUPING_ID( TIMES.FISCAL_YEAR,
   TIMES.FISCAL_QUARTER_DESC,
   TIMES.FISCAL_MONTH_DESC,
   TIMES.WEEK_ENDING_DAY,
   TIMES.TIME_ID) gid,
   max(TIMES.FISCAL_YEAR) FISCAL_YEAR_AR,
   max(TIMES.END_OF_FIS_YEAR) END_OF_FIS_YEAR_AR,
   max(TIMES.DAYS_IN_FIS_YEAR) DAYS_IN_FIS_YEAR_AR,
   max(TIMES.FISCAL_QUARTER_DESC) FISCAL_QUARTER_DESC_AR,
   max(TIMES.END_OF_FIS_QUARTER) END_OF_FIS_QUARTER_AR,
   max(TIMES.DAYS_IN_FIS_QUARTER) DAYS_IN_FIS_QUARTER_AR,
   max(TIMES.FISCAL_QUARTER_NUMBER) FISCAL_QUARTER_NUMBER_AR,
   max(TIMES.FISCAL_MONTH_DESC) FISCAL_MONTH_DESC_AR,
   max(TIMES.END_OF_FIS_MONTH) END_OF_FIS_MONTH_AR,
   max(TIMES.DAYS_IN_FIS_MONTH) DAYS_IN_FIS_MONTH_AR,
   max(TIMES.FISCAL_MONTH_NAME) FISCAL_MONTH_NAME_AR,
   max(TIMES.FISCAL_MONTH_NUMBER) FISCAL_MONTH_NUMBER_AR,
   max(TIMES.WEEK_ENDING_DAY) WEEK_ENDING_DAY_AR,
   max(TIMES.FISCAL_WEEK_NUMBER) FISCAL_WEEK_NUMBER_AR,
   max(TIMES.DAY_NUMBER_IN_WEEK) DAY_NUMBER_IN_WEEK_AR,
   max(TIMES.CALENDAR_WEEK_NUMBER) CALENDAR_WEEK_NUMBER_AR,
   max(TIMES.DAY_NUMBER_IN_MONTH) DAY_NUMBER_IN_MONTH_AR,
   max(TIMES.DAY_NAME) DAY_NAME_AR,
   TIMES.FISCAL_YEAR FISCAL_YEAR,
   TIMES.FISCAL_QUARTER_DESC FISCAL_QUARTER_DESC,
   TIMES.FISCAL_MONTH_DESC FISCAL_MONTH_DESC,
```

```
   TIMES.WEEK_ENDING_DAY WEEK_ENDING_DAY,
   TIMES.TIME_ID TIME_ID
FROM
   SH.TIMES  TIMES
GROUP BY
   TIMES.FISCAL_YEAR , ROLLUP( TIMES.FISCAL_QUARTER_DESC ,
      TIMES.FISCAL_MONTH_DESC , TIMES.WEEK_ENDING_DAY , TIMES.TIME_ID );

execute dbms_stats.gather_table_stats('SH', 'TIMES_FIS_R_OLAP',
   degree=>dbms_stats.default_degree,method_opt=>
   'for all columns size skewonly') ;
ALTER TABLE TIMES_FIS_R_OLAP MINIMIZE RECORDS_PER_BLOCK ;

CREATE BITMAP INDEX MV_FISCAL_QUARTER_DESCFI_BI8 ON TIMES_FIS_R_OLAP
   (FISCAL_QUARTER_DESC)
TABLESPACE SH_IDX
PCTFREE 0
COMPUTE STATISTICS
LOCAL
NOLOGGING;

CREATE BITMAP INDEX MV_FISCAL_MONTH_DESCFI_BI12 ON TIMES_FIS_R_OLAP
   (FISCAL_MONTH_DESC)
TABLESPACE SH_IDX
PCTFREE 0
COMPUTE STATISTICS
LOCAL
NOLOGGING;

CREATE BITMAP INDEX MV_WEEK_ENDING_DAYFI_BI16 ON TIMES_FIS_R_OLAP
   (WEEK_ENDING_DAY)
TABLESPACE SH_IDX
PCTFREE 0
COMPUTE STATISTICS
LOCAL
NOLOGGING;

CREATE BITMAP INDEX MV_TIME_IDFI_BI20 ON TIMES_FIS_R_OLAP
   (TIME_ID)
TABLESPACE SH_IDX
PCTFREE 0
COMPUTE STATISTICS
LOCAL
NOLOGGING;
```

```
CREATE BITMAP INDEX MV_GID_FI_BI_20 ON TIMES_FIS_R_OLAP
   (gid)
TABLESPACE SH_IDX
PCTFREE 0
COMPUTE STATISTICS
LOCAL
NOLOGGING;

CREATE BITMAP INDEX MV_TIMES_FIS_R_OLAP_PREL_FI ON TIMES_FIS_R_OLAP
( (CASE GID
          WHEN(15) THEN NULL
          WHEN(7) THEN TO_CHAR( FISCAL_YEAR)
          WHEN(3) THEN TO_CHAR( FISCAL_QUARTER_DESC)
          WHEN(1) THEN TO_CHAR( FISCAL_MONTH_DESC)
          ELSE TO_CHAR( WEEK_ENDING_DAY) END) )
TABLESPACE SH_IDX
PCTFREE 0
COMPUTE STATISTICS
LOCAL
NOLOGGING;

CREATE BITMAP INDEX MV_TIMES_FIS_R_OLAP_ET_FI ON TIMES_FIS_R_OLAP
( (CASE GID
          WHEN(15) THEN TO_CHAR( FISCAL_YEAR)
          WHEN(7) THEN TO_CHAR( FISCAL_QUARTER_DESC)
          WHEN(3) THEN TO_CHAR( FISCAL_MONTH_DESC)
          WHEN(1) THEN TO_CHAR( WEEK_ENDING_DAY)
          ELSE TO_CHAR( TIME_ID) END) )
TABLESPACE SH_IDX
PCTFREE 0
COMPUTE STATISTICS
LOCAL
NOLOGGING;

execute dbms_stats.gather_table_stats('SH', 'TIMES_FIS_R_OLAP',
   degree=>dbms_stats.default_degree, estimate_percent=>
   dbms_stats.auto_sample_size, method_opt=>'for all hidden columns size 254') ;
```

Table Structure of Sample TIMES_DIM Dimension Materialized View

The following table identifies the columns of the materialized view for the Times dimension CAL_ROLLUP hierarchy.

Column Name	Datatype	Description
COUNT_STAR	NUMBER	The total number of rows.
GID	NUMBER	The grouping IDs for the remaining level columns. Created by the GROUPING_ID function to identify whether a level has a value that should be included in the aggregation. A zero (0) indicates that the cell contains a value that should be included; a one (1) indicates that it is null or should not be included in the aggregation.
CALENDAR_YEAR_AR	DATE	Calendar year attribute.
END_OF_CAL_YEAR_AR	DATE	End date attribute for year level.
DAYS_IN_CAL_YEAR_AR	NUMBER	Time span attribute for year level.
CALENDAR_QUARTER_ DESC_AR	VARCHAR2	Description attribute for quarter level.
END_OF_CAL_QUARTER_AR	DATE	End date attribute for quarter level.
DAYS_IN_CAL_QUARTER_ AR	NUMBER	Time span attribute for quarter level.
CALENDAR_QUARTER_ NUMBER_AR	NUMBER	Number of quarters.
CALENDAR_MONTH_DESC_ AR	VARCHAR2	Description attribute for month level.
END_OF_CAL_MONTH_AR	DATE	End date attribute for month level.
DAYS_IN_CAL_MONTH_AR	NUMBER	Time span attribute for month level.
CALENDAR_MONTH_NAME_ AR	VARCHAR2	Name attribute for month level.
CALENDAR_MONTH_ NUMBER_AR	NUMBER	Number of months.
DAY_NUMBER_IN_WEEK_AR	NUMBER	Number of days in a week.

Column Name	Datatype	Description
CALENDAR_WEEK_NUMBER_AR	NUMBER	Number of weeks.
DAY_NUMBER_IN_MONTH_AR	NUMBER	Number of days in a month.
DAY_NAME_AR	VARCHAR2	Name attribute for day level.
CALENDAR_YEAR	NUMBER	Year level of calendar hierarchy.
CALENDAR_QUARTER_DESC	VARCHAR2	Quarter level of calendar hierarchy.
CALENDAR_MONTH_DESC	VARCHAR2	Month level of calendar hierarchy.
TIME_ID	DATE	The primary key in the dimension table. The "leaf node" in which the lowest level of data is stored.

30

Creating Fact Materialized Views With DBMS_ODM

This chapter explains how to use the `DBMS_ODM` package to create materialized views with grouping sets for the OLAP API.

> **See Also:** Chapter 10, "Creating Materialized Views for the OLAP API".

This chapter contains the following topics:

- Using the DBMS_ODM Package
- Partitioning, Statistics, and Indexes
- Sample Script for the COST Cube
- Summary of DBMS_ODM Subprograms

Using the DBMS_ODM Package

The procedures in the OLAP Data Management package, DBMS_ODM, generate scripts that create materialized views in grouping set form for fact tables. Each script generates a single MV containing all hierarchy combinations for a CWM2 cube.

The procedures in DBMS_ODM generate scripts that create materialized views, bitmap indexes, and partitions. 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.

See Also:

- "Fact Materialized Views".
- "Choosing the Right Format for Materialized Views".

Procedure: Create and Run Scripts to Generate Grouping Set Materialized Views

Follow these steps to create a grouping set materialized view for a cube:

1. Create and map a valid CWM2 cube as described in Chapter 17, "CWM2_OLAP_CUBE".
1. Enable your database to write the scripts to a file by setting the UTL_FILE_DIR parameter to a valid directory, as described in "Initialization Parameters for Oracle OLAP" on page 6-3.
2. Log into SQL*Plus using the identity of the metadata owner.
3. Delete any materialized views that currently exist for the cube.
4. Use the following three step procedure to create a script to generate a grouping set materialized view for the cube:
   a. Execute DBMS_ODM.CREATEDIMLEVTUPLE to create the table sys.olaptablevels. This table lists all the dimensions of the cube and all of the levels of each dimension.

      By default, all the levels of all the dimensions are selected for inclusion in the materialized view. You can edit the table to deselect any levels that you do not want to include.

   b. Execute DBMS_ODM.CREATECUBELEVELTUPLE to create the table sys.olaptableveltuples. This table lists all of the level combinations that will be included in the materialized view. This table is derived from the table created in the previous step.

By default, all the levels combinations are selected for inclusion in the materialized view. You can edit the table to deselect any level combinations that you do not want to include.

c. Execute DBMS_ODM.CREATEFACTMV_GS to create the script.

For example, in the Sales History sample schema, you would create a script for COST_CUBE and a script for SALES_CUBE.

5. Optionally, edit the script using any text editor.
6. Run the scripts in SQL*Plus, using commands such as the following:

```
@/users/oracle/OraHome1/olap/mvscript.sql;
```

See Also: "Summary of DBMS_ODM Subprograms" on page 30-11 for the syntax of the procedures in the DMBS_ODM package.

Partitioning, Statistics, and Indexes

The scripts generated by DBMS_ODM.CREATEFACTMV_GS include syntax for partitioning, gathering statistics, and creating bitmap indexes.

Partitioning

Partitioning can have a significant impact upon query performance. You may want to customize the partitioning of fact materialized views before running the scripts generated by DBMS_ODM.CREATEFACTMV_GS.

By default, partitioning is based on grouping IDs since most queries are based on levels. A grouping ID uniquely identifies one level combination per partition (such as CALENDAR_YEAR and PROD_TOTAL).

Statistics

Statistics are required by the optimizer in order to maximize query performance at runtime.

The following SQL statements analyze a materialized view and generate the needed information.

```
ANALYZE TABLE mv_name COMPUTE 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 ;
```

For more information about the ANALYZE TABLE statement, refer to the*Oracle9i SQL Reference*. For more information about the DBMS_STATS package, refer to the *Oracle9i Supplied PL/SQL Packages and Types Reference*.

Bitmap Indexes

Bitmap indexes optimize the performance of materialized views at runtime. Fact materialized views for the OLAP API include bitmap indexes for all columns that contain dimension values.

The following SQL statements create bitmap indexes.

```
CREATE BITMAP INDEX index_name ON mv_name (mv_colname )
TABLESPACE tblspace_name
PCTFREE 0
COMPUTE STATISTICS
LOCAL
NOLOGGING;
```

Sample Script for the COST Cube

The following sample script, generated by DBMS_ODM.CREATEFACTMV_GS, creates a materialized view in grouping set form for the COST_CUBE cube, which is mapped to the COSTS fact table in the SH schema.

This script contains all level combinations for all hierarchies. To deselect levels and level combinations, edit the tables generated by the CREATEDIMLEVTUPLE Procedure and the CREATECUBELEVELTUPLE Procedure before invoking CREATEFACTMV_GS Procedure.

```
create materialized view
COST_CUBE_2_OLAP
partition by range (gid) (
partition values less than(1),
partition values less than(62),
partition values less than(126),
partition values less than(254),
partition values less than(450),
partition values less than(454),
partition values less than(462),
partition values less than(478),
partition values less than(512),
partition values less than(574),
partition values less than(638),
```

```
partition values less than(766),
partition values less than(962),
partition values less than(966),
partition values less than(974),
partition values less than(990),
partition values less than(1536),
partition values less than(1598),
partition values less than(1662),
partition values less than(1790),
partition values less than(1986),
partition values less than(1990),
partition values less than(1998),
partition values less than(2014),
partition values less than(3584),
partition values less than(3646),
partition values less than(3710),
partition values less than(3838),
partition values less than(4034),
partition values less than(4038),
partition values less than(4046),
partition values less than(4062),
partition values less than(MAXVALUE))
pctfree 5 pctused 40
build immediate
using no index
refresh force
enable query rewrite
AS
SELECT
   GROUPING_ID(PRODUCTS.PROD_TOTAL, PRODUCTS.PROD_CATEGORY,
   PRODUCTS.PROD_SUBCATEGORY, PRODUCTS.PROD_ID,
   TIMES.CALENDAR_YEAR, TIMES.CALENDAR_QUARTER_DESC,
   TIMES.CALENDAR_MONTH_DESC, TIMES.FISCAL_YEAR,
   TIMES.FISCAL_QUARTER_DESC, TIMES.FISCAL_MONTH_DESC,
   TIMES.WEEK_ENDING_DAY, TIMES.TIME_ID) gid,
   SUM(COSTS.UNIT_COST) SUM_OF_UNIT_COST,
   SUM(COSTS.UNIT_PRICE) SUM_OF_UNIT_PRICE,
   COUNT(*) COUNT_OF_STAR,
   PRODUCTS.PROD_TOTAL PROD_TOTAL_77,
   PRODUCTS.PROD_CATEGORY PROD_CATEGORY_78,
   PRODUCTS.PROD_SUBCATEGORY PROD_SUBCATEGORY_79,
   PRODUCTS.PROD_ID PROD_ID_80,
   TIMES.CALENDAR_YEAR CALENDAR_YEAR_169,
   TIMES.CALENDAR_QUARTER_DESC CALENDAR_QUARTER_DESC_170,
   TIMES.CALENDAR_MONTH_DESC CALENDAR_MONTH_DESC_171,
```

```
   TIMES.FISCAL_YEAR FISCAL_YEAR_172,
   TIMES.FISCAL_QUARTER_DESC FISCAL_QUARTER_DESC_173,
   TIMES.FISCAL_MONTH_DESC FISCAL_MONTH_DESC_174,
   TIMES.WEEK_ENDING_DAY WEEK_ENDING_DAY_175,
   TIMES.TIME_ID TIME_ID_176
FROM
   SH.PRODUCTS PRODUCTS,
   SH.TIMES TIMES,
   SH.COSTS COSTS
WHERE
   (TIMES.TIME_ID = COSTS.TIME_ID) AND
   (PRODUCTS.PROD_ID = COSTS.PROD_ID)
    GROUP BY GROUPING SETS

      ( (PRODUCTS.PROD_TOTAL , PRODUCTS.PROD_CATEGORY ,
         PRODUCTS.PROD_SUBCATEGORY , PRODUCTS.PROD_ID , TIMES.CALENDAR_YEAR ,
         TIMES.CALENDAR_QUARTER_DESC , TIMES.CALENDAR_MONTH_DESC ,
         TIMES.FISCAL_YEAR , TIMES.FISCAL_QUARTER_DESC ,
         TIMES.FISCAL_MONTH_DESC , TIMES.WEEK_ENDING_DAY , TIMES.TIME_ID ),

        (PRODUCTS.PROD_TOTAL , PRODUCTS.PROD_CATEGORY ,
         PRODUCTS.PROD_SUBCATEGORY , PRODUCTS.PROD_ID , TIMES.FISCAL_YEAR ,
         TIMES.FISCAL_QUARTER_DESC , TIMES.FISCAL_MONTH_DESC ,
         TIMES.WEEK_ENDING_DAY ),

        (PRODUCTS.PROD_TOTAL , PRODUCTS.PROD_CATEGORY ,
         PRODUCTS.PROD_SUBCATEGORY , PRODUCTS.PROD_ID , TIMES.FISCAL_YEAR ,
         TIMES.FISCAL_QUARTER_DESC , TIMES.FISCAL_MONTH_DESC ),

        (PRODUCTS.PROD_TOTAL , PRODUCTS.PROD_CATEGORY ,
         PRODUCTS.PROD_SUBCATEGORY , PRODUCTS.PROD_ID , TIMES.CALENDAR_YEAR ,
         TIMES.CALENDAR_QUARTER_DESC , TIMES.CALENDAR_MONTH_DESC ),

        (PRODUCTS.PROD_TOTAL , PRODUCTS.PROD_CATEGORY ,
         PRODUCTS.PROD_SUBCATEGORY , PRODUCTS.PROD_ID , TIMES.FISCAL_YEAR ,
         TIMES.FISCAL_QUARTER_DESC ),

        (PRODUCTS.PROD_TOTAL , PRODUCTS.PROD_CATEGORY ,
         PRODUCTS.PROD_SUBCATEGORY , PRODUCTS.PROD_ID , TIMES.CALENDAR_YEAR ,
         TIMES.CALENDAR_QUARTER_DESC ),

         (PRODUCTS.PROD_TOTAL , PRODUCTS.PROD_CATEGORY ,
          PRODUCTS.PROD_SUBCATEGORY , PRODUCTS.PROD_ID , TIMES.FISCAL_YEAR ),

         (PRODUCTS.PROD_TOTAL , PRODUCTS.PROD_CATEGORY ,
```

```
 PRODUCTS.PROD_SUBCATEGORY , PRODUCTS.PROD_ID , TIMES.CALENDAR_YEAR ),

(PRODUCTS.PROD_TOTAL , PRODUCTS.PROD_CATEGORY ,
 PRODUCTS.PROD_SUBCATEGORY , TIMES.CALENDAR_YEAR ,
 TIMES.CALENDAR_QUARTER_DESC , TIMES.CALENDAR_MONTH_DESC ,
 TIMES.FISCAL_YEAR , TIMES.FISCAL_QUARTER_DESC ,
 TIMES.FISCAL_MONTH_DESC , TIMES.WEEK_ENDING_DAY , TIMES.TIME_ID ),

 (PRODUCTS.PROD_TOTAL , PRODUCTS.PROD_CATEGORY ,
  PRODUCTS.PROD_SUBCATEGORY , TIMES.FISCAL_YEAR ,
  TIMES.FISCAL_QUARTER_DESC , TIMES.FISCAL_MONTH_DESC ,
  TIMES.WEEK_ENDING_DAY ),

 (PRODUCTS.PROD_TOTAL , PRODUCTS.PROD_CATEGORY ,
  PRODUCTS.PROD_SUBCATEGORY , TIMES.FISCAL_YEAR ,
  TIMES.FISCAL_QUARTER_DESC , TIMES.FISCAL_MONTH_DESC ),

 (PRODUCTS.PROD_TOTAL , PRODUCTS.PROD_CATEGORY ,
  PRODUCTS.PROD_SUBCATEGORY , TIMES.CALENDAR_YEAR ,
  TIMES.CALENDAR_QUARTER_DESC , TIMES.CALENDAR_MONTH_DESC ),

 (PRODUCTS.PROD_TOTAL , PRODUCTS.PROD_CATEGORY ,
  PRODUCTS.PROD_SUBCATEGORY , TIMES.FISCAL_YEAR ,
  TIMES.FISCAL_QUARTER_DESC ),

 (PRODUCTS.PROD_TOTAL , PRODUCTS.PROD_CATEGORY ,
  PRODUCTS.PROD_SUBCATEGORY , TIMES.CALENDAR_YEAR ,
  TIMES.CALENDAR_QUARTER_DESC ),

 (PRODUCTS.PROD_TOTAL , PRODUCTS.PROD_CATEGORY ,
  PRODUCTS.PROD_SUBCATEGORY , TIMES.FISCAL_YEAR ),

 (PRODUCTS.PROD_TOTAL , PRODUCTS.PROD_CATEGORY ,
  PRODUCTS.PROD_SUBCATEGORY , TIMES.CALENDAR_YEAR ),

 (PRODUCTS.PROD_TOTAL , PRODUCTS.PROD_CATEGORY , TIMES.CALENDAR_YEAR ,
  TIMES.CALENDAR_QUARTER_DESC , TIMES.CALENDAR_MONTH_DESC ,
  TIMES.FISCAL_YEAR , TIMES.FISCAL_QUARTER_DESC ,
  TIMES.FISCAL_MONTH_DESC , TIMES.WEEK_ENDING_DAY, TIMES.TIME_ID ),

 (PRODUCTS.PROD_TOTAL , PRODUCTS.PROD_CATEGORY , TIMES.FISCAL_YEAR ,
  TIMES.FISCAL_QUARTER_DESC , TIMES.FISCAL_MONTH_DESC ,
  TIMES.WEEK_ENDING_DAY ),

 (PRODUCTS.PROD_TOTAL , PRODUCTS.PROD_CATEGORY , TIMES.FISCAL_YEAR ,
```

```
                TIMES.FISCAL_QUARTER_DESC , TIMES.FISCAL_MONTH_DESC ),

                (PRODUCTS.PROD_TOTAL , PRODUCTS.PROD_CATEGORY , TIMES.CALENDAR_YEAR ,
                 TIMES.CALENDAR_QUARTER_DESC , TIMES.CALENDAR_MONTH_DESC ),

                (PRODUCTS.PROD_TOTAL , PRODUCTS.PROD_CATEGORY , TIMES.FISCAL_YEAR ,
                 TIMES.FISCAL_QUARTER_DESC ),

               (PRODUCTS.PROD_TOTAL , PRODUCTS.PROD_CATEGORY , TIMES.CALENDAR_YEAR ,
                TIMES.CALENDAR_QUARTER_DESC ),

               (PRODUCTS.PROD_TOTAL , PRODUCTS.PROD_CATEGORY , TIMES.FISCAL_YEAR ),

               (PRODUCTS.PROD_TOTAL , PRODUCTS.PROD_CATEGORY , TIMES.CALENDAR_YEAR ),

               (PRODUCTS.PROD_TOTAL , TIMES.CALENDAR_YEAR ,
                TIMES.CALENDAR_QUARTER_DESC , TIMES.CALENDAR_MONTH_DESC ,
                TIMES.FISCAL_YEAR , TIMES.FISCAL_QUARTER_DESC ,
                TIMES.FISCAL_MONTH_DESC , TIMES.WEEK_ENDING_DAY , TIMES.TIME_ID ),

                (PRODUCTS.PROD_TOTAL , TIMES.FISCAL_YEAR , TIMES.FISCAL_QUARTER_DESC ,
                TIMES.FISCAL_MONTH_DESC , TIMES.WEEK_ENDING_DAY ),

               (PRODUCTS.PROD_TOTAL , TIMES.FISCAL_YEAR , TIMES.FISCAL_QUARTER_DESC ,
                TIMES.FISCAL_MONTH_DESC ),

               (PRODUCTS.PROD_TOTAL , TIMES.CALENDAR_YEAR ,
                TIMES.CALENDAR_QUARTER_DESC , TIMES.CALENDAR_MONTH_DESC ),

               (PRODUCTS.PROD_TOTAL , TIMES.FISCAL_YEAR ,
                TIMES.FISCAL_QUARTER_DESC ),

               (PRODUCTS.PROD_TOTAL, TIMES.CALENDAR_YEAR,
                TIMES.CALENDAR_QUARTER_DESC),

               (PRODUCTS.PROD_TOTAL , TIMES.FISCAL_YEAR ),

               (PRODUCTS.PROD_TOTAL , TIMES.CALENDAR_YEAR ) ) ;

execute dbms_stats.gather_table_stats('SH', 'COST_CUBE_2_OLAP',  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 COST_CUBE_2_OLAP MINIMIZE RECORDS_PER_BLOCK ;
```

```
CREATE BITMAP INDEX BMHIDX_COST_PROD_TOTALTAL ON COST_CUBE_2_OLAP(PROD_TOTAL_77)
LOCAL
COMPUTE STATISTICS
PARALLEL PCTFREE 0
NOLOGGING;

CREATE BITMAP INDEX BMHIDX_COST_PROD_CATEGORY ON COST_CUBE_2_OLAP
(PROD_CATEGORY_78)
LOCAL
COMPUTE STATISTICS
PARALLEL PCTFREE 0
NOLOGGING;

CREATE BITMAP INDEX BMHIDX_COST_PROD_SUBCAORY ON COST_CUBE_2_OLAP
(PROD_SUBCATEGORY_79)
LOCAL
COMPUTE STATISTICS
PARALLEL PCTFREE 0
NOLOGGING;

CREATE BITMAP INDEX BMHIDX_COST_PROD_ID_ID ON COST_CUBE_2_OLAP
(PROD_ID_80)
LOCAL
COMPUTE STATISTICS
PARALLEL PCTFREE 0
NOLOGGING;

CREATE BITMAP INDEX BMHIDX_COST_CALENDAR_YEAR ON COST_CUBE_2_OLAP
(CALENDAR_YEAR_169)
LOCAL
COMPUTE STATISTICS
PARALLEL PCTFREE 0
NOLOGGING;

CREATE BITMAP INDEX BMHIDX_COST_CALENDAR_QESC ON COST_CUBE_2_OLAP
(CALENDAR_QUARTER_DESC_170)
LOCAL
COMPUTE STATISTICS
PARALLEL PCTFREE 0
NOLOGGING;

CREATE BITMAP INDEX BMHIDX_COST_CALENDAR_MESC ON COST_CUBE_2_OLAP
(CALENDAR_MONTH_DESC_171)
LOCAL
COMPUTE STATISTICS
```

```
PARALLEL PCTFREE 0
NOLOGGING;

CREATE BITMAP INDEX BMHIDX_COST_FISCAL_YEAEAR ON COST_CUBE_2_OLAP
(FISCAL_YEAR_172)
LOCAL
COMPUTE STATISTICS
PARALLEL PCTFREE 0
NOLOGGING;

CREATE BITMAP INDEX BMHIDX_COST_FISCAL_QUAESC ON COST_CUBE_2_OLAP
(FISCAL_QUARTER_DESC_173)
LOCAL
COMPUTE STATISTICS
PARALLEL PCTFREE 0
NOLOGGING;

CREATE BITMAP INDEX BMHIDX_COST_FISCAL_MONESC ON COST_CUBE_2_OLAP
(FISCAL_MONTH_DESC_174)
LOCAL
COMPUTE STATISTICS
PARALLEL PCTFREE 0
NOLOGGING;

CREATE BITMAP INDEX BMHIDX_COST_WEEK_ENDINDAY ON COST_CUBE_2_OLAP
(WEEK_ENDING_DAY_175)
LOCAL
COMPUTE STATISTICS
PARALLEL PCTFREE 0
NOLOGGING;

CREATE BITMAP INDEX BMHIDX_COST_TIME_ID_ID ON COST_CUBE_2_OLAP(TIME_ID_176)
LOCAL
COMPUTE STATISTICS
PARALLEL PCTFREE 0
NOLOGGING;

execute dbms_stats.gather_table_stats('SH', 'COST_CUBE_2_OLAP',  degree=>
   dbms_stats.default_degree, estimate_percent=>
   dbms_stats.auto_sample_size, method_opt=>
   'for all hidden columns size 254' , granularity=>'GLOBAL') ;

execute cwm2_olap_cube.set_mv_summary_code('SH', 'COST_CUBE', 'GROUPINGSET') ;
```

Summary of DBMS_ODM Subprograms

Table 30–1 DBMS_ODM Subprograms

Subprogram	Description
CREATEDIMLEVTUPLE Procedure on page 30-11	Creates a table of levels to be included in the materialized view for a cube.
CREATECUBELEVELTUPLE Procedure on page 30-12	Creates a table of level combinations to be included in the materialized view for a cube.
CREATEFACTMV_GS Procedure on page 30-13	Generates a script that creates a fact table materialized view.
CREATEDIMMV_GS Procedure on page 30-14	Generates a script that creates a dimension table materialized view.

CREATEDIMLEVTUPLE Procedure

This procedure creates the table `sys.olaptablevels`, which lists all the levels of all of the dimensions of the cube. By default, all levels are selected for inclusion in the materialized view. You can edit the table to deselect any levels that you do not want to include.

Syntax

```
CREATE_DIMLEVTUPLE (
                cube_owner    IN varchar2,
                cube_name     IN varchar2);
```

Parameters

Table 30–2 CREATEDIMLEVTUPLE Procedure Parameters

Parameter	Description
cube_owner	Owner of the cube.
cube_name	Name of the cube.

CREATECUBELEVELTUPLE Procedure

This procedure creates the table `sys.olaptableveltuples`, which lists all the level combinations to be included in the materialized view for the cube.

The table `sys.olaptableveltuples` is created based on the table `sys.olaptablevels`, which was generated by the CREATEDIMLEVTUPLE Procedure.

> **Important:** If you do not want to include all level combinations in the materialized view for the cube, you must edit the table `sys.olaptablevels` before executing the `CREATECUBELEVELTUPLE` procedure.

Syntax

```
CREATECUBELEVELTUPLE (
               cube_owner     IN   VARCHAR2,
               cube_name      IN   VARCHAR2);
```

Parameters

Table 30–3 CREATECUBELEVELTUPLE Procedure Parameters

Parameter	Description
`cube_owner`	Owner of the cube.
`cube_name`	Name of the cube.

CREATEFACTMV_GS Procedure

This procedure generates a script that creates a fact table materialized view.

The materialized view will include all level combinations specified in the `sys.olaptableveltuples` table, which was created by the CREATECUBELEVELTUPLE Procedure.

Syntax

```
CREATEFACTMV_GS (
                cube_owner              IN   VARCHAR2,
                cube_name               IN   VARCHAR2,
                outfile                 IN   VARCHAR2,
                outfile_path            IN   VARCHAR2,
                partitioning            IN   BOOLEAN,
                tablespace_mv           IN   VARCHAR2 DEFAULT NULL,
                tablespace_index        IN   VARCHAR2 DEFAULT NULL);
```

Parameters

Table 30–4 CREATEFACTMV_GS Procedure Parameters

Parameter	Description
`cube_owner`	Owner of the cube.
`cube_name`	Name of the cube.
`output_file`	File name where the PL/SQL script will be written.
`output_path`	Directory path where `output_file` will be created.
`partitioning`	TRUE turns on index partitioning; FALSE turns it off.
`tablespace_mv`	The name of the tablespace in which the materialized view will be created. When this parameter is omitted, the materialized view is created in the user's default tablespace.
`tablespace_index`	The name of the tablespace in which the index for the materialized view will be created. When this parameter is omitted, the index is created in the user's default tablespace.

CREATEDIMMV_GS Procedure

This procedure generates a script that creates a dimension table materialized view for each hierarchy of a dimension.

Syntax

```
CREATEDIMMV_GS (
               dimension_owner    IN   VARCHAR2,
               dimension_name     IN   VARCHAR2,
               output_file        IN   VARCHAR2,
               output_path        IN   VARCHAR2,
               tablespace_mv      IN   VARCHAR2 DEFAULT NULL,
               tablespace_index   IN   VARCHAR2 DEFAULT NULL);
```

Parameters

Table 30–5 CREATEDIMMV_GS Procedure Parameters

Parameter	Description
`dimension_owner`	Owner of the dimension.
`dimension_name`	Name of the dimension.
`output_file`	File name where the PL/SQL script will be written.
`output_path`	Directory path where `output_file` will be created.
`tablespace_mv`	The name of the tablespace in which the materialized view will be created. When this parameter is omitted, the materialized view is created in the user's default tablespace.
`tablespace_index`	The name of the tablespace in which the index for the materialized view will be created. When this parameter is omitted, the index is created in the user's default tablespace.

31

Creating Fact Materialized Views With OLAP Summary Advisor

This chapter explains how to use OLAP Summary Advisor to create fact table materialized views in concatenated rollup form for the OLAP API.

> **See Also:** Chapter 10, "Creating Materialized Views for the OLAP API".

This chapter contains the following topics:

- Using the OLAP Summary Advisor Wizard
- Partitioning, Statistics, and Indexes
- The MV CREATE Statement With Concatenated Rollup
- Sample Script for the COST Cube

Using the OLAP Summary Advisor Wizard

To create concatenated rollup MVs for CWM cubes, use OLAP Summary Advisor.

Oracle Enterprise Manager has two distinct Summary Advisors. They generate very different types of materialized views. One Summary Advisor generates materialized views for Oracle OLAP, and the other generates materialized views for other types of applications.

The Summary Advisor that you need to use for OLAP is located within the OLAP Management tool. It generates materialized views that query rewrite will use for queries generated by the OLAP API.

See Also:

- "Fact Materialized Views".
- "Choosing the Right Format for Materialized Views".

Procedure: Run the OLAP Summary Advisor

Follow these steps to run the OLAP Summary Advisor wizard:

1. Start Oracle Enterprise Manager and access OLAP Management, as described in Chapter 5, "Creating OLAP Catalog Metadata".
2. Expand the OLAP folder, then fully expand the Cubes folder.
3. Right-click a cube or its Materialized Views subfolder.

   You see a popup menu.
4. Choose **Summary Advisor** from the menu.

   You see the Summary Advisor Wizard Welcome page.
5. Choose **Next**.

   The Summary Advisor analyzes the cube and makes recommendations for creating materialized views for the fact table and dimension tables associated with the selected cube. When it is done, you see the Recommendations page.
6. Choose **Next**.

   The Summary Advisor generates the scripts to create the recommended materialized views. When it is done, you see the Finish page.

7. Examine the scripts. If you have already created the materialized views for another cube that uses some of the same dimensions, delete the scripts that recreate materialized views for those dimensions.
8. To modify the scripts, choose **Save to file**. Then choose **Cancel** to close the Summary Advisor. You can edit the file, then execute it using SQL*Plus or Job Manager.

   or

   Choose **Finish** to execute the original scripts immediately. You see the Implement Recommendations page while the scripts are executing.
9. Run the OLAP Summary Advisor wizard on other cubes in your schema.

Partitioning, Statistics, and Indexes

The scripts generated by OLAP Summary Advisor include syntax for partitioning, gathering statistics and creating bitmap indexes.

Partitioning

Partitioning can have a significant impact upon query performance. You may want to customize the partitioning of fact materialized views before running the scripts generated by OLAP Summary Advisor.

By default, partitioning is based on grouping IDs since most queries are based on levels. A grouping ID uniquely identifies one level combination per partition (such as `CALENDAR_YEAR` and `PROD_TOTAL`).

Statistics

Statistics are required by the optimizer in order to maximize query performance at runtime.

The following SQL statements analyze a materialized view and generate the needed information.

```
ANALYZE TABLE mv_name COMPUTE 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 ;
```

For more information about the ANALYZE TABLE statement, refer to the*Oracle9i SQL Reference*. For more information about the DBMS_STATS package, refer to the *Oracle9i Supplied PL/SQL Packages and Types Reference*.

Bitmap Indexes

Bitmap indexes optimize the performance of materialized views at runtime. Fact materialized views for the OLAP API include bitmap indexes for all columns that contain dimension values.

The following SQL statements create bitmap indexes.

```
CREATE BITMAP INDEX index_name ON mv_name (mv_colname )
TABLESPACE tblspace_name
PCTFREE 0
COMPUTE STATISTICS
LOCAL
NOLOGGING;
```

The MV CREATE Statement With Concatenated Rollup

The following example shows the basic structure of the SQL statements generated by OLAP Summary Advisor to create a concatenated rollup style fact MV for the OLAP API. The following general characteristics apply:

- The SELECT statement contains SUM(*column*) and COUNT(*column*) function calls for all measures in the cube (that is, all aggregated columns in the fact table), and a COUNT(*) function call.
- The SELECT list contains all GROUP BY columns.
- The list of level key columns always appear in the exact same order, especially in the GROUPING_ID and GROUP BY clauses.

The following example shows the basic syntax.

```
CREATE MATERIALIZED VIEW mvname
partition by range (gid)
(partition values less than (1),
     .
     .
     .
partition values less than (MAXVALUE))
BUILD IMMEDIATE
REFRESH FORCE
```

```
ENABLE QUERY REWRITE
AS
SELECT SUM(measure1) target, COUNT(measure1) target,
   SUM(measure2) target, COUNT(measure2) target,
     .
     .
     .
   COUNT(*) COUNT_OF_STAR, select_list
   hierarch1_level1, hierarch1_level2, ...,
   hierarch2_level1, hierarch2_level2,...
   GROUPING_ID(hierarch1_level1, hierarch1_level2, ...,
    hierarch2_level1, hierarch2_level2,... ) gid
FROM dimtable1, dimtable2,...
WHERE (dim_key1=fact_key1) AND (dim_key2=fact_key2)...AND...
GROUPBY
   hierarch1_level1, ROLLUP(hierarch1_leveln2,... hierarch1_leveln),
   hierarch2_level1 ROLLUP(hierarch2_leveln2,... hierarch2_leveln,
     .
     .
     .
   hierarchn_level1 ROLLUP(hierarchn_level2... hierarchn_leveln)
```

where:

measure1, measure 2... are the measures in the fact table.

select_list are the dimension levels from *hierarch1_level1* to *hierarchn_leveln*.

hierarch1...hierarchn are the dimension hierarchies, beginning with the hierarchy with the most levels (1) and ending with the hierarchy with the fewest levels (*n*). Note that this ordering is important.

level1...leveln are the columns in the related dimension tables, from the highest (1) to the lowest (*n*) levels of aggregation.

dim_key is the key column in the dimension table.

fact_key is the related column in the fact table.

Sample Script for the COST Cube

The following sample script creates materialized views in concatenated rollup form for the COST_CUBE cube, which is mapped to the COSTS fact table in the SH schema.

This script creates two materialized views, one for each combination of hierarchies associated with the COST_CUBE cube.

```
create materialized view
COST_CUBE_1_OLAP
partition by range (gid) (
partition values less than(1),
partition values less than(3),
partition values less than(7),
partition values less than(16),
partition values less than(17),
partition values less than(19),
partition values less than(23),
partition values less than(48),
partition values less than(49),
partition values less than(51),
partition values less than(55),
partition values less than(112),
partition values less than(113),
partition values less than(115),
partition values less than(119),
partition values less than(MAXVALUE))
pctfree 5 pctused 40
tablespace SH_DATA
build immediate
using no index
refresh force
enable query rewrite
AS
SELECT
   GROUPING_ID(TIMES.CALENDAR_YEAR, TIMES.CALENDAR_QUARTER_DESC,
   TIMES.CALENDAR_MONTH_DESC, TIMES.TIME_ID, PRODUCTS.PROD_TOTAL,
   PRODUCTS.PROD_CATEGORY, PRODUCTS.PROD_SUBCATEGORY,
   PRODUCTS.PROD_ID) gid,
   SUM(COSTS.UNIT_COST) SUM_OF_UNIT_COST,
   SUM(COSTS.UNIT_PRICE) SUM_OF_UNIT_PRICE,
   COUNT(*) COUNT_OF_STAR,
   TIMES.CALENDAR_YEAR CALENDAR_YEAR_1,
   TIMES.CALENDAR_QUARTER_DESC CALENDAR_QUARTER_DESC_2,
   TIMES.CALENDAR_MONTH_DESC CALENDAR_MONTH_DESC_3,
```

```
   TIMES.TIME_ID TIME_ID_4,
   PRODUCTS.PROD_TOTAL PROD_TOTAL_10,
   PRODUCTS.PROD_CATEGORY PROD_CATEGORY_11,
   PRODUCTS.PROD_SUBCATEGORY PROD_SUBCATEGORY_12,
   PRODUCTS.PROD_ID PROD_ID_13
FROM
   SH.TIMES TIMES,
   SH.PRODUCTS PRODUCTS,
   SH.COSTS COSTS
WHERE
   (TIMES.TIME_ID = COSTS.TIME_ID) AND
   (PRODUCTS.PROD_ID = COSTS.PROD_ID)
GROUP BY
   TIMES.CALENDAR_YEAR ,
   ROLLUP
      (TIMES.CALENDAR_QUARTER_DESC, TIMES.CALENDAR_MONTH_DESC, TIMES.TIME_ID),
   PRODUCTS.PROD_TOTAL ,
   ROLLUP
      (PRODUCTS.PROD_CATEGORY, PRODUCTS.PROD_SUBCATEGORY, PRODUCTS.PROD_ID);

execute dbms_stats.gather_table_stats('SH', 'COST_CUBE_1_OLAP',  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 COST_CUBE_1_OLAP MINIMIZE RECORDS_PER_BLOCK ;

CREATE BITMAP INDEX BI_COST_CALENAR_QUESC_2_1 ON COST_CUBE_1_OLAP(CALENDAR_
QUARTER_DESC_2)
LOCAL
COMPUTE STATISTICS
TABLESPACE SH_IDX
PARALLEL PCTFREE 0
NOLOGGING;

CREATE BITMAP INDEX BI_COST_CALENAR_MOESC_3_1 ON COST_CUBE_1_OLAP(CALENDAR_
MONTH_DESC_3)
LOCAL
COMPUTE STATISTICS
TABLESPACE SH_IDX
PARALLEL PCTFREE 0
NOLOGGING;

CREATE BITMAP INDEX BI_COST_TIME_D_ID_4_1 ON COST_CUBE_1_OLAP(TIME_ID_4)
LOCAL
COMPUTE STATISTICS
```

```
TABLESPACE SH_IDX
PARALLEL PCTFREE 0
NOLOGGING;

CREATE BITMAP INDEX BI_COST_PROD_ATEGOORY_22_1 ON COST_CUBE_1_OLAP(PROD_
CATEGORY_11)
LOCAL
COMPUTE STATISTICS
TABLESPACE SH_IDX
PARALLEL PCTFREE 0
NOLOGGING;

CREATE BITMAP INDEX BI_COST_PROD_UBCATORY_24_1 ON COST_CUBE_1_OLAP(PROD_
SUBCATEGORY_12)
LOCAL
COMPUTE STATISTICS
TABLESPACE SH_IDX
PARALLEL PCTFREE 0
NOLOGGING;

CREATE BITMAP INDEX BI_COST_PROD_D_ID_26_1 ON COST_CUBE_1_OLAP(PROD_ID_13)
LOCAL
COMPUTE STATISTICS
TABLESPACE SH_IDX
PARALLEL PCTFREE 0
NOLOGGING;

execute dbms_stats.gather_table_stats('SH', 'COST_CUBE_1_OLAP',  degree=>
dbms_stats.default_degree, estimate_percent=>
dbms_stats.auto_sample_size, method_opt=>
'for all hidden columns size 254' , granularity=>'GLOBAL') ;

execute cwm2_olap_cube.set_mv_summary_code('SH', 'COST_CUBE', 'ROLLUP') ;

create materialized view
COST_CUBE_2_OLAP
partition by range (gid) (
partition values less than(1),
partition values less than(3),
partition values less than(7),
partition values less than(15),
partition values less than(32),
partition values less than(33),
partition values less than(35),
partition values less than(39),
```

```
partition values less than(47),
partition values less than(96),
partition values less than(97),
partition values less than(99),
partition values less than(103),
partition values less than(111),
partition values less than(224),
partition values less than(225),
partition values less than(227),
partition values less than(231),
partition values less than(239),
partition values less than(MAXVALUE))
pctfree 5 pctused 40
tablespace SH_DATA
build immediate
using no index
refresh force
enable query rewrite
AS
SELECT
   GROUPING_ID(PRODUCTS.PROD_TOTAL, PRODUCTS.PROD_CATEGORY,
   PRODUCTS.PROD_SUBCATEGORY, PRODUCTS.PROD_ID, TIMES.FISCAL_YEAR,
   TIMES.FISCAL_QUARTER_DESC, TIMES.FISCAL_MONTH_DESC,
   TIMES.WEEK_ENDING_DAY, TIMES.TIME_ID) gid,
   SUM(COSTS.UNIT_COST) SUM_OF_UNIT_COST,
   SUM(COSTS.UNIT_PRICE) SUM_OF_UNIT_PRICE,
   COUNT(*) COUNT_OF_STAR,
   TIMES.FISCAL_YEAR FISCAL_YEAR_5,
   TIMES.FISCAL_QUARTER_DESC FISCAL_QUARTER_DESC_6,
   TIMES.FISCAL_MONTH_DESC FISCAL_MONTH_DESC_7,
   TIMES.WEEK_ENDING_DAY WEEK_ENDING_DAY_8,
   TIMES.TIME_ID TIME_ID_9,
   PRODUCTS.PROD_TOTAL PROD_TOTAL_10,
   PRODUCTS.PROD_CATEGORY PROD_CATEGORY_11,
   PRODUCTS.PROD_SUBCATEGORY PROD_SUBCATEGORY_12,
   PRODUCTS.PROD_ID PROD_ID_13
FROM
   SH.PRODUCTS PRODUCTS,
   SH.TIMES TIMES,
   SH.COSTS COSTS
WHERE
   (PRODUCTS.PROD_ID = COSTS.PROD_ID) AND
   (TIMES.TIME_ID = COSTS.TIME_ID) GROUP BY
   PRODUCTS.PROD_TOTAL ,
```

```
     ROLLUP
      (PRODUCTS.PROD_CATEGORY, PRODUCTS.PROD_SUBCATEGORY, PRODUCTS.PROD_ID),
   TIMES.FISCAL_YEAR ,
     ROLLUP
      (TIMES.FISCAL_QUARTER_DESC, TIMES.FISCAL_MONTH_DESC,
       TIMES.WEEK_ENDING_DAY, TIMES.TIME_ID ) ;

execute dbms_stats.gather_table_stats('SH', 'COST_CUBE_2_OLAP',  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 COST_CUBE_2_OLAP MINIMIZE RECORDS_PER_BLOCK ;

CREATE BITMAP INDEX BI_COST_PROD_ATEGOORY_33_2 ON COST_CUBE_2_OLAP(PROD_
CATEGORY_11)
LOCAL
COMPUTE STATISTICS
TABLESPACE SH_IDX
PARALLEL PCTFREE 0
NOLOGGING;

CREATE BITMAP INDEX BI_COST_PROD_UBCATORY_36_2 ON COST_CUBE_2_OLAP(PROD_
SUBCATEGORY_12)
LOCAL
COMPUTE STATISTICS
TABLESPACE SH_IDX
PARALLEL PCTFREE 0
NOLOGGING;

CREATE BITMAP INDEX BI_COST_PROD_D_ID_39_2 ON COST_CUBE_2_OLAP(PROD_ID_13)
LOCAL
COMPUTE STATISTICS
TABLESPACE SH_IDX
PARALLEL PCTFREE 0
NOLOGGING;

CREATE BITMAP INDEX BI_COST_FISCA_QUARESC_24_2 ON COST_CUBE_2_OLAP(FISCAL_
QUARTER_DESC_6)
LOCAL
COMPUTE STATISTICS
TABLESPACE SH_IDX
PARALLEL PCTFREE 0
NOLOGGING;
```

```
CREATE BITMAP INDEX BI_COST_FISCA_MONTESC_28_2 ON COST_CUBE_2_OLAP(FISCAL_MONTH_
DESC_7)
LOCAL
COMPUTE STATISTICS
TABLESPACE SH_IDX
PARALLEL PCTFREE 0
NOLOGGING;

CREATE BITMAP INDEX BI_COST_WEEK_NDINGDAY_32_2 ON COST_CUBE_2_OLAP(WEEK_ENDING_
DAY_8)
LOCAL
COMPUTE STATISTICS
TABLESPACE SH_IDX
PARALLEL PCTFREE 0
NOLOGGING;

CREATE BITMAP INDEX BI_COST_TIME_D_ID_36_2 ON COST_CUBE_2_OLAP(TIME_ID_9)
LOCAL
COMPUTE STATISTICS
TABLESPACE SH_IDX
PARALLEL PCTFREE 0
NOLOGGING;

execute dbms_stats.gather_table_stats('SH', 'COST_CUBE_2_OLAP',  degree=>
   dbms_stats.default_degree, estimate_percent=>
   dbms_stats.auto_sample_size, method_opt=>
   'for all hidden columns size 254' , granularity=>'GLOBAL') ;

execute cwm2_olap_cube.set_mv_summary_code('SH', 'COST_CUBE', 'ROLLUP') ;
```

A

Upgrading From Express Server

This appendix provides upgrading instructions and identifies some of the major differences between Oracle Express Server 6.3 and Oracle9*i* OLAP. It is intended to provide a frame of reference to help you understand the material presented in this guide.

This chapter includes the following topics:

- Administration
- Data Transfer
- Localization
- Applications Support
- Programming Language Changes
- How to Upgrade an Express Database

> **See Also:** "What's New in Oracle OLAP?" for a list of major features introduced in this release.

Administration

Oracle OLAP is installed as an option in Oracle Enterprise Edition, and it is now integrated with the Oracle database. While Express Server runs in a service environment, Oracle OLAP runs within the database kernel.

In Oracle9*i*, the term **database** refers only to the relational database. Express databases are now called **analytic workspaces**. 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. There are no "`.db`" files.

The administrative tasks for Oracle OLAP are merged with the database tool set.

Authentication of Users

Oracle OLAP does not use operating system identities, except for the installation user under whose identity the RDBMS 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 the Oracle RDBMS. 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 alias** that is mapped to the physical directory path. This access is granted either to an individual user ID or to a database role.

Management Tools

Oracle Enterprise Manager encompasses the tools for administering Oracle OLAP, providing a common user interface across all platforms. Various PL/SQL packages extend the functionality currently available through Oracle Enterprise Manager and provide an alternative to its use.

Performance data can be collected in system tables the same as other Oracle database performance statistics.

OLAP Instance Manager, `oesmgr`, and `oescmd` are not available.

Data Transfer

Oracle OLAP runs within the Oracle database kernel. 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 move data between an analytic workspace objects (such as variables and dimensions) and relational tables in the following ways:

- The OLAP DML's 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.
- A PL/SQL package, `CWM2_OLAP_AW_CREATE`, provides procedures for creating an analytic workspace from relational tables and OLAP Catalog metadata, and for generating views of the workspace.
- 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.

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. The RDBMS and Oracle OLAP typically use the same character set, which is selected during installation.

If you are upgrading Express databases that used 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. If you plan to import Express databases or to use Oracle OLAP to access multibyte data in external files, then you might find Table A–1, " Multibyte Character Set Equivalents" helpful in identifying a character set. Note that the `CHARSET` option is now obsolete.

Support for Globalization Technology has been added to the OLAP DML. These options allow an application to query the current localization settings and override the behaviors controlled by the default language and territory.

Note: Oracle OLAP does not support EBCDIC character sets.

Table A–1 identifies the Unicode character sets available in Oracle that are equivalent to the Express Server character sets.:

Table A–1 Multibyte Character Set Equivalents

Express Server DefaultCharacterSet Parameter or CHARSET Option Value	Equivalent Unicode Character Set
EUC	JA16EUC
SHIFTJIS	JA16SJIS
HANGEUL	KO16KSC5601
SCHINESE	ZHS16GBK
TCHINESE	ZHT16BIG5

Applications Support

Oracle OLAP allows 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.

The CWM2_OLAP_AW_CREATE package contains procedures for creating analytic workspaces and generating views of analytic workspaces. You can create OLAP Catalog metadata for use by the OLAP API, or use SQL to run directly against these views of your multidimensional data.

You cannot run Windows C++, HTML, or Java applications that were developed for use with Express Server.

Programming Environment

Applications for Oracle OLAP can be developed in Java using the OLAP API. SQL-based applications can access OLAP data through views or manipulate it directly through SQL table functions.

OLAP Worksheet provides an interactive environment for developing stored procedures in either the OLAP DML or SQL. The `DBMS_AW` procedure executes OLAP DML commands from within a SQL program.

You cannot connect to Oracle OLAP using Express Administrator, Personal Express, or the Express Connection Utility.

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

In Oracle OLAP, the database administrator defines multidimensional objects and associated OLAP metadata in the relational database using PL/SQL packages for use by the OLAP API.

OLAP Worksheet allows DBAs and applications developers to create objects in the analytic workspace by issuing DML commands. For the OLAP API to access these objects, the appropriate analytic workspace metadata must be defined.

Oracle Express Administrator is not available in Oracle OLAP, and the Oracle Express Objects metadata that it generated is not used by the OLAP API.

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:

- Parallel aggregate
- 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

For comprehensive lists of new, obsolete, and significantly revised commands, open OLAP DML Help and click **List of Changes** on the Contents page.

UPDATE and COMMIT

The `UPDATE` command moves analytic workspace changes from a temporary area to the database table in which the workspace is stored. Your changes are not saved until you execute a `COMMIT` command, either from your Oracle OLAP session or from SQL.

If you want changes that you have made in a workspace to be committed when you execute the `COMMIT` command, then you must first update the workspace using the `UPDATE` command. Changes that have not been moved to the table are not committed.

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.

How to Upgrade an Express Database

Follow these steps to upgrade an Express database for use as an analytic workspace in Oracle9*i*:

1. Open a connection with Express Server and create an EIF file of your Express database, using a command such as this:

   ```
   EXPORT ALL TO EIF FILE 'upgrade.eif' REWRITE
   ```

   where `upgrade.eif` is the name of the file being created.

2. Copy the file to a directory that has a **directory alias** in the Oracle database and to which you have access rights.

   For information about directory aliases, refer to "Controlling Access to External Files" on page 6-10.

3. Open a connection to the Oracle database using OLAP Worksheet.

   For information about using OLAP Worksheet, refer to the *Oracle9i OLAP Developer's Guide to the OLAP DML.*

4. Create an empty analytic workspace with a command such as this:

   ```
   AW CREATE financials TABLESPACE olapts
   ```

   where `financials` is the name of the analytic workspace and `olapts` is the name of a tablespace allocated for your use. Note that the `DATABASE` command has changed to the `AW` command.

5. Copy the object definitions and data from the EIF file into the new analytic workspace with a command such as this:

   ```
   IMPORT ALL FROM EIF FILE 'alias/upgrade.eif' DATA DFNS
   ```

   where `alias` is the name of the directory alias, and `upgrade.eif` is the name of the EIF file.

6. Save your changes to the new analytic workspace:

   ```
   UPDATE
   ```

7. Commit the new analytic workspace to the Oracle database:

   ```
   COMMIT
   ```

8. Revise any programs in the analytic workspace to delete references to obsolete commands. Save these changes.

Index

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X