Oracle9*i* OLAP

User’s Guide

Release 2 (9.2)

March 2002

Part No. A95295-01

Oracle9*i* OLAP User’s Guide, Release 2 (9.2)

Part No. A95295-01

Contents

Send Us Your Comments

Oracle9*i* OLAP User's Guide, Release 2 (9.2)

Part No. A95295-01

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: "Administering Oracle OLAP"

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 data stored in relational tables, either using a utility or performing the steps manually. Describes the OLAP DML programs for creating an analytic workspace and for generating relational views of the workspace data.

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 10, "DBMS_AW"

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

Chapter 11, "OLAP_TABLE Function"

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 12, "OLAP Catalog Union Views"

Describes the views that constitute the comprehensive read API to all the OLAP metadata (both CWM1 and CWM2) defined in the database.

Chapter 13, "OLAP Catalog (CWM2-Specific) Views"

Describes the views that constitute the read API to CWM2.

Chapter 14, "OLAP Catalog Analytic Workspace Views"

Describes the views that represent analytic workspace objects registered in the OLAP Catalog.

Chapter 15, "CWM2_OLAP_AW_ACCESS"

Describes procedures for creating views of workspace objects. The views can be used by standard SQL to access data stored in the analytic workspace, or to define OLAP metadata so that OLAP API applications can access the multidimensional objects.

Chapter 16, "CWM2_OLAP_PC_TRANSFORM"

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

Chapter 17, "CWM2_OLAP_DIMENSION"

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

Chapter 18, "CWM2_OLAP_DIMENSION_ATTRIBUTE"

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

Chapter 19, "CWM2_OLAP_HIERARCHY"

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

Chapter 20, "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 21, "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 22, "CWM2_OLAP_CUBE"

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

Chapter 23, "CWM2_OLAP_MEASURE"

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

Chapter 24, "CWM2_OLAP_TABLE_MAP"

Describes procedures for mapping OLAP metadata entities to columns in your data warehouse dimension tables and fact tables.

Chapter 25, "CWM2_OLAP_AW_OBJECT"

Describes procedures for registering metadata in the OLAP catalog for data that is stored in an analytic workspace.

Chapter 26, "CWM2_OLAP_AW_MAP"

Describes procedures for mapping logical OLAP metadata entities to objects defined in analytic workspaces.

Chapter 27, "CWM_CLASSIFY"

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

Part V: "Creating Materialized Views for the OLAP API"

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

Chapter 28, "Developing a Summary Management Strategy"

Provides general information about summary management issues for 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 and the `OLAPFACTVIEW` package 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.

Convention	Meaning	Example
`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.
`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]`

...	Horizontal ellipsis points indicate either:	
	■ That we have omitted parts of the code that are not directly related to the example	`CREATE TABLE ... AS` *subquery*`;`
	■ That you can repeat a portion of the code	`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;`

Convention	Meaning	Example

`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:

- Oracle9i Release 2 (9.2) New Features in Oracle 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 workspace
- 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

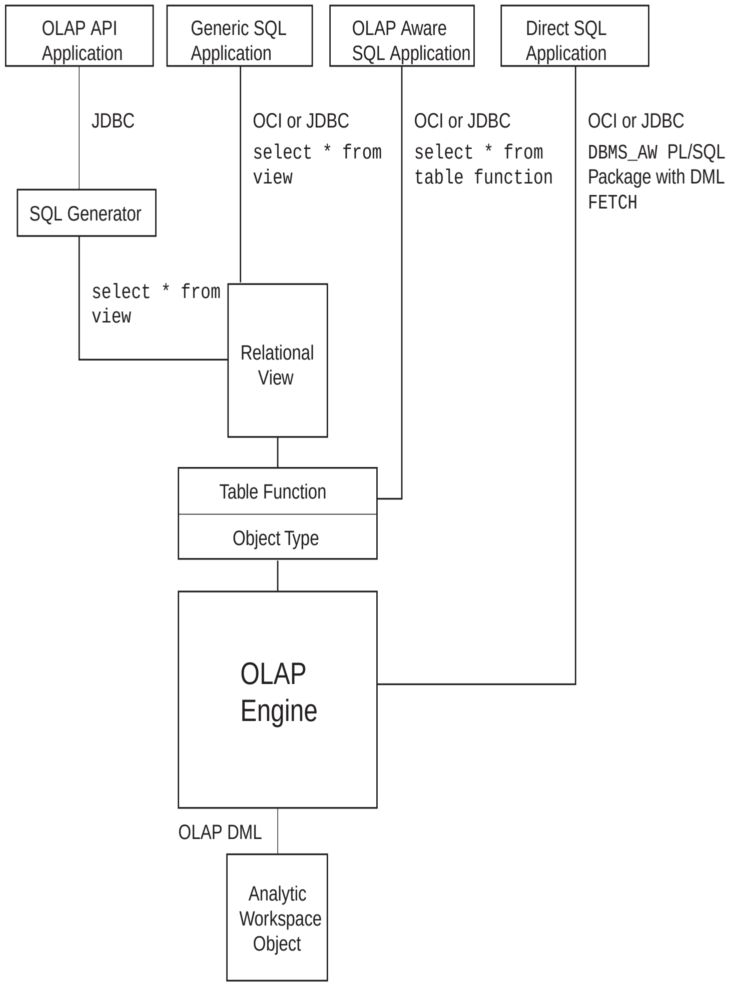
On a very basic level, all applications have access to analytic workspaces and the computational engine using SQL, but the application can be unaware of the SQL or unaware of the underlying OLAP facilities. They all use OCI or JDBC for their communications protocol.

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?
- Basic Categories of OLAP DML Commands
- Methods of Executing OLAP DML Commands

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 6-9. 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.

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 10, "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

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.

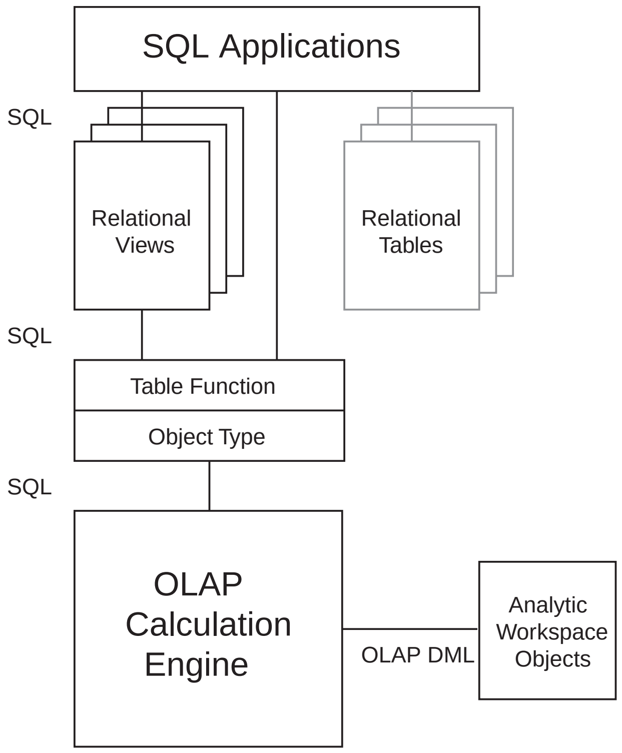
  You can use table functions to fetch data from objects in an analytic workspace. The table functions require arguments that are passed to the OLAP engine, which selects, manipulates, and returns the data. By incorporating table functions into your application, you have the most power and flexibility 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_ACCESS` PL/SQL 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. (This package is part of CWM2 because it is also used to make workspace objects accessible to the OLAP catalog metadata.)

  If you are using the `CNV_CWM.TO.ECM` OLAP DML program to create an analytic workspace, you can also use it to generate relational views of your workspace data. The CNV_CWM.TO.ECM program employs the CWM2_OLAP_AW_ACCESS package for this stage.

- 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 the CWM2_OLAP_AW_ACCESS package (which is a wrapper to this technology), 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 10, "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 10, "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.`

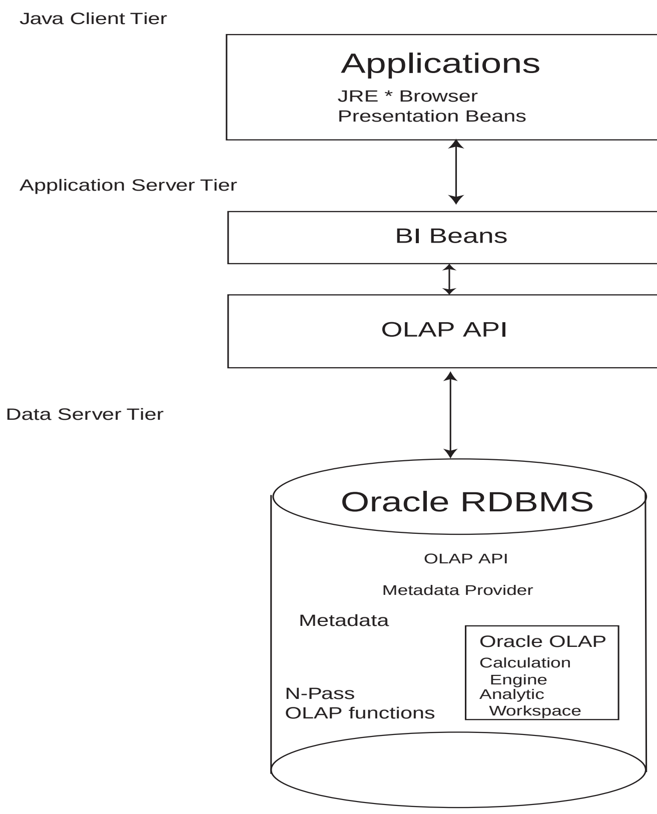
Thick-Client Configuration

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



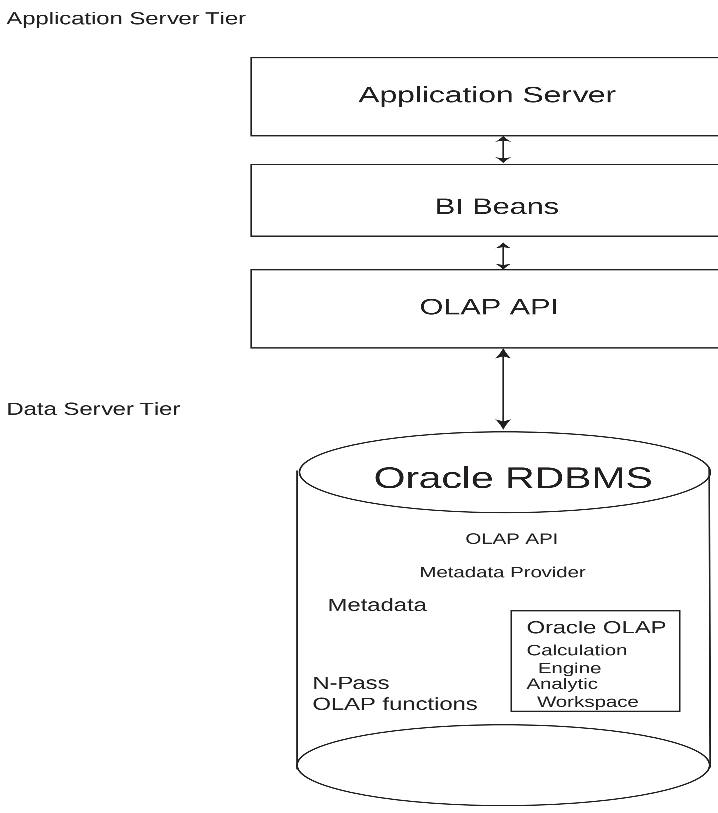
Thin-Client Configuration

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

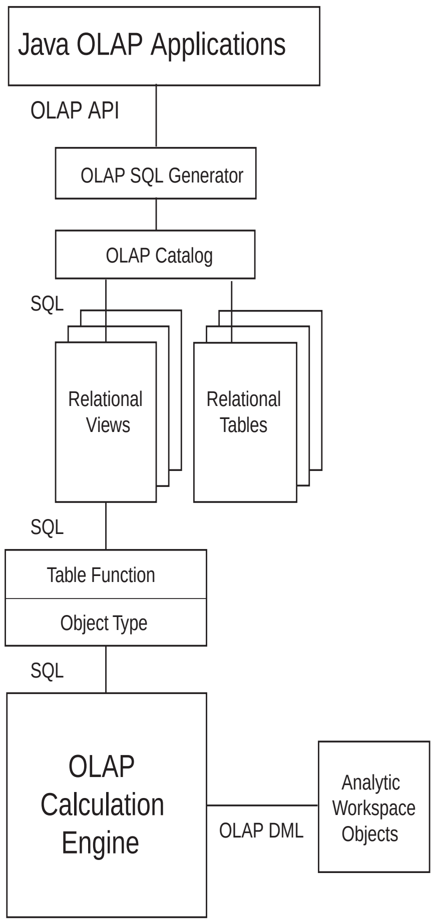
The OLAP API accesses the data through the OLAP catalog, that is, the relational tables that contain OLAP metadata. The application does not need to be aware of whether the data is located in relational tables or in an analytic workspace, nor does it need to know the mechanism for accessing that data.

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 `country`) that define the relationship between states and countries, and a third table with two columns (`state` and `country`) that define the relationship between states 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 3-14, 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.

Note: It is also possible to register metadata in the OLAP catalog to directly represent specific objects in analytic workspaces. Once this metadata is created, the OLAP API can query the OLAP catalog for information about data stored in multidimensional schemas. For more information, refer to Chapter 5, "Creating OLAP Catalog Metadata" and Chapter 25, "CWM2_OLAP_AW_OBJECT".

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.

However, in the OLAP API, a dimension does contain data, such as the names of individual products, geographic areas, and time periods. The OLAP API uses the metadata, dimension objects, and dimension tables to construct its dimensions.

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 OLAP metadata and the APIs for working with it. Detailed descriptions of each API appear in later chapters.

This chapter includes the following sections:

Choosing the Right Metadata Creation Method

There are two tools for creating OLAP metadata:

- Oracle Enterprise Manager
- OLAP Catalog APIs (`CWM2`)

The tool that you can use depends on the structure of your data warehouse.

Basic Star or Snowflake Schema

OLAP Management is a tool in Oracle Enterprise Manager that provides an easy-to-use graphical interface to release 1 of the OLAP Catalog (`CWM`). This tool creates a database `DIMENSION` object for all logical dimensions defined in the metadata, which imposes the following requirements on the dimension tables:

- Dimension values must be stored in dimension tables in a star or snowflake schema.
- All hierarchies must be level-based; the schema cannot use parent-child dimension tables.
- Multiple hierarchies defined for a dimension must have the same base level.
- Level columns cannot contain 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 all of these requirements, then you can use either Enterprise Manager or the `CWM2` APIs for defining OLAP metadata. No preprocessing steps are required.

Your ITT tool may have created metadata already. If so, you should check it using the OLAP Management tool to see if the metadata structures are “OLAP enabled.”

Dimension Tables with Complex Hierarchies

Release 2 of the OLAP Catalog (`CWM2`) supports the following variations in dimension tables:

- NULL values in level columns, such as in a skip-level hierarchy
- Data partitioning by hierarchy
- Different lowest levels for different hierarchies (sometimes called **ragged hierarchies**)

- Values mapped to different levels for multiple hierarchies
- For level-based dimensions, fact data can be either completely solved or completely unsolved. For parent/child dimensions, fact data must be completely unsolved.

If your dimension tables have any of these characteristics, then you must use the `CWM2` APIs to create OLAP metadata. No preprocessing is required.

Multidimensional Data and Parent-Child Dimensions

If your data warehouse stores data in analytic workspaces or has parent/child (instead of level-based) dimensions, then you must use the `CWM2` APIs to create OLAP metadata. For both types of warehouse storage, OLAP metadata creation is a two-step process:

1. Create relational views of the data. These views take the place of fact tables and dimension tables where they do not exist (in the case of multidimensional data) or are structurally unsuitable (in the case of parent-child dimension tables).
   - For dimension tables with parent-child relations, use the `CWM2_OLAP_PC_TRANSFORM` package to generate the views, as described in Chapter 16.
   - For multidimensional data stored in analytic workspaces, use the `CWM2_OLAP_AW_ACCESS` package, as described in Chapter 15.

   Note: When you use the `CWM2_OLAP_AW_ACCESS` package, multidimensional data is represented with the same metadata structures that represent relational data, and access to the analytic workspace is transparent to the client. You can also create metadata that directly represents the multidimensional objects in an analytic workspace. See Multidimensional Data Structures in the OLAP Catalog on page 5-4.

2. Use the OLAP Catalog `CWM2` PL/SQL procedures to generate the metadata for all of the data regardless of whether it is stored in relational tables or analytic workspaces.

See Also:

- Chapter 3, "Developing OLAP Applications" for a more detailed explanation of the technology underlying views of analytic workspace data.
- Chapter 15, "CWM2_OLAP_AW_ACCESS" for specific information about creating these views.

Multidimensional Data Structures in the OLAP Catalog

If some or all of your warehouse solution is stored in analytic workspaces, you may wish to create metadata that directly represents the multidimensional objects defined within the workspaces. This can be done using the `CWM2` APIs.

Objects within analytic workspaces are defined and manipulated by the OLAP Data Manipulation Language (DML).

To create metadata for analytic workspace objects, use the procedures in the `CWM2_OLAP_AW_OBJECT` package. To map this metadata to objects in an analytic workspace, use the procedures in the `CWM2_OLAP_AW_MAP` package.

See Also:

- Chapter 25, "CWM2_OLAP_AW_OBJECT" for specific information about creating `CWM2` metadata for analytic workspace objects, and Chapter 26, "CWM2_OLAP_AW_MAP" for specific information about mapping the metadata to objects in analytic workspaces.
- See *Oracle9i OLAP Developer's Guide to the OLAP DML* and the Oracle9i OLAP DML Reference help for specific information about the OLAP DML.

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, consisting of both `CWM` and `CWM2` metadata, 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.

Organization of the OLAP Catalog

The repository for OLAP metadata is known as the **OLAP Catalog**. This repository is included by default with the Enterprise Edition of the database when installed with either the General Purpose or Data Warehouse configuration.

The OLAP Catalog is one component of **Oracle OLAP**. The other server-side components are the OLAP calculation engine and support for analytic workspaces. On the client side, Oracle OLAP includes the OLAP API, an API for developing OLAP client applications in Java.

CWM and CWM2

The OLAP Catalog, owned by the OLAPSYS user, contains two distinct metadata repositories each with its own set of APIs:

- The CWM metadata repository, first released with Oracle 9*i* Release 1, is still supported in the current release. The CWM metadata repository, based on CWM Lite (Common Warehouse Metadata) is used by Oracle Enterprise Manager. You can create and view CWM metadata by using Enterprise Manager's OLAP Management feature.
- The CWM2 metadata repository, newly available with Release 2 (9.2), provides support for additional warehouse configurations. You can create and view CWM2 metadata by using the CWM2 PL/SQL packages and views.

Both metadata repositories include 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. The `CWM` model tables are named with the `CWM` prefix. The `CWM2` model tables are named with the `CWM2` prefix.
- **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. The `CWM` packages are named with the `CWM` prefix. The `CWM2` packages are named with the `CWM2` prefix.
- **A Read API -** A set of SQL views providing information about the metadata registered in the model tables. The `CWM` views are named with the `OLAP` prefix. The `CWM2` views are named with the `OLAP2` prefix.

`CWM2` additionally includes **Preprocessors,** PL/SQL packages for transforming certain types of warehouse data to a format supported by the `CWM2` APIs. Preprocessors are available for data stored in analytic workspaces and for parent/child dimension tables.

Depending on the organization of your warehouse and the needs of OLAP clients, you can use `CWM` metadata or `CWM2` metadata, or some combination of both.

Important: You cannot create or view `CWM2` metadata from Enterprise Manager's OLAP Manager. Conversely, you cannot create `CWM` metadata using the `CWM2` APIs.

However, `CWM2` includes a set of union views that allows you to view the OLAP metadata in both the `CWM` and `CWM2` model tables.

A separate set of `CWM2` views allows you to view metadata that directly represents objects in analytic workspaces.

Logical Steps for Creating the Metadata

Whether you create OLAP metadata programmatically (`CWM2`) or by using Oracle Enterprise Manager (`CWM`), 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.

Creating Metadata Using Oracle Enterprise Manager

If your data warehouse complies with the requirements listed in "Basic Star or Snowflake Schema" on page 5-2, 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 5-11.

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 **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. 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 4-11. 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 5-7.
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. 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.

Cubes and measures are defined entirely in the OLAP metadata; there are no corresponding database objects. In Oracle OLAP, measures are created in which to cache the data for analysis and display.

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 5-7.
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 Cubes

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 3-12. 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. Enterprise Manager's OLAP Management feature uses the OLAP Catalog Release 1 APIs (`CWM`) and can only access metadata in the `CWM` metadata repository.

Procedure: Viewing cubes

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 5-7.
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

OLAP Catalog Release 2 includes a number of related PL/SQL packages for creating CWM2 metadata. These packages contain stored procedures that can create metadata for a wide variety of data warehouses. For example, you can create metadata for parent/child dimension tables and for data stored in analytic workspaces. Before using these packages, make sure that you have performed any required preprocessing steps, as listed in "Choosing the Right Metadata Creation Method" on page 5-2.

CWM2 Packages

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

- `CWM2_OLAP_DIMENSION` contains procedures for creating OLAP dimensions and providing display names and descriptions.
- `CWM2_OLAP_HIERARCHY` contains procedures for creating OLAP hierarchies and providing display names and descriptions.
- `CWM2_OLAP_LEVEL` contains procedures for creating levels and providing display names and descriptions.
- `CWM2_OLAP_LEVEL_ATTRIBUTE` contains procedures for creating level attributes and providing display names and descriptions.
- `CWM2_OLAP_DIMENSION_ATTRIBUTE` contains procedures for creating dimension attributes and providing display names and descriptions.

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

- `CWM2_OLAP_CUBE` contains procedures for creating cubes and providing cube display names and descriptions.
- `CWM2_OLAP_MEASURE` contains procedures for creating and deleting measures and providing display names and descriptions.

The following package contains procedures that create metadata for analytic workspace objects:

- `CWM2_OLAP_AW_OBJECT` contains procedures for creating AW objects.

The following packages contain procedures that create the mapping between logical metadata entities and the warehouse structures where the data is stored:

- CWM2_OLAP_TABLE_MAP contains procedures that map metadata entities to relational fact tables and dimension tables.
- CWM2_OLAP_AW_MAP contain procedures that map metadata entities to multidimensional objects in analytic workspaces.

See Also: Part IV, "OLAP Catalog Metadata API Reference" for comprehensive syntax and descriptions of these packages.

Example 5–1 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
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 PL/SQL calls create a logical metadata dimension object, PRODUCT_DIM, for the PRODUCTS dimension table.

```
---   Create the PRODUCT Dimension    ---
cwm2_olap_dimension.create_dimension('SH', 'PRODUCT_DIM', 'Product',
          'Products', 'Product Dimension', 'Product Dimiension 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 5–2 Creating Metadata for a Fact Table

In the Sales History sample schema, COSTS is a fact table with the following columns:

Column Name	Data Type
PROD_ID	NUMBER
TIME_ID	DATE
UNIT_COST	NUMBER
UNIT_PRICE	NUMBER

The following procedures create a logical metadata cube object, ANALYTIC_CUBE, for the COSTS fact table. The dimensions of the cube are: PRODUCT_DIM, shown above, 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;');
```

Part II

Administering Oracle OLAP

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
   ```

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 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 create an analytic workspace from relational tables. Moreover, you can generate relational views of the analytic workspace data, which you can query directly using standard SQL, or use to create OLAP catalog metadata for the OLAP API.

This chapter contains the following topics:

Process Overview

This chapter primarily describes a tool for creating analytic workspaces. The OLAP Catalog metadata defines a logical analysis model and a map to a physical source. If you cannot create OLAP Catalog metadata because your database does not conform to its requirements, you can still create an analytic workspace manually. The basic steps that parallel the stages performed by the `CNV_CWM.TO.ECM` program are also described in this chapter.

The process that you will follow to create an analytic workspace from relational tables is contingent on the design of your schema.

If you have a star or snowflake schema, then you can use the tools described in "Creating an Analytic Workspace Using the CNV_CWM.TO.ECM Program" on page 9-2.

Otherwise, follow the instructions in "Manually Creating an Analytic Workspace" on page 9-4.

> **See Also:** *Oracle9i OLAP Developer's Guide to the OLAP DML* for information about OLAP Worksheet, creating analytic workspaces, aggregating data, and other multidimensional data manipulation techniques.

Creating an Analytic Workspace Using the CNV_CWM.TO.ECM Program

The following steps describe how you might approach the task of creating an analytic workspace using the `CNV_CWM.TO.ECM` program.

> **Note:** To run the `GENSQLOBJS` program, which writes a SQL file, you must have read / write access to a directory alias, as described in "Controlling Access to External Files" on page 6-9.

1. If you have not yet created OLAP metadata, then you must do that first, as described in Chapter 5, "Creating OLAP Catalog Metadata". The metadata maps the fact tables and dimension tables of your database to multidimensional objects: measures, dimensions, attributes, levels, and so forth. These object types are defined in Chapter 4, "Designing Your Database for OLAP".
2. If you plan to fetch only part of your relational schema into your analytic workspace, then identify exactly which objects you want to use.

3. Open an OLAP session using one of the methods described in "Methods of Executing OLAP DML Commands" on page 2-8.

4. Create and detach an analytic workspace. Unless you have strong reasons for doing otherwise, you should begin with an empty workspace. The following command creates an analytic workspace named `sales` in the `olapts` tablespace:

   ```
   AW CREATE sales TABLESPACE olapts
   AW DETACH sales
   ```

   `CNV_CWM.TO.ECM` will create an analytic workspace if one does not exist already, but it will be created in the default tablespace. The performance of an analytic workspace is better if it is stored in a tablespace that has been created specifically for that purpose.

5. Execute `CNV_CWM.TO.ECM`.

   The following example attaches an analytic workspace named `sales` and creates the dimensions, attributes, and hierarchies associated with the `SALES_QUANTITY` measure.

   ```
   CALL CNV_CWM.TO.ECM('sales' 'na' 'na' -
        'MEASURE::SH::SALES_CUBE::SALES_QUANTITY')
   ```

   See "CNV_CWM.TO.ECM Program" on page 9-6 for the complete syntax.

   The `CNV_CWM.TO.ECM` utility loads only the base-level data stored in relational tables. It does not load any aggregate data.

6. Create aggregation maps and generate aggregate data by using the `AGGREGATE` command.

   You now have an analytic workspace. SQL-based applications can use the `OLAP_TABLE` function for direct access to the data.

7. Run the `GENSQLOBJS` program to generate SQL scripts for creating relational views of the analytic workspace data. (Optional)

8. Generate relational views of the multidimensional data by running the SQL scripts. (Optional)

   SQL-based applications can run directly against these views using standard SQL commands, and thus have access to the workspace data.

9. You can now create OLAP catalog metadata if you wish, so that the OLAP API can access the multidimensional data in the analytic workspace. Refer to Chapter 5, "Creating OLAP Catalog Metadata". (Optional)

Manually Creating an Analytic Workspace

If your database design does not allow you to use the `CNV_CWM.TO.ECM` program, then you can still develop an analytic workspace from relational tables. However, you will need to use the various programs and procedures that underlie `CNV_CWM.TO.ECM`. Their use requires greater familiarity with the OLAP DML.

> **See Also:** *Oracle9i OLAP Developer's Guide to the OLAP DML* for information about creating analytic workspaces using the SQL command in the OLAP DML.

The following are the basic steps.

1. Browse your database schema and identify the names of the tables and columns whose data you want to fetch into an analytic workspace. Determine which columns will be used as measures, dimensions, and attributes. Refer to Chapter 4, "Designing Your Database for OLAP" for descriptions of multidimensional objects.
2. Open an OLAP session, using one of the methods described in "Methods of Executing OLAP DML Commands" on page 2-8
3. Create an analytic workspace or attach an existing workspace. The following command creates an analytic workspace named `sales` in the `olapts` tablespace:

   ```
   AW CREATE sales TABLESPACE olapts
   ```

4. Define the workspace objects in which you will store the data by using the `DEFINE` command.
5. Fetch data from the relational tables into workspace objects by using the `SQL FETCH` or the `SQL IMPORT` commands.
6. Create aggregation maps and generate aggregate data by using the `AGGREGATE` command.

   You now have an analytic workspace. SQL-based applications can use the `OLAP_TABLE` function for direct access to the data.
7. Generate relational views of the multidimensional data by using the `CWM2_OLAP_AW_ACCESS` PL/SQL package, as described in Chapter 15, "CWM2_OLAP_AW_ACCESS". (Optional)

   SQL-based applications can run directly against these views using standard SQL commands, and thus have access to the workspace data.

8. You can now create OLAP catalog metadata so that the OLAP API can access the workspace data, as described in Chapter 5, "Creating OLAP Catalog Metadata". (Optional)

CNV_CWM.TO.ECM Program

`CNV_CWM.TO.ECM` is a DML program that creates analytic workspace objects from OLAP catalog metadata and loads data into these objects. It runs in eight stages, and updates the analytic workspace after completing each stage. It does not rerun stages whose results are already saved in the analytic workspace. You can run `CNV_CWM.TO.ECM` in stages, check the results, and modify them before continuing to the next stage.

Return Value

None

Syntax

```
CNV_CWM.TO.ECM(aw_name, [stage], ['DEBUG'], [selection], [directory, filename]
```

where:

`stage` is a text expression identifying one of the following keywords:

```
GET_CWM_METADATA
GEN_RDB_STRUCTURES
GET_RDB_STRUCTURES
CRT_DIM_STRUCTURES
DFN_MEASURES
CRT_ECM_METADATA
GEN_MEASURES
GET_MEASURES
```

`selection` is a text expression in one of the following formats:

```
CATALOG::catalog
CUBE::owner::cube
MEASURE::owner::cube::measure
```

Arguments

aw_name
A text expression that identifies the name of the analytic workspace in which the objects will be created and stored. If the workspace exists, then it is attached read/write. (Note that it cannot already be attached to your session.) If it does not exist, then a new analytic workspace is created.

The `workspace` should be used only to store objects created by this program. Custom DML programs and data acquired from other sources can be stored in separate analytic workspaces.

stage
By specifying a stage, you interrupt the utility so that you can review and edit the contents of the analytic workspace before continuing. For example, the definition of a variable identifies its dimensions, and the order in which the dimensions are listed, from fastest to slowest varying, indicates how the data is stored. Depending on how users will most frequently view the data, you might want to change the order of the dimensions. If the data is sparse, you might want to create a composite dimension and use it to dimension the variable.

The following are descriptions of the various stages:

’GET_CWM_METADATA’
Stage 1: Loads all of the metadata from the database into objects in the analytic workspace.

’GEN_RDB_STRUCTURES’
Stage 2: Generates DML programs to fetch level values, attributes, and parent-child relationships.

’GET_RDB STRUCTURES’
Stage 3: Executes the DML programs created in stage 2.

’CRT_DIM_STRUCTURES’
Stage 4: Creates analytic workspace dimensions, attributes, and hierarchies.

’DFN_MEASURES’
Stage 5: Generates and executes DML programs that determine sparsity patterns and define measures and composites.

’CRT_ECM_METADATA’
Stage 6: Creates workspace objects.

’GEN_MEASURES’
Stage 7: Generates DML programs to load data from the fact tables.

’GET_MEASURES’
Stage 8: Executes the DML programs created in stage 7. (Default)

’DEBUG’
A text expression that specifies running in debug mode. In debug mode, the workspace is attached read-only and remains attached at the end of the build. You cannot save the analytic workspace when it runs in this mode.

When this argument is NA, `CNV_CWM.TO.ECM` runs in standard mode. The workspace is attached read / write, and all objects are saved at the completion of each stage. When `CNV_CWM.TO.ECM` is done, it detaches the workspace.

CATALOG
Limits the build to include only the metadata associated with a particular catalog or metadata folder. When this argument is omitted, all OLAP metadata in the schema is included in the build.

CUBE
Limits the build to include only the metadata for *cube*. When this argument is omitted, all cubes in *catalog* are included in the build.

MEASURE
Limits the build to include only the metadata for a particular measure. When this argument is omitted, all measures in *cube* are included in the build.

catalog
Identifies a particular measure folder in the schema.

owner
Identifies the owner of the schema that contains the dimension tables and fact tables.

cube
Identifies a particular cube in *catalog*.

measure
Identifies a particular measure in *cube*.

Notes

Using multiple analytic workspaces: The analytic workspace created by `CNV_CWM.TO.ECM` should be used only for this purpose so that you can periodically delete and recreate it. You can store DML programs and data from other sources in separate analytic workspaces. A session can attach numerous analytic workspaces, and objects in different analytic workspaces are fully accessible to each other.

Recreating an analytic workspace: `CNV_CWM.TO.ECM` rebuilds an analytic workspace. If you have customized a workspace that was created by `CNV_CWM.TO.ECM`, then you should export all customizations before rebuilding.

1. Export your customizations to an EIF file, using a DML command like the following one:

   ```
   export myprogs to eif file 'temp/dmlprogs.eif' rewrite
   ```

   Where:

   `temp` is the name of a directory alias to which you have write access.

   `dmlprogs.eif` is the name of the EIF file.

2. Delete the old analytic workspace. Then create a new analytic workspace and populate it using `CNV_CWM.TO.ECM`.
3. Import the custom objects from the EIF file into the original workspace, using a command like the following one:

   ```
   import all from eif file 'temp/dmlprogs.eif' dfns
   ```

4. Issue `UPDATE` and `COMMIT` commands.

Creating aggregate data: The `CNV_CWM.TO.ECM` program reads and loads only stored low-level source data from relational tables into the analytic workspace, and creates the objects that define and support dimension hierarchies. No aggregate data is loaded into the analytic workspace. The OLAP DML has very sophisticated tools for aggregating data. Refer to the chapter on aggregating data in the *Oracle9i OLAP Developer's Guide to the OLAP DML*.

Examples

This example creates or attaches an analytic workspace named `alldata` and creates all of the objects defined by the OLAP metadata for the current schema.

```
CALL CNV_CWM.TO.ECM('alldata')
```

The next example creates or attaches an analytic workspace named `saleshist` and creates all of the objects defined by the OLAP metadata for the `SH_CAT` catalog.

```
CALL CNV_CWM.TO.ECM('saleshist', na, na, 'CATALOG::SH_CAT')
```

The following example creates or attaches an analytic workspace named `sales` and creates the dimensions, attributes, and hierarchies associated with the `SALES_QUANTITY` measure.

```
CALL CNV_CWM.TO.ECM('sales', 'CRT_DIM_STRUCTURES', na, -
     'MEASURE::SH::SALES_CUBE::SALES_QUANTITY')
```

GENSQLOBJS Program

GENSQLOBJS is an OLAP DML program that generates a SQL script. The script uses the OLAP_TABLE function to generate views of multidimensional workspace data. The analytic workspace must contain ECM-type metadata, whether generated by CNV_CWM.TO.ECM or provided in a legacy Express database. The resulting dimension views and measure views comprise a star schema that represents the analytic workspace.

You can use the resulting relational views in two ways:

- Create OLAP metadata so that the OLAP API can access data in the analytic workspace.
- Use standard SQL to access data in the analytic workspace.

See Also: Chapter 15, "CWM2_OLAP_AW_ACCESS" for information about generating views of analytic workspace data.

Return Value

None

Syntax

```
GENSQLOBJS(aw_name, directory, filename [, language])
```

Arguments

aw_name
A text expression that provides the name of an ECM-type analytic workspace.

directory
A text expression that provides the name of the directory alias where the SQL script will be created. For information about obtaining access to a directory alias, see "Controlling Access to External Files" on page 6-9.

filename
A text expression that specifies the base name of the output script file. An extension of .sql will be appended to this base name.

language
A text expression that identifies the desired language dimension value. Required only when there are workspace objects that support multiple languages. For example, a variable with dimension labels might have a language dimension with values for English, French, and Spanish. When generating a relational view, you must identify which language you want the view to support.

Examples

The following command creates a SQL script with a name of `salesdat.sql` from the `sales` analytic workspace. The files are stored in the directory identified by the `scripts` directory alias. The workspace language dimension will be limited to `ENGLISH`.

```
CALL GENSQLOBJS('sales', 'scripts', 'salesdat', 'ENGLISH')
```

Analytic Workspace Metadata Catalogs

You can discover the names of objects by browsing through the metadata catalogs, which are variables named `___CWM.`*`object`*`.CAT` in the analytic workspace. Note that three underscores prefix the name.

Table 9–1 lists the names of the basic catalogs. To see a full listing of the catalogs, issue a `LISTNAMES VARIABLE` command.

Table 9–1 Workspace Metadata Catalogs

Variable Name	Description
`___CWM_CAT.CAT`	Catalog Catalog
`___CWM_CB.CAT`	Cube Catalog
`___CWM.MEAS.CAT`	Measure Catalog
`___CWM.DIM.CAT`	Dimension Catalog
`___CWM.HIER.CAT`	Hierarchy Catalog
`___CWM.LVL.CAT`	Level Catalog
`___CWM.LVLA.CAT`	Level Attribute Catalog

Catalog Catalog

The Catalog catalog provides detailed information about each measure folder. Each folder or catalog is represented by a member of the `___cwm.cat.ent` dimension. The following example shows the entries for the `xademo_cat` catalog.

```
report ___cwm.cat.cat

                     -----___CWM.CAT.CAT------
                     -----___CWM.CAT.ENT------
___CWM.CAT.PRP                  95
-------------------- ------------------------
CATALOG_NAME         XADEMO_CAT
PARENT_CATALOG_ID    NA
DESCRIPTION          XADEMO CWM Business Area
```

Cube Catalog

The Cube catalog provides detailed information about each cube. Each cube in the xademo schema is represented by a member of the ___cwm.cb.ent dimension. The following example shows the catalog entries for analytic_cube. The Description field shows that this cube contains five measures (sales, quota, cost, units, and promo), which are dimensioned by time, channel, product, and geography.

```
report ___cwm.cb.cat

                     ----------------___CWM.CB.CAT----------------
                     ----------------___CWM.CB.ENT----------------
___CWM.CB.PRP                               1
-------------------- ---------------------------------------------
OWNER                XADEMO
CUBE_NAME            ANALYTIC_CUBE
DISPLAY_NAME         Analytics
DESCRIPTION          Sales, Quota, Cost, Units, Promo <TIME
                     CHANNEL PRODUCT GEOGRAPHY>
CBID                 6 13 XADEMO ANALYTIC_CUBE
```

Measure Catalog

The Measure catalog provides detailed information about measures. Each measure in the xademo schema is represented by a member of the ___cwm.meas.ent dimension.

The following example lists the catalog entries for the f.sales measure. They show that the data from the sales column of the xademo_analytic_facts table is mapped to a workspace variable named __meas5. Note that two underscores begin the name.

```
report ___cwm.meas.cat

                     ---------------___CWM.MEAS.CAT---------------
                     ---------------___CWM.MEAS.ENT---------------
___CWM.MEAS.PRP                             5
-------------------- ---------------------------------------------
OWNER                XADEMO
CUBE_NAME            ANALYTIC_CUBE
MEASURE_NAME         F.SALES
DISPLAY_NAME         F.SALES
DESCRIPTION          Dollar Sales
DATA_TYPE            DECIMAL
```

```
FACT_TABLE_OWNER     XADEMO
FACT_TABLE_NAME      XADEMO_ANALYTIC_FACTS
COLUMN_NAME          SALES
CBID                 6 13 XADEMO ANALYTIC_CUBE
MEASID               6 13 7 XADEMO ANALYTIC_CUBE F.SALES
EXPRESS_OBJ_NAME     __MEAS5
```

Dimension Catalog

The Dimension catalog provides detailed information about dimensions. Each dimension in the `xademo` schema is represented by a member of the `___cwm.dim.ent` dimension. The following example lists the catalog entries for the `geography` measure. They show that the workspace `geography` dimension is named `__dim3`, and the workspace objects that support the geography hierarchies have an `A3.` prefix.

```
report ___cwm.dim.cat

                               ---___CWM.DIM.CAT---
                               ---___CWM.DIM.ENT---
___CWM.DIM.PRP                          3
------------------------------ --------------------
OWNER                          XADEMO
DIMENSION_NAME                 GEOGRAPHY
DISPLAY_NAME                   Geography
PLURAL_NAME                    Geographys
DESCRIPTION                    Geography Dimension
                               Values
DEFAULT_DISPLAY_HIERARCHY      STANDARD
DIMENSION_TYPE                 NA
DIMID                          6 9 XADEMO GEOGRAPHY
EXPRESS_DIM_NAME               __DIM3
EXPRESS_LVLDIM_NAME            A3.LEVELDIM
EXPRESS_LVLREL_NAME            __LREL3
EXPRESS_HIERDIM_NAME           A3.HIERDIM
EXPRESS_PARENTREL_NAME         A3.PARENTREL
EXPRESS_HIERLVLREL_NAME        A3.LVLREL
EXPRESS_HIERLVLDEPTH_NAME      A3.LVLDEPTHVAR
EXPRESS_GID_NAME               A3.GID
EXPRESS_INHIER_NAME            A3.INHIERARCHY
```

Hierarchy Catalog

The Hierarchy catalog provides detailed information about dimension hierarchies. Each hierarchy of each dimension in the xademo schema is represented by a member of the ___cwm.hier.ent dimension. The following example lists the catalog entries for the two geography hierarchies, standard and consolidated.

```
report ___cwm.hier.cat

                   ------------------___CWM.HIER.CAT------------------
                   ------------------___CWM.HIER.ENT------------------
___CWM.HIER.PRP                4                         5
------------------ ------------------------- -------------------------
OWNER              XADEMO                    XADEMO
DIMENSION_NAME     GEOGRAPHY                 GEOGRAPHY
HIERARCHY_NAME     STANDARD                  CONSOLIDATED
DISPLAY_NAME       Standard                  Consolidated
DESCRIPTION        Standard GEOGRAPHY        Executive Consolidated
                   hierarchy                 GEOGRAPHY hierarchy
DIMID              6 9 XADEMO GEOGRAPHY      6 9 XADEMO GEOGRAPHY
HIERID             6 9 8 XADEMO GEOGRAPHY    6 9 12 XADEMO GEOGRAPHY
                   STANDARD                  CONSOLIDATED
```

Level Catalog

The Level catalog provides detailed information about dimension levels. Each level of each dimension hierarchy is represented by a member of the ___cwm.lvl.ent dimension. The following example lists the catalog entries for the two of the four geography levels. From this catalog, you can learn that geography members at the city level are stored in a workspace dimension named __rdblvldim9, and geography members at the country level are stored in a workspace dimension named __rdblvldim10.

```
report ___cwm.lvl.cat
```

```
                       -----------------------___CWM.LVL.CAT------------------------
                       -----------------------___CWM.LVL.ENT------------------------
___CWM.LVL.PRP                      9                              10
---------------------- ----------------------------- -----------------------------
OWNER                  XADEMO                        XADEMO
DIMENSION_NAME         GEOGRAPHY                     GEOGRAPHY
LEVEL_NAME             L4                            L3
DISPLAY_NAME           Cities                        Countries/Areas
DESCRIPTION            Cities of standard GEOGRAPHY  Countries/Areas of standard
                       hierarchy                     GEOGRAPHY hierarchy
LEVEL_TABLE_OWNER      XADEMO                        XADEMO
LEVEL_TABLE_NAME       XADEMO_GEOGRAPHY              XADEMO_GEOGRAPHY
DIMID                  6 9 XADEMO GEOGRAPHY          6 9 XADEMO GEOGRAPHY
LVLID                  6 9 2 XADEMO GEOGRAPHY L4     6 9 2 XADEMO GEOGRAPHY L3
RDBMS_LVLDIM_NAME      __RDBLVLDIM9                  __RDBLVLDIM10
RDBMS_TO_EXPRESS_LVLREL __RDBEXPREL9                 __RDBEXPREL10
_NAME
```

Level Attribute Catalog

The Level Attribute catalog provides detailed information about level attributes. Each level attribute for each dimension level is represented by a member of the `___cwm.lvla.ent` dimension. This dimension can be quite large, so you might want to use a command like the following to limit your view to the level attributes for a single dimension:

```
limit ___cwm.lvla.ent to (___cwm.lvla.cat(-
     ___cwm.lvla.prp, 'DIMENSION_NAME') eq 'GEOGRAPHY')
```

The following example lists the catalog entries for one of the geography level attributes. From this catalog, you can learn that the long names for cities are stored in a variable named `__rdbattrvar19.`

```
report ___cwm.lvla.cat
```

```
                           ----------___CWM.LVLA.CAT----------
                           ----------___CWM.LVLA.ENT----------
___CWM.LVLA.PRP                           19
------------------------- -----------------------------------
OWNER                      XADEMO
DIMENSION_NAME             GEOGRAPHY
ATTRIBUTE_NAME             GEOG_STD_CITY_LLABEL
DISPLAY_NAME               Long City Name(s)
DESCRIPTION                Long Labels for Cities values of
                           the STANDARD GEOGRAPHY hierachy
DETERMINED_BY_LEVEL_NAME   L4
```

```
COLUMN_NAME                 GEOG_STD_CITY_LLABEL
DATA_TYPE                   TEXT
DIMID                       6 9 XADEMO GEOGRAPHY
LVLID                       6 9 2 XADEMO GEOGRAPHY L4
LVLAID                      6 9 2 20 XADEMO GEOGRAPHY L4
                            GEOG_STD_CITY_LLABEL
RDBMS_ATTRVAR_NAME          __RDBATTRVAR19
```

References to OLAP Catalog Metadata

When you are viewing the catalogs, it will help you to understand how references are formed to OLAP catalog metadata definitions in the database. They are constructed as a concatenation of the number of characters for each part of the name followed by the various parts. References to various metadata objects that appeared in "Analytic Workspace Metadata Catalogs" are described here.

Cubes

The CBID property in the catalogs references the cubes defined in the OLAP catalog. It has the following format:

```
owner_chars cube_chars owner_name cube_name
```

The following example identifies a cube owner of XADEMO (6 characters) and a cube name of ANALYTIC_CUBE (13 characters).

```
6 13 XADEMO ANALYTIC_CUBE
```

Measures

The MEASID property in the catalogs references the measures defined in the OLAP catalog. It has the following format:

```
owner_chars cube_chars meas_chars owner_name cube_name meas_name
```

The following example identifies a cube owner of XADEMO (6 characters), a cube name of ANALYTIC_CUBE (13 characters), and a measure name of F_SALES (7 characters):

```
6 13 7 XADEMO ANALYTIC_CUBE F.SALES
```

Dimensions

The DIMID property in the catalogs references the dimensions defined in the OLAP catalog. It has the following format:

```
owner_chars dim_chars owner_name dim_name
```

The following example identifies a dimension owner of XADEMO (6 characters) and a dimension name of GEOGRAPHY (9 characters):

```
6 9 XADEMO GEOGRAPHY
```

Hierarchies

The HIERID property in the catalogs references the hierarchies defined in the OLAP catalog. It has the following format:

```
owner_chars dim_chars hier_chars owner_name dim_name hier_name
```

The following example identifies a dimension owner of XADEMO (6 characters), a dimension name of GEOGRAPHY (9 characters), and a hierarchy name of STANDARD (8 characters):

```
6 9 8 XADEMO GEOGRAPHY STANDARD
```

Levels

The LVLID property in the catalogs references the hierarchy levels defined in the OLAP catalog. It has the following format:

```
owner_chars dim_chars lvl_chars owner_name dim_name lvl_name
```

The following example identifies a dimension owner of XADEMO (6 characters), a dimension name of GEOGRAPHY (9 characters), and a level name of L3 (2 characters):

```
6 9 2 XADEMO GEOGRAPHY L3
```

Level Attributes

The LVLAID property in the catalogs references the level attributes defined in the OLAP catalog. It has the following format:

```
owner_chars dim_chars lvl_chars attr_chars owner_name
     dim_name lvl_name attr_name
```

The following example identifies a dimension owner of `XADEMO` (6 characters), a dimension name of `GEOGRAPHY` (9 characters), a level of `L4` (2 characters), and an level attribute name of `GEOG_STD_CITY_LLABEL` (20):

```
6 9 2 20 XADEMO GEOGRAPHY L4 GEOG_STD_CITY_LLABEL
```

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:

10 DBMS_AW

This chapter contains reference information for the DBMS_AW package. Using the procedures and functions in the DBMS_AW package, SQL programmers can issue OLAP DML statements against analytic workspace data. They can move data from relational tables into the analytic workspace, perform advanced analysis (for example, forecasting), and move data from the analytic workspace back into relational tables. Also, once the data is in the analytic workspace, SQL programmers can issue SELECT statements against the data in the analytic workspace using the OLAP_TABLE function.

The DBMS_AW package also includes procedures and functions that SQL programmers can use to retrieve and print the session logs created by the execution of the OLAP_TABLE function or the procedures and functions in the DBMS_AW package.

This chapter includes the following topics:

Summary of DBMS_AW Subprograms

The following table describes the subprograms provided in DBMS_AW.

Table 10–1 DBMS_AW Subprograms

Subprogram	Description
"EXECUTE Procedure" on page 10-3	Executes one or more OLAP DML commands (input as a VARCHAR2 string) and prints the output of the OLAP DML commands (if any) using the DBMS_OUTPUT package.
"INTERP Function" on page 10-6	Executes one or more OLAP DML commands (input as a VARCHAR2 string) and returns the session log in which the commands were executed.
"INTERPCLOB Function" on page 10-8	Executes one or more OLAP DML commands (input as a CLOB) and returns the session log in which the commands were executed.
"GETLOG Function" on page 10-10	Returns the session log from the last execution of the DBMS_AW.INTERP function, the DBMS_AW.INTERPCLOB function, or the OLAP_TABLE function.
"PRINTLOG Procedure" on page 10-11	Prints a session log using the DBMS_OUTPUT package.

EXECUTE Procedure

This procedure executes one or more OLAP DML commands (input as a VARCHAR2 string) and prints the output of the OLAP DML commands (if any) using the DBMS_OUTPUT package.

> **See Also:** For the syntax of individual OLAP DML commands, see Oracle9i OLAP DML Reference help. For a information on analytic workspace objects, see *Oracle9i OLAP Developer's Guide to the OLAP DML.*

Syntax

The syntax for the DBMS_AW.EXECUTE procedure is shown below.

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

Parameters

Table 10–2 EXECUTE Procedure 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.

Executing Large Numbers of OLAP DML Commands

Since the olap-commands parameter of EXECUTE procedure is of type VARCHAR2, you are limited to 4,000 bytes for OLAP DML commands. For larger values, use the

"INTERPCLOB Function" on page 10-8 which allows you to input the OLAP DML commands as a CLOB.

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

Executing Multiple OLAP DML Commands

You can specify a number of OLAP DML commands in a single EXECUTE procedure. In this case, you separate each OLAP DML command with a semicolon (;). For example, the following EXECUTE procedure contains multiple lines of code prepares an analytic workspace for access by the OLAP_TABLE function with a complete limit map. It creates a grouping id variable for the Standard hierarchy of geography, execute the OLAP DML code shown below.

```
DBMS_AW.EXECUTE ('AW DETACH xademo;
AW ATTACH xademo ro;
PUSH OKNULLSTATUS;
OKNULLSTATUS = TRUE;
"Create variable for the grouping id;
DEFINE geog.gid INTEGER VARIABLE <geography>;
" Create new parent-child relations for only a single hierarchy;
" of each multidimensional hierarchical dimension;
DEFINE g0.newparent RELATION geography <geography>;
g0.newparent = g0.parent(g0.hierdim 1);
" Populate the groupid variables;
GROUPINGID g0.newparent INTO geog.gid;
" Save changes to analytic workspace;
POP OKNULLSTATUS;
ALLSTAT;
LIMIT g0.hierdim TO 1;
UPDATE;
COMMIT;')
```

Executing a Predefined OLAP DML Program

Assume that you have defined an OLAP DML program named makegroupid that performs the functionality shown in "Executing Multiple OLAP DML Commands" on page 10-4. The following code illustrates using the EXECUTE procedure to execute the makegroupid program.

```
DBMS_AW.EXECUTE (
    'AW DETACH  xademo;
    AW ATTACH  xademo ro;
    CALL makegroupid;
    IN VARCHAR');
```

Executing OLAP DML Commands Using an Infile

You can create a text file that consists only of the OLAP DML commands that you want executed. For example, you could create a text file named makegroupid.inf that had the following lines of code.

```
AW DETACH  xademo
AW ATTACH  xademo ro
CALL makegroupid
```

Assume that you have saved makegroupid.inf into the /users/oracle/sql directory. The following code illustrates using the EXECUTE procedure to execute the OLAP DML commands in xademoprep.inf.

```
-- Attach and prepare xademo analytic workspace
EXECUTE DBMS_AW.EXECUTE('INFILE /users/oracle/sql/makegroupidp.inf')
```

INTERP Function

This function executes one or more OLAP DML commands (input as a VARCHAR2 string) and returns the session log in which the commands are executed.

See Also: For the syntax of individual OLAP DML commands, see Oracle9i OLAP DML Reference help. For a information on analytic workspace objects, see *Oracle9i OLAP Developer's Guide to the OLAP DML.*

Syntax

The syntax for the INTERP function is shown below.

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

Parameters

Table 10–3 DBMS_AW.INTERP Function Parameters

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

Returns

The INTERP function returns a CLOB which is 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 DML commands when you have redirected the output by using the OLAP DML `OUTFILE` command.

Printing the Session Log

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

INTERPCLOB Function

This function executes one or more OLAP DML commands (input as a `CLOB`) and returns the session log in which the commands are executed.

> **See Also:** For the syntax of individual OLAP DML commands, see Oracle9i OLAP DML Reference help. For a information on analytic workspace objects, see *Oracle9i OLAP Developer's Guide to the OLAP DML.*

Syntax

The syntax for the `INTERPCLOB` procedure is shown below.

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

Parameters

Table 10–4 DBMS_AW.INTERPCLOB Function Parameters

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

Returns

The `INTERPCLOB` function returns a `CLOB` which is 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.

Printing the Session Log

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

GETLOG Function

This function returns the session log from the last execution of the `DBMS_AW.INTERP` function, the `DBMS_AW.INTERPCLOB` function, or the `OLAP_TABLE` function.

Syntax

The syntax for the `GETLOG` function is shown below.

```
DBMS_AW.GETLOG();
```

Returns

The `GETLOG` function returns a `CLOB` session log.

Usage Notes

Effect of the OUTFILE Command
The `DBMS_AW.INTERP` and `DBMS_AW.INTERPCLOB` functions do not return the output of the DML commands when you have redirected the output by using the OLAP DML `OUTFILE` command.

Typical use of the GETLOG Function
Since both the `DBMS_AW.INTERP` and the `DBMS_AW.INTERPCLOB` functions return a session log, typically you use the `GETLOG` function to retrieve the session log in which the `OLAP_TABLE` function was executed.

Printing the Session Log
To print the session log returned by this function, use the `DBMS_AW.PRINTLOG` procedure.

PRINTLOG Procedure

This procedure prints a session log using the DBMS_OUTPUT package. You can use this procedure to print a session log returned by the DBMS_AW.INTERP, the DBMS_AW.INTERPCLOB, or the DBMS_AW.GETLOG functions.

Syntax

The syntax for the PRINTLOG procedure is shown below.

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

Parameters

Table 10–5 DBMS_AW.PRINTLOG Procedure Parameters

Parameter	Description
session-log	The log of a session.

Usage Notes

You can use the PRINTLOG procedure to print a session log returned by the DBMS_AW.INTERP, the DBMS_AW.INTERPCLOB, or the DBMS_AW.GETLOG functions.

11 OLAP_TABLE Function

This chapter describes how SQL programers can use the OLAP_TABLE function in a SQL SELECT statement to query multidimensional data in an analytic workspace.

This chapter includes the following topics:

Accessing Multidimensional Data Using the OLAP_TABLE Function

One way to access multidimensional data is by using the OLAP_TABLE function. The OLAP_TABLE function is a predefined table function that returns a table of objects that map to analytic workspace data.

Before you can access multidimensional data using the OLAP_TABLE function, you must take the following steps:

1. Confirm that the analytic workspace has the necessary objects to support the use of the OLAP_TABLE function and define any additional objects as described in "Preparing an Analytic Workspace for Access by OLAP_TABLE" on page 11-3.
2. In SQL, define an object types to represent the analytic workspace structures as a relational table as described in "Creating Object Type Definitions Used by OLAP_TABLE" on page 11-4.

Once you have created type definitions and defined the necessary analytic workspace objects, you can make selections of that data using a SELECT statement with the OLAP_TABLE function which works like a regular table function. By using it within a SELECT statement you can select data from the analytic workspace or create a view of analytic workspace data.

The simplified syntax for using the OLAP_TABLE function in a SQL SELECT statement is shown below:

```
SELECT * FROM TABLE(OLAP_TABLE
     (aw-attach, table-name, olap-command, limit-map))
   AS table-name;
```

Additionally, by coding the OLAP_TABLE function directly in a SELECT statement of a CREATE VIEW command, you can create a relational view of analytic workspace data without using the CWM2_OLAP_AW_ACCESS PL/SQL package.

For the complete syntax of the OLAP_TABLE function, see "Syntax: OLAP_TABLE Function" on page 11-5. For an example of its use, see "Example: Using the OLAP_TABLE Function" on page 11-11.

Preparing an Analytic Workspace for Access by OLAP_TABLE

All hierarchies in an analytic workspace are defined by the following analytic workspace objects:

- A dimension whose values are all of the values of the hierarchy.
- A child-parent self-relation for that dimension whose values are the parents of the dimension values.

When you want to define a complete limit map for `OLAP_TABLE` for a hierarchy or for data that is dimensioned by a hierarchy, you must also define the following analytic workspace objects:

- A Boolean variable that has a value of `TRUE` for each dimension value that is in the hierarchy.
- A dimension whose values are integers that represent the levels of the hierarchy.
- A relation that has been populated using the OLAP DML `HIERHEIGHT` command. This relation represents the values of the hierarchy by level.

  Tip: When the hierarchical dimension is defined as a concat dimension, the values of this relation are the values of the concat dimension. If you want the base dimension values in the relational view, define a variable with the same dimensions as this relation. and populate that variable using the following syntax:

  ```
  variable = BASEVAL(relation)
  ```

- A variable that has been populated using the OLAP DML `GROUPINGID` command. The values of this variable, which is dimensioned by the hierarchy, are the grouping ids for each value of the hierarchy.

For an example of preparing an analytic workspace for access by OLAP_TABLE see, "Example: Using the OLAP_TABLE Function" on page 11-11. For more information on the analytic workspace objects that support the use of hierarchies, see "Hierarchies" on page 15-4.

Creating Object Type Definitions Used by OLAP_TABLE

Creating type definitions that are used by the `OLAP_TABLE` function involves:

1. Designing the objects that will represent the analytic workspace structures.
2. Writing the object type definitions and the table definitions to define the analytic workspace data as a table of objects.

Designing the Objects

Each object type represents a row in a table. When mapping analytic workspace structures to object types, typically, you do not define one object type for each analytic workspace structure. Instead, you map many analytic workspace structures to just a few objects:

- Objects that represent measure tables. All multidimensional analytic workspace structures that share exactly the same dimensions can be mapped into a single object.
- Objects that represent dimension tables. All one-dimensional analytic workspace structures that have exactly the same dimension can be mapped into the object type that you define for that dimension.

For a more complete discussion of the data warehouse designs that you can mimic in your design, see "Data Structures in Relational and Multidimensional Data Stores" on page 4-4.

For each object, you need to identify the attributes that correspond to the columns of the table. To do this, you first need to determine if you want to support the use of `WHERE` clauses when selecting the data. Only those attributes (table columns) that appear in the `limit-map` parameter of the `OLAP_TABLE` function can be referenced in a `WHERE` clause.

Typically, you will want to support the use of `WHERE` clauses. In this case, you need to determine the format of the `limit-map` parameter in order to determine the columns of each table. The columns of each table must correspond exactly to the columns specified in the `limit-map` parameter. For the syntax of the `limit-map` parameter, see "Syntax: OLAP_TABLE Function" on page 11-5.

Creating Type Definitions for Multidimensional Data

To create the type definitions that define the analytic workspace data as a table of objects take the following steps:

1. Create a type definition for an object that represents a row in the table and whose attributes represents the columns of the table. Simplified syntax for this definition is shown below.

   ```
   CREATE TYPE object-name AS OBJECT (
      column-first     data-type,
      column-next      data-type,
      column-last      data-type);
   ```

2. Create a type definition for a table of these objects. Simplified syntax for this definition is shown below.

   ```
   CREATE TYPE table-name AS TABLE OF object-name;
   ```

Syntax: OLAP_TABLE Function

The syntax for the `OLAP_TABLE` function is shown below.

```
OLAP_TABLE (
   aw-attach            IN VARCHAR2,
   table-name           IN VARCHAR2,
   olap-command         IN VARCHAR2,
   limit-map            IN VARCHAR2);
```

The `OLAP_TABLE` function returns a table of objects.

Parameters

aw-attach

An optional text expression that specifies the name of the analytic workspace that contains the data you want to query and whether the analytic workspace is detached after this function executes. The format of `aw-attach` is shown below:

```
[aw-name DURATION QUERY|SESSION]
```

where:

`aw-name` is the name of the analytic workspace that you want attached as the active analytic workspace for Oracle OLAP. When there is one or more analytic workspaces attached, you do not have to specify a value for the `aw_attach` parameter. In this case, Oracle OLAP searches these analytic workspaces for the analytic workspace objects referenced in the `olap-command` and `limit-map` parameters. When you do not specify a value for the `aw_attach` parameter and an analytic workspace is not attached, an error is returned.

The `DURATION` phrase specifies when the analytic workspace specified by `aw-name` is detached. When the analytic workspace specified by `aw-name` is already attached, the `DURATION` phrase is ignored. You can specify either of the following keywords:

- `QUERY` specifies that the analytic workspace is detached after this function executes.
- `SESSION` specifies that the analytic workspace is detached when the connection to the database ends.

table-name
The name of the table of objects that this function returns.

olap-command
A text expression that specifies one or more OLAP DML commands, including an OLAP DML program. When you specify a value for this parameter, Oracle OLAP executes these OLAP commands before it selects the data using the mapping provided by the `limit-map` parameter. When using quotation marks in an OLAP command, follow these guidelines:

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

The main reasons why you specify a value for the `olap-command` parameter is if you do *not* want to support the use of `WHERE` clauses. In this case, you specify a `FETCH` command (or an OLAP DML program that includes a `FETCH` command) for the value of the `olap-command` parameter instead of specifying a value for the `limit-map` parameter. For the syntax of the OLAP `FETCH` command, see the topic for that command in Oracle9i OLAP DML Reference help.

limit-map

A text expression that specifies how Oracle OLAP accesses analytic workspace data. The format of the `limit-map` parameter is shown below:

```
[MEASURE object-attribute FROM aw-measure] ...
DIMENSION [object-attribute FROM] aw-dim [WITH
   [HIERARCHY [object-attribute FROM] aw-parent-rel
        [(aw-hierdim aw-hierdim-value  [[, aw-hierdim aw-hierdim-value] ...])]
      [INHIERARCHY aw-inhier-object]
      [GID object-attribute FROM aw-gid-object]
      [PARENTGID object-attribute FROM aw-gid-object]
      [LEVELREL object-attribute-list FROM aw-level-rel USING aw-level-dim]]
   [[ATTRIBUTE object-attribute FROM aw-attribute] ...]
   [LOOP sparse-dimension]...
[PREDMLCMD olap-command]
[POSTDMLCMD olap-command]
```

`MEASURE` block — Each `MEASURE` block maps the values of one analytic workspace variable (or a function that returns an analytic workspace variable) specified by `aw-measure` to an object attribute (table column) specified by `object-attribute`. All of the analytic workspace variables mapped using a single `MEASURE` block must have exactly the same analytic workspace dimensions.

`DIMENSION` block — Each `DIMENSION` block maps the values of an analytic workspace dimension to one or more object attributes (table columns). When yo are mapping an analytic dimension, you specify that dimension using a single `DIMENSION` block. When you are mapping one or more analytic workspace variables using a `MEASURE` block, you also include one `DIMENSION` block for each dimension of the analytic workspace variables.

The syntax varies depending on whether or not the dimension is a hierarchical dimension.

When mapping a nonhierarchical dimension, use the `object-attribute FROM` clause to map the dimension values to a single object attribute (table column) specified by `object-attribute`.

When mapping a hierarchical dimension, use the `WITH HIERARCHY` clause to map the dimension values to several object attributes (table columns).

The `[object-attribute FROM] aw-parent-rel [(aw-hierdim aw-hierdim-value [[, aw-hierdim aw-hierdim-value]...])` subclause maps the values of an hierarchical analytic workspace dimension to columns in the relational object.

`object-attribute` is the name of the object attribute (relational table column) to which you want the analytic workspace value mapped.

`aw-parent-rel` is a relation that is dimensioned by `aw-dim` and all of the `aw-hierdim` dimensions. The values of `aw-parent-rel` are the values of the `aw-dim` dimension. For each unique combination of dimension values, `aw-parent-rel` has the value of `aw-dim` that is the parent. When `aw-parent-rel` represents more than one hierarchy, you use the `aw-hierdim` parameter to qualify it to a single hierarchy.

`aw-hierdim` is the name of a dimension whose values are the names of hierarchies and `aw-hierdim-value` is the name of a particular hierarchy.

`aw-inhier-object` is the name of an analytic workspace variable or relation whose non-NA values indicate membership in the hierarchy being mapped. `aw-inhier-object` must have the same dimensions as `aw-parent-rel` and `aw-gid-object` (that is, it must be dimensioned by `aw-dim` and all of the `aw-hierdim` dimensions).

- `The GID object-attribute FROM aw-gid-object` subclause maps the grouping ids of the children of an hierarchical analytic workspace dimension to a column in the relational object.

  `object-attribute` is the name of the object attribute (relational table column) to which you want the analytic workspace value mapped.

  `aw-gid-object` is the name of an analytic workspace variable whose values are the grouping ids for the hierarchy or the name of an analytic workspace relation that returns a `Number` dimension whose values are the grouping ids for the hierarchy. `aw-gid-object` must have the same dimensions as `aw-parent-rel` and `aw-inhier-object` (that is, it must be dimensioned by `aw-dim` and all of the `aw-hierdim` dimensions). When the `aw-gid-object` does not exist in the analytic workspace, you can create it using the OLAP `GROUPINGID` command. (For the syntax of `GROUPINGID`, see the topic for the command in the Oracle9i OLAP DML Reference help.)

- The `PARENTGID object-attribute FROM aw-gid-object sub`clause maps the grouping ids of the parent values of an hierarchical analytic workspace dimension to a column in the relational object.

  `object-attribute` is the name of the object attribute (relational table column) to which you want the analytic workspace value mapped.

  `aw-gid-object` is the name of an analytic workspace variable whose values are the grouping ids for the hierarchy or the name of an analytic workspace relation that returns a `Number` dimension whose values are the grouping ids for the hierarchy. `aw-gid-object` must have the same dimensions as `aw-parent-rel` and `aw-inhier-object` (that is, it must be dimensioned by `aw-dim` and all of the `aw-hierdim` dimensions). When the `aw-gid-object` does not exist in the analytic workspace, you can create it using the OLAP `GROUPINGID` command. (For the syntax of `GROUPINGID`, see the topic for the command in the Oracle9i OLAP DML Reference help.)

- The `LEVELREL object-attribute-list FROM aw-level-rel USING aw-level-dim` subclause specifies how to map the values of a single hierarchical dimension into several columns of the relational table. There is one column in the table for each level of the analytic workspace hierarchy.

  `object-attribute-list` is a list of the names of object attributes that represent columns of the relational table to which you want the values mapped. Specify one attribute (column) name for each level in the hierarchy. Separate the names of the attributes (columns) using commas. The order must be the same as the order specified in `aw-level-relation`.

  `aw-level-rel` is the name of an analytic workspace relation that is dimensioned by `aw-dim`, `aw-level-dim`, and all `aw-hierdim` dimensions. The values of `aw-level-rel` are the numerical values that represent the levels of the dimension. These are the values that will be in the table columns specified by `object-attribute-list`. When `aw-level-rel` does not exist in the analytic workspace, you can create it using the OLAP `HEIRHEIGHT` command. In this case, you can specify if you want the attributes in descending (the default) or ascending order. (For the syntax of `HIERHEIGHT`, see the topic for the command in the Oracle9i OLAP DML Reference help.)

  `aw-level-dim` is the name of an integer dimension whose values are 1 through the highest-level of hierarchy in the analytic workspace.

- The `ATTRIBUTE object-attribute FROM aw-attribute` subclause specifies how dimension table attributes are mapped.

  `object-attribute` is the name of the object attribute (relational table column) to which you want the analytic workspace value mapped.

  `aw-attribute` is the name of an analytic workspace relation, variable, or formula that is dimensioned *only* by the analytic workspace dimension specified by `aw-dim.`

- The `LOOP sparse-dimension` subclause specifies how the `OLAP_TABLE` function loops over the values of `aw-measure` to retrieve its values. You can specify this subclause for only one dimension of a measure. Using this subclause causes the function to loop sparsely over `aw-measure` using the analytic workspace composite, conjoint or multidimensional dimension specified by `sparse-dimension` rather than looping over `aw-measure` densely using the object specified by `aw-dim`. Typically, the composite specified for `sparse-dimension` is one by which measure is dimensioned.

`PREDMLCMD` and `POSTMLCMD` blocks —`PREDMLCMD` is an optional block that specifies an OLAP command to be executed before the data is fetched. `POSTDMLCMD` is an optional block that specifies an OLAP command to be executed after the data is fetched. In both blocks, the `olap-command` parameter specifies a text expression that specifies one or more OLAP commands, including an OLAP DML program. When using quotation marks in an OLAP command, follow these guidelines:

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

Keep the following points in mind when creating a limit map:

- Many of the clauses in a limit map are optional. However, only those object attributes (relational table columns) that are mapped to analytic workspace objects can effectively be used in SQL `SELECT` statements and `WHERE` clauses. When an object attribute (column) is referenced in a `SELECT` statement or a `WHERE` clause but not in the limit map, that portion of the `WHERE` clause is ignored when retrieving the data.
- A given analytic workspace object or object attribute (table column) can only be referenced once within the limit map.
- Any multidimensional analytic workspace relation must be fully qualified down to a single hierarchy.

Tip: To retrieve the Oracle OLAP session log from the last execution of the `OLAP_TABLE` function, use the `DBMS_AW.GETLOG` function. You can print the session log returned by this function by using the `DBMS_AW.PRINTLOG` procedure.

Example: Using the OLAP_TABLE Function

In the discussion on working with relational tables in *Oracle9i OLAP Developer's Guide to the OLAP DML* there is an example of creating an analytic workspace named awsh from the sample Sales History database. Assume that you want to create relational views of the cost data in the awsh analytic workspace

For cost data, the Sales History database (which is fully described in *Oracle9i Sample Schemas*) has a fact table named costs that contains columns for its keys (product_id and time_id) and columns for facts (unit_cost and unit_price). The keys of costs are primary keys of the products dimension table and the times dimension table. The products table represents one hierarchy with four levels (prod_id, prod_subcategory, prod_category, and products_all). The times table represents two hierarchies: Calendar Time and Fiscal. The Calendar Time hierarchy has five levels: date, calendar week, calendar month, calendar quarter, and calendar year. The Fiscal Time hierarchy has five levels: date, fiscal week, fiscal month, fiscal quarter, and fiscal year.

In the awsh analytic workspace, the Products and Times hierarchies are defined as a number of analytic workspace objects as illustrated in Example 11–1, "Definitions for Cost Data in the awsh Analytic Workspace" on page 11-12. For the Products hierarchy, there is a dimension for each level of a hierarchy, a concat dimension named aw_products for all of the levels, and a child-parent self-relation for the concat dimension. When designing awsh, it was determined that applications only need to summarize or aggregate data from at the year level. Consequently, there are only three levels in the Time hierarchies in the analytic workspace: time id, fiscal year, and calendar year. To define for the Time hierarchies, there is an analytic workspace dimension containing the names of the two hierarchies (Calendar and Fiscal), a dimensions for each of the three levels (time, fiscal year, and calendar year), a concat dimension named aw_times for all of the levels, and a child-parent self-relation for the concat dimension. Since there are two time hierarchies the child-parent self-relation created for aw_times is dimensioned by both the concat dimension and the hierarchies (by name). The facts in the costs table are defined as analytic workspace variables dimensioned by a composite (named aw_costsdims) of aw_products and aw_times.

To prepare awsh for access by OLAP_TABLE, you need to add the analytic workspace object definitions in Example 11–2, "Definitions for Additional Analytic Workspace Objects" on page 11-13. Example 11–3, "Preparing the awsh Analytic Workspace for OLAP_TABLE" on page 11-14 illustrates an OLAP DML program that populates these analytic workspace objects using the HEIRHEIGHT command, the GROUPINGID command, and other OLAP DML commands.

After all of the necessary analytic workspace objects are defined and populated you can use of the OLAP_TABLE function to create a relational view of the cost data in awsh. Example 11–4, "Creating Views Using the OLAP_TABLE Function" on page 11-15 is a script that creates these views.

- The aw_products_view is a view of the Products hierarchy. It includes a column for all of the values in the hierarchy, a column for the parent of each of the values in the hierarchy, columns for the values of each level in the hierarchy and, a column for the grouping id of each value in the hierarchy. The columns that contain the values of the levels are mapped to analytic workspace data using the LEVELREL clause of the limit-map parameter of the OLAP_TABLE function.
- The aw_times_view is a view of the Times hierarchies. It includes a column for all of the values in the hierarchy, a column for the parent of each values in the hierarchy, and columns for the values of each level in the hierarchy. The columns that contain the values of the levels are mapped to analytic workspace data using the ATTRIBUTE phrases in the limit-map parameter of the OLAP_TABLE function. Each ATTRIBUTE phrase maps a column in the view to a analytic workspace formula that retrieves the values of one level of the aw_times dimension.
- The aw_costs_view is a view of the actual cost facts (unit_price and unit_cost) and the analytic workspace dimensions that act as the keys to these facts.

Example 11–1 Definitions for Cost Data in the awsh Analytic Workspace

```
DEFINE aw_prod_id DIMENSION NUMBER (6)
DEFINE aw_prod_subcategory DIMENSION TEXT
DEFINE aw_prod_category DIMENSION TEXT
DEFINE aw_products_all DIMENSION TEXT
DEFINE aw_products DIMENSION CONCAT (aw_products_all -
                                  aw_prod_category -
                                  aw_prod_subcategory -
                                  aw_prod_id)
DEFINE aw_products.parents RELATION aw_products <aw_products>

DEFINE aw_time_id DIMENSION TEXT
DEFINE aw_cal_year DIMENSION NUMBER(4)
DEFINE aw_fis_year DIMENSION NUMBER(4)
DEFINE aw_times DIMENSION CONCAT (aw_cal_year -
                                  aw_fis_year -
                                  aw_time_id)
DEFINE aw_times_hiernames DIMENSION TEXT
DEFINE aw_times.parents RELATION aw_times <aw_times aw_times_hiernames>

DEFINE aw_costsdims COMPOSITE <aw_products aw_times>
DEFINE aw_unit_cost VARIABLE  NUMBER (10,2) <aw_costsdims -
        <aw_products aw_times>>
DEFINE aw_unit_price VARIABLE NUMBER (10,2) <aw_costsdims -
        <aw_products aw_times>>
```

Example 11–2 Definitions for Additional Analytic Workspace Objects

```
DEFINE temp_levelnames DIMENSION TEXT
LD A dimension used to sort names of levels
DEFINE aw_products_levelnames DIMENSION TEXT
LD Names of levels of the Products hierarchy
DEFINE aw_products_levelnums DIMENSION INTEGER
LD Levels of the Products hierarchy identified by number
DEFINE aw_times_levelnames DIMENSION TEXT
LD Names of levels of the Times hierarchy
DEFINE aw_times_levelnums DIMENSION INTEGER
LD Levels of the Times hierarchy identified by number
DEFINE aw_products_gid VARIABLE INTEGER <aw_products>
LD Levels of the Products hierarchy identified by GID
DEFINE aw_products.aw_products_levelnums RELATION aw_products <aw_products
aw_products_levelnums>
LD Concat dimension values for the Products hierarchy by level number
DEFINE aw_times.aw_times_levelnums RELATION aw_times <aw_times
aw_times_hiernames aw_times_levelnums>
LD Concat dimension values for the Times hierarchy by level number and hierarchy
DEFINE aw_products_basevalues VARIABLE aw_products <aw_products
aw_products_levelnums>

DEFINE AW_FISCAL_YEAR_FORM FORMULA -
     BASEVAL(aw_times.aw_times_levelnums(aw_times_levelnums 2 -
       aw_times_hiernames ’Fiscal’))
LD Formula that returns the base values of fiscal year
DEFINE AW_CALENDAR_YEAR_FORM FORMULA -
      BASEVAL(aw_times.aw_times_levelnums(aw_times_levelnums 1 -
         aw_times_hiernames ’Calendar’))
LD Formula that returns the base values of calendar year
```

```
DEFINE AW_TIMEID_FORM FORMULA -
       BASEVAL(aw_times.aw_times_levelnums(aw_times_levelnums 3 -
              aw_times_hiernames 'Calendar'))
LD Formula that returns the values of time_id
```

Example 11–3   Preparing the awsh Analytic Workspace for OLAP_TABLE

```
" Initialize the levelnames dimension for aw_products
"   Initialize the temp_levelnames dimension
MAINTAIN temp_levelnames DELETE ALL
"   Populate temp_levelnames with the lowercase names of
"     the base dimensions of aw_products in the order they were defined
"    which is top level (world) has position 1
MAINTAIN temp_levelnames ADD LOWCASE(OBJ(DATA 'aw_products'))
"    Sort levelnames so that bottom level (prod_id)
"    has position 1
SORT temp_levelnames A temp_levelnames
"    Populate aw_products_levelnames with sorted levelnames
MAINTAIN aw_products_levelnames ADD VALUES (temp_levelnames)
"       Populate aw_products_levelnums with integers reprsenting levels
" value of 1 represents bottom level
MAINTAIN aw_products_levelnums ADD STATLEN (aw_products_levelnames)

" Initialize the levelnames dimension for aw_times
"    Initialize the temp_levelnames dimension
MAINTAIN temp_levelnames DELETE ALL
"    Populate temp_levelnames with the lowercase names of
"    the base dimensions of aw_times in the order they were defined
MAINTAIN temp_levelnames ADD LOWCASE(OBJ(DATA 'aw_times'))
"    Sort levelnames so that bottom level has position 1
SORT temp_levelnames A temp_levelnames
"    Populate aw_times_levelnames with sorted levelnames
MAINTAIN aw_times_levelnames ADD VALUES (temp_levelnames)
"    Populate aw_products_levelnums with integers reprsenting levels
"    value of 1 represents bottom level
MAINTAIN aw_times_levelnums ADD STATLEN (aw_times_levelnames)
" Populate aw_products_gid with grouping ids
" for levels of products
GROUPINGID aw_products.parents INTO aw_products_gid

" Populate relation with concat dimension values for the Products hierarchy
HIERHEIGHT aw_products.parents INTO aw_products.aw_products_levelnums
" Populate relation with concat dimension values for the Times hierarchy
HIERHEIGHT aw_times.parents INTO aw_times.aw_times_levelnums
```

```
" Populate variable with base dimension values for the Products hierarchy
aw_products_basevals  = BASEVAL(aw_products.aw_products_levelnums)
" Populate variable with base dimension values for the Times hierarchy
aw_times_basevals  = BASEVAL(aw_times.aw_times_levelnums)

"Update the analytic workspace and make changes permanent
UPDATE
COMMIT
```

Example 11–4   Creating Views Using the OLAP_TABLE Function

```
AW CONNECT / as sysdba
SET ECHO ON
SET SERVEROUT ON

DROP TYPE aw_products_tbl;
DROP TYPE aw_products_obj;
DROP TYPE aw_times_tbl;
DROP TYPE aw_times_obj;
DROP TYPE aw_costs_tbl;
DROP TYPE aw_costs_obj;

-- Create objects and tables

-- Define an object that identifies the columns for product data

CREATE TYPE aw_products_obj AS OBJECT (
     prod_hier_value       VARCHAR2(35),
     prod_hier_parent      VARCHAR2(35),
     prod_id               VARCHAR2(10),
     prod_subcategory      VARCHAR2(20),
     prod_category         VARCHAR2(5),
     prod_all              VARCHAR2(15),
     prod_hier_gid         NUMBER(10));
```

```
-- Define an object that identifies the columns for times data


CREATE TYPE aw_times_obj AS OBJECT (
     time_hier_value       VARCHAR2(20),
     time_hier_parent      VARCHAR2(20),
     calendar_year         NUMBER(4,0),
     fiscal_year           NUMBER(4,0),
     date_id               VARCHAR2(10));


-- Define an object that identifies the columns for cost data

CREATE TYPE aw_costs_obj AS OBJECT (
     prod_hier_value       VARCHAR2(35),
     prod_hier_parent      VARCHAR2(35),
     time_hier_value       VARCHAR2(20),
     time_hier_parent      VARCHAR2(20),
     unit_cost             NUMBER(10,2),
     unit_price            NUMBER(10,2));



-- Define a table of objects for products data
CREATE TYPE aw_products_tbl AS TABLE OF aw_products_obj;
-- Define a table of objects for times data
CREATE TYPE aw_times_tbl AS TABLE OF aw_times_obj;
-- Define a table of objects for cost data
CREATE TYPE aw_costs_tbl AS TABLE OF aw_costs_obj;


-- Define a view of products data
CREATE OR REPLACE VIEW aw_products_view AS SELECT * FROM TABLE (CAST (OLAP_TABLE
(
        ’awsh duration session’, ’aw_products_tbl’, ’’,
        ’DIMENSION prod_hier_value FROM aw_products
         WITH HIERARCHY prod_hier_parent FROM aw_products.parents
         INHIERARCHY aw_products_inhier
         GID prod_hier_gid FROM aw_products_gid
         LEVELREL prod_id, prod_subcategory, prod_category, prod_all
         FROM aw_products_basevalues USING aw_products_levelnums’)
        AS aw_products_tbl));
```

```
-- Define a view of times data

CREATE OR REPLACE VIEW aw_times_view AS SELECT * FROM TABLE (CAST (OLAP_TABLE (
        'awsh duration session', 'aw_times_tbl', '',
        'DIMENSION time_hier_value FROM aw_times
         WITH HIERARCHY time_hier_parent FROM aw_times.parents
         ATTRIBUTE calendar_year FROM aw_calendar_year_form
         ATTRIBUTE fiscal_year FROM aw_fiscal_year_form
         ATTRIBUTE date_id FROM aw_timeid_form ')
        AS aw_times_tbl));

-- Define a view of cost data

CREATE OR REPLACE VIEW aw_costs_view AS SELECT * FROM TABLE (CAST (OLAP_TABLE (
        'awsh duration session', 'aw_costs_tbl', '',
        'MEASURE unit_cost FROM aw_unit_cost
         MEASURE unit_price FROM aw_unit_price
         DIMENSION prod_hier_value FROM aw_products
         WITH HIERARCHY prod_hier_parent FROM aw_products.parents
         DIMENSION time_hier_value FROM aw_times
         WITH HIERARCHY time_hier_parent FROM aw_times.parents')
        AS aw_costs_tbl));


-- Grant selection rights to the views
GRANT SELECT ON aw_products_view TO PUBLIC;
GRANT SELECT ON aw_times_view TO PUBLIC;
GRANT SELECT ON aw_costs_view TO PUBLIC;
```

```
-- Define a view of sales data
CREATE OR REPLACE VIEW olap_sales_view AS
  SELECT *
  FROM TABLE(OLAP_TABLE('XADEMO DURATION SESSION', 'XASALES_T', '',
                     'MEASURE sales FROM aw_f.sales
                      DIMENSION et_chan FROM aw_channel WITH
                        HIERARCHY aw_channel.parent
                          GID gid_chan FROM aw_channel.gid
                      DIMENSION et_prod FROM aw_product WITH
                        HIERARCHY aw_product.parent
                          GID gid_prod FROM aw_prod.gid
                      DIMENSION et_geog FROM aw_geography WITH
                        HIERARCHY aw_geography.parent
                          GID gid_geog FROM aw_geog.gid
                      DIMENSION et_time FROM aw_time WITH
                        HIERARCHY time.parent
                          GID gid_time FROM aw_time.gid'));
```

Part IV

OLAP Catalog Metadata API Reference

Part IV describes the OLAP catalog views and the PL/SQL packages for creating OLAP catalog metadata.

This part contains the following chapters:

12

OLAP Catalog Union Views

This chapter describes the views that constitute the comprehensive read API to all the OLAP metadata defined in the database. This includes OLAP metadata stored in the OLAP 2 Catalog (`CWM2`) and the OLAP 1 Catalog (`CWM`).

The OLAP Catalog union views reference the OLAP 2 Catalog views and the OLAP 1 Catalog views. For information on the OLAP 2 Catalog views, see OLAP Catalog (CWM2-Specific) Views.

This chapter discusses the following topics:

- Access to OLAP Catalog Union Views
- Summary of OLAP Catalog Union Views

Access to OLAP Catalog Union Views

The OLAP Catalog comprehensive read API (union views) 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 stored 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.

Summary of OLAP Catalog Union Views

The OLAP Catalog union views are summarized in the following table.

Table 12–1 OLAP Catalog Union Views

Synonym	View Name	Description
ALL_OLAP2_CATALOG_ENTITY_USES	ALL$OLAP2UCATALOG_ENTITY_USES	Represents measures in measure folders (catalogs).
ALL_OLAP2_CATALOGS	ALL$OLAP2UCATALOGS	Represents measure folders (catalogs).
ALL_OLAP2_CUBE_DIM_USES	ALL$OLAP2UCUBE_DIM_USES	Represents the association between cubes and their dimensions.
ALL_OLAP2_CUBE_MEAS_DIM_USES	ALL$OLAP2UCUBE_MEAS_DIM_USES	Represents the aggregation method specified for measure/dimension combinations.
ALL_OLAP2_CUBE_MEASURE_MAPS	ALL$OLAP2UCUBE_MEASURE_MAPS	Represents the mapping of measures to columns in fact tables.
ALL_OLAP2_CUBE_MEASURES	ALL$OLAP2UCUBE_MEASURES	Represents measures.
ALL_OLAP2_CUBES	ALL$OLAP2UCUBES	Represents cubes.
ALL_OLAP2_DIM_ATTR_USES	ALL$OLAP2UDIM_ATTR_USES	Represents dimension attributes and their associated level attributes.

Table 12–1 OLAP Catalog Union Views

Synonym	View Name	Description
ALL_OLAP2_DIM_ATTRIBUTES	ALL$OLAP2UDIM_ATTRIBUTES	Represents dimension attributes.
ALL_OLAP2_DIM_HIER_LEVEL_USES	ALL$OLAP2UDIM_HIER_LEVEL_USES	Represents the relationship between pairs of levels in a hierarchy.
ALL_OLAP2_DIM_HIERARCHIES	ALL$OLAP2UDIM_HIERARCHIES	Represents hierarchies.
ALL_OLAP2_DIM_LEVEL_ATTR_MAPS	ALL$OLAP2UDIM_LEVEL_ATTR_MAPS	Represents the association between levels and level attributes.
ALL_OLAP2_DIM_LEVEL_ATTRIBUTES	ALL$OLAP2UDIM_LEVEL_ATTRIBUTES	Represents level attributes.
ALL_OLAP2_DIM_LEVELS	ALL$OLAP2UDIM_LEVELS	Represents levels.
ALL_OLAP2_DIMENSIONS	ALL$OLAP2UDIMENSIONS	Represents dimensions.
ALL_OLAP2_ENTITY_DESC_USES	ALL$OLAP2UENTITY_DESC_USES	Represents the association between OLAP metadata entities and their descriptors.
ALL_OLAP2_FACT_LEVEL_USES	ALL$OLAP2UFACT_LEVEL_USES	Represents join relationships between fact tables and dimension tables.
ALL_OLAP2_FACT_TABLE_GID	ALL$OLAP2UFACT_TABLE_GID	Represents Grouping ID (GID) columns in fact tables.
ALL_OLAP2_HIER_CUSTOM_SORT	ALL$OLAP2UHIER_CUSTOM_SORT	Represents custom sorting information for hierarchies.
ALL_OLAP2_JOIN_KEY_COLUMN_USES	ALL$OLAP2UJOIN_KEY_COLUMN_USES	Represents the mapping between levels in a hierarchy.
ALL_OLAP2_LEVEL_KEY_COLUMN_USES	ALL$OLAP2ULEVEL_KEY_COLUMN_USES	Represents the mapping of levels to columns in dimension tables.

ALL_OLAP2_CATALOG_ENTITY_USES

ALL_OLAP2_CATALOG_ENTITY_USES is a synonym for ALL$OLAP2UCATALOG_ENTITY_USES.

Each row represents an entity within an OLAP measure folder (catalog). Measures are the only OLAP metadata entities that can be collected in measure folders.

Note: The classification system, used to manage measure folders and classify various OLAP metadata entities, is implemented in the OLAP 1 Catalog and referenced from the OLAP 2 Catalog.

Both OLAP 2 Catalog and OLAP 1 Catalog measure folders are displayed by ALL_OLAP2_CATALOG_ENTITY_USES.

Note: The term **catalog**, when used in the context of the OLAP metadata 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.

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_CATALOGS

ALL_OLAP2_CATALOGS is a synonym for ALL$OLAP2UCATALOGS.

Each row represents an OLAP measure folder (catalog). Measure folders are a means of grouping measures related to a given business area. For instance, all the measures that store information about a given product line might be collected in a measure folder.

Measure folders are schema independent. All users can view all the measure folders defined in the database, even if they do not have access privileges for the measures within the folders.

Measure folders can be nested within other measure folders.

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. The OLAP metadata classification system, implemented in the OLAP 1 Catalog, is used by both releases of the OLAP Catalog.

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_CUBE_DIM_USES

`ALL_OLAP2_CUBE_DIM_USES` is a synonym for `ALL$OLAP2UCUBE_DIM_USES.`

Each row represents an association between a cube and a dimension. 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.

Column	Data Type	NULL	Description
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. (OLAP 1 Catalog only)

ALL_OLAP2_CUBE_MEAS_DIM_USES

ALL_OLAP2_CUBE_MEAS_DIM_USES is a synonym for ALL$OLAP2UCUBE_MEAS_DIM_USES.

Each row represents the association of a measure with one of its dimensions, and specifies how the measure's data can be aggregated over that dimension. If no aggregation method is specified, the data is added.

Column	Data Type	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the cube that defines this measure.
CUBE_NAME	VARCHAR2(30)	NOT NULL	Name of the cube that defines 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		ID of 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_CUBE_MEASURE_MAPS

ALL_OLAP2_CUBE_MEASURE_MAPS is a synonym for ALL$OLAP2UCUBE_MEASURE_MAPS.

Each row represents the mapping of a measure to a column in a fact table.

In the OLAP 2 Catalog, measures are mapped separately for each combination of dimension hierarchies. For example, if a measure has three dimensions and each dimension has two hierarchies, then the measure has eight separate fact table mappings. These eight columns may exist within the same fact table or in separate fact tables. Each of the eight column mappings would appear as a separate row in the view.

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 defined by this cube.
DIM_HIER_COMBO_ID	NUMBER	NOT NULL	ID of the association between this measure and one combination of its dimension hierarchies. (Release 2 metadata only)
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. In Release 2 metadata, the measure's data is for one combination of its hierarchies.

ALL_OLAP2_CUBE_MEASURES

ALL_OLAP2_CUBE_MEASURES is a synonym for ALL$OLAP2UCUBE_MEASURES.

Each row represents a measure.

Column	Data Type	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the cube that defines the measure.
CUBE_NAME	VARCHAR2(30)	NOT NULL	Name of the cube that defines 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_CUBES

ALL_OLAP2_CUBES is a synonym for ALL$OLAP2UCUBES.

Each row represents a cube.

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(1)	NOT NULL	Whether or not this cube is in an invalid state. A cube is valid if it has at least one dimension, all of its dimensions are valid, and all the fact table mappings are valid.
DISPLAY_NAME	VARCHAR2(30)		Display name for the cube.
DESCRIPTION	VARCHAR2(2000)		Description of the cube.
MV_SUMMARYCODE	VARCHAR2(2)		If this cube has a materialized view in the OLAP 2 Catalog, the MV summary code specifies whether it is in GS (Grouping Set) or RU (Rolled Up) form. RU form means that all the dimension key columns are populated, and data may only be accessed when its full lineage is specified. GS form means that dimension key columns may contain null values, and data may be accessed simply by specifying one or more levels.

ALL_OLAP2_DIM_ATTR_USES

ALL_OLAP2_DIM_ATTR_USES is a synonym for ALL$OLAP2UDIM_ATTR_USES.

Each row represents the association of a level attribute with a dimension attribute. A dimension attribute is composed of a set of level attributes. The same level attribute can be included in more than one dimension attribute.

A level attribute designates a column in a dimension table that stores descriptive information about a level in the dimension. For example, there might be a color attribute pertaining to a product ID level.

A dimension attribute is a collection of level attributes. For example, the TIME_SPAN dimension attribute stores the number of days associated with each time period in a time dimension. Time periods are defined as levels, and each level has its own associated TIME_SPAN level attribute, but all the TIME_SPAN level attributes (for example, MONTH_TIME_SPAN, QUARTER_TIME_SPAN, and YEAR_TIME_SPAN) are defined as a single TIME_SPAN 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_ATTRIBUTES

ALL_OLAP2_DIM_ATTRIBUTES is a synonym for ALL$OLAP2UDIM_ATTRIBUTES.

Each row represents a dimension attribute, which is a logical attribute providing a grouping of level attributes within the dimension. The level attributes within the dimension attribute grouping can be determined from the ALL_OLAP2_DIM_ATTR_USES view.

A dimension attribute is only meaningful if it contains level attributes.

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		A classification for the dimension attribute. If a dimension attribute is classified, it may be of type LONG_DESCRIPTION, SHORT_DESCRIPTION, END_DATE, TIME_SPAN, PRIOR_PERIOD, or YEAR_AGO_PERIOD.

ALL_OLAP2_DIM_HIER_LEVEL_USES

ALL_OLAP2_DIM_HIER_LEVEL_USES is a synonym for ALL$OLAP2UDIM_HIER_LEVEL_USES.

Each row represents a hierarchical relationship between two levels in a dimension 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_DIM_HIERARCHIES

ALL_OLAP2_DIM_HIERARCHIES is a synonym for ALL$OLAP2UDIM_HIERARCHIES.

Each row represents a dimension hierarchy. The relationships between levels for each hierarchy are represented by ALL_OLAP2_DIM_HIER_LEVEL_USES.

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.

Column	Data Type	NULL	Description
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. Release 1 hierarchies are always of this type. SOLVED LEVEL-BASED, for a hierarchy that contains embedded totals, has a grouping ID, and is stored in a level-based dimension table. (Release 2 hierarchies only) SOLVED VALUE-BASED, for a hierarchy that contains embedded totals for all level combinations and is stored in a parent/child dimension table. (Release 2 hierarchies only)

ALL_OLAP2_DIM_LEVEL_ATTR_MAPS

ALL_OLAP2_DIM_LEVEL_ATTR_MAPS is a synonym for ALL$OLAP2UDIM_LEVEL_ATTR_MAPS.

Each row represents the mapping of a level attribute to its associated level.

Each level maps to one or more columns in a dimension table. Each level attribute maps to a single column in the same dimension table as its associated level.

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.

All the levels defined as OLAP metadata are represented by the ALL_OLAP2_DIM_LEVELS 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 containing this level.
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.

Column	Data Type	NULL	Description
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_DIM_LEVEL_ATTRIBUTES

ALL_OLAP2_DIM_LEVEL_ATTRIBUTES is a synonym for ALL$OLAP2UDIM_LEVEL_ATTRIBUTES.

Each row represents a level attribute. Each level attribute is a column in a dimension table. The column stores descriptive information about a level defined within the same dimension table. If the level attribute is not named, the column name is used.

The mapping of the level attribute column to its associated level columns is represented in ALL_OLAP2_DIM_LEVEL_ATTR_MAPS.

Column	Data Type	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension containing the level attribute.
DIMENSION_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension containing the level attribute.
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 to which this level attribute is mapped.

ALL_OLAP2_DIM_LEVELS

ALL_OLAP2_DIM_LEVELS is a synonym for ALL$OLAP2UDIM_LEVELS.

Each row represents a level within a dimension. A level is mapped to one or more columns within a dimension table. In a star schema, all of a dimension's levels are mapped to columns within the same table. In a snowflake schema, a dimension's levels are mapped to columns in separate tables.

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.
LEVEL_TABLE_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension table that contains the columns for this level.

ALL_OLAP2_DIMENSIONS

ALL_OLAP2_DIMENSIONS is a synonym for ALL$OLAP2UDIMENSIONS.

Each row represents a dimension. In OLAP 1 metadata, dimensions are based on Oracle dimension objects. In OLAP 2 metadata, dimensions are completely independent of Oracle 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. A dimension is valid when all of its levels and level attributes are mapped to existing columns, any dimension attributes are defined with sets of valid level attributes, and hierarchies are defined with valid levels.
DIMENSION_TYPE	VARCHAR2(10)		Not used.

ALL_OLAP2_ENTITY_DESC_USES

ALL_OLAP2_ENTITY_DESC_USES is a synonym for ALL$OLAP2UENTITY_DESC_USES.

Each row represents an association between an OLAP metadata entity and its descriptor. The OLAP metadata entities and descriptors are defined in the OLAP 1 Catalog classification system and referenced from the OLAP 2 Catalog.

The following OLAP metadata entities are represented in this view:

- Dimensions whose descriptor is Time.
- Dimension attributes whose descriptor is: Long Description, Short Description, or Description.
- Dimension attributes (for time dimensions only) whose descriptor is: End Date, Time Span, Prior Period, or Year Ago Period.
- Level attributes (for time dimensions only) whose descriptor is: Day, Month, Quarter, or Year.

Column	Data Type	NULL	Description
DESCRIPTOR_ID	NUMBER	NOT NULL	ID of the descriptor, derived from the OLAP 1 classification system.
ENTITY_OWNER	VARCHAR2(30)	NOT NULL	Owner of the entity. If the entity is a dimension attribute or level attribute, the owner is the owner of the dimension.
ENTITY_NAME	VARCHAR2(30)	NOT NULL	Name of the entity. The entity may be a dimension, a dimension attribute, or a level attribute.
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_FACT_LEVEL_USES

ALL_OLAP2_FACT_LEVEL_USES is a synonym for ALL$OLAP2UFACT_LEVEL_USES.

Each row represents a join relationship between a fact table and a dimension table. The join relationship is derived from a single key column in the fact table.

In OLAP 2 metadata, the fact table is always mapped in the context of a specific dimension hierarchy.

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. (Release 2 metadata only)
LEVEL_NAME	VARCHAR2(30)		Name of the level within the hierarchy where the mapping occurs. This represents the lowest level of aggregation defined by the foreign key / primary key for a specific dimension of the cube. In Release 1 metadata, this is always the leaf level (all the dimension's hierarchies share the same leaf level).
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. For Release 1 metadata, the key mapping type is always LL. 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). (Release 2 metadata only) RU (Rolled Up), when dimension members for each level are stored in a separate key column (multi-column key). (Release 2 metadata only)
FOREIGN_KEY_NAME	VARCHAR2(30)		Name of the foreign key constraint (OLAP 1 Catalog only) applied to the foreign key column. Constraints are not used in the OLAP 2 Catalog.

ALL_OLAP2_FACT_TABLE_GID

ALL_OLAP2_FACT_TABLE_GID is a synonym for ALL$OLAP2UFACT_TABLE_GID.

Each row represents information about a Grouping ID (GID) column in a fact table.

In the case of solved, embedded total fact table (where totals for every level combination are embedded in the table), there is a GID column corresponding to each dimension key in the fact table. For example, an embedded total fact table for sales data, which is dimensioned by product, geography and time, has three GID columns (one for each of the dimensions).

Note: This view only pertains to OLAP 2 metadata (CWM2). GID columns are not supported in OLAP 1 metadata (CWM).

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_HIER_CUSTOM_SORT

ALL_OLAP2_HIER_CUSTOM_SORT is a synonym for ALL$OLAP2UHIER_CUSTOM_SORT.

Each row provides information about custom sorting specified for a given dimension hierarchy. Custom sorting information is optional.

Custom sorting information specifies how to sort the members of a hierarchy based on columns in the associated 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 containing the column to be sorted.
TABLE_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension table containing the column to be sorted.
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_JOIN_KEY_COLUMN_USES

ALL_OLAP2_JOIN_KEY_COLUMN_USES is a synonym for ALL$OLAP2UJOIN_KEY_COLUMN_USES.

Each row represents the key information that joins two levels in a hierarchy. If the level is mapped to more than one column, each column mapping is represented in a separate row in the view.

In a snowflake schema, where levels are defined in separate dimension tables, levels in a hierarchy have a logical foreign key relationship. In a star schema, where levels are defined within the same dimension table, the child level key specifies its position in the hierarchy.

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_LEVEL_KEY_COLUMN_USES

ALL_OLAP2_LEVEL_KEY_COLUMN_USES is a synonym for ALL$OLAP2ULEVEL_KEY_COLUMN_USES.

Each row represents the logical key linking a dimension level to one of its underlying columns in a dimension table. 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).

13 OLAP Catalog (CWM2-Specific) Views

This chapter describes the views that constitute the read API to the OLAP Catalog version 2 (`CWM2`).

You can query these views to obtain information about OLAP metadata stored in the OLAP Catalog (`CWM2`) tables. To create this metadata, use the PL/SQL procedures in the OLAP Catalog (`CWM2`)write API.

Two sets of views are provided with version 2 of the OLAP Catalog. The views documented in this chapter directly represent the `CWM2` OLAP Catalog. Additionally, a set of union views, which provide a comprehensive read API for all OLAP metadata defined in the database (both `CWM` and `CWM2`), are provided. For information on the union views, see OLAP Catalog Union Views.

This chapter discusses the following topics:

- Access to OLAP Catalog (CWM2) Views
- System Tables Referenced by OLAP Catalog (CWM2) Views
- Summary of OLAP Catalog (CWM2)Views

Access to OLAP Catalog (CWM2) 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 stored 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.

System Tables Referenced by OLAP Catalog (CWM2) Views

The OLAP Catalog (`CWM2`) views present information stored in the `CWM2` tables and in several system tables.

The following system tables are referenced by the OLAP Catalog (`CWM2`) views:

- `SYS.COL$`
- `SYS.OBJ$`
- `SYS.USER$`
- `V$ENABLEDPRIVS`

System tables are referenced in the following views:

- `ALL$OLAP2_CUBES`
- `ALL$OLAP2_DIMENSIONS`
- `ALL$OLAP2_CUBE_MEASURE_MAPS`
- `ALL$OLAP2_DIM_LEVEL_ATTR_MAPS`
- `ALL$OLAP2_FACT_TABLE_GID`
- `ALL$OLAP2_FACT_LEVEL_USES`
- `ALL$OLAP2_JOIN_KEY_COLUMN_USES`
- `ALL$OLAP2_LEVEL_KEY_COLUMN_USES`

Summary of OLAP Catalog (CWM2)Views

The OLAP Catalog (CWM2) views are summarized in the following table.

Table 13–1 OLAP 2 Catalog Views

View Name	Description
ALL$OLAP2_CATALOG_ENTITY_USES	Represents measures in measure folders (catalogs).
ALL$OLAP2_CATALOGS	Represents measure folders (catalogs).
ALL$OLAP2_CUBE_DIM_USES	Represents the association between cubes and their dimensions.
ALL$OLAP2_CUBE_MEAS_DIM_USES	Represents the aggregation method specified for measure/dimension combinations.
ALL$OLAP2_CUBE_MEASURE_MAPS	Represents the mapping of measures to columns in fact tables.
ALL$OLAP2_CUBE_MEASURES	Represents measures.
ALL$OLAP2_CUBES	Represents cubes.
ALL$OLAP2_DIM_ATTR_USES	Represents dimension attributes and their associated level attributes.
ALL$OLAP2_DIM_ATTRIBUTES	Represents dimension attributes.
ALL$OLAP2_DIM_HIER_LEVEL_USES	Represents the relationship between pairs of levels in a hierarchy.
ALL$OLAP2_DIM_HIERARCHIES	Represents hierarchies.
ALL$OLAP2_DIM_LEVEL_ATTR_MAPS	Represents the association between levels and level attributes.
ALL$OLAP2_DIM_LEVEL_ATTRIBUTES	Represents level attributes.
ALL$OLAP2_DIM_LEVELS	Represents levels.
ALL$OLAP2_DIMENSIONS	Represents dimensions.
ALL$OLAP2_ENTITY_DESC_USES	Represents the association between OLAP metadata entities and their descriptors.
ALL$OLAP2_FACT_LEVEL_USES	Represents join relationships between fact tables and dimension tables.

Table 13–1 OLAP 2 Catalog Views

View Name	Description
ALL$OLAP2_FACT_TABLE_GID	Represents Grouping ID (GID) columns in fact tables.
ALL$OLAP2_HIER_CUSTOM_SORT	Represents custom sorting information for hierarchies.
ALL$OLAP2_JOIN_KEY_COLUMN_USES	Represents the mapping between levels in a hierarchy.
ALL$OLAP2_LEVEL_KEY_COLUMN_USES	Represents the mapping of levels to columns in dimension tables.

ALL$OLAP2_CATALOG_ENTITY_USES

Each row represents an entity within an OLAP measure folder (catalog). Measures are the only OLAP metadata entities that can be collected in measure folders.

Note: The classification system (CWM_CLASSIFY), used to manage measure folders and classify various OLAP metadata entities, is implemented in CWM and referenced from CWM2.

Only CWM2 measure folders are displayed by ALL$OLAP2_CATALOG_ENTITY_USES.

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.

Column	Datatype	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_CATALOGS

Each row represents an OLAP measure folder (catalog). Measure folders are a means of grouping measures related to a given business area. For instance, all the measures that store information about a given product line might be collected in a measure folder.

Measure folders are schema independent. All users can view all the measure folders defined in the database, even if they do not have access privileges for the measures within the folders.

Measure folders can be nested within other measure folders.

Note: The classification system (CWM_CLASSIFY), used to manage measure folders and classify various OLAP metadata entities, is implemented in `CWM` and referenced from `CWM2`.

Only `CWM2` measure folders are displayed by `ALL$OLAP2_CATALOG_ENTITY_USES`.

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.

Column	Datatype	NULL	Description
CATALOG_ID	NUMBER	NOT NULL	ID of the measure folder (from OLAP 1 Catalog).
CATALOG_NAME	VARCHAR2(30)	NOT NULL	Name of the measure folder (from OLAP 1 Catalog).
PARENT_CATALOG_ID	NUMBER		ID of the parent measure folder (from OLAP 1 Catalog). This column is null for measure folders at the root of the measure folder tree.
DESCRIPTION	VARCHAR2(2000)		Description of the measure folder (from OLAP 1 Catalog).

ALL$OLAP2_CUBE_DIM_USES

Each row represents an association between a cube and a dimension. 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	Datatype	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 that this association depends on.

ALL$OLAP2_CUBE_MEAS_DIM_USES

Each row represents the association of a measure with one of its dimensions, and specifies how the measure's data can be aggregated over that dimension. If no aggregation method is specified, the data is added.

Column	Datatype	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the cube that defines this measure.
CUBE_NAME	VARCHAR2(30)	NOT NULL	Name of the cube that defines 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		ID of 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_CUBE_MEASURE_MAPS

Each row represents the mapping of a measure to a column in a fact table.

Measures are mapped separately for each combination of dimension hierarchies. For example, if a measure has three dimensions and each dimension has two hierarchies, then the measure has eight separate fact table mappings. These eight columns may exist within the same fact table or in separate fact tables. Each of the eight column mappings would appear as a separate row in `ALL$OLAP_CUBE_MEASURE_MAPS`.

Column	Datatype	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 defined by 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 for this combination of hierarchies is stored.

ALL$OLAP2_CUBE_MEASURES

Each row represents a measure.

Column	Datatype	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the cube that defines the measure.
CUBE_NAME	VARCHAR2(30)	NOT NULL	Name of the cube that defines 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_CUBES

Each row represents a cube.

Column	Datatype	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the cube.
CUBE_NAME	VARCHAR2(30)	NOT NULL	Name of the cube.
INVALID	VARCHAR2(1)	NOT NULL	Whether or not this cube is in an invalid state. A cube is valid if it has at least one dimension, all of its dimensions are valid, and all the fact table mappings are valid.
DISPLAY_NAME	VARCHAR2(30)		Display name for the cube.
DESCRIPTION	VARCHAR2(2000)		Description of the cube.
MV_SUMMARYCODE	VARCHAR2(2)		If the cube is a materialized view, whether it is in GS (Grouping Set) or RU (Rolled Up) form. RU form means that all the dimension key columns are populated, and data may only be accessed when its full lineage is specified. GS form means that dimension key columns may contain null values, and data may be accessed simply by specifying one or more levels.

ALL$OLAP2_DIM_ATTR_USES

Each row represents the association of a level attribute with a dimension attribute. A dimension attribute is composed of a set of level attributes. The same level attribute can be included in more than one dimension attribute.

A level attribute designates a column in a dimension table that stores descriptive information about a level in the dimension. For example, there might be a color attribute pertaining to a product ID level.

A dimension attribute is a collection of level attributes. For example, the TIME_SPAN dimension attribute stores the number of days associated with each time period in a time dimension. Time periods are defined as levels, and each level has its own associated TIME_SPAN level attribute, but all the TIME_SPAN level attributes (for example, MONTH_TIME_SPAN, QUARTER_TIME_SPAN, and YEAR_TIME_SPAN) are defined as a single TIME_SPAN dimension attribute.

Column	Datatype	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_ATTRIBUTES

Each row represents a dimension attribute, which is a logical attribute providing a grouping of level attributes within the dimension. The level attributes within the dimension attribute grouping can be determined from the ALL$OLAP2_DIM_ATTR_USES view.

A dimension attribute is only meaningful if it contains level attributes.

Column	Datatype	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		A classification for the dimension attribute (from OLAP 1 Catalog). If a dimension attribute is classified, it may be of type LONG_DESCRIPTION, SHORT_DESCRIPTION, END_DATE, TIME_SPAN, PRIOR_PERIOD, or YEAR_AGO_PERIOD.

ALL$OLAP2_DIM_HIER_LEVEL_USES

Each row represents a hierarchical relationship between two levels in a dimension hierarchy. Within separate hierarchies, the same parent level may be hierarchically related to a different child level.

Column	Datatype	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_DIM_HIERARCHIES

Each row represents a dimension hierarchy. The relationships between levels for each hierarchy are represented by ALL$OLAP2_DIM_HIER_LEVEL_USES.

Column	Datatype	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	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 and is stored in a parent/child dimension table

ALL$OLAP2_DIM_LEVEL_ATTR_MAPS

Each row represents the mapping of a level attribute to its associated level.

Each level maps to one or more columns in a dimension table. Each level attribute maps to a single column in the same dimension table as its associated level.

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.

All the levels defined as OLAP 2 metadata are represented by the ALL$OLAP2_DIM_LEVELS view.

Column	Datatype	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.
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.
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_DIM_LEVEL_ATTRIBUTES

Each row represents a level attribute. Each level attribute is a column in a dimension table. The column stores descriptive information about a level defined within the same dimension table. If the level attribute is not named, the column name is used.

The mapping of the level attribute column to its associated level columns is represented in ALL$OLAP2_DIM_LEVEL_ATTR_MAPS.

Column	Datatype	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension containing the level attribute.
DIMENSION_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension containing the level attribute.
ATTRIBUTE_NAME	VARCHAR2(30)		Name of the level attribute. If no attribute name is specified, the column name is used. The level attribute name must be unique within the dimension.

Column	Datatype	NULL	Description
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 to which this level attribute is mapped.

ALL$OLAP2_DIM_LEVELS

Each row represents a level within a dimension. A level is mapped to one or more columns within a dimension table. In a star schema, all of a dimension's levels must be mapped to columns within the same table. In a snowflake schema, a dimension's levels may be mapped to columns in separate tables.

Column	Datatype	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. The name must be unique within the dimension.
DISPLAY_NAME	VARCHAR2(30)		Display name for the level.
DESCRIPTION	VARCHAR2(2000)		Description of the level.

ALL$OLAP2_DIMENSIONS

Each row represents an OLAP dimension. These dimensions are completely defined within OLAP 2 metadata and have no relationship to Oracle dimension objects.

Column	Datatype	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.

Column	Datatype	NULL	Description
INVALID	VARCHAR2(1)	NOT NULL	Whether or not the dimension is valid. A dimension is valid when all of its levels and level attributes are mapped to existing columns, any dimension attributes are defined with sets of valid level attributes, and hierarchies are defined with valid levels.
DIMENSION_TYPE	VARCHAR2(10)		The dimension may be a time dimension or a normal dimension.

ALL$OLAP2_ENTITY_DESC_USES

Each row represents an association between an OLAP metadata entity and its descriptor. The OLAP metadata entities and descriptors are defined in the OLAP 1 Catalog classification system and referenced from the OLAP 1 Catalog.

The following OLAP metadata entities are represented in this view:

- Dimensions whose descriptor is `Time`.
- Dimension attributes whose descriptor is: `Long Description`, `Short Description`, or `Description`.
- Dimension attributes (for time dimensions only) whose descriptor is: `End Date`, `Time Span`, `Prior Period`, or `Year Ago Period`.
- Level attributes (for time dimensions only) whose descriptor is: `Day`, `Month`, `Quarter`, or `Year`.

Column	Datatype	NULL	Description
DESCRIPTOR_ID	NUMBER	NOT NULL	ID of the descriptor, derived from the OLAP 1 classification system.
ENTITY_OWNER	VARCHAR2(30)	NOT NULL	Owner of the entity. If the entity is a dimension attribute or level attribute, the owner is the owner of the dimension.
ENTITY_NAME	VARCHAR2(30)	NOT NULL	Name of the entity. The entity may be a dimension, a dimension attribute, or a level attribute.
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_FACT_LEVEL_USES

Each row represents a join relationship between a fact table and a dimension table. The join relationship is derived from a single key column in the fact table.

The fact table is always mapped in the context of a specific dimension hierarchy.

Column	Datatype	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)	NOT NULL	Name of the level within the hierarchy where the mapping occurs. This represents the lowest level of aggregation defined by the foreign key / primary key for a specific dimension of the cube.
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 data is stored in the key column. The fact table is unsolved. ET (Embedded Totals), when 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 each level is stored in a separate key column (multi-column key).
FOREIGN_KEY_NAME	VARCHAR2(30)		Name of the foreign key constraint (OLAP 1 Catalog only) applied to the foreign key column. Constraints are not used in the OLAP 2 Catalog.

ALL$OLAP2_FACT_TABLE_GID

Each row represents information about a Grouping ID (GID) column in a fact table.

In the case of solved, embedded total fact table (where totals for every level combination are embedded in the table), there will be a GID column corresponding to each dimension key in the fact table. For example, an embedded total fact table for sales data, which is dimensioned by product, geography and time, has three GID columns (one for each of the dimensions).

Column	Datatype	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_HIER_CUSTOM_SORT

Each row provides information about custom sorting specified for a given dimension hierarchy. Custom sorting information is optional.

Custom sorting information specifies how to sort the members of a hierarchy based on columns in the associated 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.

Column	Datatype	NULL	Description
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.

Column	Datatype	NULL	Description
TABLE_OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension table containing the column to be sorted.
TABLE_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension table containing the column to be sorted.
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_JOIN_KEY_COLUMN_USES

Each row represents the key information that joins two levels in a hierarchy. Multiple column keys are represented by multiple rows in the view, one for each column use.

In a snowflake schema, where levels are defined in separate dimension tables, levels in a hierarchy have a logical foreign key relationship. When levels are defined within the same dimension table, the child level key specifies its position in the hierarchy.

Column	Datatype	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.

Column	Datatype	NULL	Description
POSITION	NUMBER		Position of column within the key. Applies to multi-column keys only.
JOIN_KEY_TYPE	VARCHAR2(30)	NOT NULL	SNOWFLAKE if the join key is a logical foreign key, or STAR if the join key refers to a column within the same table.

ALL$OLAP2_LEVEL_KEY_COLUMN_USES

Each row represents the logical key linking a dimension level to its underlying columns in a dimension table.

Multiple column keys are represented by multiple entries in the view for the level.

Column	Datatype	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 this column within the key. Applies to multi-column keys only.

14

OLAP Catalog Analytic Workspace Views

This chapter describes the views that represent analytic workspace objects registered in the OLAP Catalog.

This chapter discusses the following topics:

Summary of Analytic Workspace Object Views

The OLAP Catalog analytic workspace object views are summarized in the following table.

Table 14–1 OLAP Catalog Analytic Workspace Views

View Name	Description
ALL$OLAP2_AW_PHYS_OBJ	Represents physical objects stored in analytic workspaces.
ALL$OLAP2_AW_PHYS_OBJ_EXT	Represents additional information about analytic workspace objects.
ALL$OLAP2_AW_PHYS_OBJ_REL_OBJ	Represents objects related to other objects within analytic workspaces.
ALL$OLAP2_AW_PHYS_OBJ_PROP	Represents the properties of analytic workspace objects.
ALL$OLAP2_AW_MAP_DIM_USE	Represents the mapping of dimensions to dimensions stored in analytic workspaces.
ALL$OLAP2_AW_MAP_MEAS_USE	Represents the mapping of measures to variables in analytic workspaces.
ALL$OLAP2_AW_MAP_HIER_USE	Represents the mapping of dimension hierarchies to hierarchical information within analytic workspaces.
ALL$OLAP2_AW_MAP_LVL_USE	Represents the mapping of levels to level information within analytic workspaces.
ALL$OLAP2_AW_MAP_ATTR_USE	Represents the mapping of attributes to attribute information within analytic workspaces.

ALL$OLAP2_AW_PHYS_OBJ

Each row represents an object defined within an analytic workspace.

Column	Datatype	NULL	Description
AW_OWNER	VARCHAR2(30)	NOT NULL	Owner of the analytic workspace.
AW_NAME	VARCHAR2(30)	NOT NULL	Name of the analytic workspace.
AW_OBJECT_NAME	VARCHAR2(30)	NOT NULL	Name of the analytic workspace object, as defined by the OLAP DML.
AW_OBJECT_TYPE	VARCHAR2(30)	NOT NULL	Type of the analytic workspace object, as defined by the OLAP DML. Examples are dimensions, variables, and relations.
AW_OBJECT_DATATYPE	VARCHAR2(30)	NOT NULL	Data type of the analytic workspace object, as defined by the OLAP DML.

ALL$OLAP2_AW_PHYS_OBJ_EXT

Each row represents additional information about an AW object. If the definition of an AW object contains information other than the name and data type, that information is represented by this view. For example, an AW dimension may have a specified width, or a composite or conjoint dimension may have a specified sorting algorithm (BTREE, HASH, or NOHASH).

Column	Datatype	NULL	Description
AW_OWNER	VARCHAR2(30)	NOT NULL	Owner of the analytic workspace.
AW_NAME	VARCHAR2(30)	NOT NULL	Name of the analytic workspace.
AW_OBJECT_NAME	VARCHAR2(30)	NOT NULL	Name of the AW object, as defined by the OLAP DML.
AW_PHYS_ATTR_TYPE	VARCHAR2(30)	NOT NULL	Type of extension to the OLAP DML definition of the AW object.
AW_PHYS_ATTR_VALUE	VARCHAR2(30)	NOT NULL	Value of the extension to the OLAP DML definition of the AW object.

ALL$OLAP2_AW_PHYS_OBJ_REL_OBJ

Each row represents an AW object related to the definition of another AW object. Examples are the dimensions of variables and the related dimensions of relations.

Column	Datatype	NULL	Description
AW_OWNER	VARCHAR2(30)	NOT NULL	Owner of the analytic workspace.
AW_NAME	VARCHAR2(30)	NOT NULL	Name of the analytic workspace.
AW_OBJECT_NAME	VARCHAR2(30)	NOT NULL	Name of the AW object, as defined by the OLAP DML.
AW_OBJECT_REL_NAME	VARCHAR2(30)	NOT NULL	Name of the related AW object, as defined by the OLAP DML.
AW_REL_TYPE	VARCHAR2(30)	NOT NULL	Type of the related AW object, as defined by the OLAP DML.
POSITION	NUMBER		The position of the related AW object within the list of related objects. For example, the position of a dimension within the list of dimensions associated with a variable.

ALL$OLAP2_AW_PHYS_OBJ_PROP

Each row represents a property associated with an AW object.

Column	Datatype	NULL	Description
AW_OWNER	VARCHAR2(30)	NOT NULL	Owner of the analytic workspace.
AW_NAME	VARCHAR2(30)	NOT NULL	Name of the analytic workspace.
AW_OBJECT_NAME	VARCHAR2(30)	NOT NULL	Name of the AW object, as defined by the OLAP DML.
AW_PROP_NAME	VARCHAR2(30)	NOT NULL	Property of the AW object, as defined by the OLAP DML.
AW_PROP_DATATYPE	VARCHAR2(30)	NOT NULL	Data type of the AW object property, as defined by the OLAP DML.

ALL$OLAP2_AW_MAP_DIM_USE

Each row represents the mapping between a dimension entity within the OLAP Catalog and a dimension within an analytic workspace.

Column	Datatype	NULL	Description
DIMENSION_OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension entity within the OLAP Catalog.
DIMENSION_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension entity within the OLAP Catalog.
AW_OWNER	VARCHAR2(30)	NOT NULL	Owner of the analytic workspace.
AW_NAME	VARCHAR2(30)	NOT NULL	Name of the analytic workspace.
AW_OBJECT_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension object within the analytic workspace, as defined by the OLAP DML.

ALL$OLAP2_AW_MAP_MEAS_USE

Each row represents the mapping between a measure entity within the OLAP Catalog and a variable within an analytic workspace.

Column	Datatype	NULL	Description
CUBE_OWNER	VARCHAR2(30)	NOT NULL	Owner of the cube that defines the measure within the OLAP Catalog.
CUBE_NAME	VARCHAR2(30)	NOT NULL	Name of the cube that defines the measure within the OLAP Catalog.
MEASURE_NAME	VARCHAR2(30)	NOT NULL	Name of the measure entity within the OLAP Catalog.

Column	Datatype	NULL	Description
AW_OWNER	VARCHAR2(30)	NOT NULL	Owner of the analytic workspace.
AW_NAME	VARCHAR2(30)	NOT NULL	Name of the analytic workspace.
AW_OBJECT_NAME	VARCHAR2(30)	NOT NULL	Name of the variable within the analytic workspace, as defined by the OLAP DML.
AW_LIMIT_TYPE	VARCHAR2(30)	NOT NULL	One of the following values: ■ PROPERTY -- The logical measure will be mapped using a property of the object in the analytic workspace. ■ LIMITSET -- The logical measure will be mapped using an expression that limits one or more of the variable's dimensions. ■ NULL -- Not applicable.
AW_LIMIT_OBJ_NAME	VARCHAR2		For a limit type of LIMITSET, the names of the dimensions that are limited to a single value.
AW_LIMIT_OBJ_VALUE	VARCHAR2		For a limit type of LIMITSET, the values of the limited dimensions.

ALL$OLAP2_AW_MAP_HIER_USE

Each row represents the mapping between a dimension hierarchy entity within the OLAP Catalog and dimension hierarchy information within an analytic workspace.

Column	Datatype	NULL	Description
DIMENSION_OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension entity within the OLAP Catalog.
DIMENSION_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension entity within the OLAP Catalog.
HIERARCHY_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension hierarchy entity within the OLAP Catalog.
AW_OWNER	VARCHAR2(30)	NOT NULL	Owner of the analytic workspace.
AW_NAME	VARCHAR2(30)	NOT NULL	Name of the analytic workspace.
AW_OBJECT_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension object within the analytic workspace, as defined by the OLAP DML.

Column	Datatype	NULL	Description
AW_LIMIT_TYPE	VARCHAR2(30)	NOT NULL	One of the following values: One of the following values: ■ PROPERTY -- The logical hierarchy will be mapped using a property of the object in the analytic workspace. ■ LIMITSET -- The logical hierarchy will be mapped using an expression that limits one or more dimensions. ■ NULL -- Not applicable.
AW_LIMIT_OBJ_NAME	VARCHAR2		For a limit type of LIMITSET, the names of the dimensions that are limited to a single value.
AW_LIMIT_OBJ_ VALUE	VARCHAR2		For a limit type of LIMITSET, the values of the limited dimensions.

ALL$OLAP2_AW_MAP_LVL_USE

Each row represents the mapping between a dimension level entity within the OLAP Catalog and dimension level information within an analytic workspace.

Column	Datatype	NULL	Description
DIMENSION_OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension entity within the OLAP Catalog.
DIMENSION_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension entity within the OLAP Catalog.
LEVEL_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension level entity within the OLAP Catalog.
AW_OWNER	VARCHAR2(30)	NOT NULL	Owner of the analytic workspace.
AW_NAME	VARCHAR2(30)	NOT NULL	Name of the analytic workspace.
AW_OBJECT_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension object within the analytic workspace, as defined by the OLAP DML.
AW_LIMIT_TYPE	VARCHAR2(30)		One of the following values: ■ PROPERTY -- The logical level will be mapped using a property of the object in the analytic workspace. ■ LIMITSET -- The logical level will be mapped using an expression that limits one or more dimensions. ■ NULL -- Not applicable.

Column	Datatype	NULL	Description
AW_LIMIT_OBJ_NAME	VARCHAR2		For a limit type of LIMITSET, the names of the dimensions that are limited to a single value.
AW_LIMIT_OBJ_VALUE	VARCHAR2		For a limit type of LIMITSET, the values of the limited dimensions.

ALL$OLAP2_AW_MAP_ATTR_USE

Each row represents the mapping between a dimension attribute entity within the OLAP Catalog and dimension attribute information within an analytic workspace.

Column	Datatype	NULL	Description
DIMENSION_OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension entity within the OLAP Catalog.
DIMENSION_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension entity within the OLAP Catalog.
ATTRIBUTE_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension attribute entity within the OLAP Catalog.
AW_OWNER	VARCHAR2(30)	NOT NULL	Owner of the analytic workspace.
AW_NAME	VARCHAR2(30)	NOT NULL	Name of the analytic workspace.
AW_OBJECT_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension object within the analytic workspace, as defined by the OLAP DML.
AW_LIMIT_TYPE	VARCHAR2(30)		One of the following values: ■ PROPERTY -- The logical dimension attribute will be mapped using a property of the object in the analytic workspace. ■ LIMITSET -- The logical dimension attribute will be mapped using an expression that limits one or more dimensions. ■ NULL -- Not applicable.
AW_LIMIT_OBJ_NAME	VARCHAR2		For a limit type of LIMITSET, the names of the dimensions that are limited to a single value.
AW_LIMIT_OBJ_VALUE	VARCHAR2		For a limit type of LIMITSET, the values of the limited dimensions.

15 CWM2_OLAP_AW_ACCESS

The CWM2_OLAP_AW_ACCESS package contains procedures for generating scripts that create views of 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.

This chapter contains the following topics:

- Prerequisites
- Process Overview
- Preparing the Analytic Workspace
- Specifying the Source and Target Objects
- Example: Creating Views
- Summary of CWM2_OLAP_AW_ACCESS Subprograms

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. Create a text file for each view that you want to create. The text files map analytic workspace objects to columns in a relational 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 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 the 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 view with a command such as this one:

   ```
   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.
4. Identify the dimension attributes, which are objects that provide additional information about the dimensions.
5. If you plan to create OLAP catalog metadata, generate the additional objects that are needed by the OLAP API.

Following are descriptions of these objects.

Measures

Measures are VARIABLE, FORMULA, or RELATION objects with a numeric data type. For the best performance and data retrieval, the measures represented in a single fact view should have the exact same dimensions listed in the exact same order.

For example, these two variables can be represented very effectively in the same fact view:

```
DEFINE SALES VARIABLE SHORT <GEOGRAPHY PRODUCT CHANNEL TIME>
DEFINE COSTS VARIABLE SHORT <GEOGRAPHY PRODUCT CHANNEL TIME>
```

You can combine them with a variable that uses a composite, as long as the dimensions are listed in the same order:

```
DEFINE UNITS VARIABLE SHORT GEOGRAPHY PRODUCT SPARSE <CHANNEL TIME>
```

However, you should not combine the sales and costs variables with variables dimensioned in any of the following ways:

`<GEOGRAPHY TIME>` Fewer dimensions results in repetitive data in the view.

`<TIME GEOGRAPHY PRODUCT CHANNEL>` Different order of dimensions means that the data is stored differently and runtime performance will suffer as a result.

`<TIME SPARSE <GEOGRAPHY PRODUCT CHANNEL>>` Different order of dimensions means that the data is stored differently and runtime performance will suffer as a result.

Dimensions

If the measure is sparse, then it is probably dimensioned by a composite or a conjoint dimension. If it is dimensioned by a composite, then you will need to identify the base dimensions. If it is dimensioned by a conjoint, then you will need to identify the conjoint dimension.

A flat dimension (that is, one without a hierarchy or one in which all members are at the same level of a hierarchy) requires no supporting objects. However, a hierarchical dimension requires other objects to identify the relationships among members, as described in "Hierarchies" on page 15-4.

Hierarchies

Hierarchical dimensions are supported by several workspace objects: hierarchy dimensions, parent-child relations, level dimensions, and “in hierarchy” variables.

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.

```
DEFINE GEOGRAPHY.HIERARCHIES DIMENSION TEXT
LD Hierarchy dimension for GEOGRAPHY

REPORT geography.hierarchies

GEOGRAPHY.HIERARCHIES
----------------------
STANDARD
CONSOLIDATED
```

Parent-Child Relations

A parent- or self-relation is typically used to identify the parent of each dimension member. In the following example, the `geography` dimension has a parent relation named `geography.parentrel`.

Because geography has two hierarchies, STANDARD and CONSOLIDATED, a hierarchy dimension named `geography.hierarchies` also dimensions the parent relation. The names of the hierarchies are the members of `geography.hierarchies`.

```
DEFINE GEOGRAPHY.PARENTREL RELATION GEOGRAPHY <GEOGRAPHY GEOGRAPHY.HIERARCHIES>
LD Parent-child relation for GEOGRAPHY

REPORT DOWN geography geography.parentrel

               ---GEOGRAPHY.PARENTREL---
               --GEOGRAPHY.HIERARCHIES--
GEOGRAPHY        STANDARD   CONSOLIDATED
-------------- ----------- ------------
WORLD          NA          NA
AMERICAS       WORLD       NA
CANADA         AMERICAS    AMERICAS
TORONTO        CANADA      NA
MONTREAL       CANADA      NA
OTTAWA         CANADA      NA
```

From this example, you can see that in the STANDARD hierarchy, CANADA is the parent of TORONTO, MONTREAL, and OTTAWA, AMERICAS is the parent of CANADA, and WORLD is the parent of AMERICAS.

Level Dimensions

The levels of a dimension hierarchy are defined by the members of a dimension. This dimension often has a TEXT data type so that the members have descriptive names. For example, the `geography.leveldim` dimension might list geography levels such as CITY, STATE, COUNTRY, REGION, and so forth.

However, if you are providing support for the OLAP API, you will need to have an INTEGER dimension for the levels.

The following is the workspace definition of `geography.leveldim`, which is an dimension with an INTEGER data type required by the HIERHEIGHT command.

```
DEFINE GEOGRAPHY.LVLDIM DIMENSION INTEGER
```

The content of all INTEGER dimensions is sequential numbers. In this case, the length of the dimension is important, since it indicates the number of levels.

```
REPORT geography.lvldim

GEOGRAPHY.LVLDIM
----------------
               1
               2
               3
               4
```

You can specify the length of the INTEGER dimension based on the content of a text dimension that contains the names of the levels. For example, this command sets the length of the geography.lvldim INTEGER dimension based on the number of values currently in status for the geography.leveldim TEXT dimension:

```
LIMIT geography TO ALL
MAINTAIN geography.lvldim MERGE STATLEN(geography.leveldim)
```

In Hierarchy Variables

If a hierarchical dimension contains members that are excluded from a hierarchy, then a boolean variable is needed to identify whether a dimension member is in the hierarchy (YES) or not in the hierarchy (NO or NA). Otherwise, an NA value in the parent 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. If all the members of a dimension are included in the hierarchy, then this boolean dimension is not needed because there is no ambiguity.

The following example shows a boolean variable named time.inhierarchy defined for the time dimension, which has two hierarchies, STANDARD and YTD.

```
DESCRIBE time.inhierarchy

DEFINE TIME.INHIERARCHY VARIABLE BOOLEAN <TIME TIME.HIERARCHIES>
```

```
REPORT DOWN time time.inhierarchy time.parentrel

             -------------------------TIME.HIERARCHIES-------------------------
             ------------STANDARD------------- ---------------YTD--------------
TIME         TIME.INHIERARCHY  TIME.PARENTREL  TIME.INHIERARCHY  TIME.PARENTREL
------------ ---------------- ---------------- ---------------- ---------------
LAST.YTD                  no NA                             yes NA
CURRENT.YTD               no NA                             yes NA
JAN01                    yes Q1.01                          yes LAST.YTD
FEB01                    yes Q1.01                          yes LAST.YTD
    .                        .                                  .
    .                        .                                  .
    .                        .                                  .
Q1.02                    yes 2002                            no NA
Q2.02                    yes 2002                            no NA
2001                     yes NA                              no NA
2002                     yes NA                              no NA
```

Generated Objects

The OLAP DML contains two commands for generating objects that provide information for the OLAP API:

- `GROUPINGID`. Populates a variable that identifies the hierarchy level of each dimension value.
- `HIERHEIGHT`. Populates a relation that contains a value for each level in the hierarchy.

Use these commands to generate this information for each hierarchy of each dimension. If you do not generate these objects for use in the views, then the OLAP API performs a runtime calculation using SQL functions. This slows performance.

Grouping IDs

The following is the definition of a variable that stores the results of the `GROUPINGID` command. It is an `INTEGER` variable dimensioned by the data dimension (`geography`) and the hierarchy dimension (`geography.hierarchies`).

```
DEFINE GEOGRAPHY.GID VARIABLE INTEGER <GEOGRAPHY GEOGRAPHY.HIERARCHIES>
LD Output from GROUPINGID command
```

A command like the following populates `geography.gid`:

```
groupingid geography.parentrel INTO geography.gid
```

The contents of `geography.gid` specify the hierarchy level for each dimension value. These values correspond in order (but not value) to the level numbers. In this case, `geography.gid` identifies the four levels as 0, 1, 3, and 7.

```
             ----------GEOGRAPHY.GID----------
             ------GEOGRAPHY.HIERARCHIES------
GEOGRAPHY        STANDARD       CONSOLIDATED
------------ ---------------- ----------------
WORLD                       7                0
AMERICAS                    3                3
CANADA                      1                1
TORONTO                     0                0
MONTREAL                    0                0
OTTAWA                      0                0
VANCOUVER                   0                0
EDMONTON                    0                0
CALGARY                     0                0
```

Parent Grouping IDs

Parent grouping IDs identify the GID value of the parent for each member. You can define a formula that calculates these values based on the values of the GID variable described in "Grouping IDs" above.

For dimension members at the topmost level of a hierarchy, the parent GID must be calculated separately because their parent member is NA. That equation is 2***`levels`*-1, where *`levels`* is the number of levels in the hierarchy.

Create a formula like the following one, which calculates parent GIDs for the `geography` dimension.

```
DEFINE GEOGPARENT.GID FORMULA INTEGER <GEOGRAPHY GEOGRAPHY.HIERARCHIES>
EQ IF geography.parentrel EQ NA THEN 2**STATLEN(geography.leveldim)-1 ELSE
geography.gid(geography geography.parentrel)

LIMIT geography.hierarchies TO 1
REPORT DOWN geography geography.gid geogparent.gid
```

```
               GEOGRAPHY.HIERARCHIES
               ------STANDARD-------
               GEOGRAPHY. GEOGPARENT
GEOGRAPHY         GID        .GID
-------------- ---------- ----------
WORLD                   7         15
AMERICAS                3          7
CANADA                  1          3
TORONTO                 0          1
MONTREAL                0          1
OTTAWA                  0          1
VANCOUVER               0          1
EDMONTON                0          1
CALGARY                 0          1
```

Hierarchy Height

The following is the definition of a relation that stores the results of the `HIERHEIGHT` command. The relation must have the dimension as a data type (`geography`) and be dimensioned by the same dimensions as the parent relation (`geography` and `geography.hierarchies`) and the integer dimension for levels (`geography.lvldim`).

```
DEFINE GEOGRAPHY.HIERHEIGHT RELATION GEOGRAPHY <GEOGRAPHY GEOGRAPHY.HIERARCHIES
GEOGRAPHY.LVLDIM>
LD Output from HIERHEIGHT command
```

The following command generates the content of `geography.hierheight`:

```
HIERHEIGHT geography.parentrel INTO geography.hierheight
```

The `geography.hierheight` relation identifies the hierarchy level of each dimension value and its ancestors.

```
REPORT DOWN geography ACROSS geography.lvldim: geography.hierheight

             -----------GEOGRAPHY.HIERHEIGHT------------
             -------------GEOGRAPHY.LVLDIM-------------
GEOGRAPHY        1          2          3          4
------------ ---------- ---------- ---------- ----------
WORLD        NA         NA         NA         WORLD
AMERICAS     NA         NA         AMERICAS   WORLD
CANADA       NA         CANADA     AMERICAS   WORLD
TORONTO      TORONTO    CANADA     AMERICAS   WORLD
MONTREAL     MONTREAL   CANADA     AMERICAS   WORLD
OTTAWA       OTTAWA     CANADA     AMERICAS   WORLD
```

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

Attributes

Attributes are text variables 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. The following example shows two variables that contain short and long descriptive names for `geography` members.

```
REPORT DOWN geography geography.shortlabel geography.longlabel

GEOGRAPHY        GEOGRAPHY.SHORTLABEL       GEOGRAPHY.LONGLABEL
-------------- ------------------------ ------------------------
WORLD          World                    Regions of the World
AMERICAS       Americas                 Areas in the Americas
CANADA         Canada                   Canada
TORONTO        Toronto                  Toronto
MONTREAL       Montreal                 Montreal
OTTAWA         Ottawa                   Ottawa
VANCOUVER      Vancouver                Vancouver
EDMONTON       Edmonton                 Edmonton
CALGARY        Calgary                  Calgary
USA            USA                      United States of America
```

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 15-24) or directly in the command line (as described in "CreateAWAccessStructures Procedure" on page 15-25).

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 tables, 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. 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.

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 15-4. This keyword must appear first. Required.
`HIERARCHY`	A workspace `RELATION` that identifies the parent value for each dimension value in the hierarchy, as described in "Hierarchies" on page 15-4. Required for hierarchical dimensions.
`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 15-6. Required only when some dimension members are omitted from the hierarchy.
`HIERARCHY DIMENSION`	The workspace `DIMENSION` that contains the names of the hierarchies, as described in "Hierarchy Dimensions" on page 15-4. Required only if more than one hierarchy is defined for `DIMENSION`.
`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 15-7. Improves performance of the OLAP API.
`PARENT GID`	Identifies the grouping IDs of the parent values. You can define a formula that generates the parent values from the GID variable, as described in "Grouping IDs" on page 15-7.
`ATTRIBUTES`	One or more workspace `VARIABLE` objects that contain descriptive information about the dimension members, as described in "Attributes" on page 15-10. Optional.

Table 15–1 Keywords for Defining the Source Data for a Dimension View

Keyword	Description
`COLUMN LEVEL DIMENSION`	A workspace `DIMENSION` with an `INTEGER` data type, which contains a value for each level in the hierarchy, as described in "Level Dimensions" on page 15-5. It dimensions the `COLUMN LEVEL RELATION` object. Required for hierarchical dimensions.
`COLUMN LEVEL RELATION`	A workspace `RELATION` with a value for each level in the hierarchy. Use the `HIERHEIGHT` command in the OLAP DML to generate the values of this relation, as described in "Hierarchy Height" on page 15-9. Required for hierarchical dimensions.

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 grouping IDs from `PARENT GID`. Optional.

Table 15–2 Keywords for Defining the Target Columns for a Dimension View

Keyword	Description
`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.
`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 15-3.

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 15-3. 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.

Table 15–3 Additional Keywords for Defining a Fact View

Keyword	Description
`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`. The following are their object definitions. 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` `consolidated`	2
`product`	`standard`	1
`channel`	`standard`	1
`time`	`standard` `ytd`	2

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::geogparent.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::prodparent.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::prodp_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::chanparent.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::timeparent.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 15-17 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 PRODPARENT.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 PRODPARENT.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 15-20 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 on page 15-24	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 on page 15-25	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 CWM2_OLAP_AW_ACCESS 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 CWM2_OLAP_AW_ACCESS 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 15-10.

16 CWM2_OLAP_PC_TRANSFORM

The `CWM2_OLAP_PC_TRANSFORM` package provides a procedure that generates a SQL script for converting a parent/child dimension table to an embedded-total dimension table.

Once your parent-child tables are converted to embedded-total tables you can map OLAP metadata to them.

This chapter discusses the following topics:

- Summary of CWM2_OLAP_PC_TRANSFORM Subprograms

Summary of CWM2_OLAP_PC_TRANSFORM Subprograms

Table 16–1 CWM2_OLAP_PC_TRANSFORM

Subprogram	Description
CREATE_SCRIPT Procedure on page 16-2	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,
          t_table            IN   VARCHAR2,
          t_tablespace       IN   VARCHAR2,
          pc_root            IN   VARCHAR2   DEFAULT,
          levels             IN   VARCHAR2   DEFAULT,
          levels_list        IN   VARCHAR2   DEFAULT,
          short_description  IN   VARCHAR2   DEFAULT,
          long_description   IN   VARCHAR2   DEFAULT,
          attributes_list    IN   VARCHAR2   DEFAULT);
```

Parameters

Table 16–2 CREATE_SCRIPT Procedure Parameters

Parameter	Description
directory	Full path of a directory where the procedures will put the generated script.
schema	Schema containing the parent-child table. This schema will also contain the embedded-total table.
pc_table	Parent-child table.
pc_parent	Column containing the parent in the parent-child table.

Table 16–2 CREATE_SCRIPT Procedure Parameters

Parameter	Description
`pc_child`	Column containing the child in the parent-child table.
`t_table`	Embedded total table.
`t_tablespace`	Tablespace where the embedded total table will be created.
`pc_root`	One of the following: `null` - Root of parent-child table hierarchy is identified by `null` in the parent column. (default) *`condition`* - Root of the hierarchy in the parent-child table is a condition, for example: `'long_des = "Northeast Region"'`
`levels`	One of the following: `null` - Compute the number of levels in the parent-child table and create the embedded-total table with that number of levels. (default) *`number`* - The number of levels for the embedded-total table.
`levels_list`	One of the following: `null` - Generate the embedded-total table column names based on the parent-child column and level number. (default) *`list`* - A comma-separated list of column names for the embedded-total table.
`short_description`	One of the following: `null` - No short description in the parent-child table. Use the highest level non-null child in row of embedded-total table as short description. (default) *`column name`* - Name of the column in the parent-child table that contains the short description. Copy the short description column from the parent-child table to the embedded-total table.
`long_description`	One of the following: `null` - No long description in the parent-child table. Use the short description in the embedded-total table as the long description. (default) *`column name`* - Name of the column in the parent-child table that contains the long description. Copy the long description column from the parent-child table to the embedded-total table

Table 16–2 CREATE_SCRIPT Procedure Parameters

Parameter	Description
`attributes_list`	One of the following: `null` - No attributes. (default) *`list`* - A comma-separated list of attribute columns in the parent-child table. Copy these columns to the embedded-total table.

Usage Notes

1. The `CREATE_SCRIPT` procedure must have write access to the directory where the script will reside. The database must be able to execute scripts in this directory.
2. The script identifies the root of a parent-child hierarchy by a `null` in the `pc_parent` parameter. If the root is identified some other way, you may need to edit the generated script and change the way that the root is identified.
3. If a table with the same name as the embedded-total table already exists, the script will delete it.
4. Calculating the number of levels in the parent-child table is a relatively expensive operation. You can reduce the time required to generate the script by specifying the number of levels to be processed in the `et_levels` parameter.

17 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.

This chapter discusses the following topics:

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 2 Catalog (`CWM2`).

OLAP dimensions provide the dimensionality of OLAP measures. A measure represents data, stored in a fact table, that can be accessed by specifying values for its dimensions. For example, a measure representing sales data might be dimensioned by time, product, and location. This means that the sales data can be accessed for a given time period, a given product, and a given location. A set of measures with the same dimensionality can be represented as an OLAP cube.

OLAP dimensions represent columns in your data warehouse dimension tables as levels and attributes of levels. Each level defines a set of dimension members that share the same level of aggregation. For example, within a time dimension, the levels might be months, quarters, and years. Each level attribute holds descriptive information about dimension members within a given level. For example, the level attribute `END DATE` holds the final date for each of the months in the month level, for each of the quarters in the quarter level, and so on.

Dimensions usually have hierarchies, which define parent/child relationships between levels, and dimension attributes, which define sets of level attributes.

Note: Dimensions in the OLAP 2 Catalog map directly to columns in dimension tables and have no relationship to Oracle database dimension objects.

Dimension Table Requirements

The dimension tables that underlie OLAP dimensions must be organized in levels. The dimension members for each level are stored in one or more columns.

If your dimension tables are in parent-child format instead of level-based format, you must call `CREATE_SCRIPT` in the `CWM2_OLAP_PC_TRANSFORM` package to generate level-based views for your parent-child tables. Then you can call procedures in `CWM2_OLAP_DIMENSION` and related packages to create dimensions in the OLAP 2 Catalog.

If your source dimensions are stored in an analytic workspace within the database, you must call procedures in the `CWM2_OLAP_AW_ACCESS` package to create relational views that reference the workspace. Then you can call procedures in the `CWM2_OLAP_DIMENSION` package to create dimensions in the OLAP 2 Catalog.

Normal Dimensions and Time Dimensions

There are two types of dimensions: **normal dimensions** and **time dimensions**. The default type is normal.

A time dimension is a normal dimension with two mandatory dimension attributes: `END DATE` and `TIME SPAN`. Time dimensions typically contain levels for day, month, quarter, and year and one or more hierarchies defining the parent/child relationships between the levels. The `END DATE` and `TIME SPAN` attributes must be mapped for all the levels of a time dimension.

Dependencies Between Dimensional Entities

While a dimension is itself an OLAP metadata entity, it must have at least one child entity in order to be valid. This child entity must represent a level with a valid mapping to one or more columns in a dimension table.

The following table shows the parent/child dependencies between the metadata entities that make up a dimension.

Table 17–1 Hierarchical Relationships Between OLAP Dimensional Metadata Entities

Parent Entity	Child Entities
dimension	dimension attribute, hierarchy, level
dimension attribute	level attribute
hierarchy	level
level	level attribute

Creating Dimensions

The `CWM2_OLAP_DIMENSION` package contains procedures that establish dimension entities within the OLAP 2 Catalog.

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 of metadata does not fully define a dimension or a cube, nor does it involve any mapping to warehouse dimension tables or fact tables.

Completing the Dimension's Metadata

Creating dimensions is the first step in creating the OLAP metadata for a dimension. Typically 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.

Once you have created a dimension, you will need to call procedures in the following packages to fully define the dimension's metadata:

- CWM2_OLAP_DIMENSION_ATTRIBUTE to create dimension attributes
- CWM2_OLAP_HIERARCHY to create hierarchical relationships for the dimension's levels
- CWM2_OLAP_LEVEL to create levels and assign them to hierarchies
- CWM2_OLAP_LEVEL_ATTRIBUTE to create level attributes and assign them to dimension attributes
- CWM2_OLAP_TABLE_MAP to establish the mapping to columns in one or more dimension tables

Verifying Dimensional Metadata

To be valid, a dimension must have at least one level. If the dimension is a time dimension, it must also have the `END DATE` and `TIME SPAN` dimension attributes mapped for all levels.

All the dimension's levels and level attributes must be mapped to columns in dimension tables. Each level must map to one or more columns in one or more tables. If a level maps to multiple columns, they must be in the same table. Each level attribute must map to a single column in the same table as its associated level. If any of these columns is inaccessible, the dimension is not valid.

To test the validity of your dimensional metadata, use the `VALIDATE_DIMENSION` procedure in the `CWM2_OLAP_VALIDATE` package.

Common Logic in CWM2_OLAP_DIMENSION Subprograms

Each procedure first checks the calling user's security privileges. The calling user must be the dimension owner and must have the `OLAP_DBA` role. If the calling user does not meet the security requirements, the procedure fails with an exception.

Each procedure then checks for the existence of the dimension specified by `dimension_owner` and `dimension_name` within the OLAP 2 Catalog. All procedures, except `CREATE_DIMENSION`, return an error if the dimension does not already exist.

Case Requirements for Subprogram Parameters

You can specify arguments 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, `solved_code`), the procedure converts the argument to upper case. For all other arguments, the case that you specify is retained.

Summary of CWM2_OLAP_DIMENSION Subprograms

Table 17–2 CWM2_OLAP_DIMENSION Subprograms

Subprogram	Description
CREATE_DIMENSION Procedure on page 17-6	Creates a dimension.
DROP_DIMENSION Procedure on page 17-7	Drops a dimension.
LOCK_DIMENSION Procedure on page 17-8	Locks the dimension metadata for update.
SET_DEFAULT_DISPLAY_HIERARCHY Procedure on page 17-9	Sets the default hierarchy for a dimension.
SET_DESCRIPTION Procedure on page 17-10	Sets the description for a dimension.
SET_DIMENSION_NAME Procedure on page 17-10	Sets the name of a dimension.
SET_DISPLAY_NAME Procedure on page 17-11	Sets the display name for a dimension.
SET_PLURAL_NAME Procedure on page 17-12	Sets the plural name for a dimension.
SET_SHORT_DESCRIPTION Procedure on page 17-13	Sets the short description for a dimension.

CREATE_DIMENSION Procedure

This procedure registers a new dimension entity in the OLAP 2 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 17–3 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.
`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 17–4 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 in the OLAP 2 Catalog.

DROP_DIMENSION Procedure

This procedure drops a dimension from the OLAP 2 Catalog. All related levels, hierarchies, and dimension attributes are also dropped.

Syntax

```
DROP_DIMENSION (
          dimension_owner     IN   VARCHAR2,
          dimension_name      IN   VARCHAR2);
```

Parameters

Table 17–5 DROP_DIMENSION Procedure Parameters

Parameter	Description
`dimension_owner`	Owner of the dimension.
`dimension_name`	Name of the dimension.

Exceptions

Table 17–6 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`	Cannot drop dimension. Dimension does not exist within the OLAP 2 Catalog.

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 OLAP 2 Catalog.

Syntax

```
LOCK_DIMENSION (
          dimension_owner     IN   VARCHAR2,
          dimension_name      IN   VARCHAR2.
          wait_for_lock       IN   BOOLEAN DEFAULT FALSE);
```

Parameters

Table 17–7 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 17–8 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.
`dimension_not_found`	Cannot lock dimension. Dimension does not exist within the OLAP 2 Catalog.

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 17–9 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 17–10 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`	Cannot update dimension. Dimension does not exist within the OLAP 2 Catalog.
`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 17–11   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 17–12   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`	Cannot update dimension. Dimension does not exist within the OLAP 2 Catalog.

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 17–13 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 17–14 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`	Cannot update dimension. Dimension does not exist within the OLAP 2 Catalog.

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 17–15 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 17–16 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`	Cannot update dimension. Dimension does not exist within the OLAP 2 Catalog.

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 17–17 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 17–18 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`	Cannot update dimension. Dimension does not exist within the OLAP 2 Catalog.

SET_SHORT_DESCRIPTION Procedure

This procedure sets the short description for a dimension.

Syntax

```
SET_DESCRIPTION (
          dimension_owner       IN   VARCHAR2,
          dimension_name        IN   VARCHAR2,
          short_description     IN   VARCHAR2);
```

Parameters

Table 17–19 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 17–20 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`	Cannot update dimension. Dimension does not exist within the OLAP 2 Catalog.

18

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.

This chapter discusses the following topics:

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 2 Catalog.

Dimension attributes are child entities of dimension entities. Dimension attributes define sets of level attributes, which map to columns on a per level basis. Each level attribute column is associated with a level and must exist in the same dimension table as the level column. Dimension attributes may include level attributes for some or all of the dimension’s levels.

Examples of dimension attributes are `end date` and `time span`, which are required for time dimensions. In order for a time dimension to be valid, `end date` and `time span` dimension attributes must be defined for all levels. For example, if a time dimension has month, quarter, and year levels, `end date` must identify the last date of each month, each quarter, and each year. Likewise `time span` must identify the length of each month, each quarter, and each year.

Creating Dimension Attributes

The `CWM2_OLAP_DIMENSION_ATTRIBUTE` package contains procedures that establish attribute entities for dimensions within the OLAP 2 Catalog.

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 a dimension attribute does not fully define it, nor does it involve any mapping to warehouse dimension tables.

The parent dimension must already exist in the OLAP 2 Catalog before you can create dimension attributes for it.

Completing the Dimension's Metadata

Creating dimension attributes is one of the first steps in creating the OLAP metadata for a dimension. Once you have created the dimension attributes, you will need to call procedures in the following packages to fully define the dimension's metadata:

- CWM2_OLAP_HIERARCHY to create hierarchical relationships for the dimension's levels (if you have not already done so)
- CWM2_OLAP_LEVEL to create levels and assign them to hierarchies within the dimension
- CWM2_OLAP_LEVEL_ATTRIBUTE to create level attributes and assign them to dimension attributes within the dimension
- CWM2_OLAP_TABLE_MAP to establish the mapping to columns in one or more dimension tables

Common Logic in CWM2_OLAP_DIMENSION_ATTRIBUTE Subprograms

Each procedure first checks the calling user's security privileges. The calling user must be the dimension owner and must have the `OLAP_DBA` role. If the calling user does not meet the security requirements, the procedure fails with an exception.

Each procedure then checks for the existence of the parent dimension within the OLAP 2 Catalog. If the dimension does not exist, the procedure fails with an exception.

Each procedure then checks for the existence of the dimension attribute. All procedures, except `CREATE_DIMENSION_ATTRIBUTE`, return an error if the dimension attribute does not already exist.

Case Requirements for Subprogram Parameters

You can specify arguments in lower case, upper case, or mixed case.

If the argument is a metadata entity name (for example, `dimension_attribute_name`) or a value that will be used in further processing by other procedures, the procedure converts the argument to upper case. For all other arguments, the case that you specify is retained.

Summary of CWM2_OLAP_DIMENSION_ATTRIBUTE Subprograms

Table 18–1 CWM2_OLAP_DIMENSION_ATTRIBUTE Subprograms

Subprogram	Description
CREATE_DIMENSION_ATTRIBUTE Procedure on page 18-4	Creates a dimension attribute.
DROP_DIMENSION_ATTRIBUTE Procedure on page 18-6	Drops a dimension attribute.
LOCK_DIMENSION_ATTRIBUTE Procedure on page 18-7	Locks the dimension attribute metadata for update.
SET_DESCRIPTION Procedure on page 18-8	Sets the description for a dimension attribute.
SET_DIMENSION_ATTRIBUTE_NAME Procedure on page 18-9	Sets the name of a dimension attribute.
SET_DISPLAY_NAME Procedure on page 18-10	Sets the display name for a dimension attribute.
SET_SHORT_DESCRIPTION Procedure on page 18-11	Sets the short description for a dimension attribute.

CREATE_DIMENSION_ATTRIBUTE Procedure

This procedure registers a new dimension attribute entity in the OLAP 2 Catalog.

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 18–2 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 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 18–3 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`	Cannot create dimension attribute. Dimension attribute already exists in the OLAP 2 Catalog.

DROP_DIMENSION_ATTRIBUTE Procedure

This procedure drops a dimension attribute from the OLAP 2 Catalog.

Syntax

```
DROP_DIMENSION_ATTRIBUTE (
         dimension_owner                IN   VARCHAR2,
         dimension_name                 IN   VARCHAR2,
         dimension_attribute_name       IN   VARCHAR2);
```

Parameters

Table 18–4 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 18–5 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`	The parent dimension does not exist within the OLAP 2 Catalog.
`dimension_attribute_not_found`	Dimension attribute does not exist within the OLAP 2 Catalog.

LOCK_DIMENSION_ATTRIBUTE Procedure

This procedure locks the dimension attribute metadata for update by acquiring a database lock on the row that identifies the dimension attribute in the OLAP 2 Catalog.

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 18–6 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 18–7 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`	The parent dimension does not exist within the OLAP 2 Catalog.
`dimension_attribute_not_found`	Dimension attribute does not exist within the OLAP 2 Catalog.

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 18–8 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 18–9 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.
dimension_not_found	The parent dimension does not exist within the OLAP 2 Catalog.
dimension_attribute_not_found	Dimension attribute does not exist within the OLAP 2 Catalog.

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 18–10 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 18–11 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`	The parent dimension does not exist within the OLAP 2 Catalog.
`dimension_attribute_not_found`	Dimension attribute does not exist within the OLAP 2 Catalog.

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 18–12 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 18–13 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`	The parent dimension does not exist within the OLAP 2 Catalog.
`dimension_attribute_not_found`	Dimension attribute does not exist within the OLAP 2 Catalog.

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 18–14 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 18–15 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.
`dimension_not_found`	The parent dimension does not exist within the OLAP 2 Catalog.
`dimension_attribute_not_found`	Dimension attribute does not exist within the OLAP 2 Catalog.

19 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.

This chapter discusses the following topics:

- Understanding Hierarchies
- Creating Hierarchies
- Common Logic in CWM2_OLAP_HIERARCHY Subprograms
- Summary of CWM2_OLAP_HIERARCHY Subprograms

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 2 Catalog.

Hierarchies are child entities of dimension entities. Hierarchies define parent/child relationships between sets of levels. There can be multiple hierarchies associated with a single dimension, and the same level can be used in multiple hierarchies.

Creating Hierarchies

The `CWM2_OLAP_HIERARCHY` package contains procedures that establish hierarchy entities for dimensions within the OLAP 2 Catalog.

> **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 a hierarchy does not fully define it, nor does it involve any mapping to warehouse dimension tables.

The parent dimension must already exist in the OLAP 2 Catalog before you can create hierarchies for it.

Completing the Dimension's Metadata

Creating hierarchies is one of the first steps in creating the OLAP metadata for a dimension. Once you have created the hierarchies, you will need to call procedures in the following packages to fully define the dimension's metadata:

- CWM2_OLAP_DIMENSION_ATTRIBUTE to create dimension attributes (if you have not already done so)
- CWM2_OLAP_LEVEL to create levels and assign them to hierarchies
- CWM2_OLAP_LEVEL_ATTRIBUTE to create level attributes and assign them to dimension attributes
- CWM2_OLAP_TABLE_MAP to establish the mapping to columns in one or more dimension tables

Common Logic in CWM2_OLAP_HIERARCHY Subprograms

Each procedure first checks the calling user's security privileges. The calling user must be the dimension owner and must have the OLAP_DBA role. If the calling user does not meet the security requirements, the procedure fails with an exception.

Each procedure then checks for the existence of the parent dimension within the OLAP 2 Catalog. If the dimension does not exist, the procedure fails with an exception.

Each procedure then checks for the existence of the hierarchy. All procedures, except CREATE_HIERARCHY, return an error if the hierarchy does not already exist.

Case Requirements for Subprogram Parameters

You can specify arguments in lower case, upper case, or mixed case.

If the argument is a metadata entity name (for example, `hierarchy_name`) or a value that will be used in further processing by other procedures (for example, `solved_code`), the procedure converts the argument to upper case. For all other arguments, the case that you specify is retained.

Summary of CWM2_OLAP_HIERARCHY Subprograms

Table 19–1 CWM2_OLAP_HIERARCHY Subprograms

Subprogram	Description
CREATE_HIERARCHY Procedure on page 19-4	Creates a hierarchy.
DROP_HIERARCHY Procedure on page 19-5	Drops a hierarchy.
LOCK_HIERARCHY Procedure on page 19-6	Locks the hierarchy metadata for update.
SET_DESCRIPTION Procedure on page 19-7	Sets the description for a hierarchy.
SET_DISPLAY_NAME Procedure on page 19-8	Sets the display name for a hierarchy.
SET_HIERARCHY_NAME Procedure on page 19-9	Sets the name of a hierarchy.

Table 19–1 CWM2_OLAP_HIERARCHY Subprograms

Subprogram	Description
SET_SHORT_DESCRIPTION Procedure on page 19-10	Sets the short description for a hierarchy.
SET_SOLVED_CODE Procedure on page 19-11	Sets the solved code for a hierarchy.

CREATE_HIERARCHY Procedure

This procedure registers a new hierarchy entity in the OLAP 2 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 19–2 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 19–2 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. 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 19–3 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`	The parent dimension does not exist within the OLAP 2 Catalog.
`hierarchy_already_exists`	This hierarchy already exists for this dimension.

DROP_HIERARCHY Procedure

This procedure drops a hierarchy from the OLAP 2 Catalog.

Syntax

```
DROP_HIERARCHY (
         dimension_owner     IN   VARCHAR2,
         dimension_name      IN   VARCHAR2,
         hierarchy_name      IN   VARCHAR2);
```

Parameters

Table 19–4 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 19–5 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`	The parent dimension does not exist within the OLAP 2 Catalog.
`hierarchy_not_found`	This hierarchy does not exist for this dimension.

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 OLAP 2 Catalog.

Syntax

```
LOCK_HIERARCHY (
          dimension_owner     IN   VARCHAR2,
          dimension_name      IN   VARCHAR2,
          hierarchy_name      IN   VARCHAR2,
          wait_for_lock       IN   BOOLEAN DEFAULT FALSE);
```

Parameters

Table 19–6 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 19–7 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`	The parent dimension does not exist within the OLAP 2 Catalog.
`hierarchy_not_found`	This hierarchy does not exist for this dimension.

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 19–8 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 19–9 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`	The parent dimension does not exist within the OLAP 2 Catalog.
`hierarchy_not_found`	This hierarchy does not exist for this dimension.

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 19–10 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 19–11 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`	The parent dimension does not exist within the OLAP 2 Catalog.
`hierarchy_not_found`	This hierarchy does not exist for this dimension.

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 19–12 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 19–13 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`	The parent dimension does not exist within the OLAP 2 Catalog.
`hierarchy_not_found`	This hierarchy does not exist for this dimension.

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 19–14 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 19–15 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	The parent dimension does not exist within the OLAP 2 Catalog.
hierarchy_not_found	This hierarchy does not exist for this dimension.

SET_SOLVED_CODE Procedure

This procedure sets the solved code for a hierarchy. The 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.

Syntax

```
SET_SOLVED_CODE (
          dimension_owner     IN   VARCHAR2,
          dimension_name      IN   VARCHAR2,
          hierarchy_name      IN   VARCHAR2,
          solved_code         IN   VARCHAR2);
```

Parameters

Table 19–16 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

Exceptions

Table 19–17 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`	The parent dimension does not exist within the OLAP 2 Catalog.
`hierarchy_not_found`	This hierarchy does not exist for this dimension.

20

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.

This chapter discusses the following topics:

- Understanding Levels
- Creating Levels
- Common Logic in CWM2_OLAP_LEVEL Subprograms
- Summary of CWM2_OLAP_LEVEL Subprograms

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 2 Catalog.

A level is a child entity of a dimension entity. Each dimension must have at least one level. A level is also normally a child entity of one or more hierarchy entities. If a level is not assigned to a hierarchy, the OLAP 2 Catalog considers its hierarchy to be hidden.

Each level maps to one or more columns in a dimension table. In a star schema, the columns are in the same table. In a snowflake schema, the columns are in separate tables. If the dimension is stored in a parent/child table instead of a level-based table, levels map to columns in the view you can generate via the `CWM2_OLAP_PC_TRANSFORM` package. Similarly, if the dimension is stored in an analytic workspace, levels map to columns in the view you can generate via the `AWCONVERT` package.

Creating Levels

The `CWM2_OLAP_LEVEL` package contains procedures that establish level entities for dimensions within the OLAP 2 Catalog.

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 a level does not fully define it, nor does it involve any mapping to warehouse dimension tables.

The parent dimension must already exist in the OLAP 2 Catalog before you can create levels for it. Hierarchies must already exist for the dimension before you can assign levels to them.

Completing the Dimension’s Metadata

Creating levels is one of the steps in creating the OLAP metadata for a dimension. Once you have created the hierarchies, you will need to call procedures in the following packages to fully define the dimension’s metadata:

- CWM2_OLAP_DIMENSION_ATTRIBUTE to create dimension attributes (if you have not already done so)
- CWM2_OLAP_LEVEL_ATTRIBUTE to create level attributes and assign them to dimension attributes

- CWM2_OLAP_TABLE_MAP to map levels and level attributes to columns in one or more dimension tables

Common Logic in CWM2_OLAP_LEVEL Subprograms

Each procedure first checks the calling user's security privileges. The calling user must be the dimension owner and must have the `OLAP_DBA` role. If the calling user does not meet the security requirements, the procedure fails with an exception.

Each procedure then checks for the existence of the parent dimension within the OLAP 2 Catalog. If the dimension does not exist, the procedure fails with an exception.

Each procedure then checks for the existence of the level. All procedures, except `CREATE_LEVEL`, return an error if the level does not already exist.

Case Requirements for Subprogram Parameters

You can specify arguments in lower case, upper case, or mixed case.

If the argument is a metadata entity name (for example, `level_name`) or a value that will be used in further processing by other procedures, the procedure converts the argument to upper case. For all other arguments, the case that you specify is retained.

Summary of CWM2_OLAP_LEVEL Subprograms

Table 20–1 CWM2_OLAP_LEVEL Subprograms

Subprogram	Description
ADD_LEVEL_TO_HIERARCHY Procedure on page 20-4	Adds a level to a hierarchy.
CREATE_LEVEL Procedure on page 20-5	Creates a level.
DROP_LEVEL Procedure on page 20-6	Drops a level.
LOCK_LEVEL Procedure on page 20-7	Locks the level metadata for update.
REMOVE_LEVEL_FROM_HIERARCHY Procedure on page 20-8	Removes a level from a hierarchy.

Table 20–1 CWM2_OLAP_LEVEL Subprograms

Subprogram	Description
SET_DESCRIPTION Procedure on page 20-9	Sets the description for a level.
SET_DISPLAY_NAME Procedure on page 20-10	Sets the display name for a level.
SET_LEVEL_NAME Procedure on page 20-11	Sets the name of a level.
SET_PLURAL_NAME Procedure on page 20-12	Sets the plural name for a level.
SET_SHORT_DESCRIPTION Procedure on page 20-13	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 20–2 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 20–3 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	The parent dimension does not exist within the OLAP 2 Catalog.
hierarchy_not_found	This hierarchy does not exist for this dimension.
level_not_found	This level does not exist for this dimension.

CREATE_LEVEL Procedure

This procedure registers a new level as an entity in the OLAP 2 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 20–4 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.

Table 20–4 CREATE_LEVEL Procedure Parameters

Parameter	Description
`plural_name`	Plural name for the level.
`short_description`	Short description of the level.
`description`	Description of the level.

Exceptions

Table 20–5 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`	The parent dimension does not exist within the OLAP 2 Catalog.
`level_already_exists`	This level already exists for this dimension.

DROP_LEVEL Procedure

This procedure drops a level from the OLAP 2 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 20–6 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 20–7 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`	The parent dimension does not exist within the OLAP 2 Catalog.
`level_not_found`	This level does not exist for this dimension.

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 OLAP 2 Catalog.

Syntax

```
LOCK_LEVEL (
          dimension_owner     IN   VARCHAR2,
          dimension_name      IN   VARCHAR2,
          level_name          IN   VARCHAR2,
          wait_for_lock       IN   BOOLEAN DEFAULT FALSE);
```

Parameters

Table 20–8 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 20–9 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`	The parent dimension does not exist within the OLAP 2 Catalog.
`level_not_found`	This level does not exist for this dimension.

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 20–10 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 20–11 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.
dimension_not_found	The parent dimension does not exist within the OLAP 2 Catalog.
hierarchy_not_found	This hierarchy does not exist for this dimension.
child_level_not_found	This level is not a child of this hierarchy.

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 20–12 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 20–13 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`	The parent dimension does not exist within the OLAP 2 Catalog.
`level_not_found`	This level does not exist for this dimension.

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 20–14 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 20–15 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	The parent dimension does not exist within the OLAP 2 Catalog.
level_not_found	This level does not exist for this dimension.

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 20–16 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 20–17 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`	The parent dimension does not exist within the OLAP 2 Catalog.
`level_not_found`	This level does not exist for this dimension.

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 20–18 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 20–19 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`	The parent dimension does not exist within the OLAP 2 Catalog.
`level_not_found`	This level does not exist for this dimension.

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 20–20 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 20–21 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`	The parent dimension does not exist within the OLAP 2 Catalog.
`level_not_found`	This level does not exist for this dimension.

21

CWM2_OLAP_LEVEL_ATTRIBUTE

The CWM2_OLAP_LEVEL_ATTRIBUTE package provides a procedure for creating level attributes and associating them with dimension attributes. It also provides procedures for dropping, locking, and setting the general properties of level attributes.

This chapter discusses the following topics:

- Understanding Level Attributes
- Creating Level Attributes
- Common Logic in CWM2_OLAP_LEVEL_ATTRIBUTE Subprograms
- Summary of CWM2_OLAP_LEVEL_ATTRIBUTE Subprograms

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 2 Catalog.

A level attribute is a child entity of a level, and it may be a child entity of one or more dimension attributes.

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. If the dimension is stored in a parent/child table instead of a level-based table, level attributes map to columns in the view you can generate via the `CWM2_OLAP_PC_TRANSFORM` package. Similarly, if the dimension is stored in an analytic workspace, level attributes map to columns in the view you can generate via the `AWCONVERT` package.

Creating Level Attributes

The `CWM2_OLAP_LEVEL_ATTRIBUTE` package contains procedures that establish entities for level attributes within the OLAP 2 Catalog.

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 a level attribute does not involve any mapping to warehouse dimension tables.

The parent dimension, parent level, and parent dimension attribute must already exist in the OLAP 2 Catalog before you can create a level attribute.

Completing the Dimension's Metadata

Creating level attributes is one of the final steps in creating the OLAP metadata for a dimension. Once you have created the level attributes, you have probably created all the entities that comprise the dimension. You then need to call procedures in the CWM2_OLAP_TABLE_MAP package to map levels and level attributes to columns in one or more dimension tables.

Common Logic in CWM2_OLAP_LEVEL_ATTRIBUTE Subprograms

Each procedure first checks the calling user's security privileges. The calling user must be the dimension owner and must have the OLAP_DBA role. If the calling user does not meet the security requirements, the procedure fails with an exception.

Each procedure then checks for the existence of the parent dimension, the parent dimension attribute, and the parent level within the OLAP 2 Catalog. If any of the parent entities do not exist, the procedure fails with an exception.

Each procedure then checks for the existence of the level attribute. All procedures, except CREATE_LEVEL_ATTRIBUTE, return an error if the level attribute does not already exist.

Case Requirements for Subprogram Parameters

You can specify arguments in lower case, upper case, or mixed case.

If the argument is a metadata entity name (for example, level_attribute_name) or a value that will be used in further processing by other procedures, the procedure converts the argument to upper case. For all other arguments, the case that you specify is retained.

Summary of CWM2_OLAP_LEVEL_ATTRIBUTE Subprograms

Table 21–1 CWM2_OLAP_LEVEL_ATTRIBUTE Subprograms

Subprogram	Description
CREATE_LEVEL_ATTRIBUTE on page 21-4	Creates a level attribute.
DROP_LEVEL_ATTRIBUTE Procedure on page 21-6	Drops a level attribute.
LOCK_LEVEL_ATTRIBUTE Procedure on page 21-7	Locks the level attribute metadata for update.
SET_DESCRIPTION Procedure on page 21-8	Sets the description for a level attribute.
SET_DISPLAY_NAME Procedure on page 21-9	Sets the display name for a level attribute.

Table 21–1 CWM2_OLAP_LEVEL_ATTRIBUTE Subprograms

Subprogram	Description
SET_LEVEL_ATTRIBUTE_NAME Procedure on page 21-10	Sets the name of a level attribute.
SET_SHORT_DESCRIPTION Procedure on page 21-12	Sets the short description for a level attribute.

CREATE_LEVEL_ATTRIBUTE

This procedure registers a new level attribute as an entity in the OLAP 2 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.

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 21–2 CREATE_LEVEL_ATTRIBUTE Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.

Table 21–2 CREATE_LEVEL_ATTRIBUTE Procedure Parameters

Parameter	Description
`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.

`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 21–3 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`	The parent dimension does not exist within the OLAP 2 Catalog.
`dimension_attribute_not_found`	The parent dimension attribute does not exist for this dimension.
`level_not_found`	The parent level does not exist for this dimension.
`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 2 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 21–4 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 21–5 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`	The parent dimension does not exist within the OLAP 2 Catalog.
`dimension_attribute_not_found`	This dimension attribute does not exist for this dimension.
`level_not_found`	This level does not exist for this dimension.
`level_attribute_not_found`	This level attribute does not exist for this dimension attribute and level combination.

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 OLAP 2 Catalog.

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 21–6 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 21–7 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`	The parent dimension does not exist within the OLAP 2 Catalog.

Table 21–7 LOCK_LEVEL_ATTRIBUTE Procedure Exceptions

Exception	Description
`dimension_attribute_not_found`	This dimension attribute does not exist for this dimension.
`level_not_found`	This level does not exist for this dimension.
`level_attribute_not_found`	This level attribute does not exist for this level and dimension attribute combination.

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 21–8 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 21–9 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.
dimension_not_found	The parent dimension does not exist within the OLAP 2 Catalog.
dimension_attribute_not_found	This dimension attribute does not exist for this dimension.
level_not_found	This level does not exist for this dimension.
level_attribute_not_found	This level attribute does not exist for this dimension attribute and level combination.

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 21–10 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.

Table 21–10 SET_DISPLAY_NAME Procedure Parameters

Parameter	Description
`level_attribute_name`	Name of the level attribute.
`display_name`	Display name for the level attribute.

Exceptions

Table 21–11 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.
`dimension_not_found`	The parent dimension does not exist within the OLAP 2 Catalog.
`dimension_attribute_not_found`	This dimension attribute does not exist for this dimension.
`level_not_found`	This level does not exist for this dimension.
`level_attribute_not_found`	This level attribute does not exist for this dimension attribute and level combination.

SET_LEVEL_ATTRIBUTE_NAME Procedure

This procedure sets the name for a level attribute.

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 21–12 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.
`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 21–13 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`	The parent dimension does not exist within the OLAP 2 Catalog.
`dimension_attribute_not_found`	This dimension attribute does not exist for this dimension.
`level_not_found`	This level does not exist for this dimension.
`level_attribute_not_found`	This level attribute does not exist for this dimension attribute and level combination.

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 21–14   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 21–15   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.
`dimension_not_found`	The parent dimension does not exist within the OLAP 2 Catalog.
`dimension_attribute_not_found`	This dimension attribute does not exist for this dimension.

Table 21–15 SET_SHORT_DESCRIPTION Procedure Exceptions

Exception	Description
`level_not_found`	This level does not exist for this dimension.
`level_attribute_not_found`	This level attribute does not exist for this dimension attribute and level combination.

22 CWM2_OLAP_CUBE

The CWM2_OLAP_CUBE package provides procedures for creating, dropping, and locking cubes, and for adding dimensions to cubes. It also provides procedures for setting general properties of cubes.

This chapter discusses the following topics:

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 2 Catalog.

OLAP cubes represent dimensioned data. Cubes must exist for any data that you wish to make accessible to the Oracle OLAP API.

Cubes and Measures

A cube is a dimensional 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.

A measure's data can be accessed by specifying values for its dimensions. For example, a measure representing sales data might be dimensioned by time, product, and location. This means that the sales data can be accessed for a given time period, a given product, and a given location.

Note: A cube may contain multiple measures. All the measures of a cube share the same set of dimensions.

Fact Table Requirements

The fact tables or views that underlie a cube have data columns and key columns. The data columns store the source data for the cube's measures. The key columns reference the dimension tables that underlie the cube's dimensions.

If your measures and dimensions are stored in an analytic workspace within the database, you must call procedures in the `AWCONVERT` package to create the fact tables as relational views that reference the workspace. Then you can call procedures in the `CWM2_OLAP_CUBE` package to create cubes in the OLAP 2 Catalog.

Fact Table Key Columns

Each fact table key column references a level column in a dimension table. The level is mapped to one of the dimension's hierarchies. This means that fact data is stored on a per-hierarchy basis.

Note: The measures for a single cube may be stored in more than one fact table. The key columns of each fact table may store different hierarchies for the same dimensions.

Creating Cubes

The `CWM2_OLAP_CUBE` package contains procedures that establish cube entities within the OLAP 2 Catalog.

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 of metadata does not fully define a dimension or a cube, nor does it involve any mapping to warehouse dimension tables or fact tables.

Creating a Cube's Dimensions

Before you create a cube, you must create its dimensions by calling procedures in the following packages:

- CWM2_OLAP_DIMENSION
- CWM2_OLAP_DIMENSION_ATTRIBUTE
- CWM2_OLAP_HIERARCHY
- CWM2_OLAP_LEVEL
- CWM2_OLAP_LEVEL_ATTRIBUTE

Completing the Cube's Metadata

Once you have created a cube, you will need to call procedures in the following packages to fully define the cube's metadata:

- CWM2_OLAP_MEASURE to create measures and add them to the cube
- CWM2_OLAP_TABLE_MAP to establish the mapping to columns in one or more fact tables

Verifying the Metadata for a Cube

To be valid, a cube must have at least one valid dimension and at least one measure that is correctly mapped to a column in a fact table.

To test the validity of the cube's dimensional metadata, use the `VALIDATE_DIMENSION` procedure in the `CWM2_OLAP_VALIDATE` package.

To test the validity of the cube itself, use the `VALIDATE_CUBE` procedure in the `CWM2_OLAP_VALIDATE` package.

Common Logic in CWM2_OLAP_CUBE Subprograms

Each procedure first checks the calling user's security privileges. The calling user must be the cube owner and must have the `OLAP_DBA` role. If the calling user does not meet the security requirements, the procedure fails with an exception.

Each procedure then checks for the existence of the cube specified by `cube_owner` and `cube_name` within the OLAP 2 Catalog. All procedures, except `CREATE_CUBE`, return an error if the cube does not already exist.

Case Requirements for Subprogram Parameters

You can specify arguments in lower case, upper case, or mixed case.

If the argument is a metadata entity name (for example, `cube_name`) or a value that will be used in further processing by other procedures, the procedure converts the argument to upper case. For all other arguments, the case that you specify is retained.

Summary of CWM2_OLAP_CUBE Subprograms

Table 22–1 CWM2_OLAP_CUBE Subprograms

Subprogram	Description
ADD_DIMENSION_TO_CUBE Procedure on page 22-5	Adds a dimension to a cube.
CREATE_CUBE Procedure on page 22-6	Creates a cube.
DROP_CUBE Procedure on page 22-7	Drops a cube.

Table 22–1 CWM2_OLAP_CUBE Subprograms

Subprogram	Description
LOCK_CUBE Procedure on page 22-8	Locks a cube's metadata for update.
REMOVE_DIMENSION_FROM_CUBE Procedure on page 22-9	Removes a dimension from a cube.
SET_CUBE_NAME Procedure on page 22-10	Sets the name of a cube.
SET_DEFAULT_CUBE_DIM_CALC_HIER Procedure on page 22-11	Sets the default calculation hierarchy for a dimension of the cube.
SET_DESCRIPTION Procedure on page 22-12	Sets the description for a cube.
SET_DISPLAY_NAME Procedure on page 22-13	Sets the display name for a cube.
SET_MV_SUMMARY_CODE Procedure on page 22-13	Sets the format for materialized views associated with a cube.
SET_SHORT_DESCRIPTION Procedure on page 22-14	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 22–2 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 22–3 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.
cube_not_found	Cannot update cube. Cube does not exist within the OLAP 2 Catalog.
dimension_not_found	Dimension does not exist or is not accessible within the OLAP 2 Catalog.

CREATE_CUBE Procedure

This procedure registers a new cube entity in the OLAP 2 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 22–4 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 22–5 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`	Cannot create cube. Cube already exists within the OLAP 2 Catalog.

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 22–6 DROP_CUBE Procedure Parameters

Parameter	Description
`cube_owner`	Owner of the cube.
`cube_name`	Name of the cube.

Exceptions

Table 22–7 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`	Cannot drop cube. Cube does not exist within the OLAP 2 Catalog.

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 OLAP 2 Catalog.

Syntax

```
LOCK_CUBE (
          cube_owner       IN   VARCHAR2,
          cube_name        IN   VARCHAR2.
          wait_for_lock    IN   BOOLEAN DEFAULT FALSE);
```

Parameters

Table 22–8 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 22–9 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`	Cannot lock cube. Cube does not exist within the OLAP 2 Catalog.

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 22–10 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 22–11 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`	Cannot update cube. Cube does not exist within the OLAP 2 Catalog.
`dimension_not_found`	This dimension not found for this cube.

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 22–12 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 22–13 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`	Cannot update cube. Cube does not exist within the OLAP 2 Catalog.

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 22–14 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 22–15 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`	Cannot update cube. Cube does not exist within the OLAP 2 Catalog.

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 22–16 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 22–17 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`	Cannot update cube. Cube does not exist within the OLAP 2 Catalog.

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 22–18 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 22–19 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`	Cannot update cube. Cube does not exist within the OLAP 2 Catalog.

SET_MV_SUMMARY_CODE Procedure

This procedure specifies the form of materialized views for this cube. Materialized views may be in Grouping Set (`GS`) or Rolled Up (`RU`) 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 22–20 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: ■ RU, for Rolled Up form. ■ GS, for Grouping Set form.

Exceptions

Table 22–21 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	Cannot update cube. Cube does not exist within the OLAP 2 Catalog.

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 22–22 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 22–23 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`	Cannot update cube. Cube does not exist within the OLAP 2 Catalog.

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.

This chapter discusses the following topics:

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 2 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. Cubes must exist for any data that you wish to make accessible to the Oracle OLAP API.

A measure's data can be accessed by specifying values for its dimensions. For example, a measure representing sales data might be dimensioned by time, product, and location. This means that the sales data can be accessed for a given time period, a given product, and a given location.

Note: A cube may contain multiple measures. All the measures of a cube share the same set of dimensions.

Creating Measures

The `CWM2_OLAP_MEASURE` package contains procedures that establish measure entities within the OLAP 2 Catalog.

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 of metadata does not fully define a dimension or a cube, nor does it involve any mapping to warehouse dimension tables or fact tables.

Creating a Cube for the Measure

Before you create a measure, you must create the cube that will provide its context. To create the cube, use procedures in the CWM2_OLAP_CUBE package.

Prior to creating the cube, you must create the cube's dimensions by calling procedures in the following packages:

- CWM2_OLAP_DIMENSION
- CWM2_OLAP_DIMENSION_ATTRIBUTE

- CWM2_OLAP_HIERARCHY
- CWM2_OLAP_LEVEL
- CWM2_OLAP_LEVEL_ATTRIBUTE

Completing the Measure's Metadata

Once you have created a measure, you will need to call procedures in the CWM2_OLAP_TABLE_MAP package to establish the mapping to columns in one or more fact tables.

Common Logic in CWM2_OLAP_MEASURE Subprograms

Each procedure first checks the calling user's security privileges. The calling user must be the cube owner and must have the `OLAP_DBA` role. If the calling user does not meet the security requirements, the procedure fails with an exception.

Each procedure then checks for the existence of the cube specified by `cube_owner` and `cube_name` within the OLAP 2 Catalog. All procedures return an error if the cube does not already exist.

Each procedure then checks for the existence of the measure specified by `measure_name`. All procedures, except `CREATE_MEASURE`, return an error if the measure does not already exist for this cube.

Case Requirements for Subprogram Parameters

You can specify arguments in lower case, upper case, or mixed case.

If the argument is a metadata entity name (for example, `measure_name`) or a value that will be used in further processing by other procedures, the procedure converts the argument to upper case. For all other arguments, the case that you specify is retained.

Summary of CWM2_OLAP_MEASURE Subprograms

Table 23–1 CWM2_OLAP_MEASURE Subprograms

Subprogram	Description
CREATE_MEASURE Procedure on page 23-4	Creates a measure.
DROP_MEASURE Procedure on page 23-5	Drops a measure.
LOCK_MEASURE Procedure on page 23-6	Locks a measure's metadata for update.
SET_DESCRIPTION Procedure on page 23-7	Sets the description for a measure.
SET_DISPLAY_NAME Procedure on page 23-8	Sets the display name for a measure.
SET_MEASURE_NAME Procedure on page 23-9	Sets the name of a measure.
SET_SHORT_DESCRIPTION Procedure on page 23-10	Sets the short description for a measure.

CREATE_MEASURE Procedure

This procedure registers a new measure entity in the OLAP 2 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`	Cannot create measure. Cube does not exist within the OLAP 2 Catalog.
`measure_already_exists`	Cannot create measure. 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`	Cannot drop measure. Cube does not exist within the OLAP 2 Catalog.
`measure_not_found`	Cannot drop measure. This measure does not exist for this cube.

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 OLAP 2 Catalog.

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`	Cannot lock measure. Its cube does not exist within the OLAP 2 Catalog.
`measure_not_found`	Cannot lock measure. This measure does not exist for this cube.

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`	Cannot update measure. Its cube does not exist within the OLAP 2 Catalog.
`measure_not_found`	Cannot update measure. This measure does not exist for this cube.

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`	Cannot update measure. Its cube does not exist within the OLAP 2 Catalog.
`measure_not_found`	Cannot update measure. This measure does not exist for this cube.

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	Cannot update measure. Its cube does not exist within the OLAP 2 Catalog.
measure_not_found	Cannot update measure. This measure does not exist for this cube.

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`	Cannot update measure. Its cube does not exist within the OLAP 2 Catalog.
`measure_not_found`	Cannot update measure. This measure does not exist for this cube.

24

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.

This chapter discusses the following topics:

- Understanding OLAP Metadata Mapping
- Common Logic in CWM2_OLAP_TABLE_MAP Subprograms
- Summary of CWM2_OLAP_TABLE_MAP Subprograms

Understanding OLAP Metadata Mapping

OLAP metadata mapping is the process of establishing the links between logical metadata entities and the physical locations where the data is stored. 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.

Note: The dimension tables and fact tables may be implemented as views. For example, the views you can generate using the CWM2_OLAP_AW_ACCESS package may be a 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 Logical Metadata Entities

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).

Each dimension level attribute maps to a single column. Level attributes must be mapped within the same table as their associated levels.

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.

Joining Fact Tables with Dimension Tables

Once you have mapped the levels, 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 mapping between a fact table and its dimension tables may be specified for any of the following configurations:

- **LOWEST LEVEL.** A single fact table stores lowest level 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.

This warehouse configuration is required for metadata in the OLAP 1 Catalog. It is supported, but not required, for metadata in the OLAP 2 Catalog.

- **EMBEDDED TOTAL.** Fact tables store embedded totals and lowest level data for specific hierarchies of the cube's dimensions. Typically, the data for each combination of hierarchies is stored in a separate fact table. 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 maps to corresponding columns that identify a dimension hierarchy in a solved dimension table.

  This warehouse configuration is supported for metadata in the OLAP 2 Catalog only.

- **ROLLED UP**. This type of warehouse has embedded total fact tables and solved, hierarchical dimension tables as in EMBEDDED TOTAL mode. All mapping to dimension tables is in the context of hierarchies as in EMBEDDED TOTAL mode.

  However, in ROLLED UP mode, there are 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.

  This warehouse configuration is supported for metadata in the OLAP 2 Catalog only.

Common Logic in CWM2_OLAP_TABLE_MAP Subprograms

Each procedure first checks the calling user's security privileges. The calling user must have the `OLAP_DBA` role and must be the owner of the entity that will be mapped. If the calling user does not meet the security requirements, the procedure fails with an exception.

Each procedure then checks for the existence of the metadata entity within the OLAP 2 Catalog. All procedures return an error if the entity does not already exist.

Each procedure then checks for the existence of tables and columns. If these do not exist in the data dictionary, an error is generated.

Case Requirements for Subprogram Parameters

You can specify arguments in lower case, upper case, or mixed case.

If the argument is a metadata entity name (for example, `cube_name` or `dimension_name`) or a value that will be used in further processing by other procedures, the procedure converts the argument to upper case. For all other arguments, the case that you specify is retained.

Summary of CWM2_OLAP_TABLE_MAP Subprograms

Table 24–1 CWM2_OLAP_TABLE_MAP

Subprogram	Description
MAP_DIMTBL_HIERLEVELATTR Procedure on page 24-5	Maps a hierarchical level attribute to a column in a dimension table.
MAP_DIMTBL_HIERLEVEL Procedure on page 24-6	Maps a hierarchical level to one or more columns in a dimension table.
MAP_DIMTBL_HIERSORTKEY Procedure on page 24-8	Sorts the members of a hierarchy within a column of a dimension table.
MAP_DIMTBL_LEVELATTR Procedure on page 24-9	Maps a non-hierarchical level attribute to a column in a dimension table
MAP_DIMTBL_LEVEL Procedure on page 24-10	Maps a non-hierarchical level to one or more columns in a dimension table.
MAP_FACTTBL_LEVELKEY Procedure on page 24-12	Maps the dimensions of a cube to a fact table.
MAP_FACTTBL_MEASURE Procedure on page 24-14	Maps a measure to a column in a fact table.
REMOVEMAP_DIMTBL_HIERLEVELATTR Procedure on page 24-15	Removes the mapping of a hierarchical level attribute from a column in a dimension table.
REMOVEMAP_DIMTBL_HIERLEVEL Procedure on page 24-17	Removes the mapping of a hierarchical level from one or more columns in a dimension table.
REMOVEMAP_DIMTBL_HIERSORTKEY Procedure on page 24-18	Removes custom sorting criteria associated with columns in a dimension table.

Table 24–1 CWM2_OLAP_TABLE_MAP

Subprogram	Description
REMOVEMAP_DIMTBL_LEVELATTR Procedure on page 24-19	Removes the mapping of a non-hierarchical level attribute from a column in a dimension table.
REMOVEMAP_DIMTBL_LEVEL Procedure on page 24-20	Removes the mapping of a non-hierarchical level from one or more columns in a dimension table.
REMOVEMAP_FACTTBL_LEVELKEY Procedure on page 24-21	Removes the mapping of a cube's dimensions from a fact table.
REMOVEMAP_FACTTBL_MEASURE Procedure on page 24-22	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 24–2 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.

Table 24–2 MAP_DIMTBL_HIERLEVELATTR Procedure Parameters

Parameter	Description
`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 24–3 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 does not exist within the OLAP 2 Catalog.
`hierarchy_not_found`	This hierarchy does not exist for this dimension.
`level_not_found`	This level does not exist for this dimension.
`attribute_not_found`	This level attribute does not exist for this level.
`table_not_found`	Dimension table does not exist or is not accessible to the calling user.
`column_not_found`	This column not found in this dimension table.

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 24–4 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.
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 24–5 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.

Table 24–5 MAP_DIMTBL_HIERLEVEL Procedure Exceptions

Exception	Description
`dimension_not_found`	Dimension does not exist within the OLAP 2 Catalog.
`hierarchy_not_found`	This hierarchy does not exist for this dimension.
`level_not_found`	This level does not exist for this dimension.
`table_not_found`	Dimension table does not exist or is not accessible to the calling user.
`column_not_found`	This column not found in this dimension table.

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 24–6 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.

Table 24–6 MAP_DIMTBL_HIERSORTKEY Procedure Parameters

Parameter	Description
`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 24–7 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 does not exist within the OLAP 2 Catalog.
`hierarchy_not_found`	This hierarchy does not exist for this dimension.

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 24–8 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 24–9 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 does not exist within the OLAP 2 Catalog.
`level_not_found`	This level does not exist for this dimension.
`attribute_not_found`	This level attribute does not exist for this level.
`table_not_found`	Dimension table does not exist or is not accessible to the calling user.
`column_not_found`	This column not found in this dimension table.

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 24–10 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 24–11 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 does not exist within the OLAP 2 Catalog.
`level_not_found`	This level does not exist for this dimension.
`table_not_found`	Dimension table does not exist or is not accessible to the calling user.
`column_not_found`	This column not found in this dimension table.

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.

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 24–12 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.

Table 24–12 MAP_FACTTBL_LEVELKEY Procedure Parameters

Parameter	Description
`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 For more information on storetype, see Joining Fact Tables with Dimension Tables.
`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 24–13 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 does not exist within the OLAP 2 Catalog.
`fact_table_not_found`	Fact table does not exist or is not accessible to the calling user

Example

The following call to the MAP_FACTTBL_LEVELKEY procedure maps a cube named ANALYTIC_CUBE_AW in the schema XADEMO to a fact table named XADEMO_AW_SALES_VIEW_4 in the same schema. The fact table stores lowest level data and embedded totals for all level combinations. The cube has four dimensions: PRODUCT, CHANNEL, TIME, and GEOGRAPHY.

```
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;');
```

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 24–14 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.

Table 24–14 MAP_FACTTBL_MEASURE Procedure Parameters

Parameter	Description
`facttable_name`	Name of the fact table.
`column_name`	Column in the fact table to which the measure will be 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`*`/COL:`*`columnname`*`;` This string must also be specified as an argument to the `MAP_FACTTBL_HIERLEVELKEY` procedure.

Exceptions

Table 24–15 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 does not exist within the OLAP 2 Catalog.
`fact_table_not_found`	Fact table does not exist or is not accessible to the calling user
`measure_not_found`	This measure does not exist in this cube.
`column_not_found`	This column does not exist in this fact table.

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 24–16 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 24–17 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 does not exist within the OLAP 2 Catalog.
`hierarchy_not_found`	This hierarchy does not exist for this dimension.
`level_not_found`	This level does not exist for this dimension.
`attribute_not_found`	This level attribute does not exist for this level.

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 24–18 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 24–19 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 does not exist within the OLAP 2 Catalog.
`hierarchy_not_found`	This hierarchy does not exist for this dimension.
`level_not_found`	This level does not exist for this dimension.

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 24–20 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 24–21 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 does not exist within the OLAP 2 Catalog.
`hierarchy_not_found`	This hierarchy does not exist for this dimension.

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 24–22 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 24–23 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 does not exist within the OLAP 2 Catalog.
level_not_found	This level does not exist for this dimension.
attribute_not_found	This level attribute does not exist for this level.

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 24–24 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 24–25 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 does not exist within the OLAP 2 Catalog.
level_not_found	Level does not exist within the OLAP 2 Catalog.

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 24–26 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 24–27 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 does not exist within the OLAP 2 Catalog.
`fact_table_not_found`	Fact table does not exist or is not accessible to the calling user

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 24–28 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 24–29 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 does not exist within the OLAP 2 Catalog.
`fact_table_not_found`	Fact table does not exist or is not accessible to the calling user
`measure_not_found`	This measure does not exist in this cube.
`column_not_found`	This column does not exist in this fact table.

25

CWM2_OLAP_AW_OBJECT

The CWM2_OLAP_AW_OBJECT package provides procedures for registering metadata in the OLAP Catalog for a data warehouse that is stored in an analytic workspace.

Note: In no way does the CWM2_OLAP_AW_OBJECT package affect actual objects in the analytic workspace.

To map these metadata entities to the analytic workspace, use the procedures in the CWM2_OLAP_AW_MAP package.

This chapter discusses the following topics:

- Understanding AW Object Metadata Entities
- Common Logic in CWM2_OLAP_AW_OBJECT Subprograms
- Summary of CWM2_OLAP_AW_OBJECT Subprograms

Understanding AW Object Metadata Entities

An AW metadata object within the OLAP Catalog represents an object defined within an analytic workspace.

Objects within analytic workspaces are defined and manipulated by the OLAP Data Manipulation Language (DML).

> **See Also:** For information on the OLAP DML, see *Oracle9i OLAP Developer's Guide to the OLAP DML* and the Oracle9i OLAP DML Reference help.

Common Logic in CWM2_OLAP_AW_OBJECT Subprograms

Each procedure first checks the calling user's security privileges. The calling user must have the `OLAP_DBA` role and must be the owner of the entity that will be mapped. If the calling user does not meet the security requirements, the procedure fails with an exception.

Case Requirements for Subprogram Parameters

You can specify arguments in lower case, upper case, or mixed case.

If the argument is a metadata entity name (for example, `measure_name` or `dimension_name`) or a value that will be used in further processing by other procedures, the procedure converts the argument to upper case. For all other arguments, the case that you specify is retained.

Summary of CWM2_OLAP_AW_OBJECT Subprograms

Table 25–1 CWM2_OLAP_AW_OBJECT

Subprogram	Description
CREATE_AW_OBJECT Procedure on page 25-3	Creates an AW metadata object in the OLAP Catalog. This metadata object represents an object defined within an analytic workspace.
CREATE_AW_OBJECT_INFO Procedure on page 25-4	Creates additional information about an AW metadata object in the OLAP Catalog.
CREATE_AW_OBJECT_RELATED_OBJ Procedure on page 25-5	Creates a relationship between two AW metadata objects in the OLAP Catalog.

Table 25–1 CWM2_OLAP_AW_OBJECT

Subprogram	Description
CREATE_AW_OBJECT_PROPERTY Procedure on page 25-6	Creates a property for an AW metadata object in the OLAP Catalog.
DROP_AW_OBJECT Procedure on page 25-7	Drops an AW metadata object from the OLAP Catalog.
DROP_AW_OBJECT_INFO Procedure on page 25-7	Drops additional information about an AW metadata object from the OLAP Catalog.
DROP_AW_OBJECT_RELATED_OBJ Procedure on page 25-8	Drops a relationship between two AW metadata objects in the OLAP Catalog.
DROP_AW_OBJECT_PROPERTY Procedure on page 25-9	Drops a property of an AW metadata object in the OLAP Catalog.

CREATE_AW_OBJECT Procedure

This procedure creates an AW object within the OLAP Catalog. This object may represent a dimension, variable, relation, or any of the object types that are supported within analytic workspaces.

Syntax

```
CREATE_AW_OBJECT (
          aw_owner             IN   VARCHAR2  (30),
          aw_name              IN   VARCHAR2  (30),
          aw_object_name       IN   VARCHAR2  (30),
          aw_object_type       IN   VARCHAR2  (30),
          aw_object_datatype   IN   VARCHAR2  (30));
```

Parameters

Table 25–2 CREATE_AW_OBJECT Procedure Parameters

Parameter	Description
aw_owner	Owner of analytic workspace.
aw_name	Name of the analytic workspace.
aw_object_name	Name of the analytic workspace object.

Table 25–2 CREATE_AW_OBJECT Procedure Parameters

Parameter	Description
`aw_object_type`	Type of the analytic workspace object (dimension, variable, relation, and so on).
`aw_object_datatype`	Data type of the analytic workspace object.

CREATE_AW_OBJECT_INFO Procedure

This procedure creates additional information about an analytic workspace object that has already been registered in the OLAP Catalog by the `CREATE_AW_OBJECT` procedure.

For example, an analytic workspace dimension may have width defined to a specific value. Or a composite or conjoint dimension may have `BTREE`/`HASH`/`NOHASH` explicitly defined.

Syntax

```
CREATE_AW_OBJECT_INFO (
          aw_owner             IN   VARCHAR2  (30),
          aw_name              IN   VARCHAR2  (30),
          aw_object_name       IN   VARCHAR2  (30),
          aw_object_attr_type  IN   VARCHAR2  (30),
          aw_object_attr_name  IN   VARCHAR2  (30));
```

Parameters

Table 25–3 CREATE_AW_OBJECT_INFO Procedure Parameters

Parameter	Description
`aw_owner`	Owner of analytic workspace.
`aw_name`	Name of the analytic workspace.
`aw_object_name`	Name of the analytic workspace object.
`aw_object_attr_type`	Analytic workspace object attribute extension.
`aw_object_attr_name`	Analytic workspace object attribute extension value.

CREATE_AW_OBJECT_RELATED_OBJ Procedure

This procedure relates one AW object to another AW object. This relationship might be the dimensions associated with a variable, or the related dimensions of a relation, and so on.

Syntax

```
CREATE_AW_OBJECT_RELATED_OBJ (
          aw_owner                IN   VARCHAR2  (30),
          aw_name                 IN   VARCHAR2  (30),
          aw_object_name          IN   VARCHAR2  (30),
          aw_object_related_name  IN   VARCHAR2  (30),
          aw_object_related_type  IN   VARCHAR2  (30)
          position                IN   NUMBER);
```

Parameters

Table 25–4 CREATE_AW_OBJECT_RELATED_OBJ Procedure Parameters

Parameter	Description
`aw_owner`	Owner of analytic workspace.
`aw_name`	Name of the analytic workspace.
`aw_object_name`	Name of the AW object.
`aw_object_related_name`	The new object that is related to `aw_object_name`.
`aw_object_related_type`	How `aw_object_related_name` is related to `aw_object_name`. Values can be either: `DIMENSION` or `RELATION`.
`position`	Position order of related object in case of dimensionality of a variable.

CREATE_AW_OBJECT_PROPERTY Procedure

This procedure creates a property of an AW object.

Syntax

```
CREATE_AW_OBJECT_PROPERTY (
          aw_owner                 IN   VARCHAR2  (30),
          aw_name                  IN   VARCHAR2  (30),
          aw_object_name           IN   VARCHAR2  (30),
          aw_object_prop_name      IN   VARCHAR2  (30),
          aw_object_prop_datatype  IN   VARCHAR2  (30);
```

Parameters

Table 25–5 CREATE_AW_OBJECT_PROPERTY Procedure Parameters

Parameter	Description
`aw_owner`	Owner of analytic workspace.
`aw_name`	Name of the analytic workspace.
`aw_object_name`	Name of the AW object.
`aw_object_prop_name`	Property of the AW object.
`aw_object_prop_datatype`	Data type of the property.

DROP_AW_OBJECT Procedure

This procedure drops an AW object from the OLAP Catalog.

Syntax

```
DROP_AW_OBJECT (
          aw_owner            IN   VARCHAR2  (30),
          aw_name             IN   VARCHAR2  (30),
          aw_object_name      IN   VARCHAR2  (30));
```

Parameters

Table 25–6 DROP_AW_OBJECT Procedure Parameters

Parameter	Description
`aw_owner`	Owner of analytic workspace.
`aw_name`	Name of the analytic workspace.
`aw_object_name`	Name of the AW object.

DROP_AW_OBJECT_INFO Procedure

Drops the information that was created by `CREATE_AW_OBJECT_INFO`.

This procedure drops the additional information that was added to the AW object. It does not drop the AW object.

Syntax

```
DROP_AW_OBJECT_INFO (
          aw_owner             IN   VARCHAR2  (30),
          aw_name              IN   VARCHAR2  (30),
          aw_object_name       IN   VARCHAR2  (30),
          aw_object_attr_type  IN   VARCHAR2  (30),
          aw_object_attr_name  IN   VARCHAR2  (30));
```

Parameters

Table 25–7 DROP_AW_OBJECT_INFO Procedure Parameters

Parameter	Description
aw_owner	Owner of analytic workspace.
aw_name	Name of the analytic workspace.
aw_object_name	Name of the AW object.
aw_object_attr_type	AW object attribute extension.
aw_object_attr_name	AW object attribute extension value.

DROP_AW_OBJECT_RELATED_OBJ Procedure

This procedure drops a relationship between one AW object and another AW object. This relationship might be the dimensions associated with a variable, or the related dimensions of a relation, and so on.

Syntax

```
DROP_AW_OBJECT_RELATED_OBJ (
          aw_owner                IN   VARCHAR2  (30),
          aw_name                 IN   VARCHAR2  (30),
          aw_object_name          IN   VARCHAR2  (30),
          aw_object_related_name  IN  VARCHAR2   (30));
```

Parameters

Table 25–8 DROP_AW_OBJECT_RELATED_OBJ Procedure Parameters

Parameter	Description
aw_owner	Owner of analytic workspace.
aw_name	Name of the analytic workspace.

Table 25–8 DROP_AW_OBJECT_RELATED_OBJ Procedure Parameters

Parameter	Description
aw_object_name	Name of the AW object.
aw_object_related_ name	Related AW object.

DROP_AW_OBJECT_PROPERTY Procedure

This procedure drops a property of an AW object.

Syntax

```
DROP_AW_OBJECT_PROPERTY (
          aw_owner                 IN   VARCHAR2  (30),
          aw_name                  IN   VARCHAR2  (30),
          aw_object_name           IN   VARCHAR2  (30),
          aw_object_prop_name      IN   VARCHAR2  (30);
```

Parameters

Table 25–9 DROP_AW_OBJECT_PROPERTY Procedure Parameters

Parameter	Description
aw_owner	Owner of analytic workspace.
aw_name	Name of the analytic workspace.
aw_object_name	Name of the AW object.
aw_object_prop_name	Property of the AW object.

26

CWM2_OLAP_AW_MAP

The CWM2_OLAP_AW_MAP package provides procedures for mapping logical metadata entities to objects defined in analytic workspaces. The mapping information is stored in a set of OLAP Catalog metadata tables.

The OLAP metadata entities must have previously been created by procedures in the CWM2_OLAP_AW_OBJECT package.

This chapter discusses the following topics:

Understanding AW Object Metadata Mapping

OLAP metadata mapping is the process of establishing the links between logical metadata entities and the physical locations where the data is stored. The `CWM2_OLAP_AW_MAP` package provides procedures for linking OLAP metadata entities to objects in analytic workspaces.

Objects within analytic workspaces are defined and manipulated by the OLAP Data Manipulation Language (DML).

> **See Also:** For information on the OLAP DML, see *Oracle9i OLAP Developer's Guide to the OLAP DML* and the Oracle9i OLAP DML Reference help.

Common Logic in CWM2_OLAP_AW_MAP Subprograms

Each procedure first checks the calling user's security privileges. The calling user must have the `OLAP_DBA` role and must be the owner of the entity that will be mapped. If the calling user does not meet the security requirements, the procedure fails with an exception.

Case Requirements for Subprogram Parameters

You can specify arguments in lower case, upper case, or mixed case.

If the argument is a metadata entity name (for example, `measure_name` or `dimension_name`) or a value that will be used in further processing by other procedures, the procedure converts the argument to upper case. For all other arguments, the case that you specify is retained.

Summary of CWM2_OLAP_AW_MAP Subprograms

Table 26–1 CWM2_OLAP_AW_MAP

Subprogram	Description
`MAP_AW_ATTRIBUTE Procedure on page 26-3`	Maps a dimension attribute in the OLAP Catalog to attribute information in an analytic workspace.
`MAP_AW_DIMENSION Procedure on page 26-5`	Maps a dimension in the OLAP Catalog to a dimension in an analytic workspace.
`MAP_AW_HIERARCHY Procedure on page 26-6`	Maps a hierarchy in the OLAP Catalog to hierarchy information in an analytic workspace.

Table 26–1 CWM2_OLAP_AW_MAP

Subprogram	Description
MAP_AW_LEVEL Procedure on page 26-7	Maps a level in the OLAP Catalog to level information in an analytic workspace.
MAP_AW_MEASURE Procedure on page 26-9	Maps a measure in the OLAP Catalog to a variable in an analytic workspace.
REMOVEMAP_AW_ATTRIBUTE Procedure on page 26-11	Removes the mapping between an attribute in the OLAP Catalog and attribute information in an analytic workspace.
REMOVEMAP_AW_DIMENSION Procedure on page 26-12	Removes the mapping between a dimension in the OLAP Catalog and a dimension in an analytic workspace.
REMOVEMAP_AW_HIERARCHY Procedure on page 26-12	Removes the mapping between a hierarchy in the OLAP Catalog and hierarchy information in an analytic workspace.
REMOVEMAP_AW_LEVEL Procedure on page 26-13	Removes the mapping between a level in the OLAP Catalog and level information in an analytic workspace.
REMOVEMAP_AW_MEASURE Procedure on page 26-14	Removes the mapping between a measure in the OLAP Catalog and a variable in an analytic workspace.

MAP_AW_ATTRIBUTE Procedure

This procedure associates a logical dimension attribute in the OLAP Catalog with attribute information in an analytic workspace.

The dimension attribute entity in the OLAP Catalog must have been created by procedures in the CWM2_OLAP_AW_OBJECT package.

Note: Whereas dimensions and variables (measures) are native to the analytic workspace, there is no single native workspace object for a dimension attribute.

Syntax

```
MAP_AW_ATTRIBUTE (
          dimension_owner   IN   VARCHAR2  (30),
          dimension_name    IN   VARCHAR2  (30),
          attribute_name    IN   VARCHAR2  (30),
          aw_owner          IN   VARCHAR2  (30),
          aw_name           IN   VARCHAR2  (30),
          aw_object_name    IN   VARCHAR2  (30),
          aw_limit_type     IN   VARCHAR2  (30),
          aw_limit_value    IN   VARCHAR2  (255));
```

Parameters

Table 26–2 MAP_AW_ATTRIBUTE Procedure Parameters

Parameter	Description
`dimension_owner`	Owner of the logical dimension in the OLAP Catalog.
`dimension_name`	Name of the logical dimension in the OLAP Catalog.
`attribute_name`	Name of the logical dimension attribute in the OLAP Catalog.
`aw_owner`	Owner of analytic workspace.
`aw_name`	Name of analytic workspace.
`aw_object_name`	Name of the object in the analytic workspace.
`aw_limit_type`	One of the following values: ■ `PROPERTY` -- The logical dimension attribute will be mapped using a property of the object in the analytic workspace. ■ `LIMITSET --` The logical dimension attribute will be mapped using an expression that limits one or more dimensions.

Table 26–2 MAP_AW_ATTRIBUTE Procedure Parameters

Parameter	Description
aw_limit_value	When aw_limit_type is PROPERTY, the limit value is the property name. When aw_limit_type is LIMITSET, the limit value is a string that specifies how to limit one or more dimensions in the analytic workspace. Each dimension is limited to one of its members. The string should be enclosed in single quotes, and each dimension name and dimension member pair should be separated by a forward slash. There should be a space between the dimension name and the dimension member, but there should be no space before or after the forward slash. The format of the limit expression is as follows. *'dim1name dim1member/dim2name dim2member'*

MAP_AW_DIMENSION Procedure

This procedure associates a logical dimension in the OLAP Catalog with a dimension in an analytic workspace.

The dimension entity in the OLAP Catalog must have been created by procedures in the CWM2_OLAP_AW_OBJECT package.

Syntax

```
MAP_AW_DIMENSION (
          dimension_owner   IN   VARCHAR2  (30),
          dimension_name    IN   VARCHAR2  (30),
          aw_owner          IN   VARCHAR2  (30),
          aw_name           IN   VARCHAR2  (30),
          aw_object_name    IN   VARCHAR2  (30));
```

Parameters

Table 26–3 MAP_AW_DIMENSION Procedure Parameters

Parameter	Description
dimension_owner	Owner of the logical dimension in the OLAP Catalog.
dimension_name	Name of the logical dimension in the OLAP Catalog.
aw_owner	Owner of analytic workspace.

Table 26–3 MAP_AW_DIMENSION Procedure Parameters

Parameter	Description
`aw_name`	Name of the analytic workspace.
`aw_object_name`	Name of the dimension in the analytic workspace.

MAP_AW_HIERARCHY Procedure

This procedure associates a logical hierarchy in the OLAP Catalog with hierarchy information in an analytic workspace.

The hierarchy entity in the OLAP Catalog must have been created by procedures in the CWM2_OLAP_AW_OBJECT package.

Note: Whereas dimensions and variables (measures) are native to the analytic workspace, there is no single native workspace object for a hierarchy.

Syntax

```
MAP_AW_HIERARCHY (
          dimension_owner   IN   VARCHAR2  (30),
          dimension_name    IN   VARCHAR2  (30),
          hierarchy_name    IN   VARCHAR2  (30),
          aw_owner          IN   VARCHAR2  (30),
          aw_name           IN   VARCHAR2  (30),
          aw_object_name    IN   VARCHAR2  (30),
          aw_limit_type     IN   VARCHAR2  (30),
          aw_limit_value    IN   VARCHAR2  (255));
```

Parameters

Table 26–4 MAP_AW_HIERARCHY Procedure Parameters

Parameter	Description
`dimension_owner`	Owner of the logical dimension in the OLAP Catalog.
`dimension_name`	Name of the logical dimension in the OLAP Catalog.
`hierarchy_name`	Name of the logical hierarchy in the OLAP Catalog.
`aw_owner`	Owner of analytic workspace.
`aw_name`	Name of analytic workspace.

Table 26–4 MAP_AW_HIERARCHY Procedure Parameters

Parameter	Description
`aw_object_name`	Name of the object in the analytic workspace.
`aw_limit_type`	One of the following values: ▪ `PROPERTY` -- The logical hierarchy will be mapped using a property of the object in the analytic workspace. ▪ `LIMITSET --` The logical hierarchy will be mapped using an expression that limits one or more dimensions.
`aw_limit_value`	When `aw_limit_type` is `PROPERTY`, the limit value is the property name. When `aw_limit_type` is `LIMITSET`, the limit value is a string that specifies how to limit one or more dimensions in the analytic workspace. Each dimension is limited to one of its members. The string should be enclosed in single quotes, and each dimension name and dimension member pair should be separated by a forward slash. There should be a space between the dimension name and the dimension member, but there should be no space before or after the forward slash. The format of the limit expression is as follows. *`'dim1name dim1member/dim2name dim2member'`*

MAP_AW_LEVEL Procedure

This procedure associates a logical level in the OLAP Catalog with level information in an analytic workspace.

The level entity in the OLAP Catalog must have been created by procedures in the CWM2_OLAP_AW_OBJECT package.

Note: Whereas dimensions and variables (measures) are native to the analytic workspace, there is no single native workspace object for a level.

Syntax

```
MAP_AW_LEVEL (
          dimension_owner   IN   VARCHAR2  (30),
          dimension_name    IN   VARCHAR2  (30),
          level_name        IN   VARCHAR2  (30),
          aw_owner          IN   VARCHAR2  (30),
          aw_name           IN   VARCHAR2  (30),
          aw_object_name    IN   VARCHAR2  (30),
          aw_limit_type     IN   VARCHAR2  (30),
          aw_limit_value    IN   VARCHAR2  (255));
```

Parameters

Table 26–5 MAP_AW_LEVEL Procedure Parameters

Parameter	Description
`dimension_owner`	Owner of the logical dimension in the OLAP Catalog.
`dimension_name`	Name of the logical dimension in the OLAP Catalog.
`level_name`	Name of the logical level in the OLAP Catalog.
`aw_owner`	Owner of analytic workspace.
`aw_name`	Name of analytic workspace.
`aw_object_name`	Name of the object in the analytic workspace.
`aw_limit_type`	One of the following values: ■ `PROPERTY` -- The logical level will be mapped using a property of the object in the analytic workspace. ■ `LIMITSET --` The logical level will be mapped using an expression that limits one or more dimensions.

Table 26–5 MAP_AW_LEVEL Procedure Parameters

Parameter	Description
`aw_limit_value`	When `aw_limit_type` is `PROPERTY`, the limit value is the property name. When `aw_limit_type` is `LIMITSET`, the limit value is a string that specifies how to limit one or more dimensions in the analytic workspace. Each dimension is limited to one of its members. The string should be enclosed in single quotes, and each dimension name and dimension member pair should be separated by a forward slash. There should be a space between the dimension name and the dimension member, but there should be no space before or after the forward slash. The format of the limit expression is as follows. *`'dim1name dim1member/dim2name dim2member'`*

MAP_AW_MEASURE Procedure

This procedure associates a logical measure in the OLAP Catalog with a variable in an analytic workspace.

The measure entity in the OLAP Catalog must have been created by procedures in the CWM2_OLAP_AW_OBJECT package.

Syntax

```
MAP_AW_MEASURE (
          cube_owner        IN   VARCHAR2  (30),
          cube_name         IN   VARCHAR2  (30),
          measure_name      IN   VARCHAR   (30),
          aw_owner          IN   VARCHAR2  (30),
          aw_name           IN   VARCHAR2  (30),
          aw_object_name    IN   VARCHAR2  (30),
          aw_limit_type     IN   VARCHAR2  (30),
          aw_limit_value    IN   VARCHAR2  (255));
```

Parameters

Table 26–6 MAP_AW_MEASURE Procedure Parameters

Parameter	Description
`cube_owner`	Owner of the cube in the OLAP Catalog that contains this measure.
`cube_name`	Name of the cube in the OLAP Catalog.
`measure_name`	Name of the measure in the OLAP Catalog.
`aw_owner`	Owner of the analytic workspace.
`aw_name`	Name of the analytic workspace.
`aw_object_name`	Name of the variable in the analytic workspace.
`aw_limit_type`	One of the following values: ▪ `PROPERTY` -- The logical measure will be mapped using a property of the object in the analytic workspace. ▪ `LIMITSET --` The logical measure will be mapped using an expression that limits one or more of the variable’s dimensions.

Table 26–6 MAP_AW_MEASURE Procedure Parameters

Parameter	Description
aw_limit_value	When aw_limit_type is PROPERTY, the limit value is the property name. When aw_limit_type is LIMITSET, the limit value is a string that specifies how to limit one or more dimensions in the analytic workspace. Each dimension is limited to one of its members. The string should be enclosed in single quotes, and each dimension name and dimension member pair should be separated by a forward slash. There should be a space between the dimension name and the dimension member, but there should be no space before or after the forward slash. The format of the limit expression is as follows. *'dim1name dim1member/dim2name dim2member'*

REMOVEMAP_AW_ATTRIBUTE Procedure

This procedure removes the association between a logical dimension attribute in the OLAP Catalog and dimension attribute information in an analytic workspace.

Syntax

```
REMOVEMAP_AW_ATTRIBUTE (
          dimension_owner   IN   VARCHAR2  (30),
          dimension_name    IN   VARCHAR2  (30),
          attribute_name    IN   VARCHAR2  (30),
          aw_owner          IN   VARCHAR2  (30),
          aw_name           IN   VARCHAR2  (30),
          aw_object_name    IN   VARCHAR2  (30));
```

Parameters

Table 26–7 REMOVEMAP_AW_ATTRIBUTE Procedure Parameters

Parameter	Description
dimension_owner	Owner of the logical dimension in the OLAP Catalog.
dimension_name	Name of the logical dimension in the OLAP Catalog.
attribute_name	Name of the logical dimension attribute in the OLAP Catalog.
aw_owner	Owner of analytic workspace.

Table 26–7 REMOVEMAP_AW_ATTRIBUTE Procedure Parameters

Parameter	Description
`aw_name`	Name of analytic workspace.
`aw_object_name`	Name of the dimension in the analytic workspace.

REMOVEMAP_AW_DIMENSION Procedure

This procedure removes the association between a logical dimension in the OLAP Catalog and a dimension in an analytic workspace.

Syntax

```
REMOVEMAP_AW_DIMENSION (
          dimension_owner   IN   VARCHAR2  (30),
          dimension_name    IN   VARCHAR2  (30),
          aw_owner          IN   VARCHAR2  (30),
          aw_name           IN   VARCHAR2  (30),
          aw_object_name    IN   VARCHAR2  (30));
```

Parameters

Table 26–8 REMOVEMAP_AW_DIMENSION Procedure Parameters

Parameter	Description
`dimension_owner`	Owner of the logical dimension in the OLAP Catalog.
`dimension_name`	Name of the logical dimension in the OLAP Catalog.
`aw_owner`	Owner of analytic workspace.
`aw_name`	Name of the analytic workspace.
`aw_object_name`	Name of the dimension in the analytic workspace.

REMOVEMAP_AW_HIERARCHY Procedure

This procedure removes the association between a logical hierarchy in the OLAP Catalog and hierarchy information in an analytic workspace.

Syntax

```
REMOVEMAP_AW_HIERARCHY (
          dimension_owner   IN   VARCHAR2  (30),
          dimension_name    IN   VARCHAR2  (30),
          hierarchy_name    IN   VARCHAR2  (30),
          aw_owner          IN   VARCHAR2  (30),
          aw_name           IN   VARCHAR2  (30),
          aw_object_name    IN   VARCHAR2  (30));
```

Parameters

Table 26–9 REMOVEMAP_AW_HIERARCHY Procedure Parameters

Parameter	Description
`dimension_owner`	Owner of the logical dimension in the OLAP Catalog.
`dimension_name`	Name of the logical dimension in the OLAP Catalog.
`hierarchy_name`	Name of the logical hierarchy in the OLAP Catalog.
`aw_owner`	Owner of analytic workspace.
`aw_name`	Name of analytic workspace.
`aw_object_name`	Name of the dimension in the analytic workspace.

REMOVEMAP_AW_LEVEL Procedure

This procedure removes the association between a logical level in the OLAP Catalog and level information in an analytic workspace.

Syntax

```
REMOVEMAP_AW_LEVEL (
          dimension_owner   IN   VARCHAR2  (30),
          dimension_name    IN   VARCHAR2  (30),
          level_name        IN   VARCHAR2  (30),
          aw_owner          IN   VARCHAR2  (30),
          aw_name           IN   VARCHAR2  (30),
          aw_object_name    IN   VARCHAR2  (255));
```

Parameters

Table 26–10 REMOVEMAP_AW_LEVEL Procedure Parameters

Parameter	Description
`dimension_owner`	Owner of the logical dimension in the OLAP Catalog.
`dimension_name`	Name of the logical dimension in the OLAP Catalog.
`level_name`	Name of the logical level in the OLAP Catalog.
`aw_owner`	Owner of analytic workspace.
`aw_name`	Name of analytic workspace.
`aw_object_name`	Name of the dimension in the analytic workspace.

REMOVEMAP_AW_MEASURE Procedure

This procedure removes the association between a logical measure in the OLAP Catalog and a variable in an analytic workspace.

Syntax

```
REMOVEMAP_AW_MEASURE (
          cube_owner        IN   VARCHAR2  (30),
          cube_name         IN   VARCHAR2  (30),
          measure_name      IN   VARCHAR   (30),
          aw_owner          IN   VARCHAR2  (30),
          aw_name           IN   VARCHAR2  (30),
          aw_object_name    IN   VARCHAR2  (30));
```

Parameters

Table 26–11 MAP_AW_MEASURE Procedure Parameters

Parameter	Description
`cube_owner`	Owner of the cube in the OLAP Catalog that contains this measure.
`cube_name`	Name of the cube in the OLAP Catalog.
`measure_name`	Name of the measure in the OLAP Catalog.
`aw_owner`	Owner of analytic workspace.
`aw_name`	Name of the analytic workspace.

Table 26–11 MAP_AW_MEASURE Procedure Parameters

Parameter	Description
`aw_object_name`	Name of the variable in the analytic workspace.

27

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.

This chapter discusses the following topics:

- Understanding the OLAP Classification System
- Summary of CWM_CLASSIFY Subprograms

Understanding the OLAP Classification System

The CWM_CLASSIFY package, implementing the OLAP classification system, is used primarily to manipulate OLAP measure folders.

The OLAP classification system is the single area of overlap between the CWM and CWM2 versions the OLAP Catalog.

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 27–1 CWM_CLASSIFY Subprograms

Subprogram	Description
ADD_CATALOG_ENTITY Procedure on page 27-4	Adds a measure to a measure folder (catalog).
ADD_DESCRIPTOR_ENTITY_TYPE Procedure on page 27-5	Adds a descriptor type to an entity type.
ADD_ENTITY_DESCRIPTOR_USE Procedure on page 27-6	Attaches a descriptor to an entity.
CREATE_CATALOG Function on page 27-7	Creates a measure folder (catalog).
CREATE_DESCRIPTOR Function on page 27-8	Creates a descriptor.
CREATE_DESCRIPTOR_TYPE Procedure on page 27-9	Creates a descriptor type.
DROP_CATALOG Procedure on page 27-9	Drops a measure folder (catalog).
DROP_DESCRIPTOR Procedure on page 27-10	Drops a descriptor.
DROP_DESCRIPTOR_TYPE Procedure on page 27-11	Drops a descriptor type.
LOCK_CATALOG Procedure on page 27-12	Locks a measure folder's metadata for update.
REMOVE_CATALOG_ENTITY Procedure on page 27-12	Removes a measure from a measure folder (catalog).
REMOVE_DESCRIPTOR_ENTITY_TYPE Procedure on page 27-13	Removes a descriptor type from an entity type.
REMOVE_ENTITY_DESCRIPTOR_USE Procedure on page 27-14	Removes a descriptor from an entity.
SET_CATALOG_DESCRIPTION Procedure on page 27-15	Sets the description for a measure folder (catalog).
SET_CATALOG_PARENT Procedure on page 27-16	Sets the parent folder for a measure folder (catalog).

ADD_CATALOG_ENTITY Procedure

This procedure adds a metadata entity to a measure folder. The entity may be a measure or a cube.

Syntax

```
ADD_CATALOG_ENTITY (
   catalog_id           IN   NUMBER,
   entity_owner         IN   VARCHAR2,
   entity_name          IN   VARCHAR2,
   child_entity_name    IN   VARCHAR2);
```

Parameters

Table 27–2 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 27–3 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 does not exist or is not accessible to user.
`catalog_not_found`	Measure folder not found.

ADD_DESCRIPTOR_ENTITY_TYPE Procedure

This procedure adds a type of descriptor to a type of metadata entity.

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 27–4 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 27–5 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 27–6 ADD_ENTITY_DESCRIPTOR_USE Procedure Exceptions

Exception	Description
`entity_not_found`	Entity with this name does not exist or is not accessible to user.
`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  IN NUMBER);
```

Parameters

Table 27–7 CREATE_CATALOG Procedure 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.

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 27–8 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 27–9 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 be OLAPDBA.

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 27–10 CREATE_DESCRIPTOR_TYPE Procedure Parameters

Parameter	Description
`descriptor_type`	Name of the descriptor type. Examples might be `dimension type`, or `attribute type`.

Exceptions

Table 27–11 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 27–12 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 27–13 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 27–14 DROP_DESCRIPTOR Procedure Parameters

Parameter	Description
`descriptor_id`	Descriptor identifier

Exceptions

Table 27–15 DROP_DESCRIPTOR Procedure Exceptions

Exception	Description
`descriptor_not_found`	Descriptor not found.
`no_access_privileges`	Must be `OLAPDBA`.

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 27–16 DROP_DESCRIPTOR_TYPE Procedure Parameters

Parameter	Description
`descriptor_type`	Name of the descriptor type.

Exceptions

Table 27–17 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 metadata.

Syntax

```
LOCK_CATALOG (
   catalog_id            IN   NUMBER,
   wait_for_lock         IN   BOOLEAN);
```

Parameters

Table 27–18 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 27–19 LOCK_CATALOG Procedure Exceptions

Exception	Description
`catalog_not_found`	Measure folder does not exist.
`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 27–20 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 27–21 REMOVE_CATALOG_ENTITY Procedure Exceptions

Exception	Description
`element_not_found`	Cube or measure does not exist or is not accessible to user.
`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 27–22 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`

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 27–23 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.

Table 27–23 REMOVE_ENTITY_DESCRIPTOR_USE Procedure Parameters

Parameter	Description
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 27–24 REMOVE_ENTITY_DESCRIPTOR_USE Procedure Exceptions

Exception	Description
entity_not_found	Entity with this name does not exist or is not accessible to user.

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 27–25 SET_CATALOG_DESCRIPTION Procedure Parameters

Parameter	Description
catalog_id	Identifier of the measure folder.
catalog_description	Description of the measure folder.

Exceptions

Table 27–26 SET_CATALOG_DESCRIPTION Procedure Exceptions

Exception	Description
`catalog_not_found`	Measure folder not found.

SET_CATALOG_PARENT Procedure

Table 27–27 CREATE_CATALOG Procedure Exceptions

Exception	Description
`parent_catalog_not_found`	Parent measure folder not found.
`catalog_already_exists`	A measure folder with this name already exists.
`invalid_name`	Measure folder name may not be empty or null.

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 27–28 SET_CATALOG_PARENT Procedure Parameters

Parameter	Description
`catalog_id`	Identifier of the measure folder.
`parent_catalog_id`	Identifier of the parent measure folder.

Exceptions

Table 27–29 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.

Part V

Creating Materialized Views for the OLAP API

Part V explains how to create materialized views for queries for aggregate data from the OLAP API.

This part contains the following chapters:

28

Developing a Summary Management Strategy

This chapter provides information to help you develop a summary management strategy 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:

Optimizing the Database for OLAP

A basic feature of online analytical processing (OLAP) is the ability to analyze and view various levels of aggregate data. Queries generated by the Oracle OLAP API use the database's query rewrite capability whenever possible. Query rewrite enables a query to fetch aggregate data from materialized views rather than recomputing the aggregates at runtime.

The creation of materialized views and indexes can significantly improve the performance of analytical queries generated by the OLAP API.

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.

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 Oracle OLAP generates. The OLAP API requires a very specific set of materialized views.

Summary Management Options

To enhance the performance of queries 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.

You do not need to 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.

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.

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. You may have used Enterprise Manager to create `CWM` metadata, or you may have developed scripts to create `CWM2` metadata, or you may have some combination of `CWM` and `CWM2` metadata.

However, the structure of the metadata in the OLAP Catalog does not affect the structure of the materialized views that enable query rewrite at runtime. 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.

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

CWM Metadata

`CWM` metadata for dimension tables and fact tables is visible within the property pages for dimensions and cubes in the OLAP folder of Enterprise Manager. OLAP Summary Advisor generates concatenated rollup style MVs for cubes that are visible within Enterprise Manager.

If you wish to create grouping set style materialized views for `CWM` metadata, then you must develop your own scripts using the `DBMS_ODM` package. OLAP Summary Advisor does not support this type of materialized view.

> **See Also:** Chapter 31, "Creating Fact Materialized Views With OLAP Summary Advisor".

CWM2 Metadata

`CWM2` metadata is not visible within the OLAP folder of Enterprise Manager. Thus, you cannot use OLAP Summary Advisor with `CWM2` metadata.

To create materialized views for `CWM2` metadata, use the `DMBS_ODM` package. `DBMS_ODM` contains procedures for creating dimension MVs and grouping set style MVs.

To create materialized views in grouping set form for a star schema that is mapped to `CWM2` metadata, use the `DBMS_ODM` package.

> **See Also:** Chapter 30, "Creating Fact Materialized Views With DBMS_ODM".

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 `CWM` cube.

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

Important: 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 28–1, "SALES_CUBE Cube" lists the dimensions and hierarchies associated with the `SALES_CUBE` cube in the Sales History (`SH`) schema.

Table 28–1 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 cube that represent a star schema. The cube must be mapped to a single fact table, and the fact table may contain only lowest-level data.

For `CWM` metadata, you can choose to create fact materialized views using concatenated rollup syntax or grouping set syntax. For `CWM2` metadata, only grouping set style fact MVs are supported.

Creating Fact Materialized Views

When you use OLAP Summary Advisor, fact materialized views are created automatically for each hierarchy combination of a `CWM` cube. The materialized views are created with concatenated rollup syntax.

Alternatively, you can use the `CREATEDIMLEVTUPLE`, `CREATECUBELEVELTUPLE`, and `CREATEFACTMV_GS` procedures in the `DBMS_ODM` package to create a fact materialized view for a `CWM` or `CWM2` cube. The materialized view is created with grouping set syntax.

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 28–1, 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 Summary Management Solution

Whether you choose to use grouping set or concatenated rollup for your fact materialized views will depend primarily on the complexity of the data in your star schema. However, the nature of your existing metadata (CWM or CWM2) may also be a factor, as well as your preference for using either Enterprise Manager or your own PL/SQL scripts.

Summary Management for CWM Metadata

If you have existing CWM metadata, you can use Enterprise Manager to create all the necessary materialized views. The fact materialized views will be generated with concatenated rollup syntax. However, you are not limited to concatenated rollup style MVs if you have CWM metadata. You can create scripts that generate materialized views for your CWM metadata by using the DBMS_ODM package.

Summary Management With a Graphical User Interface

If you would rather not create your own SQL scripts for summary management and you prefer to use Enterprise Manager, you are limited to the concatenated rollup style MVs for your fact data. Moreover, the star schema must be mapped to CWM (not CWM2) metadata.

Summary Management for Multiple Hierarchies

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.

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.

Query Performance

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

29 Creating Dimension Materialized Views

This chapter explains how to create dimension 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 28-4.
- "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.

This chapter contains the following topics:

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 Summary Management Solution".

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 22, "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.

This chapter contains the following topics:

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 Summary Management Solution".

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 a alternative to its use.

Performance data can be collected in system tables the same as other Oracle database performance statistics.

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.
- An OLAP DML utility named `CNV_CWM.TO.ECM` creates an ECM-type analytic workspace from relational tables.
- Using SQL table functions, it is now possible for a SQL-based application to manipulate and extract data from an analytic workspace. Express Server did not permit a data transfer to be initiated externally.

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.

An OLAP DML program named `GENSQLOBJS` generates a SQL script that creates views of ECM-type 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 CWM2 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. However, you can use the GENSQLOBJS utility to generate relational views of variables, dimensions, and other objects in an ECM-type analytic workspace, and create OLAP catalog metadata for those views.

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 6-9.

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