Oracle® OLAP

Reference

10*g* Release 2 (10.2)

B14350-01

June 2005

ORACLE®

Oracle OLAP Reference, 10*g* Release 2 (10.2)

B14350-01

Contents

2 Creating OLAP Catalog Metadata with CWM2

3 Active Catalog Views

4 Analytic Workspace Maintenance Views

5 OLAP Catalog Metadata Views

18 CWM2_OLAP_MEASURE

19 CWM2_OLAP_METADATA_REFRESH

20 CWM2_OLAP_PC_TRANSFORM

21 CWM2_OLAP_TABLE_MAP

22 CWM2_OLAP_VALIDATE

23 CWM2_OLAP_VERIFY_ACCESS

24 DBMS_AW

25 DBMS_AW_XML

26 DBMS_AWM

27 DBMS_ODM

28 OLAP_API_SESSION_INIT

29 OLAP_CONDITION

30 OLAP_EXPRESSION

31 OLAP_EXPRESSION_BOOL

32 OLAP_EXPRESSION_DATE

33 OLAP_EXPRESSION_TEXT

34 OLAP_TABLE

Index

Preface

This reference manual describes the relational views, SQL functions, and PL/SQL packages that support the OLAP option of the Oracle Database.

This preface contains the following topics:

Audience

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

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

Documentation Accessibility

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

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

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an

otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

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

TTY Access to Oracle Support Services

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

Related Documents

For more information see these Oracle resources:

- *Oracle OLAP Application Developer's Guide*

  Explains how SQL and Java applications can extend their analytic processing capabilities by using Oracle OLAP.

- *Oracle OLAP DML Reference*

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

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

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

- *Oracle OLAP Java API Reference*

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

- *Oracle OLAP Analytic Workspace Java API Reference*

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

- *Oracle Database Data Warehousing Guide*

  Discusses the database structures, concepts, and issues involved in creating a data warehouse to support online analytical processing solutions.

- *Oracle Database PL/SQL User's Guide and Reference*

  Explains the concepts and syntax of PL/SQL, Oracle's procedural extension of SQL.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Creating Analytic Workspaces with DBMS_AWM

The DBMS_AWM package provides stored procedures for creating and maintaining analytic workspaces.

The DBMS_AWM package is used by Analytic Workspace Manager. This chapter explains how to create your own scripts that use the DBMS_AWM procedures directly.

See Also:

- Chapter 26, "DBMS_AWM" for the complete syntax of the procedures in this package
- Chapter 3, "Active Catalog Views" for descriptions of the views that expose the logical structures within analytic workspaces
- Chapter 4, "Analytic Workspace Maintenance Views" for descriptions of the views that expose information stored in the OLAP Catalog by DBMS_AWM procedures

This chapter contains the following topics:

Overview

The DBMS_AWM package provides a feature–rich set of APIs for building and maintaining analytic workspaces. These APIs use a logical cube, stored in the OLAP Catalog, to structure the workspace. The cube is mapped to a star or snowflake schema, which provides the source data for the workspace.

Applications that use the BI Beans or OLAP API can directly query any workspace created by DBMS_AWM. Other types of applications must query the workspace through relational views. These views are created by the DBMS_AWM **enablement** procedures.

> **Note:** Analytic workspaces created by the DBMS_AWM procedures are in **database standard form**, ensuring compatibility with related Oracle OLAP tools and utilities. See *Oracle OLAP Application Developer's Guide* for information about standard form.

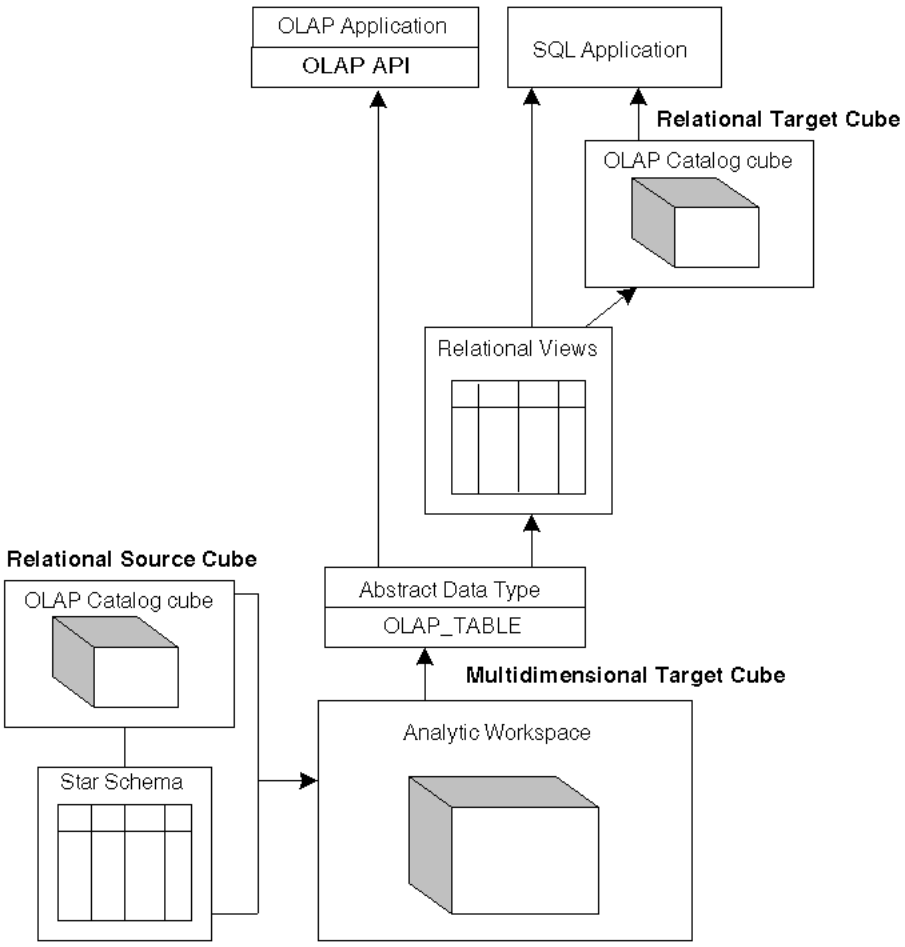
Scripts that create and maintain analytic workspaces must identify two different logical cubes: a **relational source cube** and a **multidimensional target cube**. DBMS_AWM also supports the creation of a third optional cube, a **relational target cube**, which is not used by the OLAP API.

The basic flow of events is as follows:

1. **Relational Source Cube**. This cube must exist before you call any of the DBMS_AWM procedures. The cube's metadata is defined within the OLAP Catalog. Its data is unsolved (lowest level only) and stored in a star or snowflake schema.
2. **Multidimensional Target Cube**. DBMS_AWM procedures define and populate this cube from the relational source cube. The cube's standard form metadata is defined in the analytic workspace. Its data is stored in the workspace, typically with full or partial summarization.
3. **Relational Target Cube**. DBMS_AWM enablement procedures optionally define this cube from the multidimensional target cube. The cube's metadata is defined within the OLAP Catalog. Its data is stored in the analytic workspace and accessed through relational views. The views present the data as fully solved (embedded totals for all level combinations).

The basic process of building an analytic workspace with the DBMS_AWM package is illustrated in Figure 1–1.

Figure 1–1 Creating an Analytic Workspace with DBMS_AWM



Creating OLAP Catalog Metadata for the Source Cube

Before you can use the `DBMS_AWM` procedures, you must create a cube in the OLAP Catalog and map it to the source fact table and dimension tables. The source tables must be organized as a star or snowflake schema.

You can use Oracle Enterprise Manager or Oracle Warehouse Builder to create the cube. You can also create the cube from scripts that use the `CWM2` PL/SQL packages, as described in Chapter 2.

Creating Dimensions in the Analytic Workspace

For each dimension of the source cube in the OLAP Catalog, you must run a set of procedures in the `DBMS_AWM` package to accomplish the following general tasks:

1. Create a **dimension load specification**, which contains instructions for populating the dimension in the analytic workspace. The load specification may include a filter that identifies criteria for selecting data from the source dimension tables.
2. Create containers for the dimension in an analytic workspace.
3. Use the dimension load specification to populate the dimension in the analytic workspace from the source dimension tables.

See Also: "Creating and Refreshing a Workspace Dimension" on page 1-9.

Creating Cubes in the Analytic Workspace

After creating the cube's dimensions, run another set of procedures to create and populate the cube itself.

1. Create a **cube load specification**, which contains instructions for populating the cube's measures in the analytic workspace. The load specification may include a filter that identifies criteria for selecting data from the source fact table.
2. Create a **composite specification**, which contains instructions for ordering the cube's dimensions and storing sparse data in the analytic workspace.
3. Add the composite specification to the cube load specification.
4. Create containers for the cube in an analytic workspace.
5. Use the cube load specification to populate the cube's measures in the analytic workspace from the source fact table.

See Also: "Creating and Refreshing a Workspace Cube" on page 1-10 and "Managing Sparse Data and Optimizing the Workspace Cube" on page 1-12.

Aggregating the Cube's Data in the Analytic Workspace

For the workspace cube, run a set of procedures to accomplish the following:

1. Create an **aggregation specification**, which contains instructions for storing summary data in the analytic workspace.
2. Use the aggregation specification to aggregate the workspace cube.

See Also: "Aggregating the Data in an Analytic Workspace" on page 1-14.

Enabling Access to the Analytic Workspace

Analytic workspaces created with the current release of `DBMS_AWM` and Analytic Workspace Manager are automatically accessible by applications that use the OLAP API or BI Beans. You do not need to create any views or additional metadata. If you have workspaces that were created with earlier releases of the software, you can upgrade them. Refer to the upgrade instructions in "Enabling Access by the OLAP API" on page 1-17.

To enable analytic workspaces for access by other types of SQL clients, you must create views that project the multidimensional data as logical columns and rows. The `DBMS_AWM` enablement procedures create and maintain a set of relational views for an analytic workspace. The enablement procedures can optionally create OLAP Catalog metadata that maps to the views. See "Enabling Relational Access" on page 1-17 for information on the enablement process.

Viewing Metadata Created by DBMS_AWM

Two sets of views reveal metadata related to analytic workspaces. The **Active Catalog views** reveal metadata stored within analytic workspaces. The Analytic Workspace Maintenance views reveal metadata stored within the OLAP Catalog.

Active Catalog Views

These views use OLAP_TABLE functions to return information about logical standard form objects within analytic workspaces. For example, you could query an Active Catalog view to obtain information about the dimensionality of a workspace cube. The Active Catalog view names have the prefix ALL_OLAP2_AW. For more information, see Chapter 3.

Analytic Workspace Maintenance Views

These views return information about building and maintaining analytic workspace cubes. For example, you could query an Analytic Workspace Maintenance view to obtain information about the load specifications associated with an analytic workspace dimension or cube. The Analytic Workspace Maintenance view names have the prefix ALL_AW. For more information, see Chapter 4.

Understanding the DBMS_AWM Procedures

The procedures in the DBMS_AWM package support methods on several types of logical entities. These entities are described in Table 1–1.

See Also: Chapter 26, "DBMS_AWM"

Table 1–1 Logical Entities in the DBMS_AWM Package

Entity	Description
Dimension	A dimension in the OLAP Catalog and its corresponding dimension in an analytic workspace.
Cube	A cube in the OLAP Catalog and its corresponding cube in an analytic workspace.
Dimension Load Specification	Instructions for populating an analytic workspace dimension from the dimension tables of an OLAP Catalog dimension.
Cube Load Specification	Instructions for populating an analytic workspace cube from the fact table of an OLAP Catalog cube.
Cube Aggregation Specification	Instructions for creating summary data in an analytic workspace.
Cube Composite Specification	Instructions for ordering dimensions and storing sparse data in an analytic workspace.

Methods on Dimensions

The methods you can perform on a dimension are described in Table 1–2.

Table 1–2 Methods on Dimensions in DBMS_AWM

Method	Description	Procedure
Create	Create the metadata in an analytic workspace for a dimension defined in the OLAP Catalog.	CREATE_AWDIMENSION Procedure
Refresh	Use a dimension load specification to populate an analytic workspace dimension from the dimension tables of an OLAP Catalog dimension.	REFRESH_AWDIMENSION Procedure

Table 1–2 (Cont.) Methods on Dimensions in DBMS_AWM

Method	Description	Procedure
Enable for non-OLAP clients	Create a script to enable relational access to a dimension in an analytic workspace.	CREATE_AWDIMENSION_ACCESS Procedure
	Create a script to disable relational access to a dimension in an analytic workspace.	DELETE_AWDIMENSION_ACCESS Procedure
	Create and run a script to enable relational access to a dimension in an analytic workspace	CREATE_AWDIMENSION_ACCESS_FULL Procedure
	Create and run a script to disable relational access to a dimension in an analytic workspace.	DELETE_AWDIMENSION_ACCESS_ALL Procedure
	Refresh the workspace metadata that supports user-defined view names for a dimension in an analytic workspace.	REFRESH_AWDIMENSION_VIEW_NAME Procedure
	Specify names for the relational views of a dimension in an analytic workspace.	SET_AWDIMENSION_VIEW_NAME Procedure

Methods on Cubes

The methods you can perform on a cube are described in Table 1–3.

Table 1–3 Methods on Cubes in DBMS_AWM

Method	Description	Procedure
Create	Create containers in an analytic workspace for a cube defined in the OLAP Catalog.	CREATE_AWCUBE Procedure
Refresh	Use a cube load specification to populate the measures of an analytic workspace cube from the fact table of an OLAP Catalog cube.	REFRESH_AWCUBE Procedure
Aggregate	Use an aggregation specification to aggregate the cube in the analytic workspace.	AGGREGATE_AWCUBE Procedure
Enable for non-OLAP clients	Create a script to enable relational access to a cube in an analytic workspace.	CREATE_AWCUBE_ACCESS Procedure
	Create a script to disable relational access to a cube in an analytic workspace	DELETE_AWCUBE_ACCESS Procedure
	Create and run a script to enable relational access to a cube in an analytic workspace.	CREATE_AWCUBE_ACCESS_FULL Procedure
	Create and run a script to disable relational access to a cube in an analytic workspace.	DELETE_AWCUBE_ACCESS_ALL Procedure

Table 1–3 (Cont.) Methods on Cubes in DBMS_AWM

Method	Description	Procedure
	Refresh the workspace metadata that supports user-defined view names for a cube in an analytic workspace.	REFRESH_AWCUBE_VIEW_NAME Procedure
	Specify names for the relational views of a cube in an analytic workspace.	SET_AWCUBE_VIEW_NAME Procedure

Methods on Dimension Load Specifications

The methods you can perform on a dimension load specification are described in Table 1–4.

Table 1–4 Methods on Dimension Load Specifications in DBMS_AWM

Method	Description	Procedure
Create/Delete	Create or delete a dimension load specification.	CREATE_AWDIMLOAD_SPEC Procedure DELETE_AWDIMLOAD_SPEC Procedure
Reset information	Change various components of a dimension load specification.	SET_AWDIMLOAD_SPEC_DIMENSION Procedure SET_AWDIMLOAD_SPEC_LOADTYPE Procedure SET_AWDIMLOAD_SPEC_NAME Procedure SET_AWDIMLOAD_SPEC_PARAMETER Procedure
Add/Delete filter	Add or remove a filter from a dimension load specification.	ADD_AWDIMLOAD_SPEC_FILTER Procedure DELETE_AWDIMLOAD_SPEC_FILTER Procedure

Methods on Cube Load Specifications

The methods you can perform on a cube load specification are described in Table 1–5.

Table 1–5 Methods on Cube Load Specifications in DBMS_AWM

Method	Description	Procedure
Create/Delete	Create or delete a cube load specification.	CREATE_AWCUBELOAD_SPEC Procedure DELETE_AWCUBELOAD_SPEC Procedure
Reset information	Change various components of a cube load specification.	SET_AWCUBELOAD_SPEC_CUBE Procedure SET_AWCUBELOAD_SPEC_LOADTYPE Procedure SET_AWCUBELOAD_SPEC_NAME Procedure SET_AWCUBELOAD_SPEC_PARAMETER Procedure

Table 1–5 (Cont.) Methods on Cube Load Specifications in DBMS_AWM

Method	Description	Procedure
Add/Delete filter	Add or remove a filter from a cube load specification.	ADD_AWCUBELOAD_SPEC_FILTER Procedure DELETE_AWCUBELOAD_SPEC_FILTER Procedure
Add/Delete composite specification	Add or remove a composite specification from a cube load specification.	ADD_AWCUBELOAD_SPEC_COMP Procedure DELETE_AWCUBELOAD_SPEC_COMP Procedure

Methods on Aggregation Specifications

The methods you can perform on an aggregation specification are described in Table 1–6.

Table 1–6 Methods on Aggregation Specifications in DBMS_AWM

Method	Description	Procedure
Create/Delete	Create or delete an aggregation specification.	CREATE_AWCUBEAGG_SPEC Procedure DELETE_AWCUBEAGG_SPEC_MEASURE Procedure
Set operator	Set the aggregation operator for a dimension.	SET_AWCUBEAGG_SPEC_AGGOP Procedure
Add/Delete levels	Add or remove levels from an aggregation specification.	ADD_AWCUBEAGG_SPEC_LEVEL Procedure DELETE_AWCUBEAGG_SPEC_LEVEL Procedure
Add/Delete measures	Add or remove measures from an aggregation specification.	ADD_AWCUBEAGG_SPEC_MEASURE Procedure DELETE_AWCUBEAGG_SPEC_MEASURE Procedure

Methods on Composite Specifications

The methods you can perform on a composite specification are described in Table 1–7.

Table 1–7 Methods on Composite Specifications in DBMS_AWM

Method	Description	Procedure
Create/Delete	Create or delete a composite specification.	CREATE_AWCOMP_SPEC Procedure DELETE_AWCOMP_SPEC Procedure
Reset information	Change the name of the composite specification or associate it with a different cube.	SET_AWCOMP_SPEC_CUBE Procedure SET_AWCOMP_SPEC_NAME Procedure
Add/Delete members	Add or remove members from the specification. Members can be dimensions or composites.	ADD_AWCOMP_SPEC_MEMBER Procedure DELETE_AWCOMP_SPEC_MEMBER Procedure

Table 1–7 (Cont.) Methods on Composite Specifications in DBMS_AWM

Method	Description	Procedure
Reset member information	Change information about members of the specification.	SET_AWCOMP_SPEC_MEMBER_NAME Procedure
		SET_AWCOMP_SPEC_MEMBER_POS Procedure
		SET_AWCOMP_SPEC_MEMBER_SEG Procedure
Add composite members	Add members to a composite in the specification.	ADD_AWCOMP_SPEC_COMP_MEMBER Procedure

Creating and Refreshing a Workspace Dimension

Once you have defined a dimension in the OLAP Catalog for your source dimension table, you can create the dimension in the analytic workspace.

Only one workspace dimension may be created from a given dimension in the OLAP Catalog. For example, if you have used the OLAP Catalog `PRODUCT` dimension as the source for the `PROD_AW` dimension in an analytic workspace, you cannot create another dimension `PROD_AW2` from the same source dimension in the same workspace.

> **Note:** `CREATE_AWDIMENSION` opens the analytic workspace with read/write access. It updates the workspace, but it *does not* execute a SQL `COMMIT`.
>
> The analytic workspace must already exist before you call `CREATE_AWDIMENSION` or any other procedures in the `DBMS_AWM` package.

Example 1–1 shows the procedure calls for defining and populating workspace objects for the `XADEMO.CHANNEL` dimension. The load specification includes a filter condition that causes only the row for `'DIRECT'` to be loaded.

Example 1–1 Creating the CHANNEL Dimension in an Analytic Workspace

```
--- SET UP
set serveroutput on
execute cwm2_olap_manager.set_echo_on;
create or replace directory myscripts as '/users/myxademo/myscripts';
execute cwm2_olap_manager.begin_log
          ('MYSCRIPTS' , 'channel.log');

--- CREATE THE ANALYTIC WORKSPACE
execute dbms_aw.execute ('aw create ''myaw''');

--- CREATE AND POPULATE THE DIMENSION
execute dbms_awm.create_awdimension
          ('XADEMO', 'CHANNEL', 'MYSCHEMA','MYAW', 'AW_CHAN');
execute dbms_awm.create_awdimload_spec
          ('CHAN_LOAD', 'XADEMO', 'CHANNEL', 'FULL_LOAD');
execute dbms_awm.add_awdimload_spec_filter
          ('CHAN_LOAD', 'XADEMO', 'CHANNEL', 'XADEMO',
          'XADEMO_CHANNEL', '''CHAN_STD_CHANNEL'' = ''DIRECT''' );
execute dbms_awm.refresh_awdimension
```

```
          ('MYSCHEMA', 'MYAW', 'AW_CHAN', 'CHAN_LOAD');

--- COMMIT AND WRAP UP
commit;
execute cwm2_olap_manager.set_echo_off;
execute cwm2_olap_manager.end_log
```

When you query the Active Catalog view ALL_OLAP2_AW_DIMENSIONS, you will see the following row.

```
AW_OWNER    AW_NAME     AW_LOGICAL_NAME      SOURCE_OWNER   SOURCE_NAME
----------  ----------  -------------------  -------------  -----------
MYSCHEMA    MYAW        AW_CHAN              XADEMO         CHANNEL
```

CREATE_AWDIMENSION creates the standard form metadata for the dimension in the workspace. REFRESH_AWDIMENSION loads the dimension members and attribute values.

You should refresh a dimension whenever changes occur in the source dimension tables. These changes could be additions or deletions of dimension members, for example removing a product from a Product dimension, or they could be changes to the dimension's metadata, such as adding a Day level to a time dimension.

When you refresh a dimension, you must also refresh each cube in which it participates.

When you refresh a dimension whose cube has associated stored summaries in the analytic workspace (the result of an aggregation specification), you must also reaggregate the cube.

Creating and Refreshing a Workspace Cube

Once you have defined a cube in the OLAP Catalog for your star schema, you can create the cube in the analytic workspace.

You must call CREATE_AWDIMENSION to create each of the cube's dimensions before calling CREATE_AWCUBE to create the cube. To populate the cube, you must call REFRESH_AWDIMENSION to populate each of the cube's dimensions before calling REFRESH_AWCUBE to refresh the cube's measures.

Within an analytic workspace, dimensions can be shared by more than one cube. When creating a new workspace cube, you will only call CREATE_AWDIMENSION for OLAP Catalog dimensions that have not been used as the source for dimensions of cubes that already exist in the workspace.

Note: CREATE_AWCUBE opens the analytic workspace with read/write access. It updates the workspace, but it *does not* execute a SQL COMMIT.

The analytic workspace must already exist before you call CREATE_AWCUBE or any other procedures in the DBMS_AWM package.

Example 1–2 shows the procedure calls for creating and populating the XADEMO.ANALYTIC_CUBE cube in an analytic workspace.

Example 1–2 Creating the ANALYTIC_CUBE Cube in an Analytic Workspace

```
--- SET UP
set serveroutput on
```

```
execute cwm2_olap_manager.set_echo_on;

create or replace directory myscripts as '/users/myxademo/myscripts';
execute cwm2_olap_manager.begin_log('MYSCRIPTS' , 'anacube.log');


--- CREATE THE ANALYTIC WORKSPACE
execute dbms_aw.execute ('aw create ''myaw''');

--- CREATE AND REFRESH THE DIMENSIONS
execute dbms_awm.create_awdimension
          ('XADEMO','CHANNEL','MYSCHEMA', 'MYAW', 'AW_CHAN');
execute dbms_awm.create_awdimension
          ('XADEMO','GEOGRAPHY','MYSCHEMA','MYAW', 'AW_GEOG');
execute dbms_awm.create_awdimension
          ('XADEMO','PRODUCT','MYSCHEMA', 'MYAW', 'AW_PROD');
execute dbms_awm.create_awdimension
          ('XADEMO','TIME','MYSCHEMA', 'MYAW', 'AW_TIME');
execute dbms_awm.refresh_awdimension
          ('MYSCHEMA', 'MYAW', 'AW_CHAN');
execute dbms_awm.refresh_awdimension
          ('MYSCHEMA', 'MYAW', 'AW_PROD');
execute dbms_awm.refresh_awdimension
          ('MYSCHEMA', 'MYAW', 'AW_GEOG');
execute dbms_awm.refresh_awdimension
          ('MYSCHEMA', 'MYAW', 'AW_TIME');

--- CREATE AND REFRESH THE CUBE
execute dbms_awm.create_awcube
          ('XADEMO', 'ANALYTIC_CUBE','MYSCHEMA', 'MYAW','AW_ANACUBE');
execute dbms_awm.create_awcubeload_spec
          ('AC_CUBELOADSPEC', 'XADEMO', 'ANALYTIC_CUBE', 'LOAD_DATA');
execute dbms_awm.refresh_awcube
          ('MYSCHEMA', 'MYAW', 'AW_ANACUBE', 'AC_CUBELOADSPEC');

--- COMMIT AND WRAP UP
commit;
execute cwm2_olap_manager.set_echo_off;
execute cwm2_olap_manager.end_log
```

When you query the Active Catalog view `ALL_OLAP2_AW_CUBES` , you will see the following row.

```
AW_OWNER    AW_NAME     AW_LOGICAL_NAME       SOURCE_OWNER  SOURCE_NAME
----------  ----------  -------------------  -------------  -------------
MYSCHEMA    MYAW        AW_ANACUBE            XADEMO        ANALYTIC_CUBE
```

The measures in the source fact table may have numeric, text, or date data types. When `REFRESH_AWCUBE` loads the data into a workspace cube, it converts the RDBMS data types to types that are native to analytic workspaces. The data type conversion is described in Table 1–8.

If a source measure has a data type not described in Table 1–8, the measure is ignored by `REFRESH_AWCUBE` and none of its data or metadata is loaded into the analytic workspace.

Table 1–8 Conversion of RDBMS Data Types to Workspace Data Types

RDBMS Data Type	Analytic Workspace Data Type
NUMBER	DECIMAL
CHAR, LONG, VARCHAR, VARCHAR2	TEXT
NCHAR, NVARCHAR2	NTEXT
DATE	DATE

CREATE_AWCUBE ensures that the generic standard form objects that support cubes exist in the workspace, and it registers the specified cube in the workspace. However, the metadata that defines the logical structure of this particular cube is not instantiated in the workspace until you call REFRESH_AWCUBE.

For example, if you have just created a cube AW_ANACUBE in a workspace MYAW in MYSCHEMA from the source cube XADEMO.ANALYTIC_CUBE, you can query the Active Catalog to check the workspace.

```
SQL>select * from ALL_OLAP2_AW_CUBES where AW_LOGICAL_NAME in 'AW_ANACUBE';

AW_OWNER   AW_NAME     AW_LOGICAL_NAME       SOURCE_OWNER    SOURCE_NAME
---------- ----------- --------------------- --------------- --------------
MYSCHEMA   MYAW        AW_ANACUBE            XADEMO          ANALYTIC_CUBE
```

The following query shows that there are no measures associated with the cube. The measures, dimensions, and descriptions will be instantiated when the cube is refreshed.

```
SQL>select * from ALL_OLAP2_AW_CUBE_MEASURES where AW_CUBE_NAME in 'AW_ANACUBE';

no rows selected
```

You should refresh a cube whenever changes occur in the source fact table. These changes could be additions or deletions of data, for example updating sales figures, or they could be changes to the cube's metadata, such as adding a measure or renaming a description.

When you refresh a cube, you must first refresh any of its dimensions that have changed. If you want to drop or add a dimension to a cube, you must drop the cube and re-create it.

Every time you refresh a cube that has an associated aggregation specification, you must reaggregate the cube.

If you make changes to the composite specification associated with a cube, you must drop the cube and re-create it in the analytic workspace. You cannot refresh a cube with a modified composite specification.

Managing Sparse Data and Optimizing the Workspace Cube

A **composite** is an object that is used to store sparse data compactly in a variable in an analytic workspace. A composite consists of a list of dimension-value combinations in which one value is taken from each of the dimensions on which the composite is based. Only the combinations for which data exists are included in the composite.

Composites are maintained automatically by the OLAP engine. With composites, you can keep your analytic workspace size to a minimum and promote good performance. For more information on composites, see the *Oracle OLAP DML Reference*. For

information on managing sparsity and optimizing performance in your analytic workspaces, see the *Oracle OLAP Application Developer's Guide*

For example, you might have some products in your analytic cube that are not sold in all regions. The data cells for those combinations of PRODUCT and GEOGRAPHY would be empty. In this case, you might choose to define PRODUCT and GEOGRAPHY as a composite. The OLAP DML syntax for defining the dimensionality of the Costs measure in this cube could be as follows.

```
DEFINE prod_geog COMPOSITE <product geography>
DEFINE costs VARIABLE INTEGER <time channel prod_geog<product geography>>
```

To specify that a cube's data be loaded into an analytic workspace using this definition of the cube's dimensionality, you would define a **composite specification** for the cube. The composite specification would define the following expression.

```
<time channel prod_geog<product geography>>
```

Each member of a composite specification has a name, a type, and a position. Table 1–9 identifies this information for the preceding example.

Table 1–9 Composite Spec Members for XADEMO.ANALYTIC_CUBE

Member	Type	Position
TIME	dimension	1
CHANNEL	dimension	2
PROD_GEOG	composite	3
PRODUCT	dimension	4
GEOGRAPHY	dimension	5

Dimension order determines how the cube's data is stored and accessed in the analytic workspace. The first dimension in the dimension's definition is the fastest-varying and the last is the slowest-varying.

By default, REFRESH_AWCUBE defines a workspace cube's dimensionality with Time as the fastest varying dimension followed by a composite of all the other dimensions. The dimensions in the composite are ordered according to their size. The dimension with the most members is first and the dimension with the least members is last. For example, the default dimensionality of the ANALYTIC_CUBE in an analytic workspace would be as follows.

```
<time comp_name<geography, product, channel>>
```

You can override the default dimensionality by specifying a composite specification and including it in the cube load specification.

For information on ordering dimensions and specifying segment size for dimension storage, see the *Oracle OLAP Application Developer's Guide*.

The statements in Example 1–3 create a composite specification called comp1 for the ANALYTIC_CUBE.

Example 1–3 Defining a Cube's Dimensionality in an Analytic Workspace

```
exec dbms_awm.create_awcomp_spec
          ('comp1', 'xademo', 'analytic_cube');
exec dbms_awm.add_awcomp_spec_member
         ('comp1', 'xademo', 'analytic_cube', 'comp1_time', 'dimension',
```

```
           'xademo', 'time');
exec dbms_awm.add_awcomp_spec_member
          ('comp1', 'xademo', 'analytic_cube', 'comp1_channel', 'dimension',
           'xademo', 'channel');
exec dbms_awm.add_awcomp_spec_member
          ('comp1', 'xademo', 'analytic_cube', 'comp1_prod_geog', 'composite');
exec dbms_awm.add_awcomp_spec_comp_member
          ('comp1', 'xademo', 'analytic_cube', 'comp1_prod_geog',
           'comp1_product' ,'dimension', 'xademo', 'product');
exec dbms_awm.add_awcomp_spec_comp_member
          ('comp1', 'xademo', 'analytic_cube', 'comp1_prod_geog',
           'comp1_geography' ,'dimension', 'xademo', 'geography');
exec dbms_awm.add_awcubeload_spec_comp
           ('my_cube_load', 'xademo', 'analytic_cube', 'comp1');
```

You can modify a composite specification by applying it to a different cube or giving it a different name. You can rename, move, and change the segment size of a primary member of a composite specification. However, you cannot rename, move, or change the segment size of a member of a composite. To edit the composite itself, you must delete it and define a new composite.

Suppose that you wanted to make Channel, instead of Time, the fastest varying dimension of the cube in the analytic workspace. You could reposition Channel in the composite specification as follows.

```
exec dbms_awm.set_awcomp_spec_member_pos
          ('comp1', 'xademo', 'analytic_cube', 'comp1_channel', 1);
```

Aggregating the Data in an Analytic Workspace

The `DBMS_AWM` package enables you to store aggregate data for level combinations of measures in a workspace cube.

Stored aggregates in an analytic workspace are similar to materialized views for relational data. However, a workspace cube is always presented as fully solved with embedded totals when queried by an application. If you do not preaggregate any of the workspace data, all the aggregate data is still available but it must be calculated on the fly.

Preaggregating some or all of your workspace data will improve query performance in most circumstances. For information on choosing an aggregation strategy, refer to the *Oracle OLAP Application Developer's Guide*

> **Note:** The aggregation process (`AGGREGATE_AWCUBE`) opens the analytic workspace with read/write access. It updates the workspace, but it *does not* execute a SQL `COMMIT`.

The cube refresh process stores detail data in the workspace and sets up the structures to support dynamic aggregation. If you want to preaggregate some or all of your data, you must create an aggregation specification and run a separate aggregation procedure for the workspace cube.

Example 1–4 shows sample procedure calls for preaggregating the Costs and Quota measures of the analytic workspace cube `AC2`, which was created from `XADEMO.ANALYTIC_CUBE`.

The quarter totals (level 'L2' of TIME) for product groups (level 'L3' of PRODUCT), product divisions (level 'L2' of PRODUCT), and all channels (level 'STANDARD-2' of CHANNEL) are calculated and stored in the analytic workspace.

Example 1–4 Preaggregating Costs and Quota in an Analytic Workspace

```
execute dbms_awm.create_awcubeagg_spec
          ('AGG1', 'MYSCHEMA', 'MYAW', 'AC2');
execute dbms_awm.add_awcubeagg_spec_level
          ('AGG1', 'MYSCHEMA', 'MYAW', 'AC2', 'PRODUCT', 'L3');
execute dbms_awm.add_awcubeagg_spec_level
          ('AGG1', 'MYSCHEMA', 'MYAW', 'AC2', 'PRODUCT', 'L2');
execute dbms_awm.add_awcubeagg_spec_level
          ('AGG1', 'MYSCHEMA', 'MYAW', 'AC2', 'CHANNEL', 'STANDARD_2');
execute dbms_awm.add_awcubeagg_spec_level
          ('AGG1', 'MYSCHEMA', 'MYAW', 'AC2', 'TIME', 'L2');
execute dbms_awm.add_awcubeagg_spec_measure
          ('AGG1', 'XADEMOAW', 'UK', 'AC2', 'XXF_COSTS');
execute dbms_awm.add_awcubeagg_spec_measure
          ('AGG1', 'XADEMOAW', 'UK', 'AC2', 'XXF_QUOTA');
execute dbms_awm.aggregate_awcube('MYSCHEMA', 'MYAW', 'AC2', 'AGG1');
```

The following statements show the measures and the PRODUCT levels in the aggregation plan in the analytic workspace.

```
execute dbms_aw.execute  ('aw attach MYSCHEMA.MYAW ro');
execute dbms_aw.execute  ('fulldsc agg1');

DEFINE AGG1 DIMENSION TEXT
LD List of Measures which use this AggPlan
PROPERTY 'AW$CLASS' -'IMPLEMENTATION'
PROPERTY 'AW$CREATEDBY' -'AW$CREATE'
PROPERTY 'AW$LASTMODIFIED' -'.*
PROPERTY 'AW$LOGICAL_NAME' -'AGG1'
PROPERTY 'AW$PARENT_NAME' -'AC2'
PROPERTY 'AW$ROLE' -'AGGDEF'
PROPERTY 'AW$STATE' -'ACTIVE'

execute dbms_aw.execute('rpr agg1')

AGG1
--------------
XXF.COSTS
XXF.QUOTA

execute dbms_aw.execute('fulldsc agg1_product');

DEFINE AGG1_PRODUCT VALUESET PRODUCT_LEVELLIST
LD List of Levels for this AggPlan
PROPERTY 'AW$AGGOPERATOR' -'SUM'
PROPERTY 'AW$CLASS' -'IMPLEMENTATION'
PROPERTY 'AW$CREATEDBY' -'AW$CREATE'
PROPERTY 'AW$LASTMODIFIED' -'.*
PROPERTY 'AW$PARENT_CUBE' -'AC2'
PROPERTY 'AW$PARENT_DIM' -'PRODUCT'
PROPERTY 'AW$PARENT_NAME' -'AGG1'
PROPERTY 'AW$ROLE' -'AGGDEF_LEVELS'
PROPERTY 'AW$STATE' -'ACTIVE'

execute dbms_aw.execute('shw values(agg1_product)');
```

```
L3
L2
```

An aggregation method specifies the operation used to summarize the data by level. The default aggregation method is addition. For example, sales data is typically aggregated over time by adding the values for each time period.

The OLAP Catalog supports a set of aggregation methods, which may be included in the definition of a cube. These aggregation methods are listed in Table 1–10.

When a workspace cube is refreshed, the aggregation operators specified in the OLAP Catalog are converted to the corresponding operators supported by the OLAP DML `RELATION` command. These operators are incorporated in the aggregation map that controls dynamic aggregation for the cube.

To specify a different operator for your stored aggregates, you can use the `SET_AWCUBEAGG_SPEC_AGGOP` procedure. This procedure enables you to specify any of the operators supported by the OLAP DML `RELATION` command to preaggregate your data.

> **Note:** The `DBMS_AWM` package currently does not support weighted aggregation operators. For example, if the OLAP Catalog specifies a weighted sum or weighted average for aggregation along one of the cube's dimensions, it is converted to the scalar equivalent (sum or average) when the cube is refreshed in the analytic workspace. Weighted operators specified by `SET_AWCUBEAGG_SPEC_AGGOP` are similarly converted.

The OLAP Catalog and corresponding OLAP DML aggregation operators are described in Table 1–10.

Table 1–10 Aggregation Operators

OLAP Catalog	OLAP DML	DML Abbrv	Description
SUM	SUM	SU	Sum. Adds data values (default)
SCALED SUM	SSUM	SS	Converted to Sum.
WEIGHTED SUM	WSUM	WS	Converted to Sum.
AVERAGE	AVERAGE	AV	Average. Adds data values, then divides the sum by the number of data values that were added together.
HIERARCHICAL AVERAGE	HAVERAGE	HA	Hierarchical Average. Adds data values, then divides the sum by the number of the children in the dimension hierarchy.
WEIGHTED AVERAGE	WAVERAGE	WA	Converted to Average.
	HWAVERAGE	HW	Converted to Hierarchical Average.
MAX	MAX	MA	Maximum. The largest data value among the children of any parent data value.
MIN	MIN	MI	Minimum. The smallest data value among the children of any parent data value.
FIRST	FIRST	FI	First. The first non-NA data value.
	HFIRST	HF	Hierarchical First. The first data value that is specified by the hierarchy, even if that value is NA.

Table 1–10 (Cont.) Aggregation Operators

OLAP Catalog	OLAP DML	DML Abbrv	Description
LAST	LAST	LA	Last. The last non-NA data value.
	HLAST	HL	Hierarchical Last. The last data value that is specified by the hierarchy, even if that value is NA.
AND	AND	AN	(Boolean variables only) If any child data value is FALSE, then the data value of its parent is FALSE. A parent is TRUE only when all of its children are TRUE.
OR	OR	OR	(Default for Boolean variables) If any child data value is TRUE, then the data value of its parent is TRUE. A parent is FALSE only when all of its children are FALSE.
COUNT		NO	Converted to NOAGG.
	NOAGG	NO	Do not aggregate any data for this dimension.

Enabling Access by the OLAP API

Analytic workspaces created with the current DBMS_AWM package or Analytic Workspace Manager automatically support queries by the OLAP API and BI Beans. You do not need to create relational views, abstract data types, or OLAP Catalog metadata to enable access. Current standard form metadata supports the automatic generation of the mapping information needed by OLAP queries at runtime.

The DBMS_AWM package will attempt to upgrade the standard form metadata when it attaches a workspace created in a previous version. You can explicitly upgrade the metadata with a one-time call to the CREATE_DYNAMIC_AW_ACCESS procedure. The workspace must already be in 10*g* storage format before the metadata can be upgraded.

See Also:

- "Converting an Analytic Workspace to Oracle 10g Storage Format" on page 24-2 for the storage format upgrade procedure.
- "CREATE_DYNAMIC_AW_ACCESS Procedure" on page 26-28 for the syntax of the standard form metadata upgrade procedure.

If you choose not to upgrade to the current standard form, you can continue to use the DBMS_AWM enablement procedures as you did in previous releases. The relational views, abstract data types, and OLAP Catalog metadata will still be valid. However, any views, abstract data types, and OLAP Catalog metadata associated with upgraded analytic workspaces will be ignored by the OLAP API.

Enabling Relational Access

If your analytic workspace must support ad hoc SQL queries and applications that do not use the OLAP API, you need to create views that present the workspace data in relational format.

Once you have created an analytic workspace cube and refreshed and aggregated its data, you can use the DBMS_AWM enablement procedures to create and maintain a set of views that can be queried with standard SQL. The DBMS_AWM enablement views include:

- An embedded total dimension view for each dimension hierarchy.

- An embedded total fact view for each combination of dimension hierarchies.

When you refresh a dimension or cube because of metadata change for its hierarchies, you must regenerate its enablement views. When you refresh a dimension or cube because of data changes, you can continue to use the pre-existing views.

If the enablement views do not provide the data in a useful format for your application, you can create your own views. Refer to Chapter 34, "OLAP_TABLE" for more information.

You can use DBMS_AWM enablement procedures to generate the enablement scripts and run them yourself, or you can use a one-step procedure to create and run the scripts automatically.

Procedure: Generate and Run the Enablement Scripts

Use the following steps to create and run the enablement scripts for an analytic workspace:

1. Determine how your system is configured to write to files. The enabler procedures accept either a directory object or a directory path. If you specify a directory object, make sure that your user ID has been granted the appropriate access rights to it. If you specify a path, make sure that it is the value of the UTL_FILE_DIR initialization parameter for the instance.
2. Run the REFRESH_AWCUBE and REFRESH_AWDIMENSION procedures to refresh the cube. These procedures create metadata in the analytic workspace to track the generations of enablement view names.
3. The enablement process automatically provides system-generated names for the enablement views. To provide your own view names, call REFRESH_AWDIMENSION_VIEW_NAME and REFRESH_AWCUBE_VIEW_NAME, then call SET_AWDIMENSION_VIEW_NAME and SET_AWCUBE_VIEW_NAME.
4. Call the CREATE_AWDIMENSION_ACCESS procedure for each of the cube's dimensions. Each procedure call will create an enablement script in a directory that you specify. The script will contain statements that create the dimension views and, optionally, an OLAP Catalog dimension that maps to the views.
5. Call the CREATE_AWCUBE_ACCESS procedure. This procedure call will create an enablement script in a directory that you specify. The script will contain statements that create the fact views and, optionally, an OLAP Catalog cube that maps to the views.
6. Run the enablement scripts. The scripts will delete any previous generation of views and metadata before creating new views and metadata.

Procedure: Run the Enablement Scripts Automatically

To create and run the enablement scripts automatically, use the following steps:

1. Refresh the cube and its dimensions in the analytic workspace, as described in step 2 of "Procedure: Generate and Run the Enablement Scripts" on page 1-18.
2. If you want to specify your own view names, follow step 3 of "Procedure: Generate and Run the Enablement Scripts" on page 1-18.
3. Call CREATE_AWDIMENSION_ACCESS_FULL for each of the cube's dimensions. This procedure creates the enablement scripts in temporary memory and runs the scripts to create the dimension views and, optionally, the OLAP Catalog metadata.

The scripts delete any previous views and metadata before creating new views and metadata.

4. Call the procedure `CREATE_AWCUBE_ACCESS_FULL` to create the fact views for the cube. This procedure accomplishes the same basic steps as the corresponding procedure for dimensions.

The OLAP API Enabler Procedures

The OLAP API enabler procedures are listed in Table 1–11.

Table 1–11 The OLAP API Enabler Procedures

Procedure	Description
CREATE_AWCUBE_ACCESS Procedure	Creates a script that enables relational access to a cube in an analytic workspace.
CREATE_AWCUBE_ACCESS_FULL Procedure	Enables relational access to a cube in an analytic workspace.
CREATE_AWDIMENSION_ACCESS Procedure	Creates a script that enables relational access to a dimension in an analytic workspace.
CREATE_AWDIMENSION_ACCESS_FULL Procedure	Enables relational access to a dimension in an analytic workspace.
DELETE_AWCUBE_ACCESS Procedure	Creates a script that deletes the enablement views and metadata for a cube in an analytic workspace.
DELETE_AWCUBE_ACCESS_ALL Procedure	Deletes the enablement views and metadata for a cube in an analytic workspace.
DELETE_AWDIMENSION_ACCESS Procedure	Creates a script that deletes the enablement views and metadata for a dimension in an analytic workspace.
DELETE_AWDIMENSION_ACCESS_ALL Procedure	Deletes the enablement views and metadata for a dimension in an analytic workspace.
REFRESH_AWCUBE_VIEW_NAME Procedure	Creates metadata in the analytic workspace to support user-defined view names for a cube.
REFRESH_AWDIMENSION_VIEW_NAME Procedure	Creates metadata in the analytic workspace to support user-defined view names for a dimension.
SET_AWCUBE_VIEW_NAME Procedure	Replaces the system-generated names for the views of an analytic workspace cube.
SET_AWDIMENSION_VIEW_NAME Procedure	Replaces the system-generated names for the views of an analytic workspace dimension.

Note: If you capture the SQL generated by Analytic Workspace Manager and use it to create your own scripts, you will need to edit the enablement procedure calls. Analytic Workspace Manager uses different versions of the enablement procedures. In your scripts, you must use the syntax described in this manual.

Disabling Relational Access

The enablement procedures automatically delete any previous generation of views and OLAP Catalog metadata. However, in some circumstances, you might want to drop

the views and metadata without re-creating them. In particular, if you drop the workspace cube or the workspace itself, you will need to clean up the orphaned views and metadata.

In this case, you can run the DELETE_AWDIMENSION_ACCESS and DELETE_AWCUBE_ACCESS procedures to generate scripts that will drop the views and metadata that enable relational access to the cube. These scripts do not delete any enablement metadata that is stored within the analytic workspace.

To delete all the enablement views and metadata for a dimension or a cube, use DELETE_AWCUBE_ACCESS_ALL and DELETE_AWDIMENSION_ACCESS_ALL.

Specifying Names for Dimension Views

The CREATE_AWDIMENSION_ACCESS and CREATE_AWDIMENSION_ACCESS_FULL procedures create metadata in the analytic workspace related to enablement. This metadata includes a set of default names for the enablement views.

If you want to specify your own view names, you must refresh this metadata by calling REFRESH_AWDIMENSION_VIEW_NAME. Then call SET_AWDIMENSION_VIEW_NAME to specify the names of the views.

Whenever you re-create the views, new view names are generated. If you have previously created your own names, the refresh process uses them as the basis for the new names.

The default view name for a dimension is: *aaaa_bbbbb_ccccc_ddddd*#view, where:

aaaa is the first four characters of the analytic workspace owner

bbbbb is the first five characters of the analytic workspace name

ccccc is the first five characters of the analytic workspace dimension name

ddddd is the first five characters of the analytic workspace hierarchy name

is an automatically-generated sequence number between 1 and 9,999 to ensure uniqueness.

Default names are also generated for the abstract objects (ADTs) populated by OLAP_TABLE. For example, the workspace dimension AWGEOG, in a workspace called AWTEST in the XADEMO schema could have the following system-generated names for the STANDARD hierarchy.

Default Name	Description
XADE_AWTES_AWGE0_STAND34VIEW	Name of the relational view
XADE_AWTES_AWGEOG34OBJ	Name of the abstract object that defines a row in the abstract table of objects populated by OLAP_TABLE
XADE_AWTES_AWGEOG34TBL	Name of the abstract table type populated by OLAP_TABLE

Specifying Names for Fact Views

The CREATE_AWCUBE_ACCESS and CREATE_AWCUBE_ACCESS_FULL procedures create metadata in the analytic workspace related to enablement. This metadata includes a set of default names for the enablement views.

If you want to specify your own view names, you must refresh this metadata by calling REFRESH_AWCUBE_VIEW_NAME. Then call SET_AWCUBE_VIEW_NAME to specify the names of the views.

Whenever you re-create the views, new view names are generated. If you have previously created your own names, the refresh process uses them as the basis for the new names.

The default view name for a cube is: *aaaa_bbbbb_cccccccc*#view, where:

aaaa is the first four characters of the analytic workspace owner

bbbbb is the first five characters of the analytic workspace name

cccccccc is the first eight characters of the analytic workspace cube name

is an automatically-generated sequence number between 1 and 9,999 to ensure uniqueness.

Default names are also generated for the abstract objects (ADTs) populated by OLAP_TABLE. For example, the workspace cube AWCUBE, in a workspace called AWTEST in the XADEMO schema could have the following system-generated names.

Default Name	Description
XADE_AWTES_AWCUBE8VIEW	Name of the relational fact view for the first hierarchy combination.
XADE_AWTES_AWCUBE9VIEW	Name of the relational fact view for the second hierarchy combination.
XADE_AWTES_AWCUBE10VIEW	Name of the relational fact view for the third hierarchy combination.
XADE_AWTES_AWCUBE11VIEW	Name of the relational fact view for the fourth hierarchy combination.
XADE_AWTES_AWCUBE7OBJ	Name of the abstract object that defines a row in the abstract table of objects populated by OLAP_TABLE
XADE_AWTES_AWCUBE7TBL	Name of the abstract table type populated by OLAP_TABLE

Column Structure of Dimension Views

The enablement process generates a separate view for each dimension hierarchy. For example, a workspace cube with the four dimensions shown in Table 1–12 would have six separate dimension views since two of the dimensions have two hierarchies.

Table 1–12 Sample Dimension Hierarchies

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

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

Each dimension view contains the columns described in Table 1–13.

Table 1–13 Dimension View Columns

Column	Description
ET key	The embedded-total key column stores the value of the lowest populated level in the row.
Parent ET key	The parent embedded-total key column stores the parent of each ET key value.
GID	The grouping ID column identifies the hierarchy level associated with each row, as described in "Grouping ID Column" on page 1-22.
Parent GID	The parent grouping ID column stores the parent of each GID value.
level columns	A column for each level of the dimension hierarchy. These columns provide the full ancestry of each dimension member within a single row.
level attribute columns	A column for each level attribute.

Sample Dimension View

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

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

Grouping ID Column

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

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

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

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

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

Column Structure of Fact Views

The CREATE_AWCUBE_ACCESS procedure generates a separate view for each dimension/hierarchy combination. For example, an analytic workspace cube with the four dimensions shown in Table 1–12, would have four separate fact views, one for each hierarchy combination show in Table 1–14.

Table 1–14 Sample Dimension/Hierarchy Combinations

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

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

Each fact view contains the columns described in Table 1–15.

Table 1–15 Fact View Columns

Column	Description
ET key for each dimension/hierarchy	The ET key columns are foreign keys that map to the primary keys of the associated dimension tables, and are used to join the measure table with the dimension tables.
GID for each dimension/hierarchy	The GID column provides grouping IDs needed by the OLAP API for optimal response time. It is identical to the GID column of the associated dimension table.
measure columns	Columns for each of the cube's measures.
R2C	A column that stores information used by the single-row functions. See Chapter 30, "OLAP_EXPRESSION" and "Limit Map: ROW2CELL Clause" on page 34-20.
CUST_MEAS_TEXT*n*	100 sequentially numbered empty columns with a data type of VARCHAR2(1000).
CUST_MEAS_NUM*n*	100 sequentially numbered empty columns with a data type of NUMBER(38,6).

Example: Enable a Workspace Cube for Relational Access

The following example creates, refreshes, and enables a cube AWUSR.AWTEST based on the source cube XADEMO.ANALYTIC_CUBE.

Example 1–5 Create, Refresh, and Enable a Cube

```
-- SET UP
set serveroutput on size 1000000
execute cwm2_olap_manager.set_echo_on;

create or replace directory myscripts as '/users/myxademo/myscripts';
execute cwm2_olap_manager.begin_log ('MYSCRIPTS' , 'awtest.log');
```

```
--- CREATE AW
execute dbms_aw.execute ('aw create ''AWTEST''');

-- CREATE DIMENSIONS
execute dbms_awm.create_awdimension
          ('XADEMO','CHANNEL',   'AWUSR', 'AWTEST', 'AWCHAN');
execute dbms_awm.create_awdimension
          ('XADEMO','GEOGRAPHY', 'AWUSR', 'AWTEST', 'AWGEOG');
execute dbms_awm.create_awdimension
          ('XADEMO','PRODUCT',   'AWUSR', 'AWTEST', 'AWPROD');
execute dbms_awm.create_awdimension
          ('XADEMO','TIME',      'AWUSR', 'AWTEST', 'AWTIME');

-- CREATE CUBE
execute dbms_awm.create_awcube
          ('XADEMO', 'ANALYTIC_CUBE','AWUSR', 'AWTEST','AWCUBE');

-- REFRESH DIMENSIONS
execute dbms_awm.refresh_awdimension ('AWUSR', 'AWTEST', 'AWCHAN');
execute dbms_awm.refresh_awdimension ('AWUSR', 'AWTEST', 'AWGEOG');
execute dbms_awm.refresh_awdimension ('AWUSR', 'AWTEST', 'AWPROD');
execute dbms_awm.refresh_awdimension ('AWUSR', 'AWTEST', 'AWTIME');

-- REFRESH CUBE
execute dbms_awm.refresh_awcube ('AWUSR', 'AWTEST', 'AWCUBE');

-- SET DIMENSION VIEW NAMES
exec dbms_awm.refresh_awdimension_view_name
          ('AWUSR', 'AWTEST', 'awprod');
exec dbms_awm.refresh_awdimension_view_name
          ('AWUSR', 'AWTEST', 'awchan');
exec dbms_awm.refresh_awdimension_view_name
          ('AWUSR', 'AWTEST', 'awgeog');
exec dbms_awm.refresh_awdimension_view_name
          ('AWUSR', 'AWTEST', 'awtime');
exec dbms_awm.set_awdimension_view_name
          ('AWUSR', 'AWTEST', 'awprod', 'standard',     'prod_std_view');
exec dbms_awm.set_awdimension_view_name
          ('AWUSR', 'AWTEST', 'awchan', 'standard',     'chan_std_view');
exec dbms_awm.set_awdimension_view_name
          ('AWUSR', 'AWTEST', 'awgeog', 'consolidated', 'geog_csd_view');
exec dbms_awm.set_awdimension_view_name
          ('AWUSR', 'AWTEST', 'awgeog', 'standard',     'geog_std_view');
exec dbms_awm.set_awdimension_view_name
          ('AWUSR', 'AWTEST', 'awtime', 'standard',     'time_std_view');
exec dbms_awm.set_awdimension_view_name
          ('AWUSR', 'AWTEST', 'awtime', 'ytd',          'time_ytd_view');

-- SET CUBE VIEW NAMES
exec dbms_awm.refresh_awcube_view_name
          ('AWUSR', 'AWTEST', 'awcube');
exec dbms_awm.set_awcube_view_name
          ('AWUSR', 'AWTEST', 'awcube', 1, 'AWCUBE_view1');
exec dbms_awm.set_awcube_view_name
          ('AWUSR', 'AWTEST', 'awcube', 2, 'AWCUBE_view2');
exec dbms_awm.set_awcube_view_name
          ('AWUSR', 'AWTEST', 'awcube', 3, 'AWCUBE_view3');
exec dbms_awm.set_awcube_view_name
          ('AWUSR', 'AWTEST', 'awcube', 4, 'AWCUBE_view4');
```

```
-- ENABLE DIMENSIONS
exec dbms_awm.create_AWdimension_access
          ('AWUSR', 'AWTEST', 'awprod', 'olap',
           'MYSCRIPTS', 'awprod_views.sql', 'w');
exec dbms_awm.create_AWdimension_access
          ('AWUSR', 'AWTEST', 'awchan', 'olap',
           'MYSCRIPTS', 'awchan_views.sql', 'w');
exec dbms_awm.create_AWdimension_access
          ('AWUSR', 'AWTEST', 'awgeog', 'olap',
           'MYSCRIPTS', 'awgeog_views.sql', 'w');
exec dbms_awm.create_AWdimension_access
          ('AWUSR', 'AWTEST', 'awtime', 'olap',
           'MYSCRIPTS', 'awtime_views.sql', 'w');

-- ENABLE CUBE
exec dbms_awm.create_AWcube_access
          ('AWUSR', 'AWTEST', 'awcube', 'olap',
           'MYSCRIPTS', 'awcube_views.sql', 'w');

-- COMMIT and WRAPUP
commit;
execute cwm2_olap_manager.end_log;
```

The following queries show the resulting workspace cube and dimensions with their source cubes and dimensions in the OLAP Catalog.

```
select * from all_olap2_aw_dimensions where AW_OWNER = 'AWUSER';

AW_OWNER  AW_NAME   AW_LOGICAL_NAME AW_PHYSICAL_OBJECT SOURCE_OWNER  SOURCE_NAME
--------- --------- --------------- ------------------ ------------- ---------------
AWUSER     AWTEST    AWCHAN          AWCHAN             XADEMO        CHANNEL
AWUSER     AWTEST    AWGEOG          AWGEOG             XADEMO        GEOGRAPHY
AWUSER     AWTEST    AWPROD          AWPROD             XADEMO        PRODUCT
AWUSER     AWTEST    AWTIME          AWTIME             XADEMO        TIME

select * from all_olap2_aw_CUBEs where AW_OWNER = 'AWUSER';

AW_OWNER AW_NAME  AW_LOGICAL_NAME AW_PHYSICAL_OBJECT  SOURCE_OWNER  SOURCE_NAME
-------- -------- --------------- ------------------- ------------- ---------------
AWUSER    AWTEST   AWCUBE          AWCUBE              XADEMO        ANALYTIC_CUBE
```

The following query shows the system names and user names for the dimension enablement views.

```
select * from all_aw_dim_ENABLED_VIEWS where AW_OWNER = 'AWUSER';

AW_OWNER AW_NAME  DIMENSION_ HIERARCHY_   SYSTEM_VIEWNAME               USER_VIEWNAME
-------- -------- ---------- ----------   ---------------------------   ----------------
AWUSER    AWTEST   AWCHAN     STANDARD     AWUS_AWTES_AWCHA_STAND144VIEW  CHAN_STD_VIEW
AWUSER    AWTEST   AWGEOG     CONSOLIDATED AWUS_AWTES_AWGEO_CONSO145VIEW  GEOG_CSD_VIEW
AWUSER    AWTEST   AWGEOG     STANDARD     AWUS_AWTES_AWGEO_STAND146VIEW  GEOG_STD_VIEW
AWUSER    AWTEST   AWPROD     STANDARD     AWUS_AWTES_AWPRO_STAND147VIEW  PROD_STD_VIEW
AWUSER    AWTEST   AWTIME     STANDARD     AWUS_AWTES_AWTIM_STAND148VIEW  TIME_STD_VIEW
AWUSER    AWTEST   AWTIME     YTD          AWUS_AWTES_AWTIM_YTD149VIEW    TIME_YTD_VIEW
```

The following query shows the system names and user names for the cube enablement views. Included are the hierarchy combination numbers, in this case 1 - 4, and the hierarchy strings, consisting of each unique combination of dimension hierarchies for this cube.

```
select * from all_aw_CUBE_ENABLED_VIEWS where AW_OWNER = 'AWUSER';
```

```
AW_OWN AW_NA  CUBE_NAM  HIER  HIERCOMBO_STR                          SYSTEM_VIEWNAME            USER_VIEWNAME
------ ------ --------  ----  ---------------------------------  ----------------------   -----------
AWUSER  AWTEST  AWCUBE   1     DIM:AWCHAN/HIER:STANDARD;DIM:AWGEOG   AWUS_AWTES_AWCUBE151VIEW  AWCUBE_VIEW1
                                /HIER:CONSOLIDATED;DIM:AWPROD/HIER:
                                STANDARD;DIM:AWTIME/HIER:STANDARD
AWUSER  AWTEST  AWCUBE   2     DIM:AWCHAN/HIER:STANDARD;DIM:AWGEOG   AWUS_AWTES_AWCUBE152VIEW  AWCUBE_VIEW2
                                /HIER:CONSOLIDATED;DIM:AWPROD/HIER:
                                STANDARD;DIM:AWTIME/HIER:YTD
AWUSER   AWTEST  AWCUBE  3     DIM:AWCHAN/HIER:STANDARD;DIM:AWGEOG   AWUS_AWTES_AWCUBE153VIEW  AWCUBE_VIEW3
                                /HIER:STANDARD;DIM:AWPROD/HIER:STAN
                                DARD;DIM:AWTIME/HIER:STANDARD
AWUSER   AWTEST  AWCUBE  4     DIM:AWCHAN/HIER:STANDARD;DIM:AWGEOG   AWUS_AWTES_AWCUBE154VIEW   AWCUBE_VIEW4
                                /HIER:STANDARD;DIM:AWPROD/HIER:STAN
                                DARD;DIM:AWTIME/HIER:YTD
```

The final step is to run the enablement scripts to generate the views for the analytic workspace cube. The scripts produced by this example are described as follows.

Script	Description
`awprod_views.sql`	Creates an abstract object, a table of objects, and a view for the `PRODUCT` dimension. Also creates and validates an OLAP Catalog dimension `AWUSER.AWPROD` that maps to the view.
`awchan_views.sql`	Creates an abstract object, a table of objects, and a view for the `CHANNEL` dimension. Also creates and validates an OLAP Catalog dimension `AWUSER.AWCHAN` that maps to the view.
`awgeog_views.sql`	Creates an abstract object, a table of objects, and a view for each hierarchy of the `GEOGRAPHY` dimension. Also creates and validates an OLAP Catalog dimension `AWUSER.AWGEOG` that maps to the view.
`awtime_views.sql`	Creates an abstract object, a table of objects, and a view for each hierarchy of the TIME dimension. Also creates and validates an OLAP Catalog dimension `AWUSER.AWTIME` that maps to the view.
`awcube_views.sql`	Creates an abstract object, a table of objects, and a separate view for each hierarchy combination of the `AWCUBE` cube. Also creates and validates an OLAP Catalog cube `AWUSER.AWCUBE` that maps to the view.

2

Creating OLAP Catalog Metadata with CWM2

The OLAP Catalog CWM2 PL/SQL packages provide stored procedures for creating, dropping, and updating OLAP Catalog metadata. This chapter explains how to work with the CWM2 procedures. For complete syntax descriptions, refer to the reference chapter for each package.

This chapter discusses the following topics:

Understanding OLAP Catalog Metadata

OLAP Catalog metadata presents relational data as a logical cube. It is stored in tables in the OLAP Catalog, and it can be queried using the OLAP Catalog views.

The OLAP API uses OLAP Catalog metadata to access relational data stored in star, snowflake, and embedded-total configurations.

The OLAP API does not use OLAP Catalog metadata to access data stored in analytic workspaces. However, OLAP Catalog metadata is required by the DBMS_AWM package (and Analytic Workspace Manager) for building analytic workspaces.

The DBMS_AWM enabler procedures optionally create an OLAP Catalog cube that maps to relational views of an analytic workspace. The OLAP API does not use this cube, nor its underlying views, to query an analytic workspace. The OLAP API queries data in an analytic workspace directly, using standard form metadata within the workspace.

The OLAP Analytic Workspace API does not use OLAP Catalog metadata for building analytic workspaces. See Chapter 25 for more information.

You can create OLAP Catalog metadata with Oracle Enterprise Manager, Oracle Warehouse Builder, or the CWM2 procedures. OLAP Catalog metadata created in Enterprise Manager maps to star and snowflake schemas only. You can create OLAP

Catalog metadata for embedded-total dimension and fact tables, as well as star and snowflakes, using Warehouse Builder or the CWM2 procedures.

See Also:

- "Overview" on page 1-1 for descriptions of the relational source cube used by DBMS_AWM and the relational target cube that can optionally be created by DBMS_AWM.
- Chapter 5 for descriptions of the OLAP Catalog views.

OLAP Catalog Metadata Entities

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

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

When you create an OLAP Catalog metadata entity, you are simply adding a row to an OLAP Catalog table that identifies all the entities of that type. Creating an entity does not fully define a dimension or a cube, nor does it involve any mapping to warehouse dimension tables or fact tables.

Note: All OLAP Catalog metadata entities are defined as VARCHAR(30).

To fully construct a dimension or a cube, you must understand the hierarchical relationships between the component metadata entities.

Creating a Dimension

Creating a dimension entity is only the first step in constructing the OLAP Catalog metadata for a dimension. Each dimension must have at least one level. More typically, it will have multiple levels, hierarchies, and attributes. Table 2–1 shows the parent-child relationships between the metadata components of a dimension.

Table 2–1 Hierarchical Relationships Between Components of a Dimension

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

Note: OLAP Catalog dimensions created with the CWM2 procedures are purely logical entities. They have no relationship to database dimension objects. However, OLAP Catalog dimensions created in Enterprise Manager *are* associated with database dimension objects.

Procedure: Create an OLAP Dimension

Generally, you will create hierarchies and dimension attributes after creating the dimension and before creating the dimension levels and level attributes. Once the levels and level attributes are defined, you can map them to columns in one or more warehouse dimension tables. The general steps are as follows:

1. Call procedures in `CWM2_OLAP_DIMENSION` to create the dimension.
2. Call procedures in `CWM2_OLAP_DIMENSION_ATTRIBUTE` to create dimension attributes. In general, you will need to define dimension attributes for `'long description'` and `'short description'`.

   The OLAP API requires the following dimension attributes for embedded total dimension tables (for example, views of analytic workspaces): `'ET Key'`, `'Parent ET Key'`, `'Grouping ID'`, and `'Parent Grouping ID'`. For more information, see Table 12–1, " Reserved Dimension Attributes".
3. Call procedures in `CWM2_OLAP_HIERARCHY` to define hierarchical relationships for the dimension's levels.
4. Call procedures in `CWM2_OLAP_LEVEL` to create levels and assign them to hierarchies.
5. Call procedures in `CWM2_OLAP_LEVEL_ATTRIBUTE` to create level attributes and assign them to dimension attributes. For `'long description'`, `'short description'` and other reserved dimension attributes, create level attributes with the same name for every level.

   The OLAP API requires the following level attributes for embedded total dimension tables (for example, views of analytic workspaces): `'ET Key'`, `'Parent ET Key'`, `'Grouping ID'`, and `'Parent Grouping ID'`. For more information, see Table 16–1, " Reserved Level Attributes".
6. Call procedures in `CWM2_OLAP_TABLE_MAP` to map the dimension's levels and level attributes to columns in dimension tables.

Example: Create a Product Dimension

The PL/SQL statements in Example 2–1 create a logical `CWM2` dimension, `PRODUCT_DIM`, for the `PRODUCTS` dimension table in the `SH` schema.

The following table shows the columns in the `PRODUCTS` table.

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

Column Name	Data Type
SUPPLIER_ID	NUMBER
PROD_STATUS	VARCHAR2
PROD_LIST_PRICE	NUMBER
PROD_MIN_PRICE	NUMBER
PROD_TOTAL	VARCHAR2

Example 2–1 Create an OLAP Dimension for the Products Table

```
---   CREATE THE PRODUCT DIMENSION    ---
exec cwm2_olap_dimension.create_dimension
          ('SH', 'PRODUCT_DIM', 'Product','Products', 'Product Dimension',
           'Product Dimension Values');

---   CREATE DIMENSION ATTRIBUTES  ---
exec cwm2_olap_dimension_attribute.create_dimension_attribute
          ('SH', 'PRODUCT_DIM', 'Long Description', 'Long Descriptions',
          'Long Desc', 'Long Product Descriptions', true);
exec cwm2_olap_dimension_attribute.create_dimension_attribute
          ('SH', 'PRODUCT_DIM', 'PROD_NAME_DIM', 'Product Name',
          'Prod Name', 'Product Name');

---   CREATE STANDARD HIERARCHY  ---
exec cwm2_olap_hierarchy.create_hierarchy
          ('SH', 'PRODUCT_DIM', 'STANDARD', 'Standard', 'Std Product',
           'Standard Product Hierarchy', 'Unsolved Level-Based');
exec cwm2_olap_dimension.set_default_display_hierarchy
          ('SH', 'PRODUCT_DIM', 'standard');

---   CREATE LEVELS  ---
exec cwm2_olap_level.create_level
          ('SH', 'PRODUCT_DIM', 'L4', 'Product ID', 'Product Identifiers',
          'Prod Key','Product Key');
exec cwm2_olap_level.create_level
          ('SH', 'PRODUCT_DIM', 'L3','Product Sub-Category',
          'Product Sub-Categories','Prod Sub-Category',
          'Sub-Categories of Products');
exec cwm2_olap_level.create_level
          ('SH', 'PRODUCT_DIM', 'L2','Product Category',
           'Product Categories', 'Prod Category', 'Categories of Products');
exec cwm2_olap_level.create_level
          ('SH', 'PRODUCT_DIM', 'L1', 'Total Product', 'Total Products',
           'Total Prod', 'Total Product');

---   CREATE LEVEL ATTRIBUTES  ---
exec 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);
exec 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);
exec 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);
```

```
exec cwm2_olap_level_attribute.create_level_attribute
          ('SH', 'PRODUCT_DIM', 'PROD_NAME_DIM', 'L4', 'PROD_NAME_LEV',
          'Product Name', 'Product Name', 'Product Name');

---   ADD LEVELS TO HIERARCHIES  ---
exec cwm2_olap_level.add_level_to_hierarchy
          ('SH', 'PRODUCT_DIM', 'STANDARD', 'L4', 'L3');
exec cwm2_olap_level.add_level_to_hierarchy
          ('SH', 'PRODUCT_DIM', 'STANDARD', 'L3', 'L2');
exec cwm2_olap_level.add_level_to_hierarchy
          ('SH', 'PRODUCT_DIM', 'STANDARD', 'L2', 'L1');
exec cwm2_olap_level.add_level_to_hierarchy
          ('SH', 'PRODUCT_DIM', 'STANDARD', 'L1');

---   CREATE MAPPINGS  ---
exec cwm2_olap_table_map.Map_DimTbl_HierLevel
          ('SH', 'PRODUCT_DIM', 'STANDARD', 'L4',
          'SH', 'PRODUCTS', 'PROD_ID');
exec cwm2_olap_table_map.Map_DimTbl_HierLevelAttr
          ('SH', 'PRODUCT_DIM', 'Long Description', 'STANDARD',
           'L4', 'Long Description', 'SH', 'PRODUCTS', 'PROD_DESC');
exec cwm2_olap_table_map.Map_DimTbl_HierLevelAttr
          ('SH', 'PRODUCT_DIM', 'PROD_NAME_DIM', 'STANDARD', 'L4',
          'PROD_NAME_LEV', 'SH', 'PRODUCTS', 'PROD_NAME');
exec cwm2_olap_table_map.Map_DimTbl_HierLevel
          ('SH', 'PRODUCT_DIM', 'STANDARD', 'L3','SH', 'PRODUCTS',
          'PROD_SUBCATEGORY');
exec cwm2_olap_table_map.Map_DimTbl_HierLevelAttr
          ('SH', 'PRODUCT_DIM', 'Long Description', 'STANDARD', 'L3',
          'Long Description', 'SH', 'PRODUCTS', 'PROD_SUBCATEGORY_DESC');
exec cwm2_olap_table_map.Map_DimTbl_HierLevel
          ('SH', 'PRODUCT_DIM', 'STANDARD', 'L2','SH', 'PRODUCTS',
          'PROD_CATEGORY');
exec cwm2_olap_table_map.Map_DimTbl_HierLevelAttr
          ('SH', 'PRODUCT_DIM', 'Long Description', 'STANDARD', 'L2',
          'Long Description', 'SH', 'PRODUCTS', 'PROD_CATEGORY_DESC');
exec cwm2_olap_table_map.Map_DimTbl_HierLevel
          ('SH', 'PRODUCT_DIM', 'STANDARD', 'L1','SH', 'PRODUCTS',
          'PROD_TOTAL');
```

Procedure: Create a Time Dimension

When constructing metadata for your time dimension tables, you will follow the same general procedure as for any other OLAP dimension. However, several additional requirements apply. The general steps for creating a time dimension are as follows:

1. Call procedures in `CWM2_OLAP_DIMENSION` to create the dimension. Specify `'TIME'` for the dimension type parameter.
2. Call procedures in `CWM2_OLAP_DIMENSION_ATTRIBUTE` to create dimension attributes. In addition to the dimension attributes needed for regular dimensions, define an `'End Date'` attribute and a `'Time Span'` attribute.
3. Call procedures in `CWM2_OLAP_HIERARCHY` to define hierarchical relationships for the dimension's levels. Typical hierarchies are Calendar and Fiscal.
4. Call procedures in `CWM2_OLAP_LEVEL` to create levels and assign them to hierarchies. Typical levels are Month, Quarter, and Year.
5. Call procedures in `CWM2_OLAP_LEVEL_ATTRIBUTE` to create level attributes and assign them to dimension attributes. In addition to the level attributes needed for

regular dimension attributes, create 'End Date' and 'Time Span' attributes for each level and associate them with the 'End Date' and 'Time Span' dimension attributes.

6. Call procedures in CWM2_OLAP_TABLE_MAP to map the dimension's levels and level attributes to columns in dimension tables. Map the 'End Date' level attributes to columns with a Date data type. Map the 'Time Span' level attributes to columns with a numeric data type.

Example: Create a Time Dimension

The PL/SQL statements in Example 2–1 create a logical CWM2 time dimension, TIME_DIM, for the TIMES dimension table in the SH schema.

The TIMES table includes the following columns.

Column Name	Data Type
TIME_ID	DATE
TIME_ID_KEY	NUMBER
DAY_NAME	VARCHAR2(9)
CALENDAR_MONTH_NUMBER	NUMBER(2)
CALENDAR_MONTH_DESC	VARCHAR2(8)
CALENDAR_MONTH_DESC_KEY	NUMBER
END_OF_CAL_MONTH	DATE
CALENDAR_MONTH_NAME	VARCHAR2(9)
CALENDAR_QUARTER_DESC	CHAR(7)
CALENDAR_QUARTER_DESC_KEY	NUMBER
END_OF_CAL_QUARTER	DATE
CALENDAR_QUARTER_NUMBER	NUMBER(1)
CALENDAR_YEAR	NUMBER(4)
CALENDAR_YEAR_KEY	NUMBER
END_OF_CAL_YEAR	DATE

Example 2–2 Create an OLAP TIme Dimension

```
---   CREATE THE TIME DIMENSION
exec cwm2_olap_dimension.create_dimension
          ('SH', 'TIME_DIM', 'Time','Time', 'Time Dimension',
           'Time Dimension Values', 'TIME');

---   CREATE DIMENSION ATTRIBUTE END DATE
exec cwm2_olap_dimension_attribute.create_dimension_attribute
          ('SH', 'TIME_DIM', 'END DATE', 'End Date',
          'End Date', 'Last date of time period', true);

---   CREATE CALENDAR HIERARCHY
exec cwm2_olap_hierarchy.create_hierarchy
          ('SH', 'TIME_DIM', 'CALENDAR', 'Calendar', 'Calendar Hierarchy',
           'Calendar Hierarchy', 'Unsolved Level-Based');
exec cwm2_olap_dimension.set_default_display_hierarchy
```

```
          ('SH', 'TIME_DIM', 'CALENDAR');

---   CREATE LEVELS
exec cwm2_olap_level.create_level
          ('SH', 'TIME_DIM', 'MONTH', 'Month', 'Months', 'Month','Month');
exec cwm2_olap_level.create_level
          ('SH','TIME_DIM','QUARTER','Quarter','Quarters','Quarter','Quarter');
exec cwm2_olap_level.create_level
          ('SH', 'TIME_DIM', 'YEAR','Year','Years', 'Year', 'Year');

---   CREATE LEVEL ATTRIBUTES  ---
exec cwm2_olap_level_attribute.create_level_attribute
          ('SH', 'TIME_DIM', 'END DATE', 'Month', 'END DATE',
          'End Date', 'End Date',
          'Last date of time period', TRUE);
exec cwm2_olap_level_attribute.create_level_attribute
          ('SH', 'TIME_DIM', 'END DATE', 'Quarter', 'END DATE',
          'End Date', 'End Date',
          'Last date of time period', TRUE);
exec cwm2_olap_level_attribute.create_level_attribute
          ('SH', 'TIME_DIM', 'END DATE', 'Year', 'END DATE',
          'End Date', 'End Date',
          'Last date of time period', TRUE);

---   ADD LEVELS TO HIERARCHIES
exec cwm2_olap_level.add_level_to_hierarchy
          ('SH', 'TIME_DIM', 'CALENDAR', 'Month', 'Quarter');
exec cwm2_olap_level.add_level_to_hierarchy
          ('SH', 'TIME_DIM', 'CALENDAR', 'Quarter', 'Year');
exec cwm2_olap_level.add_level_to_hierarchy
          ('SH', 'TIME_DIM', 'CALENDAR', 'Year');

---   CREATE MAPPINGS
exec cwm2_olap_table_map.Map_DimTbl_HierLevel
          ('SH', 'TIME_DIM', 'CALENDAR', 'Year',
          'SH', 'TIMES', 'CALENDAR_YEAR_ID');
exec cwm2_olap_table_map.Map_DimTbl_HierLevelAttr
          ('SH', 'TIME_DIM', 'END DATE', 'CALENDAR',
           'Year', 'END DATE', 'SH', 'TIMES', 'END_OF_CAL_YEAR');
exec cwm2_olap_table_map.Map_DimTbl_HierLevel
          ('SH', 'TIME_DIM', 'CALENDAR', 'Quarter','SH', 'TIMES',
          'CALENDAR_QUARTER_NUMBER');
exec cwm2_olap_table_map.Map_DimTbl_HierLevelAttr
          ('SH', 'TIME_DIM', 'END DATE', 'CALENDAR',
           'Quarter', 'END DATE', 'SH', 'TIMES', 'END_OF_CAL_QUARTER');
exec cwm2_olap_table_map.Map_DimTbl_HierLevel
          ('SH', 'TIME_DIM', 'CALENDAR', 'Month','SH', 'TIMES',
          'CALENDAR_MONTH_NUMBER');
exec cwm2_olap_table_map.Map_DimTbl_HierLevelAttr
          ('SH', 'TIME_DIM', 'END DATE', 'CALENDAR',
           'Month', 'END DATE', 'SH', 'TIMES', 'END_OF_CAL_MONTH');
```

Creating a Cube

Creating a cube entity is only the first step in constructing the OLAP Catalog metadata for a cube. Each cube must have at least one dimension and at least one measure. More typically, it will have multiple dimensions and multiple measures.

Procedure: Create a Cube

The general steps for constructing a cube are as follows:

1. Follow the steps in "Procedure: Create an OLAP Dimension" for each of the cube's dimensions.
2. Call procedures in CWM2_OLAP_CUBE to create the cube and identify its dimensions.
3. Call procedures in CWM2_OLAP_MEASURE to create the cube's measures.
4. Call procedures in CWM2_OLAP_TABLE_MAP to map the cube's measures to columns in fact tables and to map foreign key columns in the fact tables to key columns in the dimension tables.

Example: Create a Costs Cube

The PL/SQL statements in Example 2–3 create a logical CWM2 cube object, ANALYTIC_CUBE, for the COSTS fact table in the SH schema. The dimensions of the cube are PRODUCT_DIM, shown in Example 2–1, and TIME_DIM, shown in Example 2–2.

The COSTS fact table has the following columns.

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

Example 2–3 Create an OLAP Cube for the COSTS Fact Table

```
---   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', 'LOWESTLEVEL',
      '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;');
```

Mapping OLAP Catalog Metadata

OLAP Catalog metadata mapping is the process of establishing the links between logical metadata entities and the physical locations where the data is stored. Dimension levels and level attributes map to columns in dimension tables. Measures map to columns in fact tables. The mapping process also specifies the join relationships between a fact table and its associated dimension tables.

Note: The dimension tables and fact tables may be implemented as views. For example, the views you can generate using the `DBMS_AWM` package may be the data source for OLAP Catalog metadata. For more information, see "Overview" on page 1-1.

Mapping to Columns

The `CWM2_OLAP_TABLE_MAP` package contains the mapping procedures for `CWM2` metadata. Dimension levels, level attributes, and measures can be mapped within the context of a hierarchy or with no hierarchical context.

Mapping Dimensions

Each 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 level attribute maps to a single column in the same table as its associated level.

Mapping Measures

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.

When more than one hierarchical context is possible within a cube (at least one of the cube's dimensions has multiple hierarchies), each combination of hierarchies may be mapped to a separate fact table. In this case, each table must have columns for each of the cube's measures, and the measure columns must appear in the same order in each table.

Joining Fact Tables with Dimension Tables

Once you have mapped the levels, level attributes, and measures, you can specify the mapping of logical foreign key columns in the fact table to level key columns in dimension tables.

The `MAP_FACTTBL_LEVELKEY` procedure defines the join relationships between a cube and its dimensions. This procedure takes as input: the cube name, the fact table name, a mapping string, and a storage type indicator specifying how data is stored in the fact table.

The storage type indicator can have either of the following values:

- **`'LOWESTLEVEL'`**
  A single fact table stores unsolved data for all the measures of a cube (star schema). If any of the cube's dimensions have more than one hierarchy, they must

all have the same lowest level. Each foreign key column in the fact table maps to a level key column in a dimension table.

- **'ET'**
  Fact tables store completely solved data (with embedded totals) for specific hierarchies of the cube's dimensions. Typically, the data for each combination of hierarchies is stored in a separate fact table. Each fact table must have the same columns. Multiple hierarchies in dimensions do not have to share the same lowest level.

  An embedded total key and a grouping ID key (GID) in the fact table map to corresponding columns that identify a dimension hierarchy in a solved dimension table. The ET key identifies the lowest level value present in a row. The GID identifies the hierarchy level associated with each row. For more information, see "Grouping ID Column" on page 1-22. For more information on mapping the key relationships between fact tables and dimension tables, see "MAP_FACTTBL_LEVELKEY Procedure" on page 21-8.

When the fact table and dimension tables are joined with a storage type of `LOWESTLEVEL`, the cube's hierarchies have a `solved_code` of `'UNSOLVED LEVEL-BASED'`.

When the fact tables and dimension tables are joined with a storage type of `ET`, the cube's hierarchies have a `solved_code` of `'SOLVED LEVEL-BASED'`.

See "SET_SOLVED_CODE Procedure" on page 14-7.

Validating and Committing OLAP Catalog Metadata

None of the `CWM2` procedures that create, map, or validate OLAP Catalog metadata includes a `COMMIT`.

Scripts that create OLAP Catalog metadata should first execute all the statements that create and map new metadata, then validate the metadata, then execute a `COMMIT`.

If the metadata will be used by the OLAP API to access data stored in relational tables, you must refresh the OLAP API Metadata Reader tables after validating the metadata. The refresh process includes a `COMMIT`. See "Refreshing Metadata Tables for the OLAP API" on page 2-12.

Validating OLAP Catalog Metadata

To test the validity of OLAP Catalog metadata, use the `CWM2_OLAP_VALIDATE` and `CWM2_OLAP_VERIFY_ACCESS` packages. The validation procedures check the structural integrity of the metadata and ensure that it is correctly mapped to columns in dimension tables and fact tables. Additional validation specific to the OLAP API is done if requested.

The `CWM2_OLAP_VERIFY_ACCESS` package performs two additional checks after validating a cube. It checks that the `CWM2` metadata for the cube is consistent with the cached metadata tables queried by the OLAP API Metadata Reader. Additionally, it checks that the calling user has access to the source tables and columns.

See Also:

- "Refreshing Metadata Tables for the OLAP API" on page 2-12
- Chapter 22, "CWM2_OLAP_VALIDATE"
- Chapter 23, "CWM2_OLAP_VERIFY_ACCESS"

Note: Remember to validate metadata created or updated in Enterprise Manager as well as CWM2 metadata.

When running the validation procedures, you can choose to generate a summary or detailed report of the validation process. See "Directing Output" on page 2-13 for information about viewing output on the screen or writing output to a file.

Example 2–4 shows the statements that validate the PRODUCT dimension in XADEMO and generate a detailed validation report. The report is displayed on the screen and written to a log file.

Example 2–4 Generate a Validation Report for the PRODUCT Dimension

```
set echo on
set linesize 135
set pagesize 50
set serveroutput on size 1000000

execute cwm2_olap_manager.set_echo_on;
execute cwm2_olap_manager.begin_log('/users/myxademo/myscripts' , 'x.log');

execute cwm2_olap_validate.validate_dimension
          ('xademo','product','default','yes');

execute cwm2_olap_manager.end_log;
execute cwm2_olap_manager.set_echo_off;
```

The validation report would look like this.

```
Validate Dimension: XADEMO.PRODUCT    Type of Validation: DEFAULT    Verbose Report: YES
Validating Dimension in OLAP Catalog 1
ENTITY TYPE                   ENTITY NAME               STATUS   COMMENT
  Dimension                   .                         VALID
  Dimension                   XADEMO.PRODUCT            VALID
        LevelAttribute        PROD_STD_TOP_LLABEL       VALID    DimensionAttribute "Long Description"
          LevelAttributeMap                             VALID    Mapped to Column "XADEMO.XADEMO_PRODUCT
                                                                   .PROD_STD_TOP_LLABEL"
        LevelAttribute        PROD_STD_TOP_SLABEL       VALID    DimensionAttribute "Short Description"
          LevelAttributeMap                             VALID    Mapped to Column "XADEMO.XADEMO_PRODUCT
                                                                   .PROD_STD_TOP_SLABEL"
    Hierarchy                 STANDARD                  VALID
      Level                   L4                        VALID    Hierarchy depth 1 (Lowest Level)
        LevelMap                                        VALID    Mapped to Column "XADEMO.XADEMO_PRODUCT
                                                                   .PROD_STD_PRODUCT"
        LevelAttribute        PROD_COLOR                VALID    DimensionAttribute "Color"
          LevelAttributeMap                             VALID    Mapped to Column "XADEMO.XADEMO_PRODUCT
                                                                   .PROD_COLOR"
        LevelAttribute        PROD_SIZE                 VALID    DimensionAttribute "Size"
          LevelAttributeMap                             VALID    Mapped to Column "XADEMO.XADEMO_
                                                                    PRODUCT.PROD_SIZE"
        LevelAttribute        PROD_STD_PRODUCT_LLABEL   VALID    DimensionAttribute "Long Description"
          LevelAttributeMap                             VALID    Mapped to Column "XADEMO.XADEMO_PRODUCT
                                                                   .PROD_STD_PRODUCT_LLABEL"
        LevelAttribute        PROD_STD_PRODUCT_SLABEL   VALID    DimensionAttribute "Short Description"
          LevelAttributeMap                             VALID    Mapped to Column "XADEMO.XADEMO_PRODUCT
                                                                   .PROD_STD_PRODUCT_SLABEL"
      Level                   L3                        VALID    Hierarchy depth 2
        LevelMap                                        VALID    Mapped to Column "XADEMO.XADEMO_PRODUCT
                                                                   .PROD_STD_GROUP"
        LevelAttribute        PROD_STD_GROUP_LLABEL     VALID    DimensionAttribute "Long Description"
          LevelAttributeMap                             VALID    Mapped to Column "XADEMO.XADEMO_PRODUCT
                                                                   .PROD_STD_GROUP_LLABEL"
```

```
  LevelAttribute          PROD_STD_GROUP_SLABEL      VALID    DimensionAttribute "Short Description"
    LevelAttributeMap                                VALID    Mapped to Column "XADEMO.XADEMO_PRODUCT
                                                               .PROD_STD_GROUP_SLABEL"
Level                     L2                         VALID    Hierarchy depth 3
  LevelMap                                           VALID    Mapped to Column "XADEMO.XADEMO_PRODUCT
                                                                .PROD_STD_DIVISION"
  LevelAttribute          PROD_STD_DIVISION_LLABEL   VALID    DimensionAttribute "Long Description"
    LevelAttributeMap                                VALID    Mapped to Column "XADEMO.XADEMO_PRODUCT
                                                                .PROD_STD_DIVISION_LLABEL"
  LevelAttribute          PROD_STD_DIVISION_SLABEL   VALID    DimensionAttribute "Short Description"
    LevelAttributeMap                                VALID    Mapped to Column "XADEMO.XADEMO_PRODUCT
                                                                .PROD_STD_DIVISION_SLABEL"
Level                     L1                         VALID    Hierarchy depth 4 (Top Level)
  LevelMap                                           VALID    Mapped to Column "XADEMO.XADEMO_PRODUCT
                                                                .PROD_STD_TOP"
```

Note: When a metadata entity is invalid, the Comment column of the validation report indicates whether the problem originates with this entity or with a different entity on which it depends. For example, if a level is invalid, its dependent level attributes will also be invalid.

Viewing Validity Status

You can check the validity status of cubes and dimensions by selecting the `INVALID` column of the `ALL_OLAP2_CUBES` and `ALL_OLAP2_DIMENSIONS` views. One of the following values is displayed:

`Y` -- The cube or dimension is invalid.

`N` -- The cube or dimension has met basic validation criteria.

`O` -- The cube has met basic validation criteria and additional criteria specific to the OLAP API.

For more information, see "ALL_OLAP2_CUBES" on page 5-4 and "ALL_OLAP2_DIMENSIONS" on page 5-6.

Refreshing Metadata Tables for the OLAP API

If your metadata will be used by the OLAP API to access relational data, use the `CWM2_OLAP_METADATA_REFRESH` package to refresh the OLAP API Metadata Reader tables.

Views built on these tables present a read API to the OLAP Catalog that is optimized for queries by the OLAP API Metadata Reader. The Metadata Reader views have public synonyms with the prefix `MRV_OLAP2`. For more information, see Chapter 19.

Note: If you use Enterprise Manager to create OLAP Catalog metadata, you must run the refresh procedure separately, after the metadata has been created.

Invoking the Procedures

When using the OLAP Catalog write APIs, you should be aware of logic and conventions that are common to all the `CWM2` procedures.

Security Checks and Error Conditions

Each CWM2 procedure first checks the calling user's security privileges. The calling user must have the OLAP_DBA role. Generally, the calling user must be the entity owner. If the calling user does not meet the security requirements, the procedure fails with an exception. For example, if your identity is jsmith, you cannot successfully execute CWM2_OLAP_HIERARCHY.DROP_HIERARCHY for a hierarchy owned by jjones.

After verifying the security requirements, each procedure checks for the existence of the entity and of its parent entities. All procedures, except CREATE procedures, return an error if the entity does not already exist. For example, if you call CWM2_OLAP_LEVEL.SET_DESCRIPTION, and the level does not already exist, the procedure will fail.

Size Requirements for Parameters

CWM2 metadata entities are created with descriptions and display names. For example, the CREATE_CUBE procedure in the CWM2_OLAP_CUBE package requires the following parameters:

```
CREATE_CUBE (
          cube_owner            IN   VARCHAR2,
          cube_name             IN   VARCHAR2,
          display_name          IN   VARCHAR2,
          short_description     IN   VARCHAR2,
          description           IN   VARCHAR2);
```

Entity names and descriptions have size limitations based on the width of the columns where they are stored in the OLAP Catalog model tables. The size limitations are listed in Table 2–2.

Table 2–2 Size Limitations of CWM2 Metadata Entities

Metadata Entity	Maximum Size
entity owner	30 characters
entity name	30 characters
display name	30 characters
short description	240 characters
description	2000 characters

Case Requirements for Parameters

You can specify arguments to CWM2 procedures in lower case, upper case, or mixed case.

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

Directing Output

The CWM2_OLAP_MANAGER package, described in Chapter 17, provides procedures that direct the output of OLAP stored procedures to the screen or to a file. You can use these procedures to help you develop and debug your scripts for working with OLAP Catalog metadata and analytic workspaces.

Before calling any of the CWM2_OLAP_MANAGER procedures, you must set the serveroutput option in SQL*Plus. This causes SQL*Plus to display the contents of the SQL buffer. For more information, see *SQL*Plus User's Guide and Reference*.

```
>set serveroutput on
```

The default and minimum size of the SQL buffer is 2K. You can extend the size up to a maximum of 1MG. In general, you should set serveroutput to its maximum size to prevent buffer overflow conditions.

```
SQL>set serveroutput on size 1000000
```

To echo the output and messages from OLAP procedures to the SQL buffer, use the following statement.

```
>exec cwm2_olap_manager.set_echo_on;
```

By default, echoing is turned off. Once you have set echoing on, you can turn it off with the following statement.

```
>exec cwm2_olap_manager.set_echo_off;
```

To accommodate larger amounts of output, you should direct output to a file. Use the following statement.

```
SQL>exec cwm2_olap_manager.begin_log('directory_path','filename');
```

For *directory_path* you can specify either a directory object to which your user ID has been granted the appropriate access (in upper-case), or a directory path set by the UTL_FILE_DIR initialization parameter for the instance.

To flush the contents of the buffer and turn off logging, use the following statement.

```
SQL>exec cwm2_olap_manager.end_log;
```

Viewing OLAP Catalog Metadata

A set of views, identified by the ALL_OLAP2 prefix, presents the metadata in the OLAP Catalog. The metadata may have been created with the CWM2 PL/SQL packages, with Enterprise Manager, or with Warehouse Builder. The ALL_OLAP2 views are automatically populated whenever changes are made to the metadata.

A second set of views, identified by the MRV_OLAP prefix, also presents OLAP Catalog metadata. These views are used by the OLAP API to access data stored in relational tables, typically star schemas. These views are structured specifically to support fast querying by the OLAP API Metadata Reader. They must be explicitly refreshed whenever changes are made to the metadata.

See Also:

- Chapter 5, "OLAP Catalog Metadata Views" for more information on the ALL_OLAP2 views.
- Chapter 19, "CWM2_OLAP_METADATA_REFRESH" for more information on refreshing metadata tables for the OLAP API.

3

Active Catalog Views

This chapter describes the relational views of standard form metadata in analytic workspaces. Within the workspace, standard form objects are automatically created and populated by OLAP APIs, such as the DBMS_AWM package and the OLAP Analytic Workspace Java API.

See Also:

- Chapter 1, "Creating Analytic Workspaces with DBMS_AWM"
- *Oracle OLAP Analytic Workspace Java API Reference*

This chapter discusses the following topics:

Understanding the Active Catalog

OLAP processing depends on a data model composed of cubes, measures, dimensions, hierarchies, levels, and attributes. OLAP Catalog metadata defines this logical model for relational sources. Standard form metadata defines the logical model within analytic workspaces.

The DBMS_AWM procedures and the OLAP Analytic Workspace Java API create and maintain standard form metadata when creating and refreshing dimensions and cubes in analytic workspaces.

Whereas OLAP Catalog metadata must be explicitly created by a DBA, standard form metadata is actively generated as part of workspace management. Views of this metadata are commonly referred to as the **Active Catalog**, because they are populated with information that is automatically generated within analytic workspaces.

Active Catalog views use the OLAP_TABLE function to return information about logical objects in analytic workspaces. See Chapter 34 for more information on OLAP_TABLE.

Standard Form Classes

Each standard form workspace object belongs to one of four classes:

- **Implementation class.** Objects in this class implement the logical model.

- **Catalogs class.** Objects in this class hold information about the logical model.
- **Features class.** Objects in this class hold information about specific objects in the logical model.
- **Extensions class.** Objects in this class are proprietary.

Active Catalog and Standard Form Classes

The primary source of information for the Active Catalog views is objects in the Catalogs class. This includes a list of all the cubes, measures, dimensions, levels, and attributes in analytic workspaces.

Active Catalog views also provide information that associates logical objects from the Catalogs class with their source objects in the OLAP Catalog and with their containers in the Implementation class.

Finally, two Active Catalog views provide all the standard form objects and all the properties of those objects.

Note: Active Catalog views provide information about standard form objects in all analytic workspaces accessible to the current user.

See Also:

- *Oracle OLAP Application Developer's Guide* for information about standard form analytic workspaces
- *Oracle OLAP DML Reference* for information about the OLAP DML and the native objects within analytic workspaces

Active Catalog Metadata Cache

The Active Catalog views present information stored within analytic workspaces. This information is also stored for fast access in a separate set of cache tables in the Database.

The Active Catalog views are named with the `ALL_OLAP2_AW` prefix. The views of the cache tables, which have the same column structure, are named with the `MRV_OLAP2_AW` prefix.

Applications that require fast access to the Active Catalog should query the cached metadata in the `MRV_OLAP2_AW` views. You should continue to use `CWM2_OLAP_AC_REFRESH` to ensure that *all* the `MRV_OLAP2_AW` views are synchronized with the `ALL_OLAP2_AW` views.

The metadata cache is *not* automatically refreshed when changes are made to the Active Catalog. To refresh the cache, use the `CWM2_OLAP_METADATA_REFRESH` package, as described in Chapter 19.

The OLAP API queries the Active Catalog cache to obtain the logical model stored in an analytic workspace. You must refresh the cache for OLAP applications that use analytic workspaces.

In the current version of the Active Catalog, the performance of some of the views has been enhanced. These views no longer use the caching mechanism, and the `MRV_OLAP2_AW` views are simply copies of the `ALL_OLAP2_AW` views. However, if you are using an earlier version of standard form, caching is used for all the Active Catalog views.

See Also: "Views of Cached Active Catalog Metadata" on page 19-2.

Example: Query an Analytic Workspace Cube

Example 3–1 uses the XADEMO cube ANALYTIC_CUBE to illustrate two Active Catalog views.

Example 3–1 Query the Active Catalog for Information about a Workspace Cube

The following statements create the dimensions in the analytic workspace XADEMO.MY_AW.

```
execute dbms_awm.create_awdimension
           ('XADEMO','CHANNEL','XADEMO', 'MY_AW', 'AW_CHAN');
execute dbms_awm.create_awdimension
          ('XADEMO','PRODUCT','XADEMO', 'MY_AW', 'AW_PROD');
execute dbms_awm.create_awdimension
          ('XADEMO','GEOGRAPHY','XADEMO', 'MY_AW', 'AW_GEOG');
execute dbms_awm.create_awdimension
          ('XADEMO','TIME','XADEMO', 'MY_AW', 'AW_TIME');
```

You can view the logical dimensions in the analytic workspace with the following query.

```
SQL>select * from ALL_OLAP2_AW_DIMENSIONS;

AW_OWNER  AW_NAME  AW_LOGICAL_NAME  AW_PHYSICAL_OBJECT SOURCE_OWNER  SOURCE_NAME
--------  --------  ---------------  ------------------ ------------  ----------
XADEMO     MY_AW      AW_CHAN       AW_CHAN            XADEMO        CHANNEL
XADEMO     MY_AW      AW_PROD       AW_PROD            XADEMO        PRODUCT
XADEMO     MY_AW      AW_GEOG       AW_GEOG            XADEMO        GEOGRAPHY
XADEMO     MY_AW      AW_TIME       AW_TIME            XADEMO        TIME
```

The following statement creates the cube.

```
execute dbms_awm.create_awcube
          ('XADEMO','ANALYTIC_CUBE','XADEMO', 'MY_AW', 'MY_ANALYTIC_CUBE');
```

You can view the logical cube in the analytic workspace with the following query.

```
SQL>select * from ALL_OLAP2_AW_CUBES;

AW_OWNER AW_NAME AW_LOGICAL_NAME  AW_PHYSICAL_OBJECT  SOURCE_OWNER  SOURCE_NAME
-------- ------- ---------------  ------------------  ------------  -----------
XADEMO   MY_AW   MY_ANALYTIC_CUBE MY_ANALYTIC_CUBE    XADEMO       ANALYTIC_CUBE
```

The following query returns the analytic workspace cube with its associated dimensions.

```
SQL>select * from ALL_OLAP2_AW_CUBE_DIM_USES;

AW_OWNER AW_NAME AW_LOGICAL_NAME  DIMENSION_ DIMENSION_ DIMENSION_   DIMENSION_
                                  AW_OWNER   AW_NAME    SOURCE_OWNER SOURCE_NAME
-------- ------- ---------------- ---------  ---------- ------------ -----------
XADEMO   MY_AW   MY_ANALYTIC_CUBE  XADEMO     AW_CHAN    XADEMO       CHANNEL
XADEMO   MY_AW   MY_ANALYTIC_CUBE  XADEMO     AW_GEOG    XADEMO       GEOGRAPHY
XADEMO   MY_AW   MY_ANALYTIC_CUBE  XADEMO     AW_PROD    XADEMO       PRODUCT
XADEMO   MY_AW   MY_ANALYTIC_CUBE  XADEMO     AW_TIME    XADEMO       TIME
```

Summary of Active Catalog Views

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

Table 3–1 Active Catalog Views

PUBLIC Synonym	Description
ALL_OLAP2_AWS	List of analytic workspaces.
ALL_OLAP2_AW_ATTRIBUTES	List of dimension attributes in analytic workspaces.
ALL_OLAP2_AW_CUBES	List of cubes in analytic workspaces.
ALL_OLAP2_AW_CUBE_AGG_LVL	List of levels in aggregation plans in analytic workspaces.
ALL_OLAP2_AW_CUBE_AGG_MEAS	List of measures in aggregation plans in analytic workspaces.
ALL_OLAP2_AW_CUBE_AGG_OP	List of aggregation operators in aggregation plans in analytic workspaces.
ALL_OLAP2_AW_CUBE_AGG_SPECS	List of aggregation plans in analytic workspaces.
ALL_OLAP2_AW_CUBE_DIM_USES	List of cubes with their associated dimensions in analytic workspaces.
ALL_OLAP2_AW_CUBE_MEASURES	List of cubes with their associated measures in analytic workspaces.
ALL_OLAP2_AW_DIMENSIONS	List of dimensions in analytic workspaces.
ALL_OLAP2_AW_DIM_HIER_LVL_ORD	List of hierarchical levels in analytic workspaces.
ALL_OLAP2_AW_DIM_LEVELS	List of levels in analytic workspaces.
ALL_OLAP2_AW_PHYS_OBJ	List of standard form objects in analytic workspaces.
ALL_OLAP2_AW_PHYS_OBJ_PROP	List of properties associated with standard form objects in analytic workspaces.

ALL_OLAP2_AWS

ALL_OLAP2_AWS provides a list of all the analytic workspaces accessible to the current user. This includes both standard form and non-standard analytic workspaces.

Column	Datatype	NULL	Description
OWNER	VARCHAR2(30)		Owner of the analytic workspace.
AW	VARCHAR2(30)		Name of the analytic workspace.
AW_NUMBER	NUMBER		Unique identifier for the analytic workspace.
AW_VERSION	VARCHAR2(4)		The version of the Database in which the analytic workspace was created. If the version is 10.1 or higher, the workspace is in 10*g* storage format. Earlier versions are in 9*i* format. To upgrade to 10*g* storage format, use the DBMS_AW.CONVERT procedure, as described in "Converting an Analytic Workspace to Oracle 10g Storage Format" on page 24-2.
SF_VERSION	CHAR(8)		The version of the Database in which the standard form metadata was created. To upgrade to 10.1.0.3.1 standard form, use the DBMS_AWM.CREATE_DYNAMIC_AW_ACCESS procedure, as described in "CREATE_DYNAMIC_AW_ACCESS Procedure" on page 26-28.

ALL_OLAP2_AW_ATTRIBUTES

ALL_OLAP2_AW_ATTRIBUTES lists the attributes in standard form analytic workspaces.

The ALL_OLAP2_AW_ATTRIBUTES view uses the direct analytic workspace metadata access process to rapidly return information about standard form metadata. It does not use a caching mechanism and is unaffected by CWM2_OLAP_METADATA_REFRESH. See Chapter 19 for more information.

Column	Datatype	NULL	Description
AW_OWNER	VARCHAR2(30)		Owner of the analytic workspace.
AW_NAME	VARCHAR2(30)		Name of the analytic workspace.
AW_DIMENSION_NAME	VARCHAR2(1000)		Name of the dimension in the analytic workspace.
AW_LOGICAL_NAME	VARCHAR2(90)		Logical name for the attribute in the analytic workspace.
AW_PHYSICAL_OBJECT	VARCHAR2(1000)		Standard form name for the attribute in the analytic workspace.
DISPLAY_NAME	VARCHAR2(1000)		Display name for the attribute.
DESCRIPTION	VARCHAR2(1000)		Description of the attribute.
ATTRIBUTE_TYPE	VARCHAR2(1000)		Type of attribute. See Table 12–1, " Reserved Dimension Attributes".
SOURCE_OWNER	VARCHAR2(1000)		Owner of the source attribute in the OLAP Catalog.
SOURCE_DIMENSION_NAME	VARCHAR2(4000)		Name of the source dimension in the OLAP Catalog.
SOURCE_NAME	VARCHAR2(1000)		Name of the source attribute in the OLAP Catalog.

ALL_OLAP2_AW_CUBES

ALL_OLAP2_AW_CUBES lists the cubes in standard form analytic workspaces.

The ALL_OLAP2_AW_CUBES view uses the direct analytic workspace metadata access process to rapidly return information about standard form metadata. It does not use a caching mechanism and is unaffected by CWM2_OLAP_METADATA_REFRESH. See Chapter 19 for more information.

Column	Datatype	NULL	Description
AW_OWNER	VARCHAR2(30)		Owner of the analytic workspace.
AW_NAME	VARCHAR2(30)		Name of the analytic workspace.
AW_LOGICAL_NAME	VARCHAR2(90)		Logical name for the cube in the analytic workspace.
AW_PHYSICAL_OBJECT	VARCHAR2(1000)		Standard form name for the cube in the analytic workspace.
SOURCE_OWNER	VARCHAR2(1000)		Owner of the source cube in the OLAP Catalog.
SOURCE_NAME	VARCHAR2(1000)		Name of the source cube in the OLAP Catalog.

ALL_OLAP2_AW_CUBE_AGG_LVL

ALL_OLAP2_AW_CUBE_AGG_LVL lists the levels in aggregation specifications in standard form analytic workspaces.

Column	Datatype	NULL	Description
AW_OWNER	VARCHAR2(30)		Owner of the analytic workspace.
AW_NAME	VARCHAR2(30)		Name of the analytic workspace.
AW_CUBE_NAME	VARCHAR2(90)		Name of a cube in the analytic workspace.
AW_AGGSPEC_NAME	VARCHAR2(1000)		Name of an aggregation specification for the cube.
AW_DIMENSION_NAME	VARCHAR2(1000)		Name of a workspace dimension of the cube.
AW_LEVEL_NAME	VARCHAR2(1000)		Name of a workspace level of the dimension. This level is in the aggregation specification.

ALL_OLAP2_AW_CUBE_AGG_MEAS

ALL_OLAP2_AW_CUBE_AGG_MEAS lists the measures in aggregation specifications in standard form analytic workspaces.

Column	Datatype	NULL	Description
AW_OWNER	VARCHAR2(30)		Owner of the analytic workspace.
AW_NAME	VARCHAR2(30)		Name of the analytic workspace.
AW_CUBE_NAME	VARCHAR2(90)		Name of a cube in the analytic workspace.
AW_AGGSPEC_NAME	VARCHAR2(1000)		Name of an aggregation specification for the cube.
AW_MEASURE_NAME	VARCHAR2(1000)		Name of a workspace measure of the cube. This measure is in the aggregation specification

ALL_OLAP2_AW_CUBE_AGG_OP

ALL_OLAP2_AW_CUBE_AGG_OP lists the aggregation operators in aggregation specifications in standard form analytic workspaces.

Column	Datatype	NULL	Description
AW_OWNER	VARCHAR2(30)		Owner of the analytic workspace.
AW_NAME	VARCHAR2(30)		Name of the analytic workspace.
AW_CUBE_NAME	VARCHAR2(90)		Name of a cube in the analytic workspace.
AW_MEASURE_NAME	VARCHAR2		Name of a workspace measure to aggregate.
AW_AGGSPEC_NAME	VARCHAR2(1000)		Name of an aggregation specification for the cube.
AW_DIMENSION_NAME	VARCHAR2(1000)		Name of a workspace dimension of the cube.
OPERATOR	VARCHAR2(1000)		Operator for aggregation along this dimension. See Table 1–10, " Aggregation Operators" for a list of valid operators.

ALL_OLAP2_AW_CUBE_AGG_SPECS

ALL_OLAP2_AW_CUBE_AGG_SPECS lists the aggregation specifications in standard form analytic workspaces.

Column	Datatype	NULL	Description
AW_OWNER	VARCHAR2(30)		Owner of the analytic workspace.

Column	Datatype	NULL	Description
AW_NAME	VARCHAR2(30)		Name of the analytic workspace.
AW_CUBE_NAME	VARCHAR2(90)		Name of the cube in the analytic workspace.
AW_AGGSPEC_NAME	VARCHAR2(1000)		Name of an aggregation plan for the cube.

ALL_OLAP2_AW_CUBE_DIM_USES

ALL_OLAP2_AW_CUBE_DIM_USES lists the dimensions of cubes in standard form analytic workspaces.

Column	Datatype	NULL	Description
AW_OWNER	VARCHAR2(30)		Owner of the analytic workspace.
AW_NAME	VARCHAR2(30)		Name of the analytic workspace.
AW_LOGICAL_NAME	VARCHAR2(90)		Name of a cube in the analytic workspace.
DIMENSION_AW_OWNER	VARCHAR2(1000)		Owner of a workspace dimension of the cube.
DIMENSION_AW_NAME	VARCHAR2(1000)		Name of a workspace dimension of the cube.
DIMENSION_SOURCE_OWNER	VARCHAR2(1000)		Owner of the source dimension in the OLAP Catalog
DIMENSION_SOURCE_NAME	VARCHAR2(1000)		Name of the source dimension in the OLAP Catalog.

ALL_OLAP2_AW_CUBE_MEASURES

ALL_OLAP2_AW_CUBE_MEASURES lists the measures of cubes in standard form analytic workspaces.

The ALL_OLAP2_AW_CUBE_MEASURES view uses the direct analytic workspace metadata access process to rapidly return information about standard form metadata. It does not use a caching mechanism and is unaffected by CWM2_OLAP_METADATA_REFRESH. See Chapter 19 for more information.

Column	Datatype	NULL	Description
AW_OWNER	VARCHAR2(30)		Owner of the analytic workspace.
AW_NAME	VARCHAR2(30)		Name of the analytic workspace.
AW_CUBE_NAME	VARCHAR2(90)		Name of a cube in the analytic workspace.
AW_MEASURE_NAME	VARCHAR2(1000)		Logical name of a measure of the cube.
AW_PHYSICAL_OBJECT	VARCHAR2(4000)		Standard form name of the measure.
MEASURE_SOURCE_NAME	VARCHAR2(1000)		Name of the source measure in the OLAP Catalog.
DISPLAY_NAME	VARCHAR2(1000)		Display name for the measure in the analytic workspace.
DESCRIPTION	VARCHAR2(1000)		Description of the measure in the analytic workspace.
IS_AGGREGATEABLE	VARCHAR2(1000)		Whether or not this measure can be aggregated with the OLAP DML AGGREGATE command. The value is YES if the measure is implemented as an OLAP variable or if its underlying storage is a variable. For example, the measure could be implemented as a formula whose value is stored in a variable.

ALL_OLAP2_AW_DIMENSIONS

ALL_OLAP2_AW_DIMENSIONS lists the dimensions in standard form analytic workspaces.

The ALL_OLAP2_AW_DIMENSIONS view uses the direct analytic workspace metadata access process to rapidly return information about standard form metadata. It does not use a caching mechanism and is unaffected by CWM2_OLAP_METADATA_REFRESH. See Chapter 19 for more information.

Column	Datatype	NULL	Description
AW_OWNER	VARCHAR2(30)		Owner of the analytic workspace.
AW_NAME	VARCHAR2(30)		Name of the analytic workspace.
AW_LOGICAL_NAME	VARCHAR2(90)		Logical name of the dimension in the analytic workspace.
AW_PHYSICAL_OBJECT	VARCHAR2(1000)		Standard form name of the dimension in the analytic workspace.
SOURCE_OWNER	VARCHAR2(1000)		Owner of the source dimension in the OLAP Catalog.
SOURCE_NAME	VARCHAR2(1000)		Name of the source dimension in the OLAP Catalog.

ALL_OLAP2_AW_DIM_HIER_LVL_ORD

ALL_OLAP2_AW_DIM_HIER_LVL_ORD lists the levels in hierarchies in standard form analytic workspaces. It includes the position of each level within the hierarchy.

Column	Datatype	NULL	Description
AW_OWNER	VARCHAR2(30)		Owner of the analytic workspace.
AW_NAME	VARCHAR2(30)		Name of the analytic workspace.
AW_DIMENSION_NAME	VARCHAR2(90)		Name of a dimension in the analytic workspace.
AW_HIERARCHY_NAME	VARCHAR2(1000)		Name of a hierarchy of the workspace dimension.
IS_DEFAULT_HIER	VARCHAR2(1000)		Whether or not this hierarchy is the default hierarchy
AW_LEVEL_NAME	VARCHAR2(1000)		Name of a level of the workspace hierarchy.
POSITION	NUMBER		The position of the level in the hierarchy

ALL_OLAP2_AW_DIM_LEVELS

ALL_OLAP2_AW_DIM_LEVELS lists the levels of dimensions in standard form analytic workspaces.

Column	Datatype	NULL	Description
AW_OWNER	VARCHAR2(30)		Owner of the analytic workspace.
AW_NAME	VARCHAR2(30)		Name of the analytic workspace.
AW_LOGICAL_NAME	VARCHAR2(90)		Name of a dimension in the analytic workspace.
LEVEL_NAME	VARCHAR2(1000)		Name of a workspace level of the dimension.
DISPLAY_NAME	VARCHAR2(1000)		Display name of the level.
DESCRIPTION	VARCHAR2(1000)		Description of the level.

ALL_OLAP2_AW_PHYS_OBJ

ALL_OLAP2_AW_PHYS_OBJ lists the standard form objects in analytic workspaces.

Column	Datatype	NULL	Description
AW_OWNER	VARCHAR2(30)		Owner of the analytic workspace.
AW_NAME	VARCHAR2(30)		Name of the analytic workspace.
AW_OBJECT_NAME	VARCHAR2(90)		Name of the standard form object in the analytic workspace.
AW_OBJECT_TYPE	VARCHAR2(4000)		Type of the standard form object. The type may be any of the native object types that can be defined with the OLAP DML, including: dimensions, relations, variables, formulas, composites, and valuesets.
AW_OBJECT_DATATYPE	VARCHAR2(4000)		Data type of the standard form object. The data type may be any of the native types supported by the OLAP DML, including text, boolean, or integer, or it may be a defined type specific to standard form.

ALL_OLAP2_AW_PHYS_OBJ_PROP

ALL_OLAP2_AW_PHYS_OBJ_PROP lists the standard form objects with their properties.

Column	Datatype	NULL	Description
AW_OWNER	VARCHAR2(30)		Owner of the analytic workspace.
AW_NAME	VARCHAR2(30)		Name of the analytic workspace.
AW_OBJECT_NAME	VARCHAR2(90)		Name of the standard form object in the analytic workspace.
AW_PROP_NAME	VARCHAR2(1000)		Name of a property of the standard form object.
AW_PROP_VALUE	VARCHAR2(1000)		Value of the property.

4

Analytic Workspace Maintenance Views

This chapter describes the views you can query to obtain information about maintaining standard form analytic workspaces with the DBMS_AWM package.

See Also:

- Chapter 1, "Creating Analytic Workspaces with DBMS_AWM"
- Chapter 26, "DBMS_AWM"

This chapter discusses the following topics:

- Building and Maintaining Analytic Workspaces
- Example: Query Load and Enablement Parameters for Workspace Dimensions
- Summary of Analytic Workspace Maintenance Views

Building and Maintaining Analytic Workspaces

The DBMS_AWM package manages the life cycle of standard form analytic workspaces. This includes the creation of workspace cubes from relational sources, data loads, and the enablement of workspace cubes for relational access.

The DBMS_AWM package stores information about workspace builds in the OLAP Catalog. You can query the Analytic Workspace Maintenance views to obtain this information. For example, you could obtain a list of workspace cubes with their relational sources, a list of load specifications, or a list of composite specifications.

The DBMS_AWM package stores information about workspace enablement within the analytic workspace itself. The Analytic Workspace Maintenance views use OLAP_TABLE functions to return information about the enablement of workspace cubes. You can query these views to obtain the names of enablement views and hierarchy combinations.

Example: Query Load and Enablement Parameters for Workspace Dimensions

The following example uses the XADEMO dimensions CHANNEL and TIME to illustrate several Analytic Workspace Maintenance views.

Example 4–1 Query Load Parameters and Enablement View Names for CHANNEL and TIME

The following statements create the dimensions AW_CHAN and AW_TIME in the analytic workspace MY_SCHEMA.MY_AW.

```
execute dbms_awm.create_awdimension
           ('XADEMO','CHANNEL','MY_SCHEMA', 'MY_AW', 'AW_CHAN');
execute dbms_awm.create_awdimension
          ('XADEMO','TIME','MY_SCHEMA', 'MY_AW', 'AW_TIME');
```

The following statements create the load specifications for the dimensions.

```
execute dbms_awm.create_awdimload_spec
          ('CHAN_DIMLOADSPEC', 'XADEMO', 'CHANNEL', 'FULL_LOAD');
execute dbms_awm.add_awdimload_spec_filter
          ('CHAN_DIMLOADSPEC', 'XADEMO', 'CHANNEL', 'XADEMO', 'XADEMO_CHANNEL',
           '''CHAN_STD_CHANNEL'' = ''DIRECT''' );
execute dbms_awm.create_awdimload_spec
          ('TIME_DIMLOADSPEC', 'XADEMO', 'TIME', 'FULL_LOAD');
execute dbms_awm.add_awdimload_spec_filter
          ('TIME_DIMLOADSPEC', 'XADEMO', 'TIME', 'XADEMO', 'XADEMO_TIME',
           '''TIME_STD_YEAR'' = ''1997''' );
```

The following query returns the filter conditions associated with the dimension load specifications.

```
SQL>select * from all_aw_load_dim_filters;
```

```
OWNER     DIMENSION_NAME   LOAD_NAME          TABLE_OWNER      TABLE_NAME       FILTER_CONDITION
--------- ---------------  -----------------  --------------   ---------------  ------------------------------
XADEMO     TIME            TIME_DIMLOADSPEC     XADEMO         XADEMO_TIME      'TIME_STD_YEAR' = '1997'
XADEMO     CHANNEL         CHAN_DIMLOADSPEC     XADEMO         XADEMO_CHANNEL   'CHAN_STD_CHANNEL' = 'DIRECT'
```

The following statements load the dimensions in the analytic workspace. The system-generated names that will be used for the enablement views are created in the workspace as part of the load process.

```
execute dbms_awm.refresh_awdimension
          ('MY_SCHEMA', 'MY_AW', 'AWCHAN', 'CHAN_DIMLOADSPEC');
execute dbms_awm.refresh_awdimension
          ('MY_SCHEMA', 'MY_AW', 'AWTIME', 'TIME_DIMLOADSPEC');
```

The following query returns the system-generated enablement view names for the dimensions.

```
SQL>select * from all_aw_dim_enabled_views;
```

```
AW_OWNER   AW_NAME     DIMENSION_NAME  HIERARCHY_NAME   SYSTEM_VIEWNAME                     USER_VIEWNAME
---------- ---------- --------------  ---------------  ---------------------------------  --------------
MY_SCHEMA    MY_AW     AWCHAN          STANDARD         MY_S_MY_AW_AWCHA_STAND35VIEW
MY_SCHEMA    MY_AW     AWTIME          STANDARD         MY_S_MY_AW_AWTIM_STAND36VIEW
MY_SCHEMA    MY_AW     AWTIME          YTD              MY_S_MY_AW_AWTIM_YTD37VIEW
```

Summary of Analytic Workspace Maintenance Views

The analytic workspace maintenance views are summarized in the following table.

Table 4–1 Analytic Workspace Maintenance Views

Public Synonym	Description
ALL_AW_CUBE_AGG_LEVELS	Describes the levels in aggregation specifications for cubes.
ALL_AW_CUBE_AGG_MEASURES	Describes the measures in aggregation specifications for cubes.
ALL_AW_CUBE_AGG_PLANS	Describes the aggregation specifications for cubes.
ALL_AW_CUBE_ENABLED_HIERCOMBO	Describes the hierarchy combinations associated with cubes.

Table 4–1 (Cont.) Analytic Workspace Maintenance Views

Public Synonym	Description
ALL_AW_CUBE_ENABLED_VIEWS	Describes the fact views that can be generated for workspace cubes.
ALL_AW_DIM_ENABLED_VIEWS	Describes the dimension views that can be generated for workspace dimensions.
ALL_AW_LOAD_CUBES	Describes the load specifications for cubes.
ALL_AW_LOAD_CUBE_DIMS	Describes the composite specifications for cubes.
ALL_AW_LOAD_CUBE_FILTERS	Describes the filter conditions associated with load specifications for cubes.
ALL_AW_LOAD_CUBE_MEASURES	Describes the measures in cube load specifications.
ALL_AW_LOAD_CUBE_PARMS	Describes parameters of cube load specifications.
ALL_AW_LOAD_DIMENSIONS	Describes the load specifications for dimensions.
ALL_AW_LOAD_DIM_FILTERS	Describes the filter conditions associated with load specifications for dimensions.
ALL_AW_LOAD_DIM_PARMS	Describes parameters of dimension load specifications.
ALL_AW_OBJ	Lists the objects in all analytic workspaces available to the current user. The workspaces may have been created by DBMS_AWM or by another tool, such as the OLAP Analytic Workspace API.
ALL_AW_PROP	Lists the OLAP DML properties and their values in all analytic workspaces available to the current user. The workspaces may have been created by DBMS_AWM or by another tool, such as the OLAP Analytic Workspace API.

ALL_AW_CUBE_AGG_LEVELS

ALL_AW_CUBE_AGG_LEVELS lists the levels in aggregation specifications for cubes.

Aggregation specifications determine how data will be aggregated along the dimensions of a cube in an analytic workspace. Aggregation specifications are created by the DBMS_AWM.CREATE_AWCUBEAGG_SPEC procedure.

Column	Datatype	NULL	Description
owner	varchar2(240)		Owner of the cube.
cube_name	varchar2(240)		Name of the cube.
aggregation_name	varchar2(60)		Name of the aggregation spec.
dimension_owner	varchar2(30)		Owner of the dimension to aggregate.
dimension_name	varchar2(240)		Name of the dimension to aggregate.
level_name	archar2(240)		Name of the level of aggregation for this dimension.

ALL_AW_CUBE_AGG_MEASURES

ALL_AW_CUBE_AGG_MEASURES lists the measures in aggregation specifications for cubes.

Aggregation specifications determine how the measures will be aggregated along the dimensions of a cube in an analytic workspace. Aggregation specifications are created by the DBMS_AWM.CREATE_AWCUBEAGG_SPEC procedure.

Column	Datatype	NULL	Description
cube_owner	varchar2(240)		Owner of the cube.
cube_name	varchar2(240)		Name of the cube.
aggregation_name	varchar2(60)		Name of the aggregation spec.
measure_name	varchar2(240)		Name of the measure to aggregate.

ALL_AW_CUBE_AGG_PLANS

ALL_AW_CUBE_AGG_PLANS lists the aggregation specifications for cubes.

Aggregation specifications determine how data will be aggregated along the dimensions of a cube in an analytic workspace. Aggregation specifications are created by the DBMS_AWM.CREATE_AWCUBEAGG_SPEC procedure.

Column	Datatype	NULL	Description
owner	varchar2(240)		Owner of the cube.
cube_name	varchar2(240)		Name of the cube.
aggregation_name	varchar2(60)		Name of the aggregation spec.

ALL_AW_CUBE_ENABLED_HIERCOMBO

ALL_AW_CUBE_ENABLED_HIERCOMBO lists the hierarchy combinations associated with cubes in analytic workspaces.

Each hierarchy combination is identified by a unique number. The OLAP API Enabler creates a separate fact view for each hierarchy combination.

The information in this view is available for all standard form cubes that have been refreshed. See the DBMS_AWM.REFRESH_AWCUBE procedure and the DBMS_AWM.CREATE_AWCUBE_ACCESS procedure.

Column	Datatype	NULL	Description
aw_owner	varchar2(30)		Owner of the analytic workspace.
aw_name	varchar2(30)		Name of the analytic workspace.
cube_name	varchar2(4000)		Name of the cube in the analytic workspace.
hiercombo_num	number		Unique number that identifies the hierarchy combination.
hiercombo_str	varchar2(4000)		List of hierarchies that define the dimensionality of a fact view of the enabled cube.

ALL_AW_CUBE_ENABLED_VIEWS

ALL_AW_CUBE_ENABLED_VIEWS describes the fact views that can be generated for cubes in analytic workspaces.

Descriptions of the views are created when the cube is refreshed. The view is not instantiated until the DBMS_AWM.CREATE_AWCUBE_ACCESS has executed and the resulting script has been run.

ALL_AW_CUBE_ENABLED_VIEWS shows the descriptions of the views. The views themselves do not necessarily exist.

Metadata about fact views is generated by the DBMS_AWM.REFRESH_AWCUBE procedure. Scripts to create views of workspace cubes are created by the DBMS_AWM.CREATE_AWCUBE_ACCESS procedure.

Column	Datatype	NULL	Description
aw_owner	varchar2(30)		Owner of the analytic workspace.
aw_name	varchar2(30)		Name of the analytic workspace.
cube_name	varchar2(4000)		Name of the cube in the analytic workspace.
hiercombo_num	number		Unique number that identifies the hierarchy combination.
hiercombo_str	varchar2(4000)		List of hierarchies that define the dimensionality of a fact view of the enabled cube.
system_viewname	varchar2(4000)		Default view name created by the DBMS_AWM.REFRESH_AWCUBE procedure.
user_viewname	varchar2(4000)		User-defined view name specified by the DBMS_AWM.SET_AWCUBE_VIEWNAME procedure.

ALL_AW_DIM_ENABLED_VIEWS

ALL_AW_DIM_ENABLED_VIEWS describes the dimension views that can be generated for dimensions in analytic workspaces.

Descriptions of the views are created when the dimension is refreshed. The view is not instantiated until the DBMS_AWM.CREATE_AWDIMENSION_ACCESS has executed and the resulting script has been run.

ALL_AW_DIM_ENABLED_VIEWS shows the descriptions of the views. The views themselves do not necessarily exist.

Metadata about dimension views is generated by the DBMS_AWM.REFRESH_AWDIMENSION procedure. Scripts to create views of workspace dimensions are created by the DBMS_AWM.CREATE_AWDIMENSION_ACCESS procedure.

Column	Datatype	NULL	Description
aw_owner	varchar2(30)		Owner of the analytic workspace.
aw_name	varchar2(30)		Name of the analytic workspace.
dimension_name	varchar2(4000)		Name of the dimension in the analytic workspace.
hierarchy_name	varchar2(4000)		Name of the hierarchy in the analytic workspace.
system_viewname	varchar2(4000)		Default view name created by the DBMS_AWM.REFRESH_AWCUBE procedure.
user_viewname	varchar2(4000)		User-defined view name specified by the DBMS_AWM.SET_AWDIMENSION_VIEWNAME procedure.

ALL_AW_LOAD_CUBES

ALL_AW_LOAD_CUBES lists the load specifications for cubes.

Load specifications determine how data will be loaded from the source fact table into the analytic workspace. Cube load specifications are created by the DBMS_AWM.CREATE_AWCUBELOAD_SPEC procedure.

Column	Datatype	NULL	Description
cube_owner	varchar2(240)		Owner of the OLAP Catalog source cube.
cube_name	varchar2(240)		Name of the OLAP Catalog source cube.
load_name	varchar2(60)		Name of a load specification for the cube.
load_type	varchar2(60)		'LOAD_DATA' -- Load the data and metadata for an OLAP Catalog cube into the analytic workspace target cube. 'LOAD_PROGRAM' -- This argument is no longer used.

ALL_AW_LOAD_CUBE_DIMS

ALL_AW_LOAD_CUBE_DIMS describes the composite specifications for cubes.

Composite specifications determines how the cube's dimensions will be optimized in the analytic workspace. Composite specifications are created by the DBMS_AWM.CREATE_AWCOMP_SPEC procedure.

Column	Datatype	NULL	Description
cube_owner	varchar2(240)		Owner of the OLAP Catalog source cube.
cube_name	varchar2(240)		Name of the OLAP Catalog source cube.
cubeload_name	varchar2(60)		Name of a load specification for the cube.
compspec_name	varchar2(30)		Name of a composite specification associated with this load specification.
composite_name	varchar2(30)		Name of a composite that is a member of the specification. A composite contains sparse dimensions of the cube.
segwidth	number		Segment width for storage of the data dimensioned by this member of the specification.
compspec_position	number		Position of the member within the specification.
dimension_owner	varchar2(30)		Owner of an OLAP Catalog source dimension that is a member of the specification.
dimension_name	varchar2(240)		Name of the OLAP Catalog source dimension that is a member of the specification.
composite_position	number		Position of the member within a composite member.
nested_level	number		The level of nesting of the member of the specification. For example, a dense dimension would have a nesting level of 1. A sparse dimension within a composite would have a nesting level of 2, and a nested composite would have a nesting level of 3.
nested_type	varchar2(10)		Type of member of the specification. Either DIMENSION or COMPOSITE.
nested_name	varchar2(30)		Name of the member of the specification. This may be the name of a dimension or the name of a composite.

ALL_AW_LOAD_CUBE_FILTERS

ALL_AW_LOAD_CUBE_FILTS lists the filter conditions associated with load specifications for cubes.

Filter conditions are SQL WHERE clauses that identify a subset of the data to be loaded from the fact table to the analytic workspace.

Filter conditions are created by the DBMS_AWM.ADD_AWCUBELOAD_SPEC_FILTER procedure.

Column	Datatype	NULL	Description
owner	varchar2(240)		Owner of the OLAP Catalog source cube.
cube_name	varchar2(240)		Name of the OLAP Catalog source cube.
load_name	varchar2(60)		Name of a load specification for the cube.
table_owner	varchar2(30)		Owner of the fact table.
table_name	varchar2(30)		Name of the fact table.
filter_condition	varchar2(4000)		SQL WHERE clause.

ALL_AW_LOAD_CUBE_MEASURES

ALL_AW_LOAD_CUBE_MEASURES lists the measures in cube load specifications with their corresponding target measures in standard form analytic workspaces.

Measures are added to cube load specifications by the DBMS_AWM.ADD_AWCUBELOAD_SPEC_MEASURE procedure. This procedure enables you to specify a target name and display name for the measure in the analytic workspace. If you do not call this procedure, or if you do not specify the target names, the OLAP Catalog names are used.

Column	Datatype	NULL	Description
owner	varchar2(240)		Owner of the source cube in the OLAP Catalog.
cube_name	varchar2(240)		Name of the source cube in the OLAP Catalog.
load_name	varchar2(60)		Name of the load specification for the source cube.
measure_name	varchar2(240)		Name of a measure of the source cube.
measure_target_name	varchar2(60)		Name of the measure in the analytic workspace.
measure_target_display_name	varchar2(60)		Display name of the measure in the analytic workspace. This may be the display name from the OLAP Catalog, or it may be user-defined.
measure_target_description	varchar2(4000)		Description of the measure in the analytic workspace. This may be the description from the OLAP Catalog, or it may be user-defined.

ALL_AW_LOAD_CUBE_PARMS

ALL_AW_LOAD_CUBE_PARMS lists the parameters in cube load specifications.

Cube load specifications determine how a cube's data will be loaded from the fact table into the analytic workspace.

Parameters are set for cube load specifications by the DBMS_AWM.SET_AWCUBELOAD_SPEC_PARAMETER procedure.

Column	Datatype	NULL	Description
owner	varchar2(240)		Owner of the source cube in the OLAP Catalog.
cube_name	varchar2(240)		Name of the source cube in the OLAP Catalog.
load_name	varchar2(60)		Name of the load specification for the source cube.
parm_name	varchar2(16)		The name of the parameter. Currently only 'DISPLAY NAME' is available. If you do not set this parameter, the cube display name from the OLAP Catalog is used in the analytic workspace.
parm_value	varchar2(4000)		The display name to use for the target cube in the analytic workspace.

ALL_AW_LOAD_DIMENSIONS

ALL_AW_LOAD_DIMENSIONS lists the load specifications for dimensions.

Dimension load specifications are created by the DBMS_AWM.CREATE_AWDIMLOAD_SPEC procedure.

Column	Datatype	NULL	Description
owner	varchar2(30)		Owner of the source dimension in the OLAP Catalog.
dimension_name	varchar2(30)		Name of the source dimension in the OLAP Catalog.
load_name	varchar2(60)		Name of the load specification.
load_type	varchar2(60)		'FULL_LOAD_ADDITIONS_ONLY' -- Only new dimension members will be loaded when the dimension is refreshed. (Default) 'FULL_LOAD' -- When the dimension is refreshed, all dimension members in the workspace will be deleted, then all the members of the source dimension will be loaded.

ALL_AW_LOAD_DIM_FILTERS

ALL_AW_LOAD_DIM_FILTERS lists the filter conditions associated with load specifications for dimensions.

Filter conditions are SQL WHERE clauses that identify a subset of the data to be loaded from the dimension table to the analytic workspace.

Filter conditions are created by the DBMS_AWM.ADD_AWDIMLOAD_SPEC_FILTER procedure.

Column	Datatype	NULL	Description
owner	varchar2(30)		Owner of the source dimension in the OLAP Catalog.
dimension_name	varchar2(30)		Name of the source dimension in the OLAP Catalog.
load_name	varchar2(60)		Name of the dimension load specification.
table_owner	varchar2(30)		Owner of the dimension table.
table_name	varchar2(30)		Name of the dimension table.
filter_condition	varchar2(4000)		SQL WHERE clause.

ALL_AW_LOAD_DIM_PARMS

ALL_AW_LOAD_DIM_PARMS lists the parameters in dimension load specifications.

Dimension load specifications determine how dimension members will be loaded from the dimension table into the analytic workspace.

Parameters are set for dimension load specifications by the DBMS_AWM.SET_AWDIMLOAD_SPEC_PARAMETER procedure.

Column	Datatype	NULL	Description
owner	varchar2(30)		Owner of the source dimension in the OLAP Catalog.
dimension_name	varchar2(30)		Name of the source dimension in the OLAP Catalog.
load_name	varchar2(60)		Name of the dimension load specification.
parm_name	varchar2(16)		'UNIQUE_RDBMS_KEY' -- Whether or not the members of this dimension are unique across all levels in the source tables. 'DISPLAY_NAME' -- Display name for the target dimension in the analytic workspace. 'PLURAL_DISPLAY_NAME' -- Plural display name for the target dimension in the analytic workspace.
parm_value	varchar2(4000)		Values of UNIQUE_RDBMS_KEY: NO-- Dimension member names are not unique across levels in the RDBMS tables. The corresponding dimension member names in the analytic workspace include the level name as a prefix. (Default) YES -- Dimension member names are unique across levels in the RDBMS tables. The corresponding dimension member names in the analytic workspace have the same names as in the source relational dimension. Value of DISPLAY_NAME is the display name for the target dimension in the analytic workspace. Value of PLURAL_DISPLAY_NAME is the plural display name for the target dimension in the analytic workspace.

ALL_AW_OBJ

ALL_AW_OBJ lists the current objects in all analytic workspaces that are accessible to the user. The workspaces may have been created by DBMS_AWM or by another tool, such as the OLAP Analytic Workspace API.

Column	Datatype	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	User name of the analytic workspace owner
AW_NUMBER	NUMBER	NOT NULL	Unique identifier within the database for the analytic workspace
AW_NAME	VARCHAR2(30)		Name of the analytic workspace
OBJ_ID	NUMBER(20)		Unique identifier for the object within the analytic workspace
OBJ_NAME	VARCHAR2(256)		Name of the object
OBJ_TYPE	NUMBER(4)		Data type of the object

Column	Datatype	NULL	Description
PART_NAME	VARCHAR2(256)		Name of the partition for the object
FULL_PROPERTY_VALUE			

ALL_AW_PROP

ALL_AW_PROP lists the current OLAP DML properties and their values in all analytic workspaces that are accessible to the user. The workspaces may have been created by DBMS_AWM or by another tool, such as the OLAP Analytic Workspace API.

Column	Datatype	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	User name of the analytic workspace owner
AW_NUMBER	NUMBER	NOT NULL	Unique identifier within the database for the analytic workspace
AW_NAME	VARCHAR2(30)		Name of the analytic workspace
OBJ_ID	NUMBER(20)		Unique identifier for the object within the analytic workspace
OBJ_NAME	VARCHAR2(256)		Name of the object
PROPERTY_NAME	VARCHAR2(256)		Name of the property
PROPERTY_TYPE	VARCHAR2(4000)		Data type of the property value
PROPERTY_VALUE	VARCHAR2(4000)		Value of the property
FULL_PROPERTY_VALUE	CLOB		Full value of the property
PROPERTY_VALUE_LENGTH	NUMBER		Length of the property value

5

OLAP Catalog Metadata Views

This chapter describes the OLAP Catalog metadata views. All OLAP Catalog metadata, whether created with the CWM2 PL/SQL packages, with Enterprise Manager, or with Warehouse Builder, is presented in these views.

> **See Also:** Chapter 2, "Creating OLAP Catalog Metadata with CWM2".

This chapter discusses the following topics:

- Access to OLAP Catalog Views
- OLAP Catalog Metadata Cache
- Views of the Dimensional Model
- Views of Mapping Information

Access to OLAP Catalog Views

The OLAP Catalog read API consists of two sets of corresponding views:

- ALL_ views displaying all valid OLAP Catalog metadata accessible to the current user.
- DBA_ views displaying all OLAP Catalog 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 Catalog metadata in these tables, the user must have the OLAP_DBA role.

The columns of the ALL_ and DBA_ views are identical. Only the ALL_ views are listed in this chapter.

OLAP Catalog Metadata Cache

The OLAP Catalog views present information stored in the base tables of the OLAP Catalog. This information is also stored for fast access in a separate set of cache tables.

The OLAP Catalog views are named with the ALL_OLAP2 or DBA_OLAP2 prefix. The views of the cache tables, which have the same column structure, are named with the MRV_OLAP2 prefix.

Applications that require fast access to OLAP Catalog metadata should query the cached metadata in the MRV_OLAP2 views.

The metadata cache is *not* automatically refreshed when changes are made to the base metadata tables. To refresh the cache, use the CWM2_OLAP_METADATA_REFRESH package.

Note: If your data is stored in relational tables (not in analytic workspaces), you must refresh the OLAP Catalog metadata cache for applications that use the OLAP API.

See Also:

- Chapter 19, "CWM2_OLAP_METADATA_REFRESH"
- "Validating and Committing OLAP Catalog Metadata" on page 2-10

Views of the Dimensional Model

The following views show the basic dimensional model of OLAP Catalog metadata.

For more information on the logical model, see the *Oracle OLAP Application Developer's Guide*.

Table 5–1 OLAP Catalog Dimensional Model Views

View Name Synonym	Description
ALL_OLAP2_AGGREGATION_USES	Lists the aggregation operators that can be used in relational cubes based on star or snowflake schemas.
ALL_OLAP2_CATALOGS	List all measure folders (catalogs) within the Oracle instance.
ALL_OLAP2_CATALOG_ENTITY_USES	Lists the measures within each measure folder.
ALL_OLAP2_CUBES	Lists all cubes in an Oracle instance.
ALL_OLAP2_CUBE_DIM_USES	Lists the dimensions within each cube.
ALL_OLAP2_CUBE_MEASURES	Lists the measures within each cube.
ALL_OLAP2_CUBE_MEAS_DIM_USES	Shows how each measure is aggregated along each of its dimensions.
ALL_OLAP2_DIMENSIONS	Lists all OLAP dimensions in an Oracle instance.
ALL_OLAP2_DIM_ATTRIBUTES	Lists the dimension attributes within each dimension.
ALL_OLAP2_DIM_ATTR_USES	Shows how level attributes are associated with each dimension attribute.
ALL_OLAP2_DIM_HIERARCHIES	Lists the hierarchies within each dimension.
ALL_OLAP2_DIM_HIER_LEVEL_USES	Show how levels are ordered within each hierarchy.
ALL_OLAP2_DIM_LEVELS	Lists the levels within each dimension.
ALL_OLAP2_DIM_LEVEL_ATTRIBUTES	Lists the level attributes within each level.

Table 5–1 (Cont.) OLAP Catalog Dimensional Model Views

View Name Synonym	Description
ALL_OLAP2_ENTITY_DESC_USES	Lists the reserved attributes that have application-specific meanings. Examples are dimension attributes that are used for long and short descriptions and time-series calculations (end date, time span, period ago, and so on).
ALL_OLAP2_ENTITY_EXT_PARMS	Lists the metadata descriptors.
ALL_OLAP2_ENTITY_PARAMETERS	Lists the parameters for the metadata descriptors.

Views of Mapping Information

The following views show how the basic dimensional model is mapped to relational tables or views.

Table 5–2 OLAP Catalog Mapping Views

View Synonym Name	Description
ALL_OLAP2_CUBE_MEASURE_MAPS	Shows the mapping of each measure to a column.
ALL_OLAP2_DIM_LEVEL_ATTR_MAPS	Shows the mapping of each level attribute to a column.
ALL_OLAP2_FACT_LEVEL_USES	Shows the joins between dimension tables and fact tables in a star or snowflake schema.
ALL_OLAP2_FACT_TABLE_GID	Shows the Grouping ID column for each hierarchy in each fact table.
ALL_OLAP2_HIER_CUSTOM_SORT	Shows the default sort order for level columns within hierarchies.
ALL_OLAP2_JOIN_KEY_COLUMN_USES	Shows the joins between two levels in a hierarchy.
ALL_OLAP2_LEVEL_KEY_COL_USES	Shows the mapping of each level to a unique key column.

ALL_OLAP2_AGGREGATION_USES

`ALL_OLAP2_AGGREGATION_USES` lists the aggregation operators associated with cubes that map to relational tables organized as star or snowflake schemas.

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_NAME`	`VARCHAR2(30)`	`NOT NULL`	Name of the dimensions of the cube.
`HIERARCHY_NAME`	`VARCHAR2(30)`		Name of the hierarchies of the cube's dimensions.
`DIM_HIER_COMBO_ID`	`NUMBER`	`NOT NULL`	Identifier of a hierarchy combination within the cube.
`AGGREGATION_NAME`	`VARCHAR2(240)`		Name of the aggregation operator for this dimension. (See Table 1–10, " Aggregation Operators" on page 1-16.)
`AGGREGATION_ORDER`	`NUMBER`		The order of precedence of the aggregation operator.
`TABLE_OWNER`	`VARCHAR2(30)`		Owner of the table that contains the weightby factors for weighted operators. If the operator is not weighted, this column is null.

Column	Data Type	NULL	Description
TABLE_NAME	VARCHAR2(30)		Name of the table that contains the weightby factors for weighted operators. If the operator is not weighted, this column is null.
COLUMN_NAME	VARCHAR2(30)		Name of the column that contains the weightby factors for weighted operators. If the operator is not weighted, this column is null.

ALL_OLAP2_CATALOGS

ALL_OLAP2_CATALOGS lists all the measure folders (catalogs) within the Oracle instance.

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

ALL_OLAP2_CATALOG_ENTITY_USES

ALL_OLAP2_CATALOG_ENTITY_USES lists the measures within each measure folder.

Column	Data Type	NULL	Description
CATALOG_ID	NUMBER	NOT NULL	ID of the measure folder.
ENTITY_OWNER	VARCHAR2(30)	NOT NULL	Owner of the measure's cube.
ENTITY_NAME	VARCHAR2(30)	NOT NULL	Name of the measure's cube.
CHILD_ENTITY_NAME	VARCHAR2(30)	NOT NULL	Name of the measure in the measure folder.

ALL_OLAP2_CUBES

ALL_OLAP2_CUBES lists all cubes in an Oracle instance .

Column	Data Type	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the cube that contains the measure.
CUBE_NAME	VARCHAR2(30)	NOT NULL	Name of the cube that contains the measure.
INVALID	VARCHAR2(2)	NOT NULL	Whether or not this cube is in an invalid state. See "Validating and Committing OLAP Catalog Metadata" on page 2-10.
DISPLAY_NAME	VARCHAR2(30)		Display name for the cube.
DESCRIPTION	VARCHAR2(2000)		Description of the cube.
MV_SUMMARYCODE	VARCHAR2(2)		If this cube has an associated materialized view, the MV summary code specifies whether it is in Grouping Set (**groupingset**) or Rolled Up (**rollup**) form. See Chapter 27, "DBMS_ODM".

ALL_OLAP2_CUBE_DIM_USES

ALL_OLAP2_CUBE_DIM_USES lists the dimensions within each cube.

A dimension may be associated more than once with the same cube, but each association is specified in a separate row, under its own unique dimension alias.

Column	Data Type	NULL	Description
CUBE_DIMENSION_USE_ID	NUMBER	NOT NULL	ID of the association between a cube and a dimension.
OWNER	VARCHAR2(30)	NOT NULL	Owner of the cube.
CUBE_NAME	VARCHAR2(30)	NOT NULL	Name of the cube.
DIMENSION_OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension.
DIMENSION_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension.
DIMENSION_ALIAS	VARCHAR2(30)		Alias of the dimension, to provide unique identity of dimension use within the cube.
DEFAULT_CALC_HIERARCHY_NAME	VARCHAR2(30)		The default hierarchy to be used for drilling up or down within the dimension.
DEPENDENT_ON_DIM_USE_ID	NUMBER		ID of the cube/dimension association on which this cube/dimension association depends.

ALL_OLAP2_CUBE_MEASURES

ALL_OLAP2_CUBE_MEASURES lists the measures within each cube .

Column	Data Type	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the cube that contains the measure.
CUBE_NAME	VARCHAR2(30)	NOT NULL	Name of the cube that contains the measure.
MEASURE_NAME	VARCHAR2(30)	NOT NULL	Name of the measure.
DISPLAY_NAME	VARCHAR2(30)		Display name for the measure.
DESCRIPTION	VARCHAR2(2000)		Description of the measure.

ALL_OLAP2_CUBE_MEASURE_MAPS

ALL_OLAP2_CUBE_MEASURE_MAPS shows the mapping of each measure to a column.

Column	Data Type	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the cube.
CUBE_NAME	VARCHAR2(30)	NOT NULL	Name of the cube.
MEASURE_NAME	VARCHAR2(30)	NOT NULL	Name of the measure contained in this cube.
DIM_HIER_COMBO_ID	NUMBER	NOT NULL	ID of the association between this measure and one combination of its dimension hierarchies.
FACT_TABLE_OWNER	VARCHAR2(30)	NOT NULL	Owner of the fact table.
FACT_TABLE_NAME	VARCHAR2(30)	NOT NULL	Name of the fact table.
COLUMN_NAME	VARCHAR2(30)	NOT NULL	Name of the column in the fact table where this measure's data is stored.

ALL_OLAP2_CUBE_MEAS_DIM_USES

ALL_OLAP2_CUBE_MEAS_DIM_USES shows how each measure is aggregated along each of its dimensions. The default aggregation method is addition.

Column	Data Type	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the cube that contains this measure.
CUBE_NAME	VARCHAR2(30)	NOT NULL	Name of the cube that contain this measure.
MEASURE_NAME	VARCHAR2(30)	NOT NULL	Name of the measure.
DIMENSION_OWNER	VARCHAR2(30)	NOT NULL	Owner of a dimension associated with this measure.
DIMENSION_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension.
DIMENSION_ALIAS	VARCHAR2(30)		Alias of the dimension.
DEFAULT_AGGR_FUNCTION_USE_ID	NUMBER		The default aggregation method used to aggregate this measure's data over this dimension. If this column is null, the aggregation method is addition.

ALL_OLAP2_DIMENSIONS

ALL_OLAP2_DIMENSIONS lists all the OLAP dimensions in the Oracle instance.

OLAP dimensions created with the CWM2 APIs have no association with database dimension objects. OLAP dimensions created in Enterprise Manager are based on database dimension objects.

Column	Data Type	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension.
DIMENSION_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension.
PLURAL_NAME	VARCHAR2(30)		Plural name for the dimension. Used for display.
DISPLAY_NAME	VARCHAR2(30)		Display name for the dimension.
DESCRIPTION	VARCHAR2(2000)		Description of the dimension.
DEFAULT_DISPLAY_HIERARCHY	VARCHAR2(30)	NOT NULL	Default display hierarchy for the dimension.
INVALID	VARCHAR2(1)	NOT NULL	Whether or not the dimension is valid. See "Validating and Committing OLAP Catalog Metadata" on page 2-10
DIMENSION_TYPE	VARCHAR2(10)		Not used.

ALL_OLAP2_DIM_ATTRIBUTES

ALL_OLAP2_DIM_ATTRIBUTES lists the dimension attributes within each dimension.

Column	Data Type	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension.
DIMENSION_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension.
ATTRIBUTE_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension attribute.
DISPLAY_NAME	VARCHAR2(30)		Display name for the dimension attribute.
DESCRIPTION	VARCHAR2(2000)		Description of the dimension attribute.

Column	Data Type	NULL	Description
DESC_ID	NUMBER		If the attribute is reserved, its type is listed in this column. Examples of reserved dimension attributes are long and short descriptions and time-related attributes, such as end date, time span, and period ago.

ALL_OLAP2_DIM_ATTR_USES

ALL_OLAP2_DIM_ATTR_USES shows how level attributes are associated with each dimension attribute.

The same level attribute can be included in more than one dimension attribute.

Column	Data Type	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension.
DIMENSION_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension.
DIM_ATTRIBUTE_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension attribute.
LEVEL_NAME	VARCHAR2(30)	NOT NULL	Name of a level within the dimension.
LVL_ATTRIBUTE_NAME	VARCHAR2(30)	NOT NULL	Name of an attribute for this level. This level attribute is included in the dimension attribute.

ALL_OLAP2_DIM_HIERARCHIES

ALL_OLAP2_DIM_HIERARCHIES lists the hierarchies within each dimension.

Column	Data Type	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension.
DIMENSION_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension.
HIERARCHY_NAME	VARCHAR2(30)	NOT NULL	Name of the hierarchy.
DISPLAY_NAME	VARCHAR2(30)		Display name for the hierarchy.
DESCRIPTION	VARCHAR2(2000)		Description of the hierarchy.
SOLVED_CODE	VARCHAR2(2)	NOT NULL	The solved code may be one of the following: UNSOLVED LEVEL-BASED, for a hierarchy that contains no embedded totals and is stored in a level-based dimension table. SOLVED LEVEL-BASED, for a hierarchy that contains embedded totals, has a grouping ID, and is stored in a level-based dimension table. SOLVED VALUE-BASED, for a hierarchy that contains embedded totals and is stored in a parent-child dimension table. For information about mapping hierarchies with different solved codes, see "Joining Fact Tables with Dimension Tables" on page 2-9.

ALL_OLAP2_DIM_HIER_LEVEL_USES

ALL_OLAP2_DIM_HIER_LEVEL_USES shows how levels are ordered within each hierarchy.

Within separate hierarchies, the same parent level may be hierarchically related to a different child level.

Column	Data Type	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension.
DIMENSION_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension.
HIERARCHY_NAME	VARCHAR2(30)	NOT NULL	Name of the hierarchy.
PARENT_LEVEL_NAME	VARCHAR2(30)	NOT NULL	Name of the parent level.
CHILD_LEVEL_NAME	VARCHAR2(30)	NOT NULL	Name of the child level.
POSITION	NUMBER	NOT NULL	Position of this parent-child relationship within the hierarchy, with position 1 being the most detailed.

ALL_OLAP2_DIM_LEVELS

ALL_OLAP2_DIM_LEVELS lists the levels within each dimension.

Column	Data Type	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension containing this level.
DIMENSION_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension containing this level.
LEVEL_NAME	VARCHAR2(30)	NOT NULL	Name of the level.
DISPLAY_NAME	VARCHAR2(30)		Display name for the level.
DESCRIPTION	VARCHAR2(2000)		Description of the level.
LEVEL_TABLE_OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension table that contains the columns for this level.
LEVEL_TABLE_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension table that contains the columns for this level.

ALL_OLAP2_DIM_LEVEL_ATTRIBUTES

ALL_OLAP2_DIM_LEVEL_ATTRIBUTES lists the level attributes within each level.

Column	Data Type	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension containing the level.
DIMENSION_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension containing the level.
ATTRIBUTE_NAME	VARCHAR2(30)		Name of the level attribute. If no attribute name is specified, the column name is used.
DISPLAY_NAME	VARCHAR2(30)		Display name for the level attribute.
DESCRIPTION	VARCHAR2(2000)		Description of the level attribute.
DETERMINED_BY_ LEVEL_NAME	VARCHAR2(30)	NOT NULL	Name of the level.

ALL_OLAP2_DIM_LEVEL_ATTR_MAPS

ALL_OLAP2_DIM_LEVEL_ATTR_MAPS shows the mapping of each level attribute to a column.

The mapping of level attributes to levels is dependent on hierarchy. The same level may have different attributes when it is used in different hierarchies.

Column	Data Type	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension.
DIMENSION_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension.
HIERARCHY_NAME	VARCHAR2(30)		Name of the hierarchy containing this level.
ATTRIBUTE_NAME	VARCHAR2(30)		Name of a dimension attribute grouping containing this level attribute.
LVL_ATTRIBUTE_NAME	VARCHAR2(30)	NOT NULL	Name of the level attribute, or name of the column if the level attribute name is not specified.
LEVEL_NAME	VARCHAR2(30)	NOT NULL	Name of the level.
TABLE_OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension table containing the level and level attribute.
TABLE_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension table containing the level and level attribute columns.
COLUMN_NAME	VARCHAR2(30)	NOT NULL	Name of the column containing the level attribute.
DTYPE	VARCHAR2(10)	NOT NULL	Data type of the column containing the level attribute.

ALL_OLAP2_ENTITY_DESC_USES

ALL_OLAP2_ENTITY_DESC_USES lists the reserved attributes and shows whether or not dimensions are time dimensions.

Column	Data Type	NULL	Description
DESCRIPTOR_ID	NUMBER	NOT NULL	Name of the reserved attribute or dimension type. The reserved dimension attributes are listed in Table 12–1, " Reserved Dimension Attributes" on page 12-1. The reserved level attributes are listed in Table 16–1, " Reserved Level Attributes" on page 16-1.
ENTITY_OWNER	VARCHAR2(30)	NOT NULL	Owner of the metadata entity.
ENTITY_NAME	VARCHAR2(30)	NOT NULL	Name of the metadata entity.
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_ENTITY_EXT_PARMS

ALL_OLAP2_ENTITY_EXT_PARMS lists the following metadata descriptors: Default Member, Dense Indicator, Fact Table Join, and Estimated Cardinality.

The metadata descriptors are described in Table 8–1, " OLAP Catalog Metadata Descriptors" on page 8-1.

Column	Data Type	NULL	Description
DESCRIPTOR_ID	NUMBER(38)		ID of the metadata descriptor.
DESCRIPTOR_NAME	VARCHAR2(240)		One of the following metadata descriptor names: **Default Member** — The default dimension member within a hierarchy. The Default Member descriptor is set by the CWM2_OLAP_CLASSIFY.ADD_ENTITY_DEFAULTMEMBER_USE procedure (described on page 8-5). **Dense Indicator** — Specifies whether the data is sparse or dense over a dimension of a cube. The Dense Indicator descriptor is set by the CWM2_OLAP_CLASSIFY.ADD_ENTITY_DENSEINDICATOR_USE procedure (described on page 8-6). **Fact Table Join** — Specifies the key columns in a dimension table that satisfy the foreign key columns in the fact table. The Fact Table Join descriptor applies only to CWM2 metadata. The Fact Table Join descriptor is set by the CWM2_OLAP_CLASSIFY.ADD_ENTITY_FACTJOIN_USE procedure (described on page 8-7). **Estimated Cardinality** — The Estimated Cardinality descriptor is set by the CWM2_OLAP_CLASSIFY.ADD_ENTITY_CARDINALITY_USE procedure (described on page 8-5).
ENTITY_OWNER	VARCHAR2(240)		Schema of the cube or dimension.
ENTITY_NAME	VARCHAR2(240)		Name of the cube or dimension.
CHILD_ENTITY_NAME	VARCHAR2(30)		Name of a child of the cube or dimension. For example, a dimension attribute is a child of a dimension, and a measure is a child of a cube. If the descriptor applies to a cube or dimension, this parameter is NULL.
SECONDARY_CHILD_ENTITY_NAME	VARCHAR2(30)		Name of a child of the child entity. For example, a level attribute is a child of a level, which is a child of a dimension. If the descriptor applies to a cube or dimension, or a child of a cube or dimension, this parameter is NULL.
PARAMETER_NAME	VARCHAR2(80)		User-defined label for the descriptor.
PARAMETER_VALUE	VARCHAR2(4000)		Value of the descriptor. For the Fact Table Join descriptor, this parameter contains the table owner.
PARAMETER_VALUE2	VARCHAR2(4000)		Table name for Fact Table Join descriptor.
PARAMETER_VALUE3	VARCHAR2(4000)		Column name for Fact Table Join descriptor.
PARAMETER_VALUE4	VARCHAR2(4000)		Hierarchy name for Fact Table Join descriptor.
POSITION	NUMBER		Position in multi-column key for Fact Table Join descriptor.

ALL_OLAP2_ENTITY_PARAMETERS

ALL_OLAP2_ENTITY_PARAMETERS lists the metadata descriptors not listed in ALL_OLAP2_ENTITY_EXT_PARMS. Additionally, it includes all the descriptors from ALL_OLAP2_ENTITY_DESC_USES.

The metadata descriptors are described in Table 8–1, " OLAP Catalog Metadata Descriptors" on page 8-1.

Column	Data Type	NULL	Description
DESCRIPTOR_ID	NUMBER(38)		ID of metadata descriptor.
DESCRIPTOR_NAME	VARCHAR2(240)		Name of the metadata descriptor.
ENTITY_OWNER	VARCHAR2(240)		Schema of the cube or dimension.
ENTITY_NAME	VARCHAR2(240)		Name of the cube or dimension.
CHILD_ENTITY_NAME	VARCHAR2(240)		Name of a child of the cube or dimension. For example, a dimension attribute is a child of a dimension, and a measure is a child of a cube. If the descriptor applies to a cube or dimension, this parameter is NULL.
SECONDARY_CHILD_ENTITY_NAME	VARCHAR2(240)		Name of a child of the child entity. For example, a level attribute is a child of a level, which is a child of a dimension. If the descriptor applies to a cube or dimension, or a child of a cube or dimension, this parameter is NULL.
PARAMETER_NAME	VARCHAR2(30)		User-defined label for the descriptor.
PARAMETER_VALUE	VARCHAR2(80)		Value of the descriptor.

ALL_OLAP2_FACT_LEVEL_USES

ALL_OLAP2_FACT_LEVEL_USES shows the joins between dimension tables and fact tables in a star or snowflake schema. For more information, see "Joining Fact Tables with Dimension Tables" on page 2-9.

Column	Data Type	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the cube.
CUBE_NAME	VARCHAR2(30)	NOT NULL	Name of the cube.
DIMENSION_OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension.
DIMENSION_NAME	NUMBER	NOT NULL	Name of the dimension.
DIMENSION_ALIAS	VARCHAR2(30)		Dimension alias (if applicable).
HIERARCHY_NAME		NOT NULL	Name of the hierarchy.
DIM_HIER_COMBO_ID	NUMBER	NOT NULL	ID of the dimension hierarchy combination associated with this fact table.
LEVEL_NAME	VARCHAR2(30)		Name of the level within the hierarchy where the mapping occurs.
FACT_TABLE_OWNER	VARCHAR2(30)	NOT NULL	Owner of the fact table.
FACT_TABLE_NAME	VARCHAR2(30)	NOT NULL	Name of the fact table.
COLUMN_NAME	VARCHAR2(30)	NOT NULL	Name of the foreign key column in the fact table.
POSITION	NUMBER		Position of this column within a multi-column key.
DIMENSION_KEYMAP_TYPE	VARCHAR2(30)	NOT NULL	Type of key mapping for the fact table. Values may be: LL (Lowest Level), when only lowest-level dimension members are stored in the key column. The fact table is unsolved. ET (Embedded Totals), when dimension members for all level combinations are stored in the key column. The fact table is solved (contains embedded totals for all level combinations).

Column	Data Type	NULL	Description
FOREIGN_KEY_NAME	VARCHAR2(30)		Name of the foreign key constraint applied to the foreign key column. Constraints are not used by the CWM2 APIs.

ALL_OLAP2_FACT_TABLE_GID

ALL_OLAP2_FACT_TABLE_GID shows the Grouping ID column for each hierarchy in each fact table. For more information, see "Grouping ID Column" on page 1-22.

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 shows the sort order for level columns within hierarchies. Custom sorting information is optional.

Custom sorting information specifies how to sort the members of a hierarchy based on columns in the dimension table. The specific columns in the dimension tables may be the same as the key columns or may be related attribute columns.

Custom sorting can specify that the column be sorted in ascending or descending order, with nulls first or nulls last. Custom sorting can be applied at multiple levels of a dimension.

Column	Data Type	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension.
DIMENSION_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension.
HIERARCHY_NAME	VARCHAR2(30)	NOT NULL	Name of the hierarchy.
TABLE_OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension table.
TABLE_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension table.
COLUMN_NAME	VARCHAR2(30)	NOT NULL	Name of the column to be sorted.
POSITION	NUMBER	NOT NULL	Represents the position within a multi-column SORT_POSITION. In most cases, a single column represents SORT_POSITION, and the value of POSITION is 1.
SORT_POSITION	NUMBER	NOT NULL	Position within the sort order of the level to be sorted.
SORT_ORDER	VARCHAR2(4)	NOT NULL	Sort order. Can be either Ascending or Descending.
NULL_ORDER	VARCHAR2(5)	NOT NULL	Where to insert null values in the sort order. Can be either Nulls First or Nulls Last.

ALL_OLAP2_JOIN_KEY_COLUMN_USES

ALL_OLAP2_JOIN_KEY_COLUMN_USES shows the joins between two levels in a hierarchy. The joins are between dimension tables in a snowflake schema, and between level columns in a star schema.

If the level is mapped to more than one column, each column mapping is represented in a separate row in the view.

Column	Data Type	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension.
DIMENSION_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension.
HIERARCHY_NAME	VARCHAR2(30)	NOT NULL	Name of the hierarchy.
CHILD_LEVEL_NAME	VARCHAR2(30)	NOT NULL	Child level in the hierarchy.
TABLE_OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension table.
TABLE_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension table.
COLUMN_NAME	VARCHAR2(30)	NOT NULL	Name of the child level column in the dimension table. In a star schema, this is the column associated with CHILD_LEVEL_NAME. In a snowflake schema, this is the parent column of CHILD_LEVEL_NAME in the same dimension table.
POSITION	NUMBER		Position of column within the key. Applies to multi-column keys only (where the level is mapped to more than one column).
JOIN_KEY_TYPE	VARCHAR2(30)	NOT NULL	The key is of type SNOWFLAKE if the join key is a logical foreign key. The key is of type STAR if the join key refers to a column within the same table.

ALL_OLAP2_LEVEL_KEY_COL_USES

ALL_OLAP2_LEVEL_KEY_COL_USES shows the mapping of each level to a unique key column.

If the level is mapped to more than one column, each column mapping is represented in a separate row in the view.

Column	Data Type	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension.
DIMENSION_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension.
HIERARCHY_NAME	VARCHAR2(30)		Name of the hierarchy that includes this level.
CHILD_LEVEL_NAME	VARCHAR2(30)	NOT NULL	Name of the level.
TABLE_OWNER	VARCHAR2(30)	NOT NULL	Owner of the dimension table.
TABLE_NAME	VARCHAR2(30)	NOT NULL	Name of the dimension table.
COLUMN_NAME	VARCHAR2(30)	NOT NULL	Name of the column that stores CHILD_LEVEL_NAME.
POSITION	NUMBER		Position of the column within the key. Applies to multi-column keys only (where the level is mapped to more than one column).

6

OLAP Dynamic Performance Views

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

> **See Also:** For additional information about fixed tables and views, refer to the following:
>
> - *Oracle Database Reference*
> - *Oracle Database Performance Tuning Guide*

This chapter contains the following topics:

V$ Tables for OLAP

Each Oracle database instance maintains a set of virtual tables that record current database activity and store data about the instance. These tables are called the **V$** tables. They are also referred to as the **dynamic performance tables**, because they store information that pertains primarily to performance. Views of the V$ tables are sometimes called **fixed views** because they cannot be altered or removed by the database administrator.

The V$ tables collect data on internal disk structures and memory structures. They are continuously updated while the database is in use. Among them are tables that collect data on Oracle OLAP.

The `SYS` user owns the V$ 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 also owned by `SYS`, but the DBA can grant access to them to a wider range of users.

The names of the OLAP V$ tables begin with `V$AW`. The view names also begin with `V$AW`. 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$AW_AGGREGATE_OP
V$AW_ALLOCATE_OP
V$AW_CALC
V$AW_LONGOPS
V$AW_OLAP
V$AW_SESSION_INFO
```

See Also: For more information on the V$ views in the Database, see the *Oracle Database Reference*.

Summary of OLAP Dynamic Performance Views

Table 6–1 briefly describes each OLAP dynamic performance view.

Table 6–1 OLAP Fixed Views

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

V$AW_AGGREGATE_OP

V$AW_AGGREGATE_OP lists the aggregation operators available in the OLAP DML. You can use this view in an application to provide a list of choices.

Column	Datatype	Description
NAME	VARCHAR2(14)	Operator keyword used in the OLAP DML RELATION command
LONGNAME	VARCHAR2(30)	Descriptive name for the operator
DEFAULT_WEIGHT	NUMBER	Default weight factor for weighted operators

V$AW_ALLOCATE_OP

V$AW_ALLOCATE_OP lists the allocation operators available in the OLAP DML. You can use this view in an application to provide a list of choices.

Column	Datatype	Description
NAME	VARCHAR2(14)	Operator keyword used in the OLAP DML RELATION command
LONGNAME	VARCHAR2(30)	Descriptive name for the operator

V$AW_CALC

V$AW_CALC reports on the effectiveness of various caches used by Oracle OLAP and the status of processing by the AGGREGATE function.

OLAP Caches

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 internal cache used by the aggregation subsystem during querying. It stores the children of a given dimension member, such as Q1-04, Q2-04, Q3-04, and Q4-04 as the children of 2004.
- **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 User Global Area (UGA), which Oracle OLAP maintains for the session. The page pool is associated with a particular session and caches records from all the analytic workspaces attached in that session. 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. The size of the page pool is controlled by the OLAP_PAGE_POOL initialization parameter.
- **Database cache**. The larger cache maintained by the Oracle RDBMS for the database instance.

> **See Also:** *Oracle OLAP Application Developer's Guide* for full discussions of data storage issues and aggregation.

Dynamic Aggregation

V$AW_CALC provides status information about dynamic aggregation in each OLAP session. Dynamic aggregation is performed by the OLAP DML AGGREGATE function.

V$AW_CALC reports the number of logical NAs generated when AGGINDEX is set. AGGINDEX is an index of all composite tuples for the data. When a composite tuple does not exist, the AGGREGATE function returns NA.

V$AW_CALC also reports the number of times the AGGREGATE function uses a precomputed aggregate, and the number of times the AGGREGATE function has to calculate an aggregate value.

See Also: *Oracle OLAP DML Reference* for information about the AGGREGATE function.

Column	Datatype	Description
SESSION_ID	NUMBER	A unique numeric identifier for the session.
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 kilobytes in the OLAP page pool.
CURR_DML_COMMAND	VARCHAR2(64)	The OLAP DML command currently being executed.
PREV_DML_COMMAND	VARCHAR2(64)	The OLAP DML command most recently completed.
AGGR_FUNC_LOGICAL_NA	NUMBER	The number of times the AGGREGATE function returns a logical NA because AGGINDEX is on and the composite tuple does not exist.
AGGR_FUNC_PRECOMPUTE	NUMBER	The number of times the AGGREGATE function finds a value in a position that it was called to calculate.
AGGR_FUNC_CALCS	NUMBER	The number of times the AGGREGATE function calculates a parent value based on the values of its children.

V$AW_LONGOPS

V$AW_LONGOPS provides status information about active SQL cursors initiated in the OLAP DML.

A cursor can be initiated within the OLAP DML using SQL FETCH, SQL IMPORT, or SQL EXECUTE, that is, SQL statements that can be declared and executed.

Column	Datatype	Description
SESSION_ID	NUMBER	The identifier for the session in which the fetch is executing. This table can be joined with V$SESSION to get the user name.
CURSOR_NAME	VARCHAR2(64)	The name assigned to the cursor in an OLAP DML SQL DECLARE CURSOR or SQL PREPARE CURSOR command.
COMMAND	VARCHAR2(7)	An OLAP DML command (SQL IMPORT, SQL FETCH, or SQL EXECUTE) that is actively fetching data from relational tables.
STATUS	VARCHAR2(9)	One of the following values: ■ EXECUTING. The command has begun executing. ■ FETCHING. Data is being fetched into the analytic workspace. ■ FINISHED. The command has finished executing. This status appears very briefly before the record disappears from the table.
ROWS_PROCESSED	NUMBER	The number of rows already inserted, updated, or deleted.
START_TIME	TIMESTAMP(3)	The time the command started executing.

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 numeric identifier for a session.
AW_NUMBER	NUMBER	A unique numeric identifier for an analytic workspace. To get the name of the analytic workspace, join this column to the AW_NUMBER column of the USER_AWS view or to the AWSEQ# column of the AW$ table
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 OLAP DML commands can execute within a single transaction, such as in a call to the DBMS_AW.EXECUTE procedure.

Column	Datatype	Description
SESSION_ID	NUMBER	A unique numeric identifier for a session.
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
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
TRANSACTION_TIME	NUMBER	The elapsed time in milliseconds of the mostly recently completed transaction.
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

7

CWM2_OLAP_CATALOG

The CWM2_OLAP_CATALOG package provides procedures for managing measure folders.

Note: The term **catalog**, when used in the context of the CWM2_OLAP_CATALOG package, refers to a measure folder.

See Also:

- Chapter 18, "CWM2_OLAP_MEASURE"
- Chapter 2, "Creating OLAP Catalog Metadata with CWM2"

This chapter discusses the following topics:

- Understanding Measure Folders
- Example: Creating a Measure Folder
- Summary of CWM2_OLAP_CATALOG Subprograms

Understanding Measure Folders

A measure folder is an OLAP Catalog metadata entity. This means that it is a logical object, identified by name and owner, within the OLAP Catalog.

Use the procedures in the CWM2_OLAP_CATALOG package to create, populate, drop, and lock measure folders, and to specify descriptive information for display purposes.

Measure folders provide a mechanism for grouping related measures. They can contain measures and nested measure folders. Access to measure folders is schema-independent. All measure folders are visible to any client. However, access to the measures themselves depends on the client's access rights to the underlying tables.

See Also: *Oracle OLAP Application Developer's Guide* for more information on measure folders and the OLAP Catalog metadata model.

Example: Creating a Measure Folder

The following statements create a measure folder called PHARMACEUTICALS and add the measure UNIT_COST from the cube SH.COST_CUBE. The measure folder is at the root level.

```
execute cwm2_olap_catalog.create_catalog
```

```
     ('PHARMACEUTICALS', 'Pharmaceutical Sales and Planning');
execute cwm2_olap_catalog.add_catalog_entity
     ('PHARMACEUTICALS',  'SH', 'COST_CUBE', UNIT_COST');
```

Summary of CWM2_OLAP_CATALOG Subprograms

Table 7–1 CWM2_OLAP_CATALOG Subprograms

Subprogram	Description
ADD_CATALOG_ENTITY Procedure on page 7-4	Adds a measure to a measure folder.
CREATE_CATALOG Procedure on page 7-4	Creates a measure folder.
DROP_CATALOG Procedure on page 7-4	Drops a measure folder.
LOCK_CATALOG Procedure on page 7-5	Locks a measure folder.
REMOVE_CATALOG_ENTITY Procedure on page 7-5	Removes a measure from a measure folder.
SET_CATALOG_NAME Procedure on page 7-5	Sets the name of a measure folder.
SET_DESCRIPTION Procedure on page 7-6	Sets the description of a measure folder.
SET_PARENT_CATALOG Procedure on page 7-6	Sets the parent folder of a measure folder.

ADD_CATALOG_ENTITY Procedure

This procedure adds a measure to a measure folder.

Syntax

```
ADD_CATALOG_ENTITY  (
               catalog_name     IN   VARCHAR2,
               cube_owner       IN   VARCHAR2,
               cube_name        IN   VARCHAR2,
               measure_name     IN   VARCHAR2);
```

Parameters

Table 7–2 ADD_CATALOG_ENTITY Procedure Parameters

Parameter	Description
`catalog_name`	Name of the measure folder.
`cube_owner`	Owner of the cube.
`cube_name`	Name of the cube.
`measure_name`	Name of the measure to be added to the measure folder.

CREATE_CATALOG Procedure

This procedure creates a new measure folder.

Descriptions and display properties must also be established as part of measure folder creation. Once the measure folder has been created, you can override these properties by calling other procedures in this package.

Syntax

```
CREATE_CATALOG (
               catalog_name       IN   VARCHAR2,
               description        IN   VARCHAR2,
               parent_catalog     IN   VARCHAR2 DEFAULT NULL);
```

Parameters

Table 7–3 CREATE_CATALOG Procedure Parameters

Parameter	Description
`catalog_name`	Name of the measure folder.
`description`	Description of the measure folder.
`parent_catalog`	Optional parent measure folder.

DROP_CATALOG Procedure

This procedure drops a measure folder. If the measure folder contains other measure folders, they are also dropped.

Syntax

```
DROP_CATALOG (
               catalog_name     IN   VARCHAR2);
```

Parameters

Table 7–4 DROP_CATALOG Procedure Parameters

Parameter	Description
catalog_name	Name of the measure_folder.

LOCK_CATALOG Procedure

This procedure locks the measure folder's metadata for update by acquiring a database lock on the row that identifies the measure folder in the CWM2 model table.

Syntax

```
LOCK_CATALOG (
               catalog_name     IN   VARCHAR2,
               wait_for_lock    IN   BOOLEAN DEFAULT FALSE);
```

Parameters

Table 7–5 LOCK_CATALOG Procedure Parameters

Parameter	Description
catalog_name	Name of the measure folder
wait_for_lock	(Optional) Whether or not to wait for the measure folder 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.

REMOVE_CATALOG_ENTITY Procedure

This procedure removes a measure from a measure folder.

Syntax

```
REMOVE_CATALOG_ENTITY  (
               catalog_name      IN   VARCHAR2,
               cube_owner        IN   VARCHAR2,
               cube_name         IN   VARCHAR2,
               measure_name      IN   VARCHAR2);
```

Parameters

Table 7–6 REMOVE_CATALOG_ENTITY Procedure Parameters

Parameter	Description
catalog_name	Name of the measure folder.
cube_owner	Owner of the cube.
cube_name	Name of the cube.
measure_name	Name of the measure to be removed from the measure folder.

SET_CATALOG_NAME Procedure

This procedure sets the name for a measure folder.

Syntax

```
SET_CATALOG_NAME (
               old_catalog_name     IN   VARCHAR2,
               new_catalog_name     IN   VARCHAR2);
```

Parameters

Table 7–7 SET_CATALOG_NAME Procedure Parameters

Parameter	Description
old_catalog_name	Old measure folder name.
new_catalog_name	New measure folder name.

SET_DESCRIPTION Procedure

This procedure sets the description for a measure folder.

Syntax

```
SET_DESCRIPTION (
               catalog_name     IN   VARCHAR2,
               description      IN   VARCHAR2);
```

Parameters

Table 7–8 SET_DESCRIPTION Procedure Parameters

Parameter	Description
catalog_name	Name of the measure folder
description	Description of the measure folder.

SET_PARENT_CATALOG Procedure

This procedure sets a parent measure folder for a measure folder.

Syntax

```
SET_PARENT_CATALOG (
               catalog_name            IN   VARCHAR2,
               parent_catalog_name     IN   VARCHAR2   DEFAULT NULL);
```

Parameters

Table 7–9 SET_PARENT_CATALOG Procedure Parameters

Parameter	Description
catalog_name	Name of the measure folder.
parent_catalog_name	Name of the parent measure folder. If the measure folder is at the root level, this parameter is null.

8

CWM2_OLAP_CLASSIFY

The CWM2_OLAP_CLASSIFY package provides procedures for managing metadata extensions for the OLAP API.

This chapter discusses the following topics:

- OLAP Catalog Metadata Descriptors
- Example: Creating Descriptors
- Summary of CWM2_OLAP_CLASSIFY Subprograms

OLAP Catalog Metadata Descriptors

The OLAP Catalog metadata descriptors provide additional information about your data. These descriptors can be used by the OLAP API.

The OLAP Catalog metadata descriptors are described in Table 8–1, " OLAP Catalog Metadata Descriptors".

You can view the descriptors that have been set for your OLAP Catalog metadata in the views ALL_OLAP2_ENTITY_EXT_PARMS (described on page 5-9) and ALL_OLAP2_ENTITY_PARAMETERS (described on page 5-10).

Table 8–1 OLAP Catalog Metadata Descriptors

Descriptor	Applies To	Description
`Level Standard`	level	The level is not in a time dimension.
`Level Year`	level	The year level in a time dimension.
`Level HalfYear`	level	The half year level in a time dimension.
`Level Quarter`	level	The quarter level in a time dimension.
`Level Month`	level	The month level in a time dimension.
`Level Week`	level	The week level in a time dimension.
`Level Day`	level	The day level in a time dimension.
`Level Hour`	level	The hour level in a time dimension.
`Level Minute`	level	The minutes level in a time dimension.
`Level Second`	level	The seconds level in a time dimension.
`Value Separator`	dimension	The separator character used by the OLAP API to construct the names of dimension members. The default separator is "::".

Table 8–1 (Cont.) OLAP Catalog Metadata Descriptors

Descriptor	Applies To	Description
Skip Level	hierarchy	Whether or not the hierarchy supports skip levels. An example of a skip level hierarchy is City-State-Country, where Washington D.C. is a City whose parent is a Country.
Measure Format	measure	The display format for a measure.
Measure Unit	measure	The unit of measurement of a measure.
Fact Table Join	hierarchy	The key columns in a dimension table that satisfy the join to a fact table. This descriptor applies to CWM2 metadata only.
Default Member	hierarchy	The default dimension member in a hierarchy.
Dense Indicator	dimension	Whether or not the data over a given dimension of a cube is dense or sparse.
Estimated Cardinality	level	Estimated number of dimension members in a given level.

Example: Creating Descriptors

The following examples show how to set some of the metadata descriptors.

> **Note:** If you have used Enterprise Manager to create your OLAP Catalog metadata, be sure to respect the case of metadata names.

The following statements specify the quarter, month, and year levels in the time dimension XADEMO.TIME.

```
execute cwm2_olap_classify.add_entity_descriptor_use
     ('Level Year', 'LEVEL', 'XADEMO', 'TIME', 'L1');
execute cwm2_olap_classify.add_entity_descriptor_use
     ('Level Quarter', 'LEVEL', 'XADEMO', 'TIME', 'L2');
execute cwm2_olap_classify.add_entity_descriptor_use
     ('Level Month', 'LEVEL', 'XADEMO', 'TIME', 'L3');
```

The following statement indicates that the value separator used by the OLAP API to construct dimension member names for XADEMO.TIME is the default ("::").

```
execute cwm2_olap_classify.add_entity_descriptor_use
     ('Value Separator', 'DIMENSION', 'XADEMO', 'TIME', NULL, NULL,
      'Value Separator','::');
```

The following statement indicates that the data in the cube XADEMO.ANALYTIC_CUBE is dense over Time and Geography, but sparse over Channel and Product.

```
execute cwm2_olap_classify.add_entity_denseindicator_use
     ('XADEMO', 'ANALYTIC_CUBE', 'XADEMO', 'TIME', 'YES');
execute cwm2_olap_classify.add_entity_denseindicator_use
     ('XADEMO', 'ANALYTIC_CUBE', 'XADEMO', 'GEOGRAPHY', 'YES');
execute cwm2_olap_classify.add_entity_denseindicator_use
     ('XADEMO', 'ANALYTIC_CUBE', 'XADEMO', 'CHANNEL', 'NO');
execute cwm2_olap_classify.add_entity_denseindicator_use
     ('XADEMO', 'ANALYTIC_CUBE', 'XADEMO', 'PRODUCT', 'NO');
```

The following statement removes the Dense Indicator descriptors from XADEMO.ANALYTIC_CUBE.

```
execute cwm2_olap_classify.remove_entity_descriptor_use
  ('Dense Indicator', 'DENSE INDICATOR','XADEMO', 'ANALYTIC_CUBE',
   'XADEMO', 'CHANNEL');
execute cwm2_olap_classify.remove_entity_descriptor_use
  ('Dense Indicator', 'DENSE INDICATOR','XADEMO', 'ANALYTIC_CUBE',
   'XADEMO', 'PRODUCT');
execute cwm2_olap_classify.remove_entity_descriptor_use
  ('Dense Indicator', 'DENSE INDICATOR','XADEMO', 'ANALYTIC_CUBE',
   'XADEMO', 'GEOGRAPHY');
execute cwm2_olap_classify.remove_entity_descriptor_use
  ('Dense Indicator', 'DENSE INDICATOR','XADEMO', 'ANALYTIC_CUBE',
   'XADEMO', 'TIME');
```

Summary of CWM2_OLAP_CLASSIFY Subprograms

Table 8–2 CWM2_OLAP_CLASSIFY Subprograms

Subprogram	Description
ADD_ENTITY_CARDINALITY_USE on page 8-5	Adds the `Estimated Cardinality` descriptor to a level of a hierarchy.
ADD_ENTITY_DEFAULTMEMBER_USE on page 8-5	Adds the `Default Member` descriptor to a hierarchy.
ADD_ENTITY_DENSEINDICATOR_USE on page 8-6	Adds the `Dense Indicator` descriptor to a dimension of a cube.
ADD_ENTITY_DESCRIPTOR_USE on page 8-7	Applies a descriptor to a metadata entity.
ADD_ENTITY_FACTJOIN_USE on page 8-7	Adds the `Fact Table Join` descriptor to a CWM2 hierarchy.
REMOVE_ENTITY_DESCRIPTOR_USE on page 8-8	Removes a descriptor from a metadata entity.

ADD_ENTITY_CARDINALITY_USE

This procedure adds the `Estimated Cardinality` descriptor to a level of a hierarchy.

The OLAP Catalog metadata descriptors are described in Table 8–1, " OLAP Catalog Metadata Descriptors".

Syntax

```
ADD_ENTITY_CARDINALITY_USE  (
               dimension_owner         IN   VARCHAR2,
               dimension_name          IN   VARCHAR2,
               hierarchy_name          IN   VARCHAR2,
               level_name              IN   VARCHAR2,
               estimated_cardinality   IN   NUMBER);
```

Parameters

Table 8–3 ADD_ENTITY_CARDINALITY_USE Procedure Parameters

Parameter	Description
`dimension_owner`	Owner of the dimension.
`dimension_name`	Name of the dimension.
`hierarchy_name`	Hierarchy within the dimension. If the dimension has no hierarchy, specify `NULL`.
`level_name`	Level within the hierarchy.
`estimated_cardinality`	Estimated number of dimension members in the level.

Example

The following statement sets the estimated cardinality of a level in the Standard hierarchy of the Geography dimension.

```
execute cwm2_olap_classify.add_entity_cardinality_use
         ('XADEMO', 'GEOGRAPHY', 'STANDARD', 'L4', 60);
```

ADD_ENTITY_DEFAULTMEMBER_USE

This procedure adds the `Default Member` descriptor to a hierarchy.

The OLAP Catalog metadata descriptors are described in Table 8–1, " OLAP Catalog Metadata Descriptors".

Syntax

```
ADD_ENTITY_DEFAULTMEMBER_USE  (
               dimension_owner        IN   VARCHAR2,
               dimension_name         IN   VARCHAR2,
               hierarchy_name         IN   VARCHAR2,
               default_member         IN   VARCHAR2,
               default_member_level   IN   VARCHAR2,
               position               IN   NUMBER DEFAULT NULL);
```

Parameters

Table 8–4 ADD_ENTITY_DEFAULTMEMBER_USE Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
hierarchy_name	Name of the hierarchy.
default_member	Name of a dimension member in the hierarchy.
default_member_level	Level of the default dimension member.
position	Position of the default member within a multi-column key. If position is not meaningful, this parameter is NULL (default).

Example

The following statement sets the default member of the Standard hierarchy in the Geography dimension to Paris.

```
execute cwm2_olap_classify.add_entity_defaultmember_use
          ('XADEMO', 'GEOGRAPHY', 'STANDARD', 'Paris', 'L4');
```

ADD_ENTITY_DENSEINDICATOR_USE

This procedure adds the `Dense Indicator` descriptor to a dimension of a cube.

The OLAP Catalog metadata descriptors are described in Table 8–1, " OLAP Catalog Metadata Descriptors".

Syntax

```
ADD_ENTITY_DENSEINDICATOR_USE  (
               cube_owner        IN   VARCHAR2,
               cube_name         IN   VARCHAR2,
               dimension_owner   IN   VARCHAR2,
               dimension_name    IN   VARCHAR2,
               dense_indicator   IN   VARCHAR2 );
```

Parameters

Table 8–5 ADD_ENTITY_DENSEINDICATOR_USE Procedure Parameters

Parameter	Description
cube_owner	Owner of the cube.
cube_name	Name of the cube.
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
dense_indicator	YES indicates that the data over this dimension is dense. This means that data exists for most dimension members. NO indicates that the data over this dimension is sparse. This means that there is no data for many of the dimension members.

Example

See "Example: Creating Descriptors" on page 8-2.

ADD_ENTITY_DESCRIPTOR_USE

This procedure adds a descriptor to a metadata entity.

The OLAP Catalog metadata descriptors are described in Table 8–1, " OLAP Catalog Metadata Descriptors".

Syntax

```
ADD_ENTITY_DESCRIPTOR_USE  (
               descriptor_name              IN   VARCHAR2,
               entity_type                  IN   VARCHAR2,
               entity_owner                 IN   VARCHAR2,
               entity_name                  IN   VARCHAR2,
               entity_child_name            IN   VARCHAR2 DEFAULT NULL,
               entity_secondary_child_name  IN   VARCHAR2 DEFAULT NULL,
               parameter_name               IN   VARCHAR2 DEFAULT NULL,
               parameter_value              IN   VARCHAR2 DEFAULT NULL);
```

Parameters

Table 8–6 ADD_ENTITY_DESCRIPTOR_USE Procedure Parameters

Parameter	Description
descriptor_name	Name of the descriptor.
entity_type	Type of metadata entity to which the descriptor applies. Types are: DIMENSION HIERARCHY LEVEL LEVEL ATTRIBUTE DIMENSION ATTRIBUTE CUBE MEASURE
entity_owner	Schema of the cube or dimension.
entity_name	Name of the cube or dimension.
entity_child_name	Name of a child of the cube or dimension. For example, a dimension attribute is a child of a dimension, and a measure is a child of a cube. If the descriptor applies to a cube or dimension, this parameter is NULL.
entity_secondary_child_name	Name of a child of the child entity. For example, a level attribute is a child of a level, which is a child of a dimension. If the descriptor applies to a cube or dimension, or a child of a cube or dimension, this parameter is NULL.
parameter_name	Label for the descriptor. You can specify any label that you choose.
parameter_value	Value of the descriptor.

Example

See "Example: Creating Descriptors" on page 8-2.

ADD_ENTITY_FACTJOIN_USE

This procedure adds the `Fact Table Join` descriptor to a cube. The `Fact Table Join` descriptor applies to CWM2 metadata only.

The OLAP Catalog metadata descriptors are described in Table 8–1, " OLAP Catalog Metadata Descriptors".

Syntax

```
ADD_ENTITY_FACTJOIN_USE  (
               cube_owner              IN   VARCHAR2,
               cube_name               IN   VARCHAR2,
               dimension_owner         IN   VARCHAR2,
               dimension_name          IN   VARCHAR2,
               hierarchy_name          IN   VARCHAR2,
               dim_table_owner         IN   VARCHAR2,
               dim_table_name          IN   VARCHAR2,
               dim_table_column_name   IN   VARCHAR2,
               position                IN   NUMBER DEFAULT NULL);
```

Parameters

Table 8–7 ADD_ENTITY_FACTJOIN_USE Procedure Parameters

Parameter	Description
cube_owner	Owner of the cube.
cube_name	Name of the cube.
dimension_owner	Owner of a dimension of the cube.
dimension_name	Name of the dimension.
hierarchy_name	Name of a hierarchy of the dimension.
dim_table_owner	Owner of the dimension table.
dim_table_name	Name of the dimension table.
dim_table_column_name	Key column in the dimension table that maps to a foreign key column in the fact table.
position	Position of the key column in a multi-column key. If the key is in a single column, this parameter is NULL (Default).

Example

The following statement adds Fact Table Join descriptor to the Standard hierarchy of the Geography dimension of the ANALYTIC_CUBE.

```
execute cwm2_olap_classify.add_entity_factjoin_use
          ('XADEMO', 'ANALYTIC_CUBE', 'XADEMO','GEOGRAPHY, 'STANDARD',
           'XADEMO', 'XADEMO_GEOGRAPHY', 'GEOG_STD_CITY');
```

REMOVE_ENTITY_DESCRIPTOR_USE

This procedure removes a descriptor from an entity.

The OLAP Catalog metadata descriptors are described in Table 8–1, " OLAP Catalog Metadata Descriptors".

Syntax

```
REMOVE_ENTITY_DESCRIPTOR_USE  (
               descriptor_name              IN   VARCHAR2,
               entity_type                  IN   VARCHAR2,
               entity_owner                 IN   VARCHAR2,
               entity_name                  IN   VARCHAR2,
               entity_child_name            IN   VARCHAR2 DEFAULT NULL,
               entity_secondary_child_name  IN   VARCHAR2 DEFAULT NULL);
```

Parameters

Table 8–8 REMOVE_ENTITY_DESCRIPTOR_USE Procedure Parameters

Parameter	Description
descriptor_name	Name of the descriptor to remove.
entity_type	Type of metadata entity to which the descriptor applies. Types are: DIMENSION HIERARCHY LEVEL LEVEL ATTRIBUTE DIMENSION ATTRIBUTE CUBE MEASURE ESTIMATED CARDINALITY DEFAULT MEMBER DENSE INDICATOR FACT TABLE JOIN
entity_owner	Schema of the cube or dimension.
entity_name	Name of the cube or dimension.
entity_child_name	Name of a child of the cube or dimension. For example, a dimension attribute is a child of a dimension, and a measure is a child of a cube. If the descriptor applies to a cube or dimension, this parameter is NULL.
entity_secondary_child_name	Name of a child of the child entity. For example, a level attribute is a child of a level, which is a child of a dimension. If the descriptor applies to a cube or dimension, or a child of a cube or dimension, this parameter is NULL.

Example

See "Example: Creating Descriptors" on page 8-2.

9

CWM2_OLAP_CUBE

The CWM2_OLAP_CUBE package provides procedures managing cubes.

> **See Also:** Chapter 2, "Creating OLAP Catalog Metadata with CWM2"

This chapter discusses the following topics:

- Understanding Cubes
- Example: Creating a Cube
- Summary of CWM2_OLAP_CUBE Subprograms

Understanding Cubes

A cube is an OLAP Catalog metadata entity. This means that it is a logical object, identified by name and owner, within the OLAP Catalog.

A cube is a multidimensional framework to which you can assign measures. A measure represents data stored in fact tables. The fact tables may be relational tables or views. The views may reference data stored in analytic workspaces.

Use the procedures in the CWM2_OLAP_CUBE package to create, drop, and lock cubes, to associate dimensions with cubes, and to specify descriptive information for display purposes.

You must create the cube before using the CWM2_OLAP_MEASURE package to create the cube's measures.

> **See Also:**
>
> - Chapter 18, "CWM2_OLAP_MEASURE"
> - *Oracle OLAP Application Developer's Guide* for more information about cubes and the OLAP Catalog metadata model.

Example: Creating a Cube

The following statements drop the cube SALES_CUBE, re-create it, and add the dimensions TIME_DIM, GEOG_DIM, and PRODUCT_DIM.

Dropping the cube removes the cube entity, along with its measures, from the OLAP Catalog. However, dropping the cube does not cause the cube's dimensions to be dropped.

```
execute cwm2_olap_cube.drop_cube('JSMITH', 'SALES_CUBE');
```

```
execute cwm2_olap_cube.create_cube
     ('JSMITH', 'SALES_CUBE', 'Sales', 'Sales Cube',
      'Sales dimensioned over geography, product, and time' );
execute cwm2_olap_cube.add_dimension_to_cube
     ('JSMITH', 'SALES_CUBE', 'JSMITH', 'TIME_DIM');
execute cwm2_olap_cube.add_dimension_to_cube
     ('JSMITH', 'SALES_CUBE', 'JSMITH', 'GEOG_DIM');
execute cwm2_olap_cube.add_dimension_to_cube
     ('JSMITH', 'SALES_CUBE', 'JSMITH', 'PRODUCT_DIM');
```

Summary of CWM2_OLAP_CUBE Subprograms

Table 9–1 CWM2_OLAP_CUBE Subprograms

Subprogram	Description
ADD_DIMENSION_TO_CUBE Procedure on page 9-4	Adds a dimension to a cube.
CREATE_CUBE Procedure on page 9-4	Creates a cube.
DROP_CUBE Procedure on page 9-4	Drops a cube.
LOCK_CUBE Procedure on page 9-5	Locks a cube's metadata for update.
REMOVE_DIMENSION_FROM_CUBE Procedure on page 9-5	Removes a dimension from a cube.
SET_AGGREGATION_OPERATOR Procedure on page 9-6	Sets the aggregation operators for rolling up the cube's data.
SET_CUBE_NAME Procedure on page 9-7	Sets the name of a cube.
SET_DEFAULT_CUBE_DIM_CALC_HIER Procedure on page 9-8	Sets the default calculation hierarchy for a dimension of the cube.
SET_DESCRIPTION Procedure on page 9-8	Sets the description for a cube.
SET_DISPLAY_NAME Procedure on page 9-8	Sets the display name for a cube.
SET_MV_SUMMARY_CODE Procedure on page 9-9	Sets the format for materialized views associated with a cube.
SET_SHORT_DESCRIPTION Procedure on page 9-9	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 9–2 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.

CREATE_CUBE Procedure

This procedure creates a new cube in the OLAP Catalog.

Descriptions and display properties must also be established as part of cube creation. Once the cube has been created, you can override these properties by calling other procedures in this package.

Syntax

```
CREATE_CUBE (
          cube_owner             IN   VARCHAR2,
          cube_name              IN   VARCHAR2,
          display_name           IN   VARCHAR2,
          short_description      IN   VARCHAR2,
          description            IN   VARCHAR2);
```

Parameters

Table 9–3 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.

DROP_CUBE Procedure

This procedure drops a cube from the OLAP 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 9–4 DROP_CUBE Procedure Parameters

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

LOCK_CUBE Procedure

This procedure locks the cube's metadata for update by acquiring a database lock on the row that identifies the cube in the CWM2 model table.

Syntax

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

Parameters

Table 9–5 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.

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 9–6 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.

SET_AGGREGATION_OPERATOR Procedure

This procedure sets the aggregation operator for rolling up a cube's data over its dimensions. The cube must be mapped to a star schema, with a storage type indicator of 'LOWESTLEVEL'. (See "Joining Fact Tables with Dimension Tables" on page 2-9.)

The aggregation operators supported by the OLAP Catalog are listed in Table 1–10, " Aggregation Operators" on page 1-16.

When no aggregation operator is specified, the operator is addition. The view ALL_OLAP2_AGGREGATION_USES lists the nondefault aggregation operators that have been specified for cubes. See "ALL_OLAP2_AGGREGATION_USES" on page 5-3.

Syntax

```
SET_AGGREGATION_OPERATOR (
          cube_owner        IN   VARCHAR2,
          cube_name         IN   VARCHAR2,
          aggop_spec        IN   VARCHAR2);
```

Parameters

Table 9–7 SET_AGGREGATION_OPERATOR Procedure Parameters

Parameter	Description
cube_owner	Owner of the cube.
cube_name	Name of the cube.
aggop_spec	A string that specifies the aggregation operators for the cube. Each aggregation operator that you specify applies to all of the cube's measures over a given hierarchy of a given dimension of the cube. If you do not specify a hierarchy, the operator applies to all hierarchies of the dimension. By default, the aggregation operator is addition. Enclose the string in single quotes, and separate each dimension/operator clause with a semicolon as follows: 'DIM:*dim1_owner.dim1_name*/AGGOP:*operator*; DIM:*dim2_owner.dim2_name*/AGGOP:*operator*;.....' If the operator should apply to a specific hierarchy of a dimension, use the optional 'HIER' clause after the DIM clause: /HIER:*hiername1* For weighted operators, the 'AGGOP' clause may optionally be followed with a WEIGHTBY clause: /WEIGHTBY:*TblOwner.TblName.ColName;* *NOTE:* The cube's data will be aggregated in the order of the dimension clauses in the aggregation specification.

Example

The following example specifies that data in the ANALYTIC_CUBE should be aggregated using addition over the Standard hierarchies of the Product and Channel dimensions, using the MAX operator over the Standard hierarchy of Geography, and using AVERAGE over the Year to Date hierarchy of the Time dimension. Any unspecified hierarchies will use addition.

```
execute cwm2_olap_cube.set_aggregation_operator
            ('XADEMO', 'ANALYTIC_CUBE',
             'DIM:XADEMO.PRODUCT/HIER:STANDARD/AGGOP:SUM;
              DIM:XADEMO.GEOGRAPHY/HIER:STANDARD/AGGOP:MAX;
              DIM:XADEMO.TIME/HIER:YTD/AGGOP:AVERAGE;
              DIM:XADEMO.CHANNEL/HIER:STANDARD/AGGOP:SUM;');
```

The following example shows the same specification including a weighted operator for Product.

```
execute cwm2_olap_cube.set_aggregation_operator
            ('XADEMO', 'ANALYTIC_CUBE',
             'DIM:XADEMO.PRODUCT/HIER:STANDARD/AGGOP:SUM/
                                WEIGHTBY:XADEMO.XADEMO_SALES_VIEW.COSTS;
              DIM:XADEMO.GEOGRAPHY/HIER:STANDARD/AGGOP:MAX;
              DIM:XADEMO.TIME/HIER:YTD/AGGOP:AVERAGE;
              DIM:XADEMO.CHANNEL/HIER:STANDARD/AGGOP:SUM;');
```

In the following example, aggregation operators are specified for all hierarchies of each dimension.

```
execute cwm2_olap_cube.set_aggregation_operator
            ('XADEMO', 'ANALYTIC_CUBE',
              DIM:XADEMO.PRODUCT/AGGOP:SUM;
              DIM:XADEMO.GEOGRAPHY/AGGOP:MAX;
              DIM:XADEMO.TIME/AGGOP:AVERAGE;
              DIM:XADEMO.CHANNEL/AGGOP:SUM;');
```

See Also

"Aggregating the Cube's Data in the Analytic Workspace" on page 1-4

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 9–8 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.

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 9–9 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.

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 9–10 SET_DESCRIPTION Procedure Parameters

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

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 9–11 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.

SET_MV_SUMMARY_CODE Procedure

This procedure specifies the form of materialized views for this cube. Materialized views may be in Grouping Set (groupingset) or Rolled Up (rollup) form.

In a materialized view in Rolled Up form, all the dimension key columns are populated, and data may only be accessed when its full lineage is specified.

In a materialized view in Grouping Set form, dimension key columns may contain null values, and data may be accessed simply by specifying one or more levels.

Syntax

```
SET_MV_SUMMARY_CODE (
          cube_owner            IN   VARCHAR2,
          cube_name             IN   VARCHAR2,
          summary_code          IN   VARCHAR2);
```

Parameters

Table 9–12 SET_MV_SUMMARY_CODE Procedure Parameters

Parameter	Description
cube_owner	Owner of the cube.
cube_name	Name of the cube.
summary_code	One of the following case-insensitive values: ■ rollup, for Rolled Up form. ■ groupingset, for Grouping Set form.

SET_SHORT_DESCRIPTION Procedure

This procedure sets the short description for a cube.

Syntax

```
SET_SHORT_DESCRIPTION (
          cube_owner            IN   VARCHAR2,
          cube_name             IN   VARCHAR2,
          short_description     IN   VARCHAR2);
```

Parameters

Table 9–13 SET_SHORT_DESCRIPTION Procedure Parameters

Parameter	Description
cube_owner	Owner of the cube.

Table 9–13 (Cont.) SET_SHORT_DESCRIPTION Procedure Parameters

Parameter	Description
cube_name	Name of the cube.
short_description	Short description of the cube.

10

CWM2_OLAP_DELETE

The CWM2_OLAP_DELETE package provides procedures for deleting OLAP Catalog metadata.

See Also: Chapter 13, "CWM2_OLAP_EXPORT"

This chapter discusses the following topics:

- Deleting OLAP Catalog Metadata
- Summary of CWM2_OLAP_DELETE Subprograms

Deleting OLAP Catalog Metadata

You can use the CWM2_OLAP_DELETE package to delete individual cubes, dimensions, or measure folders, or the entire contents of the OLAP Catalog. CWM2_OLAP_DELETE deletes CWM2 metadata created by the CWM2 PL/SQL packages and CWM1 metadata created by Oracle Enterprise Manager. CWM2_OLAP_DELETE deletes both valid and invalid metadata.

OLAP dimensions created in Oracle Enterprise Manager use Oracle Database dimension objects. When deleting these CWM1 dimensions, you can choose whether or not to delete the associated dimension objects. For more information on Oracle dimension objects, see "CREATE DIMENSION" in the *Oracle Database SQL Reference*.

Rebuilding OLAP Catalog Metadata

To rebuild the OLAP Catalog metadata for a relational data source, you can export the data and metadata, delete it, then import it. Use the CWM2_OLAP_EXPORT package and the Oracle Export utility to do the export. Use CWM2_OLAP_DELETE to delete the metadata. Drop the source tables, then use the Oracle import utility to do the import. See Chapter 13, "CWM2_OLAP_EXPORT".

To rebuild analytic workspaces, use the OLAP DML to export the contents of the workspace to an EIF file, then import it in a new workspace. See "Procedure: Import a workspace from a 9i Database into a 10g Database" on page 24-3. If you are running in Oracle9*i* compatibility mode, you will need to re-enable the workspaces and re-create the metadata for the workspaces. See "Enabling Relational Access" on page 1-17.

Using Wildcards to Identify Metadata Entities

You can use wildcard characters to delete cubes, dimensions, and measure folders whose names meet certain criteria.

Wildcard characters are the underscore "_" and the percent sign "%". An underscore replaces any single character, and a percent sign replaces any zero or more characters. An underscore, but not a percent sign, is also a legal character in a metadata owner or entity name. Any underscore character in the owner or entity name is treated as a wildcard, unless you precede it with a backslash "\" which acts as an escape character.

For example, the following command deletes all the cubes belonging to the owner 'GLOBAL'.

```
>execute cwm2_olap_delete.delete_cube('GLOBAL', '%', 'yes', 'yes');
```

The following command deletes all the cubes in the GLOBAL schema whose names start with 'a'.

```
>execute cwm2_olap_delete.delete_cube('GLOBAL', 'a%', 'yes', 'yes');
```

If your database includes users 'TESTUSER1' and 'TESTUSER2', you could delete the 'TEST' cube belonging to each of these users with the following command.

```
>execute cwm2_olap_delete.delete_cube('TESTUSER_', 'TEST', 'yes', 'yes');
```

If your database includes users 'TEST_USER1' and 'TEST_USER2', you could delete the 'TEST' cube belonging to each of these users with the following command.

```
>execute cwm2_olap_delete.delete_cube('TEST/_USER_', 'TEST', 'yes', 'yes');
```

Using a Command Report

Each procedure in the CWM2_OLAP_DELETE package accepts a parameter that causes a command report to be written to the SQL buffer. You can generate this report without deleting any metadata. A separate parameter controls whether or not you actually execute the delete commands.

> **See Also:** "Directing Output" on page 2-13 for more information on the SQL buffer and directing the output of OLAP procedures.

Depending on the metadata entities that you want to delete, the report will list commands like the following.

```
EXECUTE cwm2_olap_cube.drop_cube           ( 'cubeowner', 'cubename')
EXECUTE cwm2_olap_dimension.drop_dimension ( 'dimowner', 'dimname')
EXECUTE cwm2_olap_catalog.drop_catalog     ( 'catalogowner', 'catalogname')
```

If you choose to drop the dimension objects associated with CWM1 dimensions, the report will also include the following command.

```
EXECUTE cwm_utility.Collect_Garbage
```

Use the CWM2_OLAP_MANAGER.SET_ECHO_ON procedure to display the command report on the screen. Use the CWM2_OLAP_MANAGER.BEGIN_LOG procedure to direct the report to a log file. See "Directing Output" on page 2-13 for more information.

As long as you have directed the output of the SQL buffer to the screen or to a file, you will see messages describing the success or failure of each stored procedure call. If you choose to delete a cube without generating a command report, you will see only the following.

```
AMD-00003  dropped Cube "CUBEOWNER.CUBENAME"
```

If you choose to delete a cube *and* generate a command report, you will see the following.

```
EXECUTE cwm2_olap_cube.Drop_Cube('CUBEOWNER', 'CUBENAME');
AMD-00003  dropped Cube "CUBEOWNER.CUBENAME"
```

Summary of CWM2_OLAP_DELETE Subprograms

Table 10–1 CWM2_OLAP_DELETE

Subprogram	Description
DELETE_CUBE Procedure on page 10-5	Deletes a cube in the OLAP Catalog.
DELETE_DIMENSION Procedure on page 10-6	Deletes a dimension in the OLAP Catalog.
DELETE_MEASURE_CATALOG Procedure on page 10-7	Deletes a measure folder in the OLAP Catalog.
DELETE_OLAP_CATALOG Procedure on page 10-8	Deletes all the metadata in the OLAP Catalog.

DELETE_CUBE Procedure

This procedure can be used to delete a cube or group of cubes in the OLAP Catalog. You can also use this procedure to list the commands that will delete the cubes. You can choose to execute these commands or simply list them, without actually deleting the cubes. See "Using a Command Report" on page 10-2.

You can identify a group of cubes by specifying wildcard characters in the `cube_owner` and `cube_name` parameters. See "Using Wildcards to Identify Metadata Entities" on page 10-1.

When you delete a cube, its dimensions are not deleted.

OLAP Catalog cubes are displayed in the view ALL_OLAP2_CUBES.

Syntax

```
DELETE_CUBE (
          cube_owner      IN   VARCHAR2,
          cube_name       IN   VARCHAR2,
          delete_report   IN   VARCHAR2,
          delete_cube     IN   VARCHAR2);
```

Parameters

Table 10–2 DELETE_CUBE Procedure Parameters

Parameter	Description
`cube_owner`	The owner of the cube. See "Using Wildcards to Identify Metadata Entities" on page 10-1.
`cube_name`	The name of the cube. See "Using Wildcards to Identify Metadata Entities" on page 10-1.
`delete_report`	Whether or not to list the commands that will delete the cubes. Specify 'YES' to list the commands. Otherwise specify 'NO'. To display the output on the screen, use the `CWM2_OLAP_MANAGER.SET_ECHO_ON` procedure. To send the output to a file, use the `CWM2_OLAP_MANAGER.BEGIN_LOG` procedure. See "Directing Output" on page 2-13 for more information.
`delete_cube`	Whether or not to actually delete the cubes. Specify 'YES' to delete the cubes. Otherwise specify 'NO'.

Example

The following example first generates a command report for deleting the `cwm2` cube `PRICE_COST` in the `GLOBAL` schema, then actually deletes the cube.

```
>set serveroutput on size 1000000
>execute cwm2_olap_manager.set_echo_on;
>select * from all_olap2_cubes where OWNER ='GLOBAL';

     OWNER  CUBE_NAME  NVALID DISPLAY_NAME SHORT_DESCRIPTION DESCRIPTION  MV
     ------ ---------  ------ ------------ ----------------- -----------  --
     GLOBAL PRICE_CUBE  O    PRICE_CUBE                                   RU
     GLOBAL UNITS_CUBE  O    UNITS_CUBE                                   RU
     GLOBAL PRICE_COST  N    PRICE_COST                                   GS

>execute cwm2_olap_delete.delete_cube('GLOBAL', 'PRICE_COST', 'yes', 'no');
```

```
     EXECUTE cwm2_olap_cube.Drop_Cube('GLOBAL', 'PRICE_COST');

>select * from all_olap2_cubes where OWNER ='GLOBAL';

     OWNER  CUBE_NAME  NVALID DISPLAY_NAME SHORT_DESCRIPTION DESCRIPTION  MV
     ------ ---------  ------ ------------ ----------------- -----------  --
     GLOBAL PRICE_CUBE  O    PRICE_CUBE                                   RU
     GLOBAL UNITS_CUBE  O    UNITS_CUBE                                   RU
     GLOBAL PRICE_COST  N    PRICE_COST                                   GS

>execute cwm2_olap_delete.delete_cube('GLOBAL', 'PRICE_COST', 'yes', 'yes');

     EXECUTE cwm2_olap_cube.Drop_Cube('GLOBAL', 'PRICE_COST');
     AMD-00003  dropped Cube "GLOBAL.PRICE_COST"

>select * from all_olap2_cubes where OWNER ='GLOBAL';

     OWNER  CUBE_NAME  NVALID DISPLAY_NAME SHORT_DESCRIPTION DESCRIPTION  MV
     ------ ---------  ------ ------------ ----------------- -----------  --
     GLOBAL PRICE_CUBE  O    PRICE_CUBE                                   RU
     GLOBAL UNITS_CUBE  O    UNITS_CUBE                                   RU
```

DELETE_DIMENSION Procedure

This procedure can be used to delete a dimension or group of dimensions in the OLAP Catalog. You can also use this procedure to list the commands that will delete the dimensions. You can choose to execute these commands or simply list them, without actually deleting the dimensions. See "Using a Command Report" on page 10-2.

You can identify a group of dimensions by specifying wildcard characters in the `dimension_owner` and `dimension_name` parameters. See "Using Wildcards to Identify Metadata Entities" on page 10-1.

If the dimension was created in Oracle Enterprise Manager, it is a `CWM1` dimension. `CWM1` dimensions have OLAP Catalog metadata and an associated Oracle dimension object.

When you delete a dimension, all references within cubes to the dimension are also deleted. This causes any cubes that used the dimension to become invalid.

OLAP Catalog dimensions are displayed in the view ALL_OLAP2_DIMENSIONS.

Syntax

```
DELETE_DIMENSION (
          dimension_owner         IN   VARCHAR2,
          dimension_name          IN   VARCHAR2,
          delete_cwm1_dimension   IN   VARCHAR2,
          delete_report           IN   VARCHAR2,
          delete_dimension        IN   VARCHAR2);
```

Parameters

Table 10–3 DELETE_DIMENSION Procedure Parameters

Parameter	Description
`dimension_owner`	The owner of the dimension. See "Using Wildcards to Identify Metadata Entities" on page 10-1.
`dimension_name`	The name of the dimension. See "Using Wildcards to Identify Metadata Entities" on page 10-1.

Table 10–3 (Cont.) DELETE_DIMENSION Procedure Parameters

Parameter	Description
delete_cwm1_dimension	Whether or not to delete the Oracle dimension object associated with a CWM1 dimension. Specify 'YES' to delete the Oracle dimension object. Otherwise specify 'NO'. This parameter has no effect on CWM2 dimensions.
delete_report	Whether or not to list the commands that will delete the dimensions. Specify 'YES' to list the commands. Otherwise specify 'NO'. To display the output on the screen, use the CWM2_OLAP_MANAGER.SET_ECHO_ON procedure. To send the output to a file, use the CWM2_OLAP_MANAGER.BEGIN_LOG procedure. See "Directing Output" on page 2-13 for more information.
delete_dimension	Whether or not to actually delete the dimensions. Specify 'YES' to delete the dimensions. Otherwise specify 'NO'.

Example

The following example first generates a command report for deleting the PROD dimension in the GLOBAL schema, then actually deletes the dimension. Since the dimension is a CWM2 dimension, the third parameter to the DELETE_DIMENSION procedure is ignored.

```
>set serveroutput on size 1000000
>execute cwm2_olap_manager.set_echo_on;
>execute cwm2_olap_delete.delete_dimension
     ('GLOBAL', 'PROD', 'no','yes', 'no');

     EXECUTE cwm2_olap_dimension.Drop_Dimension('GLOBAL', 'PROD');

>execute cwm2_olap_delete.delete_dimension
     ('GLOBAL', 'PROD', 'no','yes', 'yes');

     EXECUTE cwm2_olap_dimension.Drop_Dimension('GLOBAL', 'PROD');
     AMD-00003  dropped Dimension "GLOBAL.PROD"
```

DELETE_MEASURE_CATALOG Procedure

This procedure can be used to delete a measure folder or group of measure folders in the OLAP Catalog. You can also use this procedure to list the commands that will delete the measure folders. You can choose to execute these commands or simply list them, without actually deleting the measure folders. See "Using a Command Report" on page 10-2.

You can identify a group of measure folders by specifying wildcard characters in the measure_folder_name parameter. See "Using Wildcards to Identify Metadata Entities" on page 10-1.

OLAP Catalog measure folders are displayed in the view ALL_OLAP2_CATALOGS.

Syntax

```
DELETE_MEASURE_CATALOG (
          measure_folder_name     IN   VARCHAR2,
          delete_report           IN   VARCHAR2,
          delete_measure_catalog  IN   VARCHAR2);
```

Parameters

Table 10–4 DELETE_MEASURE_CATALOG Procedure Parameters

Parameter	Description
measure_folder_name	The name of the measure folder. See "Using Wildcards to Identify Metadata Entities" on page 10-1.
delete_report	Whether or not to list the commands that will delete the measure folders. Specify 'YES' to list the commands. Otherwise specify 'NO'. To display the output on the screen, use the CWM2_OLAP_MANAGER.SET_ECHO_ON procedure. To send the output to a file, use the CWM2_OLAP_MANAGER.BEGIN_LOG procedure. See "Directing Output" on page 2-13.
delete_measure_catalog	Whether or not to actually delete the measure folders. Specify 'YES' to delete the measure folder. Otherwise specify 'NO'.

Example

The following example deletes the two measure folders whose names start with 'TEMP'.

```
>set serveroutput on size 1000000
>execute cwm2_olap_manager.set_echo_on;
>execute cwm2_olap_delete.delete_measure_catalog
     ('TEMP%', 'no', 'yes');

     AMD-0003 dropped Catalog "Temp1"
     AMD-0003 dropped Catalog "Temp2"
```

DELETE_OLAP_CATALOG Procedure

This procedure can be used to delete all the metadata in the OLAP Catalog. You can also use this procedure to list the commands that will drop each metadata entity. You can choose to execute these commands or simply list them, without actually deleting the metadata. See "Using a Command Report" on page 10-2.

OLAP Catalog metadata is displayed in the OLAP Catalog metadata views, described in Chapter 5.

Syntax

```
DELETE_OLAP_CATALOG (
          delete_cwm1_dimension   IN   VARCHAR2,
          delete_report           IN   VARCHAR2,
          delete_olap_catalog     IN   VARCHAR2);
```

Parameters

Table 10–5 DELETE_OLAP_CATALOG Procedure Parameters

Parameter	Description
delete_cwm1_dimension	Whether or not to delete the Oracle dimension object associated with each CWM1 dimension. Specify 'YES' to delete the Oracle dimension object. Otherwise specify 'NO'. This parameter has no effect on CWM2 dimensions.
delete_report	Whether or not to list the commands that will delete the metadata. Specify 'YES' to list the commands. Otherwise specify 'NO'. To display the output on the screen, use the CWM2_OLAP_MANAGER.SET_ECHO_ON procedure. To send the output to a file, use the CWM2_OLAP_MANAGER.BEGIN_LOG procedure. See "Directing Output" on page 2-13 for more information.
delete_olap_catalog	Whether or not to actually delete all the metadata in the OLAP Catalog. Specify 'YES' to delete the metadata. Otherwise specify 'NO'.

Example

The following example deletes all the metadata in the OLAP Catalog without generating a command report. Any associated Oracle dimension objects are not deleted.

```
>set serveroutput on size 1000000
>execute cwm2_olap_manager.set_echo_on;
>execute cwm2_olap_delete.delete_olap_catalog('no', 'no', 'yes');
```

11

CWM2_OLAP_DIMENSION

The CWM2_OLAP_DIMENSION package provides procedures for managing dimensions.

> **See Also:** Chapter 2, "Creating OLAP Catalog Metadata with CWM2"

This chapter discusses the following topics:

- Understanding Dimensions
- Example: Creating a CWM2 Dimension
- Summary of CWM2_OLAP_DIMENSION Subprograms

Understanding Dimensions

A dimension is an OLAP Catalog metadata entity. This means that it is a logical object, identified by name and owner, within the OLAP Catalog. Logical OLAP dimensions are fully described in.

> **Note:** Dimensions in CWM2 map directly to columns in dimension tables and have no relationship to Oracle database dimension objects.

Use the procedures in the CWM2_OLAP_DIMENSION package to create, drop, and lock CWM2 dimension entities and to specify descriptive information for display purposes. To fully define a CWM2 dimension, follow the steps listed in "Creating a Dimension" on page 2-2.

> **See Also:** *Oracle OLAP Application Developer's Guide* for more information on dimensions and the OLAP Catalog metadata model.

Example: Creating a CWM2 Dimension

The following statement creates a CWM2 dimension entity, PRODUCT_DIM, in the JSMITH schema. The display name is Product, and the plural name is Products. The short description is Prod, and the description is Product.

```
execute cwm2_olap_dimension.create_dimension
     ('JSMITH', 'PRODUCT_DIM', 'Product', 'Products', 'Prod', 'Product');
```

The following statements change the short description to Product and the long description to Product Dimension.

```
execute cwm2_olap_dimension.set_short_description
```

```
     ('JSMITH', 'PRODUCT_DIM', 'Product');
execute cwm2_olap_dimension.set_description
     ('JSMITH', 'PRODUCT_DIM', 'Product Dimension');
```

Summary of CWM2_OLAP_DIMENSION Subprograms

Table 11–1 CWM2_OLAP_DIMENSION Subprograms

Subprogram	Description
CREATE_DIMENSION Procedure on page 11-4	Creates a dimension.
DROP_DIMENSION Procedure on page 11-4	Drops a dimension.
LOCK_DIMENSION Procedure on page 11-5	Locks the dimension metadata for update.
SET_DEFAULT_DISPLAY_HIERARCHY Procedure on page 11-5	Sets the default hierarchy for a dimension.
SET_DESCRIPTION Procedure on page 11-5	Sets the description for a dimension.
SET_DIMENSION_NAME Procedure on page 11-6	Sets the name of a dimension.
SET_DISPLAY_NAME Procedure on page 11-6	Sets the display name for a dimension.
SET_PLURAL_NAME Procedure on page 11-7	Sets the plural name for a dimension.
SET_SHORT_DESCRIPTION Procedure on page 11-7	Sets the short description for a dimension.

CREATE_DIMENSION Procedure

This procedure creates a new dimension entity in the OLAP Catalog.

By default the new dimension is a normal dimension, but you can specify the value `TIME` for the `dimension_type` parameter to create a time dimension.

Descriptions and display properties must also be established as part of dimension creation. Once the dimension has been created, you can override these properties by calling other procedures in this package.

Syntax

```
CREATE_DIMENSION (
          dimension_owner       IN   VARCHAR2,
          dimension_name        IN   VARCHAR2,
          display_name          IN   VARCHAR2,
          plural_name           IN   VARCHAR2,
          short_description     IN   VARCHAR2,
          description           IN   VARCHAR2,
          dimension_type        IN   VARCHAR2 DEFAULT NULL);
```

Parameters

Table 11–2 CREATE_DIMENSION Procedure Parameters

Parameter	Description
`dimension_owner`	Owner of the dimension.
`dimension_name`	Name of the dimension.
`display_name`	Display name for the dimension.
`plural_name`	Plural name for the dimension.
`short_description`	Short description of the dimension.
`description`	Description of the dimension.
`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.

DROP_DIMENSION Procedure

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

Syntax

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

Parameters

Table 11–3 DROP_DIMENSION Procedure Parameters

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

LOCK_DIMENSION Procedure

This procedure locks the dimension metadata for update by acquiring a database lock on the row that identifies the dimension in the CWM2 model table.

Syntax

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

Parameters

Table 11–4 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.

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 11–5 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.

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 11–6 SET_DESCRIPTION Procedure Parameters

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

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 11–7 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.

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 11–8 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.

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 11–9 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.

SET_SHORT_DESCRIPTION Procedure

This procedure sets the short description for a dimension.

Syntax

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

Parameters

Table 11–10 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.

12

CWM2_OLAP_DIMENSION_ATTRIBUTE

The CWM2_OLAP_DIMENSION_ATTRIBUTE package provides procedures managing dimension attributes.

> **See Also:** Chapter 2, "Creating OLAP Catalog Metadata with CWM2".

This chapter discusses the following topics:

- Understanding Dimension Attributes
- Example: Creating a Dimension Attribute
- Summary of CWM2_OLAP_DIMENSION_ATTRIBUTE Subprograms

Understanding Dimension Attributes

A dimension attribute is an OLAP Catalog metadata entity. This means that it is a logical object, identified by name and owner, within the OLAP Catalog.

Dimension attributes define sets of level attributes for a dimension. Dimension attributes may include level attributes for some or all of the dimension's levels. For time dimensions, the dimension attributes `end date` and `time span` must be defined for all levels.

Use the procedures in the CWM2_OLAP_DIMENSION_ATTRIBUTE package to create, drop, and lock dimension attributes and to specify descriptive information for display purposes.

Several dimension attribute names are reserved, because they have special significance within CWM2. The level attributes comprising a reserved dimension attribute will be mapped to columns containing specific information. The reserved dimension attributes are listed in Table 12–1.

Table 12–1 Reserved Dimension Attributes

Dimension Attribute	Description
`Long Description`	A long description of the dimension member.
`Short Description`	A short description of the dimension member.
`End Date`	For a time dimension, the last date in a time period. (Required)
`Time Span`	For a time dimension, the number of days in a time period. (Required)
`Prior Period`	For a time dimension, the time period before this time period.

Table 12–1 (Cont.) Reserved Dimension Attributes

Dimension Attribute	Description
`Year Ago Period`	For a time dimension, the period a year before this time period.
`ET Key`	For an embedded total dimension, the embedded total key, which identifies the dimension member. (Required)
`Parent ET Key`	For an embedded total dimension, the dimension member that is the parent of the ET key. (Required)
`Grouping ID`	For an embedded total dimension, the grouping ID (GID), which identifies the hierarchical level for a row of the dimension table. (Required)
`Parent Grouping ID`	For an embedded total dimension, the dimension member that is the parent of the grouping ID. (Required)

The parent dimension must already exist before you can create dimension attributes for it. To fully define a dimension, follow the steps listed in "Creating a Dimension" on page 2-2.

See Also:

- Chapter 16, "CWM2_OLAP_LEVEL_ATTRIBUTE"
- *Oracle OLAP Application Developer's Guide* for more information about dimension attributes and the OLAP Catalog metadata model

Example: Creating a Dimension Attribute

The following statement creates a dimension attribute, `PRODUCT_DIM_BRAND`, for the `PRODUCT_DIM` dimension in the `JSMITH` schema. The display name is `Brand`. The short description is `Brand Name`, and the description is `Product Brand Name`.

```
execute cwm2_olap_dimension_attribute.create_dimension_attribute
     ('JSMITH', 'PRODUCT_DIM', 'PRODUCT_DIM_BRAND',
      'Brand', 'Brand Name', 'Product Brand Name');
```

The following statement creates a dimension attribute, `'Short Description'`, for the `PRODUCT_DIM` dimension in the `JSMITH` schema. `Short Description` is a reserved dimension attribute.

```
execute cwm2_olap_dimension_attribute.create_dimension_attribute
     ('JSMITH', 'PRODUCT_DIM', 'Short Description',
      'Short Product Names', 'Short Desc Product',
      'Short Name of Products', TRUE);
```

Summary of CWM2_OLAP_DIMENSION_ATTRIBUTE Subprograms

Table 12–2 CWM2_OLAP_DIMENSION_ATTRIBUTE Subprograms

Subprogram	Description
CREATE_DIMENSION_ATTRIBUTE Procedure on page 12-4	Creates a dimension attribute.
DROP_DIMENSION_ATTRIBUTE Procedure on page 12-5	Drops a dimension attribute.
LOCK_DIMENSION_ATTRIBUTE Procedure on page 12-5	Locks the dimension attribute for update.
SET_DESCRIPTION Procedure on page 12-6	Sets the description for a dimension attribute.
SET_DIMENSION_ATTRIBUTE_NAME Procedure on page 12-6	Sets the name of a dimension attribute.
SET_DISPLAY_NAME Procedure on page 12-7	Sets the display name for a dimension attribute.
SET_SHORT_DESCRIPTION Procedure on page 12-7	Sets the short description for a dimension attribute.

CREATE_DIMENSION_ATTRIBUTE Procedure

This procedure creates a new dimension attribute.

If the dimension attribute is reserved, you can specify the reserved name as the dimension attribute name or as a type associated with a name that you specify. The reserved dimension attributes are listed in Table 12–1, " Reserved Dimension Attributes".

If the dimension attribute name should be reserved for mapping specific groups of level attributes, you can set the `RESERVED_DIMENSION_ATTRIBUTE` argument to `TRUE`. For more information, see Table 12–1, " Reserved Dimension Attributes".

Descriptions and display properties must also be established as part of dimension attribute creation. Once the dimension attribute has been created, you can override these properties by calling other procedures in this package.

Syntax

```
CREATE_DIMENSION_ATTRIBUTE (
          dimension_owner               IN   VARCHAR2,
          dimension_name                IN   VARCHAR2,
          dimension_attribute_name      IN   VARCHAR2,
          display_name                  IN   VARCHAR2,
          short_description             IN   VARCHAR2,
          description                   IN   VARCHAR2,
            type                        IN   VARCHAR2             );
            use_name_as_type            IN   BOOLEAN DEFAULT FALSE);
```

Parameters

Table 12–3 CREATE_DIMENSION_ATTRIBUTE Procedure Parameters

Parameter	Description
`dimension_owner`	Owner of the dimension.
`dimension_name`	Name of the dimension.
`dimension_attribute_name`	Name of the dimension attribute.
`display_name`	Display name for the dimension attribute.
`short_description`	Short description of the dimension attribute.

Table 12–3 (Cont.) CREATE_DIMENSION_ATTRIBUTE Procedure Parameters

Parameter	Description
description	Description of the dimension attribute.
type *or* use_name_as_type	This argument can be one of the following: ▪ type a VARCHAR2 argument whose value is one of the reserved names from Table 12–1, " Reserved Dimension Attributes". Specify this argument if you want to create your own name for a reserved dimension attribute. ▪ use_name_as_type a BOOLEAN argument that defaults to FALSE. This argument specifies whether or not the dimension attribute name is a reserved name. If this argument is TRUE, the value of the dimension_attribute_name argument must be a reserved name from Table 12–1, " Reserved Dimension Attributes". If you do not specify a value for this argument, the dimension attribute is not reserved.

DROP_DIMENSION_ATTRIBUTE Procedure

This procedure drops a dimension attribute.

Syntax

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

Parameters

Table 12–4 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.

LOCK_DIMENSION_ATTRIBUTE Procedure

This procedure locks the dimension attribute for update by acquiring a database lock on the row that identifies the dimension attribute in the CWM2 model table.

Syntax

```
LOCK_DIMENSION_ATTRIBUTE (
           dimension_owner              IN   VARCHAR2,
           dimension_name               IN   VARCHAR2,
           dimension_attribute_name     IN   VARCHAR2,
           wait_for_lock                IN   BOOLEAN DEFAULT FALSE);
```

Parameters

Table 12–5 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.

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 12–6 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.

SET_DIMENSION_ATTRIBUTE_NAME Procedure

This procedure sets the name for a dimension attribute.

If the dimension attribute is reserved, you can specify the reserved name as the dimension attribute name or as a type associated with a name that you specify. The reserved dimension attributes are listed in Table 12–1, " Reserved Dimension Attributes".

Syntax

```
SET_DIMENSION_ATTRIBUTE_NAME (
          dimension_owner                IN   VARCHAR2,
          dimension_name                 IN   VARCHAR2,
          dimension_attribute_name       IN   VARCHAR2,
          set_dimension_attribute_name   IN   VARCHAR2,
            type                         IN   VARCHAR2            );
            use_name_as_type             IN   BOOLEAN DEFAULT FALSE);
```

Parameters

Table 12–7 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.
type *or* use_name_as_type	This argument can be one of the following: ■ type a VARCHAR2 argument whose value is one of the reserved names from Table 12–1, " Reserved Dimension Attributes". Specify this argument if you want to create your own name for a reserved dimension attribute. ■ use_name_as_type a BOOLEAN argument that defaults to FALSE. This argument specifies whether or not the dimension attribute name is a reserved name. If this argument is TRUE, the value of the dimension_attribute_name argument must be a reserved name from Table 12–1, " Reserved Dimension Attributes". If you do not specify a value for this argument, the dimension attribute is not reserved.

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 12–8 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.

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 12–9 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.

13

CWM2_OLAP_EXPORT

The CWM2_OLAP_EXPORT package provides procedures you can use to export OLAP Catalog metadata and its underlying fact tables and dimension tables. You can rebuild the metadata and import the data within the same database instance or in a different database instance.

See Also:

- "Original Export and Import" in *Oracle Database Utilities*.
- Chapter 10, "CWM2_OLAP_DELETE".

This chapter discusses the following topics:

Exporting and Importing OLAP Catalog Metadata

You can use the CWM2_OLAP_EXPORT package to export individual cubes or dimensions or the entire contents of the OLAP Catalog. CWM2_OLAP_EXPORT exports CWM2 metadata created by the CWM2 PL/SQL packages and CWM1 metadata created by Oracle Enterprise Manager.

You can use CWM2_OLAP_EXPORT if your mapped data is stored in relational tables or views of relational tables. If your data is stored in analytic workspaces, use the OLAP DML to export and import the contents of the workspace. See "Procedure: Import a workspace from a 9i Database into a 10g Database" on page 24-3.

Procedures in CWM2_OLAP_EXPORT produce a metadata command script and an Export parameter file. The metadata command script contains the CWM1, CWM2, and Oracle Database commands that build the metadata. The Export parameter file can be used with the Oracle Export utility to export the dimension tables and fact tables that underlie the metadata.

Exporting and importing OLAP Catalog metadata is a four-step process:

1. Run the CWM2_OLAP_EXPORT procedure, specifying a metadata command script file and an Export parameter file.
2. Run the Oracle Export utility, using the Export parameter file you produced in Step 1. The Export utility will create an Export dump file.

3. In the database instance where you want to re-create the data and metadata, run the Oracle Import utility using the Export dump file you produced in Step 2. The Import utility will import the underlying dimension tables and fact tables.
4. After running the Oracle Import utility, run the metadata command script you produced in Step 1. This script will rebuild the metadata that maps to the underlying dimension tables and fact tables.

> **Note:** The database in which you re-create your OLAP Catalog metadata must be OLAP-enabled. You can only use `CWM2_OLAP_EXPORT` to replicate your OLAP Catalog metadata within an environment where the OLAP Catalog is already defined.

Rebuilding OLAP Catalog Metadata

To rebuild the OLAP Catalog metadata for a relational data source, you can export the data and metadata, delete it, then import it. Use the `CWM2_OLAP_EXPORT` package and the Oracle Export utility to do the export. Use `CWM2_OLAP_DELETE` to delete the metadata. Drop the tables, then use the Oracle import utility to do the import. See Chapter 10, "CWM2_OLAP_DELETE".

To rebuild analytic workspaces, use the OLAP DML to export the contents of the workspace to an EIF file, then import it in a new workspace. See "Procedure: Import a workspace from a 9i Database into a 10g Database" on page 24-3. If you are running in Oracle9*i* compatibility mode, you will need to re-enable the workspaces and re-create the metadata for the workspaces. See "Enabling Relational Access" on page 1-17.

Using the Oracle Export and Import Utilities

The `CWM2_OLAP_EXPORT` package works with the Export and Import utilities that are invoked with the `exp` and `imp` commands. In *Oracle Database Utilities* these are called the original Export and Import utilities to differentiate them from the new Data Pump Export and Import utilities available with Oracle Database 10*g*. The `CWM2_OLAP_EXPORT` package was not designed to work with the new Data Pump Export and Import utilities.

The original Export and Import utilities provide a simple way for you to transfer data objects between Oracle databases, even if they reside on platforms with different hardware and software configurations. When you run Export against an Oracle database, objects (such as tables) are extracted, followed by their related objects (such as indexes, comments, and grants), if any. The extracted data is written to an export dump file. The Import utility reads the object definitions and table data from the dump file.

The Export parameter file created by `CWM2_OLAP_EXPORT` specifies the tables where your dimension and fact data are stored. The Export utility supports many options and parameters. Refer to *Oracle Database Utilities* for specific information about exporting and importing tables with `exp` and `imp`.

Using Wildcards to Identify Metadata Entities

You can use wildcard characters to export cubes and dimensions whose names meet certain criteria.

Wildcard characters are the underscore "_" and the percent sign "%". An underscore replaces any single character, and a percent sign replaces any zero or more characters. An underscore, but not a percent sign, is also a legal character in a metadata owner or

entity name. Any underscore character in the owner or entity name is treated as a wildcard, unless you precede it with a backslash "\" which acts as an escape character.

For example, the following command exports all the cubes belonging to the owner 'GLOBAL'.

```
>execute cwm2_olap_export.export_cube('GLOBAL', '%', '/scripts_dir',
                                      'global_cmd_file', 'global_tbl_file');
```

The following command exports all the cubes in the GLOBAL schema whose names start with 'a'.

```
>execute cwm2_olap_export.export_cube('GLOBAL', 'a%', '/scripts_dir',
                                      'global_cmd_file', 'global_tbl_file');
```

If your database includes users 'TESTUSER1' and 'TESTUSER2', you could export the 'TEST' cube belonging to each of these users with the following command.

```
>execute cwm2_olap_export.export_cube('TESTUSER_', 'TEST', '/scripts_dir',
                                      'global_cmd_file', 'global_tbl_file');
```

If your database includes users 'TEST_USER1' and 'TEST_USER2', you could export the 'TEST' cube belonging to each of these users with the following command.

```
>execute cwm2_olap_export.export_cube('TEST/_USER_', 'TEST', '/scripts_dir',
                                      'global_cmd_file', 'global_tbl_file');
```

Creating a Metadata Command Script

Each procedure in the CWM2_OLAP_EXPORT package accepts a parameter that identifies a metadata command script file. The contents of this script are the commands that build the metadata.

The commands in the command script file may be CWM2 procedure calls, CWM1 procedure calls, CREATE DIMENSION statements to re-create the dimension objects associated with CWM1 dimensions, and CREATE VIEW statements to re-create the views when the metadata is mapped to views of the source dimension tables and fact tables.

You can create a metadata command script without creating an Export parameter file. However, if you run the script in a different database without importing or re-creating the source tables, the metadata will be invalid.

Example 13–1 shows the metadata command script for the GLOBAL.PRODUCT dimension. This dimension was created in Enterprise Manager, therefore it has an associated Oracle dimension object and its metadata was defined using the CWM1 APIs.

Example 13–1 Metadata Command Script for GLOBAL.PRODUCT

The following command creates a metadata command script for the GLOBAL.PRODUCT dimension. It does not create an Export parameter file.

```
>EXECUTE cwm2_olap_export.export_dimension
            ('GLOBAL', 'PRODUCT', '/myscripts', 'GLOBALPROD_CMD_SCRIPT.SQL');
```

To re-create the metadata, transfer the GLOBALPROD_CMD_SCRIPT.SQL file to a directory that can be accessed by the database. In SQL*Plus, navigate to this directory and run the script with a command like the following.

```
>@GLOBALPROD_CMD_SCRIPT.SQL
```

The contents of GLOBALPROD_CMD_SCRIPT.SQL are shown as follows.

```
CREATE DIMENSION GLOBAL.PRODUCT
  LEVEL CLASS IS (GLOBAL.PRODUCT_DIM.CLASS_ID)
  LEVEL FAMILY IS (GLOBAL.PRODUCT_DIM.FAMILY_ID)
  LEVEL ITEM IS (GLOBAL.PRODUCT_DIM.ITEM_ID)
  LEVEL TOTAL_PRODUCT IS (GLOBAL.PRODUCT_DIM.TOTAL_PRODUCT_ID)
  HIERARCHY PRODUCT_ROLLUP
  ( ITEM CHILD OF
    FAMILY CHILD OF
    CLASS CHILD OF
    FAMILY CHILD OF
    CLASS CHILD OF
    TOTAL_PRODUCT
  )
  ATTRIBUTE CLASS DETERMINES
  ( CLASS_DSC
  )
  ATTRIBUTE FAMILY DETERMINES
  ( FAMILY_DSC
  )
  ATTRIBUTE ITEM DETERMINES
  ( ITEM_DSC
  )
  ATTRIBUTE TOTAL_PRODUCT DETERMINES
  ( TOTAL_PRODUCT_DSC
  )
  ATTRIBUTE ITEM DETERMINES
  ( ITEM_PACKAGE_ID
  )
  ATTRIBUTE CLASS DETERMINES
  ( CLASS_DSC
  )
  ATTRIBUTE FAMILY DETERMINES
  ( FAMILY_DSC
  )
  ATTRIBUTE ITEM DETERMINES
  ( ITEM_DSC
  )
  ATTRIBUTE TOTAL_PRODUCT DETERMINES
  ( TOTAL_PRODUCT_DSC
  )
  ;
EXECUTE cwm_olap_dimension.Set_Description('GLOBAL', 'PRODUCT', '');
EXECUTE cwm_olap_dimension.Set_Display_Name('GLOBAL', 'PRODUCT', 'Product');
EXECUTE cwm_olap_dimension.Set_Plural_Name('GLOBAL', 'PRODUCT', 'PRODUCT');
EXECUTE cwm_olap_dim_attribute.Create_Dimension_Attribute
           ('GLOBAL', 'PRODUCT', 'Long_Description', 'Long Description', '');
EXECUTE cwm_olap_dim_attribute.Create_Dimension_Attribute
           ('GLOBAL', 'PRODUCT', 'Package', 'Package', '');
EXECUTE cwm_olap_dim_attribute.Create_Dimension_Attribute
           ('GLOBAL', 'PRODUCT', 'Short_Description', 'Short Description', '');
EXECUTE cwm_olap_hierarchy.Set_Description('GLOBAL', 'PRODUCT', 'PRODUCT_ROLLUP', '');
EXECUTE cwm_olap_hierarchy.Set_Display_Name('GLOBAL', 'PRODUCT', 'PRODUCT_ROLLUP', 'Product
Rollup');
EXECUTE cwm_olap_dimension.Set_Default_Display_Hierarchy('GLOBAL', 'PRODUCT', 'PRODUCT_ROLLUP');
EXECUTE cwm_olap_level.Set_Description('GLOBAL', 'PRODUCT', 'CLASS', '');
EXECUTE cwm_olap_level.Set_Display_Name('GLOBAL', 'PRODUCT', 'CLASS', 'Class');
EXECUTE cwm_olap_level.Set_Description('GLOBAL', 'PRODUCT', 'FAMILY', '');
EXECUTE cwm_olap_level.Set_Display_Name('GLOBAL', 'PRODUCT', 'FAMILY', 'Family');
EXECUTE cwm_olap_level.Set_Description('GLOBAL', 'PRODUCT', 'ITEM', '');
EXECUTE cwm_olap_level.Set_Display_Name('GLOBAL', 'PRODUCT', 'ITEM', 'Item');
```

```
EXECUTE cwm_olap_level.Set_Description('GLOBAL', 'PRODUCT', 'TOTAL_PRODUCT', '');
EXECUTE cwm_olap_level.Set_Display_Name('GLOBAL', 'PRODUCT', 'TOTAL_PRODUCT', 'Total Product');
EXECUTE cwm_olap_level_attribute.Set_Name('GLOBAL', 'PRODUCT', 'CLASS', 'CLASS_DSC', 'CLASS_DSC');
EXECUTE cwm_olap_dim_attribute.Add_Level_Attribute
           ('GLOBAL','PRODUCT','Short_Description','CLASS','CLASS_DSC');
EXECUTE cwm_olap_level_attribute.Set_Description('GLOBAL', 'PRODUCT', 'CLASS', 'CLASS_DSC', '');
EXECUTE cwm_olap_level_attribute.Set_Display_Name('GLOBAL', 'PRODUCT', 'CLASS', 'CLASS_DSC', '');
EXECUTE cwm_olap_level_attribute.Set_Name
           ('GLOBAL', 'PRODUCT', 'FAMILY', 'FAMILY_DSC', 'FAMILY_DSC');
EXECUTE cwm_olap_dim_attribute.Add_Level_Attribute
           ('GLOBAL','PRODUCT','Short_Description','FAMILY','FAMILY_DSC');
EXECUTE cwm_olap_level_attribute.Set_Description
           ('GLOBAL', 'PRODUCT', 'FAMILY', 'FAMILY_DSC', '');
EXECUTE cwm_olap_level_attribute.Set_Display_Name
           ('GLOBAL', 'PRODUCT', 'FAMILY', 'FAMILY_DSC', '');
EXECUTE cwm_olap_level_attribute.Set_Name('GLOBAL', 'PRODUCT', 'ITEM', 'ITEM_DSC', 'ITEM_DSC');
EXECUTE cwm_olap_dim_attribute.Add_Level_Attribute('GLOBAL','PRODUCT','Short_
Description','ITEM','ITEM_DSC');
EXECUTE cwm_olap_level_attribute.Set_Description('GLOBAL', 'PRODUCT', 'ITEM', 'ITEM_DSC', '');
EXECUTE cwm_olap_level_attribute.Set_Display_Name('GLOBAL', 'PRODUCT', 'ITEM', 'ITEM_DSC', '');
EXECUTE cwm_olap_level_attribute.Set_Name('GLOBAL', 'PRODUCT', 'ITEM', 'ITEM_PACKAGE_ID', 'ITEM_
PACKAGE_ID');
EXECUTE cwm_olap_dim_attribute.Add_Level_Attribute('GLOBAL', 'PRODUCT', 'Package', 'ITEM', 'ITEM_
PACKAGE_ID');
EXECUTE cwm_olap_level_attribute.Set_Description('GLOBAL', 'PRODUCT', 'ITEM', 'ITEM_PACKAGE_ID',
'');
EXECUTE cwm_olap_level_attribute.Set_Display_Name('GLOBAL', 'PRODUCT', 'ITEM', 'ITEM_PACKAGE_ID',
'');
EXECUTE cwm_olap_level_attribute.Set_Name
          ('GLOBAL','PRODUCT','TOTAL_PRODUCT','TOTAL_PRODUCT_DSC','TOTAL_PRODUCT_DSC');
EXECUTE cwm_olap_dim_attribute.Add_Level_Attribute
          ('GLOBAL','PRODUCT','Short_Description','TOTAL_PRODUCT','TOTAL_PRODUCT_DSC');
EXECUTE cwm_olap_level_attribute.Set_Description
          ('GLOBAL','PRODUCT','TOTAL_PRODUCT', 'TOTAL_PRODUCT_DSC', '');
EXECUTE cwm_olap_level_attribute.Set_Display_Name
           ('GLOBAL', 'PRODUCT','TOTAL_PRODUCT','TOTAL_PRODUCT_DSC','');
```

Creating an Export Parameter File

Each procedure in the CWM2_OLAP_EXPORT package accepts a parameter that identifies an Export parameter file. The contents of this file are a series of comments and a table specification that can be used by the Oracle Export utility.

You can create an Export parameter file without creating a metadata command script. You can use the parameter file to export and import the base tables, but without a metadata command script you will not be able to restore the metadata.

Example 13–2 shows the Export parameter file for the GLOBAL.PRODUCT dimension.

Example 13–2 Export Parameter File for GLOBAL.PRODUCT

The following command creates an Export parameter file for the GLOBAL.PRODUCT dimension. It does not create a metadata command script file.

```
>execute cwm2_olap_export.export_dimension
            ('GLOBAL', 'PRODUCT','/myuser/scripts',' ' ,'GLOBALPROD_EXP_PARAM.DAT');
```

To export the dimension table used by GLOBAL.PRODUCT, run the Export utility in SQL*Plus using a command like the following.

```
>exp username/password PARFILE=GLOBALPROD_EXP_PARAM.DAT
```

The contents of GLOBALPROD_EXP_PARAM.DAT are shown as follows.

```
#Export Dimension: GLOBAL.PRODUCT   Directory: /myuser/scripts   Command File:
#Table File: GLOBALPROD_EXP_PARAM.DAT
#
#ORACLE RDBMS EXPORT UTILITY PARFILE
#
# Cube "GLOBAL.PRICE_CUBE"
#   Dimension "GLOBAL.PRODUCT" Mapped to Table "GLOBAL.PRODUCT_DIM"
#
# Cube "GLOBAL.UNITS_CUBE"
#   Dimension "GLOBAL.PRODUCT" Mapped to Table "GLOBAL.PRODUCT_DIM"
TABLES = (
GLOBAL.PRODUCT_DIM
)
```

To re-create the dimension table for GLOBAL.PRODUCT, transfer the dump file generated by the Export utility (by default, expdat.dmp) to a directory that can be accessed by the database. In SQL*Plus, navigate to this directory and run the Import utility with a command like the following.

```
>imp username/password FILE=expdat.dmp
```

Summary of CWM2_OLAP_Export Subprograms

Table 13–1 CWM2_OLAP_DELETE

Subprogram	Description
EXPORT_CUBE Procedure on page 13-8	Exports a cube in the OLAP Catalog.
EXPORT_DIMENSION Procedure on page 13-8	Exports a dimension in the OLAP Catalog.
EXPORT_OLAP_CATALOG Procedure on page 13-9	Exports all the metadata in the OLAP Catalog.

EXPORT_CUBE Procedure

EXPORT_CUBE produces a metadata command script and an Export parameter file for a cube that is based on relational tables.

You can use the Export parameter file to export the underlying fact and dimension tables to a dump file, then you can import the dump file to re-create the tables. You can run the metadata command script to rebuild the metadata that maps to the tables. See "Exporting and Importing OLAP Catalog Metadata" on page 13-1.

You can identify a group of cubes to export by specifying wildcard characters in the cube_owner and cube_name parameters. See "Using Wildcards to Identify Metadata Entities" on page 13-2.

EXPORT_CUBE includes each of the cube's dimensions. You do not have to export the dimensions separately.

OLAP Catalog cubes are displayed in the view ALL_OLAP2_CUBES.

Syntax

```
EXPORT_CUBE (
          cube_owner                    IN   VARCHAR2,
          cube_name                     IN   VARCHAR2,
          directory_name                IN   VARCHAR2,
          metadata_command_file_name    IN   VARCHAR2,
          export_parameter_file_name    IN   VARCHAR2);
```

Parameters

Table 13–2 EXPORT_CUBE Procedure Parameters

Parameter	Description
cube_owner	The owner of the cube. See "Using Wildcards to Identify Metadata Entities" on page 13-2.
cube_name	The name of the cube. See "Using Wildcards to Identify Metadata Entities" on page 13-2.
directory_name	The directory where the metadata command file and the Export parameter file will be written.
metadata_command_file_name	The name of the metadata command file that will contain the SQL and PL/SQL commands for rebuilding the metadata.
export_parameter__file_name	The name of the Export parameter file that will identify the dimension tables and fact tables for the Oracle Export utility.

Example

See "Creating a Metadata Command Script" and "Creating an Export Parameter File" on page 13-5. These examples illustrate the process of exporting and importing a dimension. The process is exactly the same for a cube.

EXPORT_DIMENSION Procedure

EXPORT_DIMENSION produces a metadata command script and an Export parameter file for a dimension that is based on relational dimension tables.

You can use the Export parameter file to export the underlying dimension tables to a dump file, then you can import the dump file to re-create the tables. You can run the

metadata command script to rebuild the metadata that maps to the tables. See "Exporting and Importing OLAP Catalog Metadata" on page 13-1.

You can identify a group of dimensions to export by specifying wildcard characters in the `dimension_owner` and `dimension_name` parameters. See "Using Wildcards to Identify Metadata Entities" on page 13-2.

Dimensions are exported along with cubes. You only need to use `EXPORT_DIMENSION` if the dimension does not participate in a cube or if the owning cube will not be exported.

OLAP Catalog dimensions are displayed in the view ALL_OLAP2_DIMENSIONS.

Syntax

```
EXPORT_DIMENSION (
         dimension_owner              IN   VARCHAR2,
         dimension_name               IN   VARCHAR2,
         directory_name               IN   VARCHAR2,
         metadata_command_file_name   IN   VARCHAR2,
         export_parameter_file_name   IN   VARCHAR2);
```

Parameters

Table 13–3 EXPORT_DIMENSION Procedure Parameters

Parameter	Description
`dimension_owner`	The owner of the dimension. See "Using Wildcards to Identify Metadata Entities" on page 13-2.
`dimension_name`	The name of the dimension. See "Using Wildcards to Identify Metadata Entities" on page 13-2.
`directory_name`	The directory where the metadata command file and the Export parameter file will be written.
`metadata_command_file_name`	The name of the metadata command file that will contain the SQL and PL/SQL commands for rebuilding the metadata.
`export_parameter__file_name`	The name of the Export parameter file that will identify the dimension tables for the Oracle Export utility.

Example

See "Creating a Metadata Command Script" and "Creating an Export Parameter File" on page 13-5.

EXPORT_OLAP_CATALOG Procedure

`EXPORT_OLAP_CATALOG` produces a metadata command script and an Export parameter file for all the metadata (both `CWM1` and `CWM2`) in the OLAP Catalog.

You can use the Export parameter file to export the underlying fact and dimension tables to a dump file, then you can import the dump file to re-create the tables. You can run the metadata command script to rebuild the metadata that maps to the tables. See "Exporting and Importing OLAP Catalog Metadata" on page 13-1.

OLAP Catalog metadata is displayed in the OLAP Catalog metadata views, described in Chapter 5.

Syntax

```
EXPORT_OLAP_CATALOG (
```

```
directory_name               IN   VARCHAR2,
metadata_command_file_name   IN   VARCHAR2,
export_parameter_file_name   IN   VARCHAR2);
```

Parameters

Table 13–4 EXPORT_OLAP_CATALOG Procedure Parameters

Parameter	Description
directory_name	The directory where the metadata command file and the Export parameter file will be written.
metadata_command_ file_name	The name of the metadata command file that will contain the SQL and PL/SQL commands for rebuilding the metadata
export_parameter__ file_name	The name of the Export parameter file that will identify the fact and dimension tables for the Oracle Export utility.

Example

See "Creating a Metadata Command Script" and "Creating an Export Parameter File" on page 13-5. These examples illustrate the process of exporting and importing a dimension. To use EXPORT_OLAP_CATALOG, follow the same procedure without specifying a dimension name.

14

CWM2_OLAP_HIERARCHY

The CWM2_OLAP_HIERARCHY package provides procedures managing hierarchies.

> **See Also:** Chapter 2, "Creating OLAP Catalog Metadata with CWM2".

This chapter discusses the following topics:

- Understanding Hierarchies
- Example: Creating a Hierarchy
- Summary of CWM2_OLAP_HIERARCHY Subprograms

Understanding Hierarchies

A hierarchy is an OLAP Catalog metadata entity. This means that it is a logical object, identified by name and owner, within the OLAP Catalog.

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

Use the procedures in the CWM2_OLAP_HIERARCHY package to create, drop, and lock hierarchies and to specify descriptive information for display purposes.

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

> **See Also:**
>
> - Chapter 15, "CWM2_OLAP_LEVEL"
> - *Oracle OLAP Application Developer's Guide* for more information about hierarchies and the OLAP Catalog metadata model

Example: Creating a Hierarchy

The following statement creates a dimension hierarchy PRODUCT_DIM_ROLLUP, for the PRODUCT_DIM dimension in the JSMITH schema. The display name is Standard. The short description is Std Product, and the description is Standard Product Hierarchy. The solved code is SOLVED LEVEL-BASED, meaning that this hierarchy will be mapped to an embedded total dimension table, and that the fact table associated with this dimension hierarchy will store fully solved data.

```
execute cwm2_olap_hierarchy.create_hierarchy
     ('JSMITH', 'PRODUCT_DIM', 'PRODUCT_DIM_ROLLUP',
```

```
'Standard', 'Std Product', 'Standard Product Hierarchy',
'SOLVED LEVEL-BASED');
```

Summary of CWM2_OLAP_HIERARCHY Subprograms

Table 14–1 CWM2_OLAP_HIERARCHY Subprograms

Subprogram	Description
CREATE_HIERARCHY Procedure on page 14-4	Creates a hierarchy.
DROP_HIERARCHY Procedure on page 14-4	Drops a hierarchy.
LOCK_HIERARCHY Procedure on page 14-5	Locks the hierarchy for update.
SET_DESCRIPTION Procedure on page 14-5	Sets the description for a hierarchy.
SET_DISPLAY_NAME Procedure on page 14-6	Sets the display name for a hierarchy.
SET_HIERARCHY_NAME Procedure on page 14-6	Sets the name of a hierarchy.
SET_SHORT_DESCRIPTION Procedure on page 14-6	Sets the short description for a hierarchy.
SET_SOLVED_CODE Procedure on page 14-7	Sets the solved code for a hierarchy.

CREATE_HIERARCHY Procedure

This procedure creates a new hierarchy in the OLAP Catalog.

You must specify descriptions and display properties as part of hierarchy creation. Once the hierarchy has been created, you can override these properties by calling other procedures in the CWM2_OLAP_HIERARCHY package.

Syntax

```
CREATE_HIERARCHY (
          dimension_owner       IN   VARCHAR2,
          dimension_name        IN   VARCHAR2,
          hierarchy_name        IN   VARCHAR2,
          display_name          IN   VARCHAR2,
          short_description     IN   VARCHAR2,
          description           IN   VARCHAR2,
          solved_code           IN   VARCHAR2);
```

Parameters

Table 14–2 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.
solved_code	Specifies whether or not the hierarchy includes embedded totals and whether it is mapped to a level-based dimension table or a parent-child dimension table. For information about mapping hierarchies with different solved codes, see "Joining Fact Tables with Dimension Tables" on page 2-9. 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

DROP_HIERARCHY Procedure

This procedure drops a hierarchy from the OLAP Catalog.

Syntax

```
DROP_HIERARCHY (
         dimension_owner     IN   VARCHAR2,
         dimension_name      IN   VARCHAR2,
```

```
        hierarchy_name      IN   VARCHAR2);
```

Parameters

Table 14–3 DROP_HIERARCHY Procedure Parameters

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

LOCK_HIERARCHY Procedure

This procedure locks the hierarchy metadata for update by acquiring a database lock on the row that identifies the hierarchy in the CWM2 model table.

Syntax

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

Parameters

Table 14–4 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.

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 14–5 SET_DESCRIPTION Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.

Table 14–5 (Cont.) SET_DESCRIPTION Procedure Parameters

Parameter	Description
dimension_name	Name of the dimension.
hierarchy_name	Name of the hierarchy.
description	Description of the hierarchy.

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 14–6 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.

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 14–7 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.

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 14–8 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.

SET_SOLVED_CODE Procedure

This procedure sets the solved code for a hierarchy. The solved code specifies whether or not the data dimensioned by this hierarchy includes embedded totals and whether it is mapped to a level-based dimension table or a parent-child dimension table. If mapped to a parent-child dimension table, it cannot be accessed by the OLAP API.

For more information on mapping solved and unsolved data, see "Joining Fact Tables with Dimension Tables" on page 2-9.

Syntax

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

Parameters

Table 14–9 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.

Table 14–9 (Cont.) SET_SOLVED_CODE Procedure Parameters

Parameter	Description
`solved_code`	Specifies whether or not the hierarchy includes embedded totals and whether it is mapped to a level-based dimension table or a parent-child dimension table. For information about mapping hierarchies with different solved codes, see "Joining Fact Tables with Dimension Tables" on page 2-9. 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

15

CWM2_OLAP_LEVEL

The CWM2_OLAP_LEVEL package provides procedures for managing levels.

> **See Also:** Chapter 2, "Creating OLAP Catalog Metadata with CWM2".

This chapter discusses the following topics:

- Understanding Levels
- Example: Creating a Level
- Summary of CWM2_OLAP_LEVEL Subprograms

Understanding Levels

A level is an OLAP Catalog metadata entity. This means that it is a logical object, identified by name and owner, within the OLAP Catalog.

Dimension members are organized in levels that map to columns in dimension tables or views. Levels are typically organized in hierarchies. Every dimension must have at least one level. Levels are fully described in

Use the procedures in the CWM2_OLAP_LEVEL package to create, drop, and lock levels, to assign levels to hierarchies, and to specify descriptive information for display purposes.

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

> **See Also:**
>
> - Chapter 14, "CWM2_OLAP_HIERARCHY"
> - *Oracle OLAP Application Developer's Guide* for more information about levels and the OLAP Catalog metadata model

Example: Creating a Level

The following statements create four levels for the PRODUCT_DIM dimension and assign them to the PRODUCT_DIM_ROLLUP hierarchy.

```
execute cwm2_olap_level.create_level
     ('JSMITH', 'PRODUCT_DIM', 'TOTALPROD_LVL',
      'Total Product', 'All Products', 'Total',
      'Equipment and Parts of standard product hierarchy');
execute cwm2_olap_level.create_level
```

```
     ('JSMITH', 'PRODUCT_DIM', 'PROD_CATEGORY_LVL',
      'Product Category', 'Product Categories', 'Category',
      'Categories of standard product hierarchy');
execute cwm2_olap_level.create_level
     ('JSMITH', 'PRODUCT_DIM', 'PROD_SUBCATEGORY_LVL',
      'Product Sub-Category', 'Product Sub-Categories', 'Sub-Category',
      'Sub-Categories of standard product hierarchy');
execute cwm2_olap_level.create_level
     ('JSMITH', 'PRODUCT_DIM', 'PRODUCT_LVL',
      'Product', 'Products', 'Product',
      'Individual products of standard product hierarchy');

execute cwm2_olap_level.add_level_to_hierarchy
     ('JSMITH', 'PRODUCT_DIM', 'PRODUCT_DIM_ROLLUP',
      'PRODUCT_LVL', 'PROD_SUBCATEGORY_LVL');
execute cwm2_olap_level.add_level_to_hierarchy
     ('JSMITH', 'PRODUCT_DIM', 'PRODUCT_DIM_ROLLUP',
      'PROD_SUBCATEGORY_LVL', 'PROD_CATEGORY_LVL');
execute cwm2_olap_level.add_level_to_hierarchy
     ('JSMITH', 'PRODUCT_DIM', 'PRODUCT_DIM_ROLLUP',
      'PROD_CATEGORY_LVL', 'TOTALPROD_LVL');
execute cwm2_olap_level.add_level_to_hierarchy
     ('JSMITH', 'PRODUCT_DIM', 'PRODUCT_DIM_ROLLUP', 'TOTALPROD_LVL');
```

Summary of CWM2_OLAP_LEVEL Subprograms

Table 15–1 CWM2_OLAP_LEVEL Subprograms

Subprogram	Description
ADD_LEVEL_TO_HIERARCHY Procedure on page 15-4	Adds a level to a hierarchy.
CREATE_LEVEL Procedure on page 15-4	Creates a level.
DROP_LEVEL Procedure on page 15-5	Drops a level.
LOCK_LEVEL Procedure on page 15-5	Locks the level metadata for update.
REMOVE_LEVEL_FROM_HIERARCHY Procedure on page 15-6	Removes a level from a hierarchy.
SET_DESCRIPTION Procedure on page 15-6	Sets the description for a level.
SET_DISPLAY_NAME Procedure on page 15-6	Sets the display name for a level.
SET_LEVEL_NAME Procedure on page 15-7	Sets the name of a level.
SET_PLURAL_NAME Procedure on page 15-7	Sets the plural name for a level.
SET_SHORT_DESCRIPTION Procedure on page 15-8	Sets the short description for a level.

ADD_LEVEL_TO_HIERARCHY Procedure

This procedure adds a level to a hierarchy. A hierarchy can have a maximum of 31 levels.

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 NULL);
```

Parameters

Table 15–2 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.

CREATE_LEVEL Procedure

This procedure creates a new level in the OLAP Catalog.

You must specify descriptions and display properties as part of level creation. Once the level has been created, you can override these properties by calling other procedures in the CWM2_OLAP_LEVEL package.

Syntax

```
CREATE_LEVEL (
          dimension_owner       IN   VARCHAR2,
          dimension_name        IN   VARCHAR2,
          level_name            IN   VARCHAR2,
          display_name          IN   VARCHAR2,
          plural_name           IN   VARCHAR2,
          short_description     IN   VARCHAR2,
          description           IN   VARCHAR2);
```

Parameters

Table 15–3 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 15–3 (Cont.) 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.

DROP_LEVEL Procedure

This procedure drops a level from the OLAP Catalog. All related level attributes are also dropped.

Syntax

```
DROP_LEVEL (
         dimension_owner     IN   VARCHAR2,
         dimension_name      IN   VARCHAR2,
         level_name          IN   VARCHAR2);
```

Parameters

Table 15–4 DROP_LEVEL Procedure Parameters

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

LOCK_LEVEL Procedure

This procedure locks the level metadata for update by acquiring a database lock on the row that identifies the level in the CWM2 model table.

Syntax

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

Parameters

Table 15–5 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.

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 15–6 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.

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 15–7 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.

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 15–8 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.

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 15–9 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.

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 15–10 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.

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 15–11 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.

16

CWM2_OLAP_LEVEL_ATTRIBUTE

The CWM2_OLAP_LEVEL_ATTRIBUTE package provides procedures for managing level attributes.

> **See Also:** Chapter 2, "Creating OLAP Catalog Metadata with CWM2".

This chapter discusses the following topics:

- Understanding Level Attributes
- Example: Creating Level Attributes
- Summary of CWM2_OLAP_LEVEL_ATTRIBUTE Subprograms

Understanding Level Attributes

A level attribute is an OLAP Catalog metadata entity. This means that it is a logical object, identified by name and owner, within the OLAP Catalog.

A level attribute is a child entity of a level and a dimension attribute. A level attribute stores descriptive information about its related level. For example, a level containing product identifiers might have an associated level attribute that contains color information for each product.

Each level attribute maps to a column in a dimension table. The level attribute column must be in the same table as the column (or columns) for its associated level. Level attributes are fully described in.

Use the procedures in the CWM2_OLAP_LEVEL_ATTRIBUTE package to create, drop, and lock level attributes, to assign level attributes to levels and dimension attributes, and to specify descriptive information for display purposes.

Several level attribute names are reserved, because they have special significance within CWM2. Reserved level attributes are associated with reserved dimension attributes of the same name. Reserved level attributes will be mapped to columns containing specific information. The reserved level attributes are listed in Table 16–1.

Table 16–1 Reserved Level Attributes

Dimension Attribute	Description
Long Description	A long description of the dimension member.
Short Description	A short description of the dimension member.
End Date	For a time dimension, the last date in a time period. (Required)

Table 16–1 (Cont.) Reserved Level Attributes

Dimension Attribute	Description
Time Span	For a time dimension, the number of days in a time period. (Required)
Prior Period	For a time dimension, the time period before this time period.
Year Ago Period	For a time dimension, the period a year before this time period.
ET Key	For an embedded total dimension, the embedded total key, which identifies the dimension member at the lowest level in a row of the dimension table. (Required)
Parent ET Key	For an embedded total dimension, the dimension member that is the parent of the ET key. (Required)
Grouping ID	For an embedded total dimension, the grouping ID (GID), which identifies the hierarchical level for a row of the dimension table. (Required)
Parent Grouping ID	For an embedded total dimension, the dimension member that is the parent of the grouping ID. (Required)

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

See Also:

- Chapter 12, "CWM2_OLAP_DIMENSION_ATTRIBUTE"
- *Oracle OLAP Application Developer's Guide* for more information about level attributes and the OLAP Catalog metadata model

Example: Creating Level Attributes

The following statements create a color attribute for the lowest level and long descriptions for all four levels of the PRODUCT_DIM dimension.

```
execute cwm2_olap_level_attribute.create_level_attribute
     ('JSMITH', 'PRODUCT_DIM', 'Product Color', 'PRODUCT_LVL', 'Product Color',
      'PROD_STD_COLOR', 'Prod Color', 'Product Color');

execute cwm2_olap_level_attribute.create_level_attribute
     ('JSMITH', 'PRODUCT_DIM', 'Long Description', 'PRODUCT_LVL',
      'Long Description','PRODUCT_STD_LLABEL', 'Product',
      'Long Labels for individual products of the PRODUCT hierarchy', TRUE);

execute cwm2_olap_level_attribute.create_level_attribute
     ('JSMITH', 'PRODUCT_DIM', 'Long Description', 'PROD_SUBCATEGORY_LVL',
      'Long Description', 'PROD_STD_LLABEL', 'Product Sub Category',
      'Long Labels for subcategories of the PRODUCT hierarchy', TRUE);

execute cwm2_olap_level_attribute.create_level_attribute
     ('JSMITH', 'PRODUCT_DIM', 'Long Description', 'PROD_CATEGORY_LVL',
      'Long Description', 'PROD_STD_LLABEL', 'Product Category',
      'Long Labels for categories of the PRODUCT hierarchy', TRUE);

execute cwm2_olap_level_attribute.create_level_attribute
     ('JSMITH', 'PRODUCT_DIM', 'Long Description', 'TOTALPROD_LVL',
      'Long Description', 'PROD_STD_LLABEL', 'Total Product',
      'Long Labels for total of the PRODUCT hierarchy', TRUE);
```

Summary of CWM2_OLAP_LEVEL_ATTRIBUTE Subprograms

Table 16–2 CWM2_OLAP_LEVEL_ATTRIBUTE Subprograms

Subprogram	Description
CREATE_LEVEL_ATTRIBUTE Procedure on page 16-4	Creates a level attribute.
DROP_LEVEL_ATTRIBUTE Procedure on page 16-5	Drops a level attribute.
LOCK_LEVEL_ATTRIBUTE Procedure on page 16-5	Locks the level attribute metadata for update.
SET_DESCRIPTION Procedure on page 16-6	Sets the description for a level attribute.
SET_DISPLAY_NAME Procedure on page 16-6	Sets the display name for a level attribute.
SET_LEVEL_ATTRIBUTE_NAME Procedure on page 16-7	Sets the name of a level attribute.
SET_SHORT_DESCRIPTION Procedure on page 16-8	Sets the short description for a level attribute.

CREATE_LEVEL_ATTRIBUTE Procedure

This procedure creates a new level attribute in the OLAP Catalog and associates the level attribute with a level and with a dimension attribute.

If the level attribute is reserved, you can specify the reserved name as the level attribute name or as a type associated with a name that you specify. The reserved level attributes are listed in Table 16–1, " Reserved Level Attributes".

You must specify descriptions and display properties as part of level attribute creation. Once the level attribute has been created, you can override these properties by calling other procedures in the CWM2_OLAP_LEVEL_ATTRIBUTE package.

Syntax

```
CREATE_LEVEL_ATTRIBUTE (
          dimension_owner            IN   VARCHAR2,
          dimension_name             IN   VARCHAR2,
          dimension_attribute_name   IN   VARCHAR2,
          level_name                 IN   VARCHAR2,
          level_attribute_name       IN   VARCHAR2,
          display_name               IN   VARCHAR2,
          short_description          IN   VARCHAR2,
          description                IN   VARCHAR2,
            type                     IN   VARCHAR2             );
            use_name_as_type         IN   BOOLEAN DEFAULT FALSE);
```

Parameters

Table 16–3 CREATE_LEVEL_ATTRIBUTE Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
dimension_attribute_name	Name of the dimension attribute that includes this level attribute.
level_name	Name of the level.
level_attribute_name	Name of the level attribute.
display_name	Display name for the level attribute.

Table 16–3 (Cont.) CREATE_LEVEL_ATTRIBUTE Procedure Parameters

Parameter	Description
short_description	Short description of the level attribute.
description	Description of the level attribute.
type *or* use_name_as_type	This argument can be one of the following: ■ type a VARCHAR2 argument whose value is one of the reserved names from Table 16–1, " Reserved Level Attributes". Specify this argument if you want to create your own name for a reserved level attribute. ■ use_name_as_type a BOOLEAN argument that defaults to FALSE. This argument specifies whether or not the level attribute name is a reserved name. If this argument is TRUE, the value of the level_attribute_name argument must be a reserved name from Table 16–1, " Reserved Level Attributes". If you do not specify a value for this argument, the level attribute is not reserved.

DROP_LEVEL_ATTRIBUTE Procedure

This procedure drops a level attribute from the OLAP Catalog.

Syntax

```
DROP_LEVEL_ATTRIBUTE (
         dimension_owner            IN   VARCHAR2,
         dimension_name             IN   VARCHAR2,
         dimension_attribute_name   IN   VARCHAR2,
         level_name                 IN   VARCHAR2,
         level_attribute_name       IN   VARCHAR2);
```

Parameters

Table 16–4 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.

LOCK_LEVEL_ATTRIBUTE Procedure

This procedure locks the level attribute metadata for update by acquiring a database lock on the row that identifies the level attribute in the CWM2 model table.

Syntax

```
LOCK_LEVEL_ATTRIBUTE (
          dimension_owner            IN   VARCHAR2,
          dimension_name             IN   VARCHAR2,
          dimension_attribute_name   IN   VARCHAR2,
```

```
          level_name                IN   VARCHAR2,
          level_attribute_name      IN   VARCHAR2,
          wait_for_lock             IN   BOOLEAN DEFAULT FALSE);
```

Parameters

Table 16–5 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.

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 16–6 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.

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 16–7 SET_DISPLAY_NAME Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
dimension_attribute_name	Name of the dimension attribute.
level_name	Name of the level.
level_attribute_name	Name of the level attribute.
display_name	Display name for the level attribute.

SET_LEVEL_ATTRIBUTE_NAME Procedure

This procedure sets the name for a level attribute.

If the level attribute is reserved, you can specify the reserved name as the level attribute name or as a type associated with a name that you specify. The reserved level attributes are listed in Table 16–1, " Reserved Level Attributes".

Syntax

```
SET_LEVEL_ATTRIBUTE_NAME (
          dimension_owner             IN   VARCHAR2,
          dimension_name              IN   VARCHAR2,
          dimension_attribute_name    IN   VARCHAR2,
          level_name                  IN   VARCHAR2,
          level_attribute_name        IN   VARCHAR2,
          set_level_attribute_name    IN   VARCHAR2,
            type                      IN   VARCHAR2             );
            use_name_as_type          IN   BOOLEAN DEFAULT FALSE);
```

Parameters

Table 16–8 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.

Table 16–8 (Cont.) SET_LEVEL_ATTRIBUTE_NAME Procedure Parameters

Parameter	Description
level_attribute_name	Original name for the level attribute.
set_level_attribute_name	New name for the level attribute.
type *or* use_name_as_type	This argument can be one of the following: ▪ type a VARCHAR2 argument whose value is one of the reserved names from Table 16–1, " Reserved Level Attributes". Specify this argument if you want to create your own name for a reserved level attribute. ▪ use_name_as_type a BOOLEAN argument that defaults to FALSE. This argument specifies whether or not the level attribute name is a reserved name. If this argument is TRUE, the value of the level_attribute_name argument must be a reserved name from Table 16–1, " Reserved Level Attributes". If you do not specify a value for this argument, the level attribute is not reserved.

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 16–9 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.

17

CWM2_OLAP_MANAGER

The CWM2_OLAP_MANAGER package provides procedures that manage output generated by the OLAP PL/SQL packages. These procedures will help you develop and debug your PL/SQL scripts.

See Also:

- "Directing Output" in Chapter 2
- *SQL*Plus User's Guide and Reference*

This chapter discusses the following topics:

Managing Output in a SQL*Plus Session

SQL*Plus maintains system variables (also called SET command variables) to enable you to set up a particular environment for a SQL*Plus session. You can change these system variables with the SET command and list them with the SHOW command.

In SQL*Plus, the output of stored procedures is sent to the SQL buffer. The default size of the buffer is 2K. The SERVEROUTPUT system variable controls whether or not SQL*Plus displays the contents of the SQL buffer.

When using the OLAP stored procedures, you should set SERVEROUTPUT and extend the size of the buffer to its maximum size.

```
>set serveroutput on size 1000000
```

After setting SERVEROUTPUT, use the CWM2_OLAP_MANAGER procedure SET_ECHO_ON to display the output of OLAP procedures.

```
>execute cwm2_olap_manager.set_echo_on;
```

Several OLAP packages generate reports. For example, the CWM2_OLAP_VALIDATE package generates a metadata validation report, and the CWM2_OLAP_DELETE package generates a delete command report . For procedures that generate reports or other lengthy output, you should direct the output to a file. Use the CWM2_OLAP_MANAGER procedure BEGIN_LOG.

```
>execute cwm2_olap_manager.begin_log;
```

Example: Using a Log File

The following example shows how to use the CWM2_OLAP_MANAGER package to direct a validation report to a log file.

Example 17–1 Direct a Validation Report to a File

```
>set linesize 135
>set pagesize 50

>execute cwm2_olap_manager.begin_log
       ('/users/myuser' , 'Metadata_Validation_Report');
>execute cwm2_olap_manager.log_note
       ('OLAP Metadata Validation Report' );
>execute cwm2_olap_validate.validate_olap_catalog
       ('OLAP API');
>execute cwm2_olap_manager.end_log;
```

The log file would look something like this.

```
BEGIN: CwM2_OLAP Log  Date: 2004 APRIL     05  Time: 17:10:20 User: MYUSER.

Log Directory: /users/myuser  Log File: Metadata_Validation_Report.

OLAP Metadata Validation Report

.Validate Olap Catalog

.Validate Dimension: GLOBAL.CHANNEL    Type of Validation: OLAP API    Verbose
Report: YES
.Validating Dimension Metadata in OLAP Catalog 1  Date: 2004 APRIL     05  Time:
17:11:45
                                                  User: MYUSER  031201

.ENTITY TYPE   ENTITY NAME        STATUS    COMMENT

Dimension       GLOBAL.CHANNEL    VALID    Default_Display_Hierarchy: "CHANNEL_
ROLLUP".
.
.
.
.
.
.
END: CwM2_OLAP Log  Date: 2004 APRIL     05  Time: 17:12:11 User: MYUSER.
Log Directory: /users/myuser  Log File: Metadata_Validation_Report.
```

Summary of CWM2_OLAP_MANAGER Subprograms

Table 17–1 CWM2_OLAP_MANAGER

Subprogram	Description
BEGIN_LOG Procedure on page 17-4	Turns on logging.
END_LOG Procedure on page 17-4	Turns off logging.
LOG_NOTE Procedure on page 17-4	Writes a text string in the log file.
SET_ECHO_OFF Procedure on page 17-5	Turns on echoing to the screen.
SET_ECHO_ON Procedure on page 17-5	Turns off echoing to the screen.

BEGIN_LOG Procedure

The BEGIN_LOG procedure directs the output from OLAP PL/SQL packages to a log file.

Syntax

```
BEGIN_LOG (
          output_directory     IN   VARCHAR2,
          file_name            IN   VARCHAR2,
          append_to_file       IN   VARCHAR2 DEFAULT 'NO');
```

Parameters

Table 17–2 BEGIN_LOG Procedure Parameters

Parameter	Description
output_directory	Output directory for the log file. You can either specify a directory object, to which your user ID has been granted the appropriate access, or a path set by the UTL_FILE_DIR initialization parameter for the instance.
file_name	Name of log file.
append_to_file	Specify 'YES' to append the output to the end of the file. Specify 'NO' to delete the previous contents of the file before writing to it. The default is 'NO'.

Example

See "Example: Using a Log File" on page 17-2.

END_LOG Procedure

The END_LOG procedure turns off logging.

Syntax

```
END_LOG;
```

Example

See "Example: Using a Log File" on page 17-2.

LOG_NOTE Procedure

When logging is turned on, the LOG_NOTE procedure writes the text that you specify to the log file. If logging is not turned on, this procedure has no effect.

Syntax

```
LOG_NOTE  (
          message_text   IN   VARCHAR2);
```

Parameters

Table 17–3 LOG_NOTE Procedure Parameters

Parameter	Description
message_text	Text to write to the log file.

Example

See "Example: Using a Log File" on page 17-2.

SET_ECHO_OFF Procedure

SET_ECHO_OFF prevents the display of output generated by OLAP stored procedures.

Syntax

```
SET_ECHO_OFF;
```

Example

The following example illustrates how the output from an OLAP DML command is displayed when echoing is turned on, but suppressed when echoing is turned off.

The PL/SQL calls are listed to the left, and the screen output is shown indented to the right.

```
>execute cwm2_olap_manager.set_echo_on;

          PL/SQL procedure successfully completed.

>execute dbms_aw.execute ('listnames');

          1 DIMENSION
          ------------------
          MYDIM

          PL/SQL procedure successfully completed.

>execute cwm2_olap_manager.set_echo_off;

          PL/SQL procedure successfully completed.

>execute dbms_aw.execute ('listnames');

          PL/SQL procedure successfully completed.
```

SET_ECHO_ON Procedure

SET_ECHO_ON causes the output generated by OLAP stored procedures to be displayed on the screen.

Syntax

```
SET_ECHO_ON;
```

Example

See the example in "SET_ECHO_OFF Procedure" on page 17-5.

18

CWM2_OLAP_MEASURE

The CWM2_OLAP_MEASURE package provides procedures for managing measures.

> **See Also:** Chapter 2, "Creating OLAP Catalog Metadata with CWM2".

This chapter discusses the following topics:

- Understanding Measures
- Example: Creating a Measure
- Summary of CWM2_OLAP_MEASURE Subprograms

Understanding Measures

A measure is an OLAP Catalog metadata entity. This means that it is a logical object, identified by name and owner, within the OLAP Catalog.

Measures represent data stored in fact tables. The fact tables may be relational tables or views. The views may reference data stored in analytic workspaces.

Measures exist within the context of cubes, which fully specify the dimensionality of the measures' data. Measures are fully described in.

Use the procedures in the CWM2_OLAP_MEASURE package to create, drop, and lock measures, to associate a measure with a cube, and to specify descriptive information for display purposes.

The parent cube must already exist in the OLAP Catalog before you can create a measure.

> **See Also:**
>
> - Chapter 9, "CWM2_OLAP_CUBE"
> - *Oracle OLAP Application Developer's Guide* for more information about measures and the OLAP Catalog metadata model

Example: Creating a Measure

The following statements create the SALES_AMOUNT and SALES_QUANTITY measures for the SALES_CUBE cube.

```
execute cwm2_olap_measure.create_measure
     ('JSMITH', 'SALES_CUBE', 'SALES_AMOUNT', 'Sales Amount',
      '$ Sales', 'Dollar Sales');
execute cwm2_olap_measure.create_measure
```

```
('JSMITH', 'SALES_CUBE', 'SALES_QUANTITY', 'Sales Quantity',
 'Sales Quantity', 'Quantity of Items Sold');
```

Summary of CWM2_OLAP_MEASURE Subprograms

Table 18–1 CWM2_OLAP_MEASURE Subprograms

Subprogram	Description
CREATE_MEASURE Procedure on page 18-4	Creates a measure.
DROP_MEASURE Procedure on page 18-4	Drops a measure.
LOCK_MEASURE Procedure on page 18-5	Locks a measure's metadata for update.
SET_DESCRIPTION Procedure on page 18-5	Sets the description for a measure.
SET_DISPLAY_NAME Procedure on page 18-5	Sets the display name for a measure.
SET_MEASURE_NAME Procedure on page 18-6	Sets the name of a measure.
SET_SHORT_DESCRIPTION Procedure on page 18-6	Sets the short description for a measure.

CREATE_MEASURE Procedure

This procedure creates a new measure in the OLAP Catalog.

A measure can only be created in the context of a cube. The cube must already exist before you create the measure.

Descriptions and display properties must also be established as part of measure creation. Once the measure has been created, you can override these properties by calling other procedures in this package.

Syntax

```
CREATE_MEASURE (
          cube_owner            IN   VARCHAR2,
          cube_name             IN   VARCHAR2,
          measure_name          IN   VARCHAR2,
          display_name          IN   VARCHAR2,
          short_description     IN   VARCHAR2,
          description           IN   VARCHAR2);
```

Parameters

Table 18–2 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.

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 18–3 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.

LOCK_MEASURE Procedure

This procedure locks the measure's metadata for update by acquiring a database lock on the row that identifies the measure in the CWM2 model table.

Syntax

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

Parameters

Table 18–4 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.

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 18–5 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.

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 18–6 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.

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 18–7 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.

SET_SHORT_DESCRIPTION Procedure

This procedure sets the short description for a measure.

Syntax

```
SET_SHORT_DESCRIPTION (
          cube_owner             IN   VARCHAR2,
          cube_name              IN   VARCHAR2,
          measure_name           IN   VARCHAR2,
          short_description      IN   VARCHAR2);
```

Parameters

Table 18–8 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.

19

CWM2_OLAP_METADATA_REFRESH

The CWM2_OLAP_METADATA_REFRESH package provides procedures that refresh cached OLAP Catalog metadata.

See Also:

- "Validating and Committing OLAP Catalog Metadata" on page 2-10
- Chapter 5, "OLAP Catalog Metadata Views"
- Chapter 3, "Active Catalog Views"

This chapter discusses the following topics:

Views of Cached OLAP Catalog Metadata

The Metadata Reader Views for the OLAP Catalog, named with the prefix MRV_OLAP2, present a read API to a set of cache tables for OLAP Catalog metadata. These views and tables are structured to facilitate query performance for the OLAP API.

The MRV_OLAP2 views correspond to the ALL_OLAP2 views, which provide information about OLAP Catalog metadata. Each MRV_OLAP2 view has the same name and column structure as its corresponding ALL_OLAP2 view. If you require fast access to OLAP Catalog metadata, you should query the cached metadata through the MRV_OLAP2 views.

The cache tables are *not* automatically refreshed when changes are made to the metadata. To refresh the cache, call the CWM2_OLAP_METADATA_REFRESH.MR_REFRESH procedure.

Note: If your data is stored in relational tables (not in analytic workspaces), you must refresh the OLAP Catalog metadata cache for applications that use the OLAP API.

Views of Cached Active Catalog Metadata

The Metadata Reader Views for the Active Catalog, named with the prefix MRV_OLAP2_AW, present a read API to a set of cache tables for the Active Catalog. These views and tables are structured to facilitate query performance for the OLAP API.

The MRV_OLAP2_AW views correspond to the ALL_OLAP2_AW views, which provide information about standard form metadata within analytic workspaces. Each MRV_OLAP2_AW view has the same name and column structure as its corresponding ALL_OLAP2_AW view. If you require fast access to the Active Catalog, you should query the cached metadata through the MRV_OLAP2_AW views.

The cache tables are *not* automatically refreshed when changes are made to the Active Catalog. To refresh the cache, call the CWM2_OLAP_METADATA_REFRESH.MR_AC_REFRESH procedure.

Note: If your data is stored in analytic workspaces, you should refresh the Active Catalog cache for applications that use the OLAP API.

Summary of CWM2_OLAP_METADATA_REFRESH Subprograms

Table 19–1 CWM2_OLAP_METADATA_REFRESH Subprograms

Subprogram	Description
MR_REFRESH Procedure	Refreshes the OLAP Catalog metadata cache.
MR_AC_REFRESH Procedure	Refreshes the Active Catalog metadata cache.

MR_REFRESH Procedure

This procedure refreshes the OLAP Catalog metadata cache tables that underlie the MRV_OLAP2 views. You must refresh the cache for applications that use the OLAP API with a relational data source.

The MR_REFRESH procedure includes a COMMIT.

Syntax

```
MR_REFRESH;
```

See Also

"Validating and Committing OLAP Catalog Metadata" on page 2-10 and "OLAP Catalog Metadata Cache" on page 5-1.

MR_AC_REFRESH Procedure

This procedure refreshes the Active Catalog metadata cache tables that underlie the MRV_OLAP2_AW views. You must refresh the cache for applications that use the OLAP API with a multidimensional data source in analytic workspaces.

The MR_AC_REFRESH procedure includes a COMMIT.

Syntax

```
MR_AC_REFRESH;
```

See Also

"Active Catalog Metadata Cache" on page 3-2.

20

CWM2_OLAP_PC_TRANSFORM

The CWM2_OLAP_PC_TRANSFORM package contains a procedure for generating a SQL script that creates a solved, level-based dimension table from a parent-child dimension table.

After running the script and creating the new table, you can define OLAP Catalog metadata so that OLAP API applications can access the dimension.

See Also:

- *Oracle OLAP Application Developer's Guide* for information about types of data warehouse tables supported by OLAP Catalog metadata.
- Chapter 14, "CWM2_OLAP_HIERARCHY" for information about creating OLAP Catalog metadata for dimension hierarchies.

This chapter discusses the following topics:

Prerequisites

Before running the CWM2_OLAP_PC_TRANSFORM.CREATE_SCRIPT procedure, ensure that the RDBMS is enabled to write to a file. To specify a directory, you can use either a directory object to which your user ID has been granted the appropriate access, or a path set by the UTL_FILE_DIR initialization parameter for the instance.

A parent-child dimension table must exist and be accessible to the CWM2_OLAP_PC_TRANSFORM.CREATE_SCRIPT procedure.

Parent-Child Dimensions

A **parent-child dimension table** is one in which the hierarchical relationships are defined by a parent column and a child column. Since the hierarchy is defined by the relationship between the *values* within two columns, a parent-child dimension is sometimes referred to as having a **value-based hierarchy**.

Sample Parent-Child Dimension Table Columns

The following example illustrates the relationships between the values in the child and parent columns. A description column, which is an attribute of the child, is also included.

```
CHILD           PARENT         DESCRIPTION
-------------   -----------    -------------
World                          World
USA             World          United States of America
Northeast       USA            North East Region
Southeast       USA            South East Region
MA              Northeast      Massachusetts
Boston          MA             Boston, MA
Burlington      MA             Burlington, MA
NY              Northeast      New York State
New York City   NY             New York, NY
GA              Southeast      Georgia
Atlanta         GA             Atlanta,GA
Canada          World          Canada
```

If you choose to create OLAP Catalog metadata to represent a parent-child dimension, set the `solved_code` for the hierarchy to `'SOLVED VALUE-BASED'`, as described in Chapter 14, "CWM2_OLAP_HIERARCHY".

Note: You can create OLAP Catalog metadata to represent value-based hierarchies, but this type of hierarchy is not accessible to applications that use the OLAP API.

Solved, Level-Based Dimensions

The script generated by `OLAP_PC_TRANSFORM.CREATE_SCRIPT` creates a table that stores the values from the parent-child table in levels.

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

If you want to enable parent-child dimension tables for access by the OLAP API, you must convert them to solved, level-based dimension tables. The OLAP API requires that dimensions have levels and that they include a GID (Grouping ID) column and an Embedded Total (ET) key column. GIDs and ET key columns are described in Example: Creating a Solved, Level-Based Dimension Table.

The following example illustrates how the parent-child relationships in "Parent-Child Dimensions" on page 20-1would be represented as solved levels.

```
TOT_GEOG COUNTRY REGION    STATE CITY           DESCRIPTION
-------- ------- --------- ----- -------        ----------------------
World    USA     Northeast  MA   Boston         Boston, MA
World    USA     Northeast  MA   Burlington     Burlington, MA
World    USA     Northeast  NY   New York City  New York, NY
World    USA     Southeast  GA   Atlanta        Atlanta, GA
World    USA     Northeast  MA                  Massachusetts
World    USA     Northeast  NY                  New York State
World    USA     Southeast  GA                  Georgia
World    USA     Northeast                      North East Region
World    USA     Southeast                      South East Region
World    USA                                    United States of America
World    Canada                                 Canada
World                                           World
```

When creating OLAP Catalog metadata to represent a solved, level-based dimension hierarchy, specify a `solved_code` of `'SOLVED LEVEL-BASED'`, as described in Chapter 14, "CWM2_OLAP_HIERARCHY".

Example: Creating a Solved, Level-Based Dimension Table

Assuming a parent-child dimension table with the `PARENT` and `CHILD` columns shown in "Parent-Child Dimensions" on page 20-1, you could use a command like the following to represent these columns in a solved, level-based dimension table.

```
execute cwm2_olap_pc_transform.create_script
     ('/dat1/scripts/myscripts' ,
     'jsmith' ,
     'input_tbl' ,
     'PARENT' ,
     'CHILD' ,
     'output_tbl' ,
     'jsmith_data');
```

This statement creates a script in the directory `/dat1/scripts/myscripts`. The script will convert the parent-child table `input_tbl` to the solved, level-based table `output_tbl`. Both tables are in the `jsmith_data` tablespace of the `jsmith` schema.

You can run the resulting script with the following command.

```
@create_output_tbl
```

You can view the resulting table with the following command.

```
select * from output_tbl_view
```

The resulting table would look like this.

```
GID SHORT_DESC    LONG_DESC     CHILD1 CHILD2 CHILD3    CHILD4 CHILD5
--- -----------   ------------  ------ -----  --------  ------ -------
0   Boston        Boston        World  USA    Northeast  MA    Boston
0   Burlington    Burlington    World  USA    Northeast  MA    Burlington
0   New York City New York City World  USA    Northeast  NY    New York City
0   Atlanta       Atlanta       World  USA    Southeast  GA    Atlanta
1   MA            MA            World  USA    Northeast  MA
1   NY            MA            World  USA    Northeast  NY
1   GA            GA            World  USA    Southeast  GA
3   Northeast     Northeast     World  USA    Northeast
3   Southeast     Southeast     World  USA    Southeast
7   USA           USA           World  USA
7   Canada        Canada        World  Canada
15  World         World         World
```

Grouping ID Column

The script automatically creates a GID column, as required by the OLAP API. The GID identifies the hierarchy level associated with each row by assigning a zero to each non-null value and a one to each null value in the level columns. The resulting binary number is the value of the GID. For example, a GID of 3 is assigned to the row with the level values World, USA, Northeast, since the three highest levels are assigned zeros and the two lowest levels are assigned ones.

```
CHILD1 CHILD2 CHILD3    CHILD4 CHILD5
```

```
------ -----  --------  ------ -------
World  USA    Northeast
0      0      0          1      1
```

Embedded Total Key Column

The script automatically generates columns for long description and short description. If you have columns in the input table that contain this information, you can specify them as parameters to the `CREATE_SCRIPT` procedure.

If you do not specify a column for the short description, the script creates the column and populates it with the lowest-level child value represented in each row. If you do not specify a column for the long description, the script simply replicates the short description.

The ET key column required by the OLAP API is the short description column that is created by default.

Summary of CWM2_OLAP_PC_TRANSFORM Subprograms

Table 20–1 CWM2_OLAP_PC_TRANSFORM

Subprogram	Description
CREATE_SCRIPT Procedure on page 20-6	Generates a script that converts a parent-child table to an embedded-total table.

CREATE_SCRIPT Procedure

This procedure generates a script that converts a parent-child dimension table to an embedded-total dimension table.

Syntax

```
CREATE_SCRIPT (
         directory          IN   VARCHAR2,
         schema             IN   VARCHAR2,
         pc_table           IN   VARCHAR2,
         pc_parent          IN   VARCHAR2,
         pc_child           IN   VARCHAR2,
         slb_table          IN   VARCHAR2,
         slb_tablespace     IN   VARCHAR2,
         pc_root            IN   VARCHAR2   DEFAULT  NULL,
         number_of_levels   IN   NUMBER     DEFAULT  NULL,
         level_names        IN   VARCHAR2   DEFAULT  NULL,
         short_description  IN   VARCHAR2   DEFAULT  NULL,
         long_description   IN   VARCHAR2   DEFAULT  NULL,
         attribute_names    IN   VARCHAR2   DEFAULT  NULL);
```

Parameters

Table 20–2 CREATE_SCRIPT Procedure Parameters

Parameter	Description
`directory`	The directory that will contain the generated script. This may be either a directory object or a directory path specified in the `UTL_FILE_DIR` initialization parameter.
`schema`	Schema containing the parent-child table. This schema will also contain the solved, level-based table.
`pc_table`	Name of the parent-child table.
`pc_parent`	Name of the column in `pc_table` that contains the parent values.
`pc_child`	Name of the column in `pc_table` that contains the child values.
`slb_table`	Name of the solved, level-based table that will be created.
`slb_tablespace`	Name of the tablespace where the solved, level-based table will be created.
`pc_root`	One of the following: `null` - Root of the parent-child hierarchy is identified by `null` in the parent column. (default) *`condition`* - Root of the parent-child hierarchy is a condition, for example: `'long_des = "All Countries"'`
`number_of_levels`	One of the following: `null` - The number of levels in the solved, level-based table will be all the levels of the hierarchy in the parent-child table. (default) *`number`* - The number of levels to be created in the solved, level-based table.

Table 20–2 (Cont.) CREATE_SCRIPT Procedure Parameters

Parameter	Description
level_names	One of the following: null - The column names in the solved, level-based table will be the source child column name concatenated with the level number. (default) *list* - A comma-delimited list of column names for the solved, level-based table.
short_description	One of the following: null - There is no short description in the parent-child table. The highest level non-null child value in each row of the solved, level-based table will be used as the short description. This constitutes the ET key column (default) *column name* - Name of the column in the parent-child table that contains the short description. This column will be copied from the parent-child table to the solved, level-based table.
long_description	One of the following: null - There is no long description in the parent-child table. The short description will be used. (default) *column name* - Name of the column in the parent-child table that contains the long description. This column will be copied from the parent-child table to the solved, level-based table.
attribute_names	One of the following: null - There are no attributes in the parent-child table. (default) *list* - A comma-delimited list of attribute columns in the parent-child table. These columns will be copied from the parent-child table to the solved, level-based table

Usage Notes

1. If a table with the same name as the solved, level-based table already exists, the script will delete it.
2. You can reduce the time required to generate the script by specifying the number of levels in the number_of_levels parameter. If you do not specify a value for this parameter, the CREATE_SCRIPT procedure calculates all the levels from the parent-child table.
3. To define additional characteristics of the solved, level-based table, you can modify the generated script file before executing it.

21

CWM2_OLAP_TABLE_MAP

The CWM2_OLAP_TABLE_MAP package provides procedures for mapping OLAP Catalog metadata entities to columns in your data warehouse dimension tables and fact tables.

> **See Also:** Chapter 2, "Creating OLAP Catalog Metadata with CWM2"

This chapter discusses the following topics:

- Understanding OLAP Catalog Metadata Mapping
- Example: Mapping a Dimension
- Example: Mapping a Cube
- Summary of CWM2_OLAP_TABLE_MAP Subprograms

Understanding OLAP Catalog Metadata Mapping

The CWM2_OLAP_TABLE_MAP package provides procedures for linking OLAP Catalog metadata entities to columns in fact tables and dimension tables and for establishing the join relationships between a fact table and its associated dimension tables.

Dimension levels and level attributes are mapped to columns in dimension tables. Typically, they are mapped by hierarchy. Measures are mapped to columns in fact tables.

The join relationship between the fact table and dimension tables may be specified for solved or unsolved data stored in a single fact table, or for solved data stored in a single fact table for each hierarchy combination.

> **See Also:** "Mapping OLAP Catalog Metadata" on page 2-9.

Example: Mapping a Dimension

The following statements map the four levels of the STANDARD hierarchy in the XADEMO.PRODUCT_AW dimension to columns in the XADEMO_AW_VIEW_PRODUCT dimension table. A long description attribute is mapped for each level.

```
execute cwm2_olap_table_map.Map_DimTbl_HierLevel
    ('XADEMO', 'PRODUCT_AW', 'STANDARD', 'L4',
     'XADEMO', 'XADEMO_AW_VIEW_PRODUCT', 'L4', 'L3');
execute cwm2_olap_table_map.Map_DimTbl_HierLevelAttr
    ('XADEMO', 'PRODUCT_AW', 'Long Description', 'STANDARD', 'L4',
     'Long Description', 'XADEMO', 'XADEMO_AW_VIEW_PRODUCT', 'PROD_STD_LLABEL');
```

```
execute cwm2_olap_table_map.Map_DimTbl_HierLevel
    ('XADEMO', 'PRODUCT_AW', 'STANDARD', 'L3',
     'XADEMO', 'XADEMO_AW_VIEW_PRODUCT', 'L3', 'L2');
execute cwm2_olap_table_map.Map_DimTbl_HierLevelAttr
    ('XADEMO', 'PRODUCT_AW', 'Long Description', 'STANDARD', 'L3',
     'Long Description', 'XADEMO', 'XADEMO_AW_VIEW_PRODUCT', 'PROD_STD_LLABEL');

execute cwm2_olap_table_map.Map_DimTbl_HierLevel
    ('XADEMO', 'PRODUCT_AW', 'STANDARD', 'L2',
     'XADEMO', 'XADEMO_AW_VIEW_PRODUCT', 'L2', 'L1');
execute cwm2_olap_table_map.Map_DimTbl_HierLevelAttr
    ('XADEMO', 'PRODUCT_AW', 'Long Description', 'STANDARD', 'L2',
     'Long Description', 'XADEMO', 'XADEMO_AW_VIEW_PRODUCT', 'PROD_STD_LLABEL');

execute cwm2_olap_table_map.Map_DimTbl_HierLevel
    ('XADEMO', 'PRODUCT_AW', 'STANDARD', 'L1',
     'XADEMO', 'XADEMO_AW_VIEW_PRODUCT', 'L1', null);

execute cwm2_olap_table_map.Map_DimTbl_HierLevelAttr
    ('XADEMO', 'PRODUCT_AW', 'Long Description', 'STANDARD', 'L1',
     'Long Description', 'XADEMO', 'XADEMO_AW_VIEW_PRODUCT', 'PROD_STD_LLABEL');
```

Example: Mapping a Cube

The following statement maps the dimension join keys for a cube named ANALYTIC_CUBE_AW in the XADEMO schema. Join key relationships are specified for four dimension/hierarchy combinations:

```
PRODUCT_AW/STANDARD
CHANNEL_AW/STANDARD
TIME_AW/YTD
GEOGRAPHY_AW/CONSOLIDATED.
```

The fact table is called XADEMO_AW_SALES_VIEW_4. It stores lowest level data and embedded totals for all level combinations.

```
execute cwm2_olap_table_map.Map_FactTbl_LevelKey
        ('XADEMO', 'ANALYTIC_CUBE_AW','XADEMO', 'XADEMO_AW_SALES_VIEW_4', 'ET',
         'DIM:XADEMO.PRODUCT_AW/HIER:STANDARD/GID:PRODUCT_GID/LVL:L4/COL:PRODUCT_ET;
          DIM:XADEMO.CHANNEL_AW/HIER:STANDARD/GID:CHANNEL_GID/LVL:STANDARD_1/COL:CHANNEL_ET;
          DIM:XADEMO.TIME_AW/HIER:YTD/GID:TIME_YTD_GID/LVL:L3/COL:TIME_YTD_ET;
          DIM:XADEMO.GEOGRAPHY_AW/HIER:CONSOLIDATED/GID:GEOG_CONS_GID/LVL:L4/COL:GEOG_CONS_ET;');
```

The following statement maps the F.SALES_AW measure to the SALES column in the fact table.

```
execute cwm2_olap_table_map.Map_FactTbl_Measure
        ('XADEMO', 'ANALYTIC_CUBE_AW', 'F.SALES_AW',
         'XADEMO', 'XADEMO_AW_SALES_VIEW_4', 'SALES',
         'DIM:XADEMO.PRODUCT_AW/HIER:STANDARD/LVL:L4/COL:PRODUCT_ET;
          DIM:XADEMO.CHANNEL_AW/HIER:STANDARD/LVL:STANDARD_1/COL:CHANNEL_ET;
          DIM:XADEMO.TIME_AW/HIER:YTD/LVL:L3/COL:TIME_YTD_ET;
          DIM:XADEMO.GEOGRAPHY_AW/HIER:CONSOLIDATED/LVL:L4/COL:GEOG_CONS_ET;');
```

Summary of CWM2_OLAP_TABLE_MAP Subprograms

Table 21–1 CWM2_OLAP_TABLE_MAP

Subprogram	Description
MAP_DIMTBL_HIERLEVELATTR Procedure on page 21-4	Maps a hierarchical level attribute to a column in a dimension table.
MAP_DIMTBL_HIERLEVEL Procedure on page 21-4	Maps a hierarchical level to one or more columns in a dimension table.
MAP_DIMTBL_HIERSORTKEY Procedure on page 21-5	Sorts the members of a hierarchy within a column of a dimension table.
MAP_DIMTBL_LEVELATTR Procedure on page 21-6	Maps a non-hierarchical level attribute to a column in a dimension table
MAP_DIMTBL_LEVEL Procedure on page 21-7	Maps a non-hierarchical level to one or more columns in a dimension table.
MAP_FACTTBL_LEVELKEY Procedure on page 21-8	Maps the dimensions of a cube to a fact table.
MAP_FACTTBL_MEASURE Procedure on page 21-9	Maps a measure to a column in a fact table.
REMOVEMAP_DIMTBL_HIERLEVELATTR Procedure on page 21-10	Removes the mapping of a hierarchical level attribute from a column in a dimension table.
REMOVEMAP_DIMTBL_HIERLEVEL Procedure on page 21-11	Removes the mapping of a hierarchical level from one or more columns in a dimension table.
REMOVEMAP_DIMTBL_HIERSORTKEY Procedure on page 21-11	Removes custom sorting criteria associated with columns in a dimension table.
REMOVEMAP_DIMTBL_LEVELATTR Procedure on page 21-12	Removes the mapping of a non-hierarchical level attribute from a column in a dimension table.
REMOVEMAP_DIMTBL_LEVEL Procedure on page 21-12	Removes the mapping of a non-hierarchical level from one or more columns in a dimension table.
REMOVEMAP_FACTTBL_LEVELKEY Procedure on page 21-13	Removes the mapping of a cube's dimensions from a fact table.
REMOVEMAP_FACTTBL_MEASURE Procedure on page 21-13	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 21–2 MAP_DIMTBL_HIERLEVELATTR Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
dimension_name	Name of the dimension.
dimension_attribute_name	Name of the dimension attribute.
hierarchy_name	Name of the hierarchy.
level_name	Name of the level.
level_attribute_name	Name of the level attribute associated with this level.
table_owner	Owner of the dimension table.
table_name	Name of the dimension table.
attrcol	Column in the dimension table to which this level attribute should be mapped.

Example

See Example 2–1, "Create an OLAP Dimension for the Products Table" and "Example: Mapping a Dimension" on page 21-1.

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 21–3 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.

Example

See Example 2–1, "Create an OLAP Dimension for the Products Table" and "Example: Mapping a Dimension" on page 21-1.

MAP_DIMTBL_HIERSORTKEY Procedure

This procedure specifies how to sort the members of a hierarchy within one or more columns of a dimension table.

Custom sorting can be specified for level columns or related attribute columns. Columns can be sorted in ascending or descending order, with nulls first or nulls last. By default, columns are sorted in ascending order and nulls are first.

Custom sorting information is optional. You can define a valid hierarchy without using the MAP_DIMTBL_HIERSORTKEY procedure.

Syntax

```
MAP_DIMTBL_HIERSORTKEY (
        dimension_owner     IN   VARCHAR2,
        dimension_name      IN   VARCHAR2,
        hierarchy_name      IN   VARCHAR2,
        sortcols            IN   VARCHAR2);
```

Parameters

Table 21–4 MAP_DIMTBL_HIERSORTKEY Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.

Table 21–4 (Cont.) MAP_DIMTBL_HIERSORTKEY Procedure Parameters

Parameter	Description
dimension_name	Name of the dimension.
hierarchy_name	Name of the hierarchy.
sortcols	A string specifying how to sort the values stored in one or more columns of a dimension table. For each column, the string specifies whether to sort in ascending or descending order, and whether to place nulls first or last. The default order is ascending with nulls first. Specify the columns in the order in which they should be sorted. The string should be enclosed in single quotes, and it should be in the following form. 'TBL:*table1_owner.table1_name* /COL:*column1_name* /ORD:ASC|DSC/NULL:FIRST|LAST; TBL:*table2_owner.table2_name* /COL:*column2_name* /ORD:ASC|DSC/NULL:FIRST|LAST;.....' **NOTE**: You do not need to repeat the table name for columns in the same table. You do not need to repeat the column names for a group of columns that share the same sorting attributes.

Example

The GLOBAL.CUSTOMER dimension, based on the table GLOBAL.CUSTOMER_DIM, has two hierarchies: SHIPMENTS_ROLLUP and MARKET_ROLLUP.

The MARKET_ROLLUP hierarchy has four levels: TOTAL_MARKET, MARKET_SEGMENT, ACCOUNT, and SHIP_TO. Each level has a corresponding attribute column containing a short description of the level. The attribute column names are: TOTAL_MARKET_DSC, MARKET_SEGMENT_DSC, ACCOUNT_DSC, and SHIP_TO_DSC.

The following command specifies that all the levels within the MARKET_ROLLUP hierarchy should be sorted in ascending order by description. The three most aggregate levels should be sorted with nulls first; the lowest level, with attribute column SHIP_TO_DSC, should be sorted with nulls last.

```
>EXECUTE cwm2_olap_table_map.map_dimtbl_hiersortkey
         ('GLOBAL', 'CUSTOMER', 'MARKET_ROLLUP',
          'TBL:GLOBAL.CUSTOMER_DIM/COL:TOTAL_MARKET_DSC/COL:MARKET_SEGMENT_DSC
                                  /COL:ACCOUNT_DSC/ORD:ASC/NULL:FIRST
                                  /COL:SHIP_TO_DSC/ORD:ASC/NULL:LAST');
```

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 21–5 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.

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 21–6 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.

Table 21–6 (Cont.) MAP_DIMTBL_LEVEL Procedure Parameters

Parameter	Description
keycol	Column in the dimension table to which this level should be mapped. This column will be the key for this level column in the fact table. If the level is stored in more than one column, separate the column names with commas. These columns will be the multicolumn key for these level columns in the fact table.

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 follows.

```
Dim1_Hier1, Dim2_Hier, Dim3_Hier1
Dim1_Hier1, Dim2_Hier, Dim3_Hier2
Dim1_Hier2, Dim2_Hier, Dim3_Hier1
Dim1_Hier2, Dim2_Hier, Dim3_Hier2
```

Typically the data for each hierarchy combination would be stored in a separate fact table.

For more information, see"Joining Fact Tables with Dimension Tables" on page 2-9.

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,
          dimkeytype           IN   VARCHAR2 DEFAULT NULL);
```

Parameters

Table 21–7 MAP_FACTTBL_LEVELKEY Procedure Parameters

Parameter	Description
cube_owner	Owner of the cube.
cube_name	Name of the cube.
facttable_owner	Owner of the fact table.
facttable_name	Name of the fact table.

Table 21–7 (Cont.) MAP_FACTTBL_LEVELKEY Procedure Parameters

Parameter	Description
storetype	One of the following: 'LOWESTLEVEL', for a fact table that stores only lowest level data 'ET', for a fact table that stores embedded totals for all level combinations in addition to lowest level data
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 as follows: 'DIM:*dim1_name*/HIER:*hier_name*; /GID:*gid_column*/LVL:*level_name* /COL:*map_column*; 'DIM:*dim2_name*/HIER:*hier_name*; /GID:*gid_column*/LVL:*level_name* /COL:*map_column*;.....' **Note**: The GID clause of the mapping string is only applicable to embedded totals. If you specify 'LOWESTLEVEL' for the *storetype* argument, do not include a GID clause in the mapping string. This string must also be specified as an argument to the MAP_FACTTBL_MEASURE procedure.
dimkeytype	This parameter is not currently used.

Example

Example 2–3, "Create an OLAP Cube for the COSTS Fact Table" illustrates the mapping commands for a fact table with a *storetype* of 'LOWESTLEVEL'.

"Example: Mapping a Cube" on page 21-2 illustrates the mapping commands for a fact table with a *storetype* of '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 21–8 MAP_FACTTBL_MEASURE Procedure Parameters

Parameter	Description
cube_owner	Owner of the cube.

Table 21–8 (Cont.) MAP_FACTTBL_MEASURE Procedure Parameters

Parameter	Description
cube_name	Name of the cube.
measure_name	Name of the measure to be mapped.
facttable_owner	Owner of the fact table.
facttable_name	Name of the fact table.
column_name	Column in the fact table to which the measure will be mapped.
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 as follows: 'DIM:*dim1_name*/HIER:*hier_name*; /GID:*gid_column*/LVL:*level_name* /COL:*map_column*; 'DIM:*dim2_name*/HIER:*hier_name*; /GID:*gid_column*/LVL:*level_name* /COL:*map_column*;.....' **Note**: The GID clause of the mapping string is only applicable to embedded totals. If you specify 'LOWESTLEVEL' for the *storetype* argument, do not include a GID clause in the mapping string. This string must also be specified as an argument to the MAP_FACTTBL_LEVELKEY procedure.

Example

See Example 2–3, "Create an OLAP Cube for the COSTS Fact Table" and "Example: Mapping a Cube" on page 21-2.

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 21–9 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.

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 21–10 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.

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 21–11 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.

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 21–12 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.

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 21–13 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.

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 21–14 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.

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 21–15 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 as follows: 'DIM:*dimname1*/HIER:*hiername1* /GID:*gid_columnname1*/LVL:*levelname1* /COL:*map_columnname1*; DIM:*dimname2*/HIER:*hiername2* /GID:*gid_columnname2*/LVL:*levelname2* /COL:*map_columnname2*;.....' Note that the GID clause of the mapping string is only applicable to embedded totals. If the measure contained only detail data and was mapped with a storage type of 'LOWESTLEVEL', do not include a GID clause in the mapping string. This string must also be specified as an argument to the MAP_FACTTBL_MEASURE and MAP_FACTTBL_LEVELKEY procedures.

22

CWM2_OLAP_VALIDATE

The CWM2_OLAP_VALIDATE package provides procedures for validating OLAP Catalog metadata.

See Also:

- "Validating OLAP Catalog Metadata" on page 2-10
- Chapter 23, "CWM2_OLAP_VERIFY_ACCESS"

This chapter discusses the following topics:

- About OLAP Catalog Metadata Validation
- Summary of CWM2_OLAP_VALIDATE Subprograms

About OLAP Catalog Metadata Validation

The validation process checks the structural integrity of the metadata and ensures that it is correctly mapped to columns in dimension tables and fact tables. Additional validation specific to the OLAP API is done if requested.

The procedures in CWM2_OLAP_VALIDATE validate the OLAP Catalog metadata created by Enterprise Manager as well as the metadata created by CWM2 procedures.

See Also: "Validating and Committing OLAP Catalog Metadata" on page 2-10 for additional information.

Structural Validation

Structural validation ensures that cubes and dimensions have all their required component parts. All the procedures in CWM2_OLAP_VALIDATE perform structural validation by default.

Cubes

To be structurally valid, a cube must meet the following criteria:

- It must have at least one valid dimension.
- It must have at least one measure.

Dimensions

To be structurally valid, a dimension must meet the following criteria:

- It must have at least one level.

- It may have one or more hierarchies. Each hierarchy must have at least one level.
- It may have one or more dimension attributes. Each dimension attribute must have at least one level attribute.

Mapping Validation

Mapping validation ensures that the metadata has been properly mapped to columns in tables or views. All the procedures in `CWM2_OLAP_VALIDATE` perform mapping validation by default.

Cubes

To be valid, a cube's mapping must meet the following criteria:

- It must be mapped to one or more fact tables.
- All of the cube's measures must be mapped to existing columns in a fact table. If there are multiple fact tables, all the measures must be in each one.
- Every dimension/hierarchy combination must be mapped to one of the fact tables.

Dimensions

To be valid, a dimension's mapping must meet the following criteria:

- All levels must be mapped to existing columns in a dimension table.
- Level attributes must be mapped to columns in the same table as the corresponding levels.

Validation Type

All the procedures in `CWM2_OLAP_VALIDATE` package take a validation type argument. The validation type can be one of the following:

`DEFAULT` -- Validates the basic structure of the metadata and its mapping to the source tables. To be valid, the metadata must meet the criteria specified in "Structural Validation" and "Mapping Validation" on page 22-2.

`OLAP API` -- Performs default validation plus the following:

- Validates that each dimension of an ET-style cube has dimension and level attributes `'ET KEY'` and `'GROUPING ID'` for all levels.
- Validates that time dimensions have dimension and level attributes `'END DATE'` and `'TIME SPAN'` for all levels.

Using Wildcards to Identify Metadata Entities

You can use wildcard characters to validate cubes and dimensions whose names meet certain criteria.

Wildcard characters are the underscore "_" and the percent sign "%". An underscore replaces any single character, and a percent sign replaces any zero or more characters. An underscore, but not a percent sign, is also a legal character in a metadata owner or entity name. Any underscore character in the owner or entity name is treated as a wildcard, unless you precede it with a backslash "\" which acts as an escape character.

For example, the following command validates all the cubes belonging to the owner 'GLOBAL'.

```
execute cwm2_olap_validate.validate_cube('GLOBAL', '%');
```

The following command validates all the cubes in the GLOBAL schema whose names start with 'a'.

```
execute cwm2_olap_validate.validate_cube('GLOBAL', 'a%');
```

If your database includes users 'TESTUSER1' and 'TESTUSER2', you could validate the 'TEST' cube belonging to each of these users with the following command.

```
execute cwm2_olap_validate.validate_cube('TESTUSER_', 'TEST');
```

If your database includes users 'TEST_USER1' and 'TEST_USER2', you could validate the 'TEST' cube belonging to each of these users with the following command.

```
execute cwm2_olap_validate.validate_cube('TEST/_USER_', 'TEST');
```

Summary of CWM2_OLAP_VALIDATE Subprograms

Table 22–1 CWM2_OLAP_VALIDATE

Subprogram	Description
VALIDATE_ALL_CUBES Procedure on page 22-5	Validates all the cubes in the OLAP Catalog.
VALIDATE_ALL_DIMENSIONS Procedure on page 22-5	Validates all the dimensions in the OLAP Catalog.
VALIDATE_CUBE Procedure on page 22-5	Validates one or more cubes in the OLAP Catalog.
VALIDATE_DIMENSION Procedure on page 22-6	Validates one or more dimensions in the OLAP Catalog.
VALIDATE_OLAP_CATALOG Procedure on page 22-7	Validates all the cubes and all the dimensions in the OLAP Catalog.

VALIDATE_ALL_CUBES Procedure

This procedure validates all the cubes the OLAP Catalog. This includes validation of all the dimensions associated with the cubes.

Cube validity status is displayed in the view ALL_OLAP2_CUBES.

Syntax

```
VALIDATE_ALL_CUBES (
              type_of_validation     IN   VARCHAR2 DEFAULT 'DEFAULT',
              verbose_report         IN   VARCHAR2 DEFAULT 'YES');
```

Parameters

Table 22–2 VALIDATE_ALL_CUBES Procedure Parameters

Parameter	Description
type_of_validation	'DEFAULT' or 'OLAP API'. See "Validation Type" on page 22-2.
verbose_report	'YES' or 'NO'. Whether to report all validation checks or only major events and errors. By default, all validation checks are reported.

VALIDATE_ALL_DIMENSIONS Procedure

This procedure validates all the dimensions in the OLAP Catalog.

Dimension validity status is displayed in the view ALL_OLAP2_DIMENSIONS.

Syntax

```
VALIDATE_ALL_DIMENSIONS (
              type_of_validation     IN   VARCHAR2 DEFAULT 'DEFAULT',
              verbose_report         IN   VARCHAR2 DEFAULT 'YES');
```

Parameters

Table 22–3 VALIDATE_ALL_DIMENSIONS Procedure Parameters

Parameter	Description
type_of_validation	'DEFAULT' or 'OLAP API'. See "Validation Type" on page 22-2.
verbose_report	'YES' or 'NO'. Whether to report all validation checks or only major events and errors. By default, all validation checks are reported.

VALIDATE_CUBE Procedure

This procedure validates a cube or group of cubes in the OLAP Catalog. This includes validation of all the dimensions associated with the cubes.

You can identify a group of cubes by specifying wildcard characters in the cube_owner and cube_name parameters. See "Using Wildcards to Identify Metadata Entities" on page 22-2.

The validity status of a cube is displayed in the view ALL_OLAP2_CUBES.

Syntax

```
VALIDATE_CUBE (
          cube_owner          IN   VARCHAR2,
          cube_name           IN   VARCHAR2,
          type_of_validation  IN   VARCHAR2 DEFAULT 'DEFAULT',
          verbose_report      IN   VARCHAR2 DEFAULT 'YES');
```

Parameters

Table 22–4 VALIDATE_CUBE Procedure Parameters

Parameter	Description
cube_owner	Owner of the cube. See "Using Wildcards to Identify Metadata Entities" on page 22-2.
cube_name	Name of the cube. See "Using Wildcards to Identify Metadata Entities" on page 22-2.
type_of_validation	'DEFAULT' or 'OLAP API'. See "Validation Type" on page 22-2.
verbose_report	'YES' or 'NO'. Whether to report all validation checks or only major events and errors. By default, all validation checks are reported.

VALIDATE_DIMENSION Procedure

This procedure validates a dimension or group of dimensions in the OLAP Catalog.

You can identify a group of dimensions by specifying wildcard characters in the cube_owner and cube_name parameters. See "Using Wildcards to Identify Metadata Entities" on page 22-2.

The validity status of an OLAP dimension is displayed in the view ALL_OLAP2_DIMENSIONS.

Syntax

```
VALIDATE_DIMENSION (
          dimension_owner      IN   VARCHAR2,
          dimension_name       IN   VARCHAR2,
          type_of_validation   IN    VARCHAR2 DEFAULT 'DEFAULT',
          verbose_report       IN   VARCHAR2 DEFAULT 'YES');
```

Parameters

Table 22–5 VALIDATE_DIMENSION Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension. See "Using Wildcards to Identify Metadata Entities" on page 22-2.
dimension_name	Name of the dimension. "Using Wildcards to Identify Metadata Entities" on page 22-2.
type_of_validation	'DEFAULT' or 'OLAP API'. See "Validation Type" on page 22-2.
verbose_report	'YES' or 'NO'. Whether to report all validation checks or only major events and errors. By default, all validation checks are reported.

VALIDATE_OLAP_CATALOG Procedure

This procedure validates all the metadata in the OLAP Catalog. This includes all the cubes (with their dimensions) and all the dimensions that are not associated with cubes.

VALIDATE_OLAP_CATALOG validates each standalone dimension in alphabetical order, then it validates each cube in alphabetical order.

Syntax

```
VALIDATE_OLAP_CATALOG (
               type_of_validation     IN   VARCHAR2 DEFAULT 'DEFAULT',
               verbose_report         IN   VARCHAR2 DEFAULT 'YES');
```

Parameters

Table 22–6 VALIDATE_OLAP_CATALOG Procedure Parameters

Parameter	Description
type_of_validation	'DEFAULT' or 'OLAP API'. See "Validation Type" on page 22-2.
verbose_report	'YES' or 'NO'. Whether to report all validation checks or only major events and errors. By default, all validation checks are reported.

23

CWM2_OLAP_VERIFY_ACCESS

The CWM2_OLAP_VERIFY_ACCESS package provides a procedure for validating an OLAP cube and verifying its accessibility to the OLAP API.

See Also:

- "Validating and Committing OLAP Catalog Metadata" on page 2-10
- Chapter 19, "CWM2_OLAP_METADATA_REFRESH"
- Chapter 22, "CWM2_OLAP_VALIDATE"

This chapter discusses the following topics:

Validating the Accessibility of an OLAP Cube

Cube validation procedures in the CWM2_OLAP_VALIDATE package validate the logical structure of an OLAP cube and check that it is correctly mapped to columns in dimension tables and fact tables. However, a cube may be entirely valid according to this criteria and still be inaccessible to your application.

For this reason, you may need to use the CWM2_OLAP_VERIFY_ACCESS package to check that the following additional criteria have also been met:

- The metadata tables used by the OLAP API Metadata Reader must be refreshed with the latest changes in the cube's metadata. If these MRV$ tables have not been updated, you must run the procedures in the CWM2_OLAP_METADATA_REFRESH package to enable access by the OLAP API.
- The identity of the application must have access to the source data that underlies the cube. The validation procedures in CWM2_OLAP_VALIDATE run under the SYS identity. These procedures may indicate that the cube is entirely valid, and yet the application may not be able to access it. If this is the case, you must grant the appropriate rights to the calling user.

Summary of CWM2_OLAP_VERIFY_ACCESS Subprograms

Table 23–1 CWM2_OLAP_VERIFY_ACCESS

Subprogram	Description
VERIFY_CUBE_ACCESS Procedure on page 23-3	Validates the cube and verifies its accessibility to an OLAP application.

VERIFY_CUBE_ACCESS Procedure

This procedure first validates a cube by calling the VALIDATE_CUBE procedure in the CWM2_OLAP_VALIDATE package. Additionally it checks that an OLAP API application running under the identity of the calling user has access to the cube.

Cube accessibility requirements are described in "Validating the Accessibility of an OLAP Cube" on page 23-1.

Syntax

```
VERIFY_CUBE_ACCESS (
          cube_owner             IN   VARCHAR2,
          cube_name              IN   VARCHAR2,
          type_of_validation     IN   VARCHAR2 DEFAULT 'DEFAULT',
          verbose_report         IN   VARCHAR2 DEFAULT 'YES');
```

Parameters

Table 23–2 VERIFY_CUBE_ACCESS Procedure Parameters

Parameter	Description
cube_owner	Owner of the cube.
cube_name	Name of the cube.
type_of_validation	'DEFAULT' or 'OLAP API'. See "Validation Type" on page 22-2.
verbose_report	'YES' or 'NO'. Whether to report all validation checks or only major events and errors. By default, all validation checks are reported.

24

DBMS_AW

The DBMS_AW package provides procedures and functions for interacting with analytic workspaces. With DBMS_AW, you can:

- Create, delete, copy, rename, and update analytic workspaces.
- Convert analytic workspaces from Oracle9*i* to Oracle 10*g* storage format.
- Attach analytic workspaces for processing within your session.
- Execute OLAP DML commands.
- Obtain information to help you manage sparsity and summary data within analytic workspaces.

See Also:

- *Oracle OLAP DML Reference* for information on analytic workspace objects and the syntax of individual OLAP DML commands.
- *Oracle OLAP Application Developer's Guide* for information about using analytic workspaces.

This chapter includes the following topics:

Managing Analytic Workspaces

To interact with Oracle OLAP, you must attach an analytic workspace to your session. From within SQL*Plus, you can use the following command to attach a workspace with read-only access.

```
SQL>execute dbms_aw.aw_attach ('awname');
```

Each analytic workspace is associated with a list of analytic workspaces. The read-only workspace EXPRESS.AW, which contains the OLAP engine code, is always attached last in the list. When you create a new workspace, it is attached first in the list by default.

You can reposition a workspace within the list by using keywords such as FIRST and LAST. For example, the following commands show how to move a workspace called GLOBAL.TEST2 from the second position to the first position on the list.

```
SQL>execute dbms_aw.execute ('aw list');

     TEST1 R/O UNCHANGED GLOBAL.TEST1
     TEST2 R/O UNCHANGED GLOBAL.TEST2
     EXPRESS R/O UNCHANGED SYS.EXPRESS

SQL>execute dbms_aw.aw_attach ('test2', false, false, 'first');
SQL>execute dbms_aw.execute ('aw list');

     TEST2 R/O UNCHANGED GLOBAL.TEST2
     TEST1 R/O UNCHANGED GLOBAL.TEST1
     EXPRESS R/O UNCHANGED SYS.EXPRESS
```

From within SQL*Plus, you can rename workspaces and make copies of workspaces. If you have a workspace attached with read/write access, you can update the workspace and save your changes in the permanent database table where the workspace is stored. You must do a SQL COMMIT to save the workspace changes within the database.

The following commands make a copy of the objects and data in workspace test2 in a new workspace called test3, update test3, and commit the changes to the database.

```
SQL>execute dbms_aw.aw_copy('test2', 'test3');
SQL>execute dbms_aw.aw_update('test3');
SQL>commit;
```

Converting an Analytic Workspace to Oracle 10*g* Storage Format

Analytic workspaces are stored in tables within the Database. The storage format for Oracle 10*g* analytic workspaces is different from the storage format used in Oracle9*i*. Analytic workspace storage format is described in the *Oracle OLAP Application Developer's Guide*.

When you upgrade an Oracle9*i* database to Oracle 10*g*, the upgraded database is automatically in Oracle9*i* compatibility mode, and the analytic workspaces are still in 9*i* storage format. If you want to use new Oracle 10*g* OLAP features, such as dynamic enablement and multi-writer, you must use DBMS_AW.CONVERT to convert these workspaces to the new storage format.

See Also:

- *Oracle Database Upgrade Guide* for more information on Database compatibility mode.
- Oracle *MetaLink* at http://metalink.oracle.com for more information about upgrading analytic workspaces.

Procedure: Convert an Analytic Workspace from 9*i* to10*g* Storage Format

To convert an Oracle9*i* compatible analytic workspace to Oracle 10*g* storage format, follow these steps:

1. Change the compatibility mode of the database to 10.0.0 or higher.
2. Log into the database with the identity of the analytic workspace.

3. In Oracle Database 10*g* SQL*Plus, use the following procedure to convert the workspace to the new storage format.
   - Rename the analytic workspace to a name like `aw_temp`.

     ```
     SQL>execute dbms_aw.aw_rename ('my_aw', 'aw_temp');
     ```

   - Convert the workspace to 10*g* storage format in a workspace with the original name.

     ```
     SQL>execute dbms_aw.convert ('aw_temp', 'my_aw');
     ```

     Note that standard form analytic workspaces typically include the workspace name in fully-qualified logical object names. For this reason, the upgraded workspace must have the same name as the original Oracle9*i* workspace.
4. Because you changed the Database compatibility mode to Oracle Database 10*g*, any new workspaces that you create are in the new storage format.

Procedure: Import a workspace from a 9*i* Database into a 10*g* Database

If you install Oracle Database 10*g* separately from your old Oracle9*i* Database installation, you must export the Oracle9*i* workspaces and import them into Oracle Database 10*g*. The export and import processes automatically convert the workspaces to the new storage format. Therefore you do not need to use `DBMS_AW.CONVERT` in this case.

Use the following procedure to export an Oracle9*i* analytic workspace and import it in an Oracle 10*g* database.

In Oracle Database 9*i* SQL*Plus, export the analytic workspace to the directory identified by the `SCRIPTS` directory object.

```
SQL>execute dbms_aw.execute ('aw attach ''awname''');
SQL>execute dbms_aw.execute ('allstat');
SQL>execute dbms_aw.execute ('cda scripts');
SQL>execute dbms_aw.execute ('export all to eif file ''filename''');
```

In Oracle 10*g* SQL*Plus, create a new workspace with the same name and schema, and import the EIF file from the `SCRIPTS` directory.

```
SQL>execute dbms_aw.execute ('aw create awname');
SQL>execute dbms_aw.execute ('cda scripts');
SQL>execute dbms_aw.execute ('import all from eif file ''filename''');
SQL>execute dbms_aw.execute ('update');
```

You can also use Oracle export and import utilities to move the entire schema, including the analytic workspaces to an Oracle 10*g* database. See *Oracle Database Utilities* and *Oracle Database Upgrade Guide*.

Embedding OLAP DML in SQL Statements

With the `DBMS_AW` package you can perform the full range of OLAP processing within analytic workspaces. You can import data from legacy workspaces, relational tables, or flat files. You can define OLAP objects and perform complex calculations.

Note: If you use the DBMS_AW package to create analytic workspaces from scratch, you may not be able to use OLAP utilities, such as Analytic Workspace Manager and the DBMS_AW Aggregate Advisor, which require standard form.

Methods for Executing OLAP DML Commands

The DBMS_AW package provides several procedures for executing ad hoc OLAP DML commands. Using the EXECUTE or INTERP_SILENT procedures or the INTERP or INTERCLOB functions, you can execute a single OLAP DML command or a series of commands separated by semicolons.

Which procedures you use will depend on how you want to direct output and on the size of the input and output buffers. For example, the EXECUTE procedure directs output to a printer buffer, the INTERP_SILENT procedure suppresses output, and the INTERP function returns the session log.

The DBMS_AW package also provides functions for evaluating OLAP expressions. The EVAL_TEXT function returns the result of a text expression, and EVAL_NUMBER returns the result of a numeric expression.

See Also: *Oracle OLAP DML Reference* for complete information about OLAP DML expressions.

Do not confuse the DBMS_AW functions EVAL_NUMBER and EVAL_TEXT with the SQL function OLAP_EXPRESSION. See Chapter 30, "OLAP_EXPRESSION" for more information.

Guidelines for Using Quotation Marks in OLAP DML Commands

The SQL processor evaluates the embedded 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 of DBMS_AW procedures:

- Wherever you would normally use a 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.

Using the Sparsity Advisor

Data can be stored in several different forms in an analytic workspace, depending on whether it is dense, sparse, or very sparse. The Sparsity Advisor is a group of subprograms in DBMS_AW that you can use to analyze the relational source data and get recommendations for storing it in an analytic workspace.

Data Storage Options in Analytic Workspaces

Analytic workspaces analyze and manipulate data in a multidimensional format that allocates one cell for each combination of dimension members. The cell can contain a data value, or it can contain an NA (null). Regardless of its content, the cell size is defined by the data type, for example, every cell in a DECIMAL variable is 8 bytes.

Variables can be either dense (they contain 30% or more cells with data values) or sparse (less than 30% data values). Most variables are sparse and many are extremely sparse.

Although data can also be stored in the multidimensional format used for analysis, other methods are available for storing sparse variables that make more efficient use of disk space and improve performance. Sparse data can be stored in a variable defined with a **composite** dimension. A composite has as its members the dimension-value combinations (called **tuples**) for which there is data. When a data value is added to a variable dimensioned by a composite, that action triggers the creation of a composite tuple. A composite is an index into one or more sparse data variables, and is used to store sparse data in a compact form. Very sparse data can be stored in a variable defined with a **compressed composite**, which uses a different algorithm for data storage from regular composites.

Selecting the Best Data Storage Method

In contrast to dimensional data, relational data is stored in tables in a very compact format, with rows only for actual data values. When designing an analytic workspace, you may have difficulty manually identifying sparsity in the source data and determining the best storage method. The Sparsity Advisor analyzes the source data in relational tables and recommends a storage method. The recommendations may include the definition of a composite and partitioning of the data variable.

The Sparsity Advisor consists of these procedures and functions:

SPARSITY_ADVICE_TABLE Procedure
ADD_DIMENSION_SOURCE Procedure
ADVISE_SPARSITY Procedure
ADVISE_DIMENSIONALITY Function
ADVISE_DIMENSIONALITY Procedure

The Sparsity Advisor also provides a public table type for storing information about the dimensions of the facts being analyzed. Three objects are used to define the table type:

```
DBMS_AW$_COLUMNLIST_T
DBMS_AW$_DIMENSION_SOURCE_T
DBMS_AW$_DIMENSION_SOURCES_T
```

The following SQL `DESCRIBE` statements show the object definitions.

```
SQL> describe dbms_aw$_columnlist_t
 dbms_aw$_columnlist_t TABLE OF VARCHAR2(100)

SQL> describe dbms_aw$_dimension_source_t
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 DIMNAME                                            VARCHAR2(100)
 COLUMNNAME                                         VARCHAR2(100)
 SOURCEVALUE                                        VARCHAR2(32767)
 DIMTYPE                                            NUMBER(3)
 HIERCOLS                                           DBMS_AW$_COLUMNLIST_T
 PARTBY                                             NUMBER(9)

SQL> describe dbms_aw$_dimension_sources_t
 dbms_aw$_dimension_sources_t TABLE OF DBMS_AW$_DIMENSION_SOURCE_T
```

Using the Sparsity Advisor

Take these steps to use the Sparsity Advisor:

1. Call SPARSITY_ADVICE_TABLE to create a table for storing the evaluation of the Sparsity Advisor.
2. Call ADD_DIMENSION_SOURCE for each dimension related by one or more columns to the fact table being evaluated.

   The information that you provide about these dimensions is stored in a DBMS_AW$_DIMENSION_SOURCES_T variable.
3. Call ADVISE_SPARSITY to evaluate the fact table.

   Its recommendations are stored in the table created by SPARSITY_ADVICE_TABLE. You can use these recommendations to make your own judgements about defining variables in your analytic workspace, or you can continue with the following step.
4. Call the ADVISE_DIMENSIONALITY procedure to get the OLAP DML object definitions for the recommended composite, partitioning, and variable definitions.

   or

   Use the ADVISE_DIMENSIONALITY function to get the OLAP DML object definition for the recommended composite and the dimension order for the variable definitions for a specific partition.

Example: Evaluating Sparsity in the GLOBAL Schema

Example 24–1 provides a SQL script for evaluating the sparsity of the UNITS_HISTORY_FACT table in the GLOBAL schema. In the GLOBAL analytic workspace, UNITS_HISTORY_FACT defines the Units Cube and will be the source for the UNITS variable. UNITS_HISTORY_FACT is a fact table with a primary key composed of foreign keys from four dimension tables. A fifth column contains the facts for Unit Sales.

The CHANNEL_DIM and CUSTOMER_DIM tables contain all of the information for the Channel and Customer dimensions in a basic star configuration. Three tables in a snowflake configuration provide data for the Time dimension: MONTH_DIM, QUARTER_DIM, and YEAR_DIM. The PRODUCT_CHILD_PARENT table is a parent-child table and defines the Product dimension.

Example 24–1 Sparsity Advisor Script for GLOBAL

```
CONNECT global/global
SET ECHO ON
SET LINESIZE 300
SET PAGESIZE 300
SET SERVEROUT ON FORMAT WRAPPED

-- Define and initialize an advice table named GLOBAL_SPARSITY_ADVICE
BEGIN
     dbms_aw.sparsity_advice_table();
EXCEPTION
     WHEN OTHERS THEN NULL;
END;
/

TRUNCATE TABLE aw_sparsity_advice;
```

```
DECLARE
     dimsources dbms_aw$_dimension_sources_t;
     dimlist VARCHAR2(500);
     sparsedim VARCHAR2(500);
     defs CLOB;
BEGIN
-- Provide information about all dimensions in the cube
     dbms_aw.add_dimension_source('channel', 'channel_id', dimsources,
         'channel_dim', dbms_aw.hier_levels,
          dbms_aw$_columnlist_t('channel_id', 'total_channel_id'));
     dbms_aw.add_dimension_source('product', 'item_id', dimsources,
          'product_child_parent', dbms_aw.hier_parentchild,
           dbms_aw$_columnlist_t('product_id', 'parent_id'));
     dbms_aw.add_dimension_source('customer', 'ship_to_id', dimsources,
         'customer_dim', dbms_aw.hier_levels,
          dbms_aw$_columnlist_t('ship_to_id', 'warehouse_id', 'region_id',
               'total_customer_id'));
     dbms_aw.add_dimension_source('time', 'month_id', dimsources,
           'SELECT m.month_id, q.quarter_id, y.year_id
                FROM time_month_dim m, time_quarter_dim q, time_year_dim y
                WHERE m.parent=q.quarter_id AND q.parent=y.year_id',
            dbms_aw.hier_levels,
            dbms_aw$_columnlist_t('month_id', 'quarter_id', 'year_id'));

-- Analyze fact table and provide advice without partitioning
     dbms_aw.advise_sparsity('units_history_fact', 'units_cube',
          dimsources, dbms_aw.advice_default, dbms_aw.partby_none);

commit;

-- Generate OLAP DML for composite and variable definitions
dimlist := dbms_aw.advise_dimensionality('units_cube', sparsedim,
           'units_cube_composite');
dbms_output.put_line('Dimension list:  ' || dimlist);
dbms_output.put_line('Sparse dimension:  ' || sparsedim);
dbms_aw.advise_dimensionality(defs, 'units_cube');
dbms_output.put_line('Definitions:  ');
dbms_aw.printlog(defs);

END;
/
```

Advice from Sample Program

The script in Example 24–1 generates the following information.

```
Dimension list:  <channel units_cube_composite<product customer time>>
Sparse dimension:  DEFINE units_cube_composite COMPOSITE <product customer time>
Definitions:
DEFINE units_cube.cp COMPOSITE <product customer time>
DEFINE units_cube NUMBER VARIABLE <channel units_cube.cp<product customer time>>
PL/SQL procedure successfully completed.
```

Information Stored in GLOBAL_SPARSITY_ADVICE Table

This SQL `SELECT` statement shows some of the columns from the `GLOBAL_SPARSITY_ADVICE` table, which is the basis for the recommended OLAP DML object definitions.

```
SELECT fact, dimension, dimcolumn, membercount nmem, leafcount nleaf,
     advice, density
```

```
    from aw_sparsity_advice
    WHERE cubename='units_cube';

FACT                DIMENSION    DIMCOLUMN     NMEM  NLEAF ADVICE        DENSITY
------------------- ------------ ------------ ----- ------ ------------ --------
units_history_fact  channel      channel_id       3      3 DENSE           .46182
units_history_fact  product      item_id         48     36 SPARSE          .94827
units_history_fact  customer     ship_to_id      61     61 SPARSE          .97031
units_history_fact  time         month_id        96     79 SPARSE          .97664
```

Using the Aggregate Advisor

The management of aggregate data within analytic workspaces can have significant performance implications. To determine an optimal set of dimension member combinations to preaggregate, you can use the ADVISE_REL and ADVISE_CUBE procedures in the DBMS_AW package. These procedures are known together as the **Aggregate Advisor.**

Based on a percentage that you specify, ADVISE_REL suggests a set of dimension members to preaggregate. The ADVISE_CUBE procedure suggests a set of members for each dimension of a cube.

Aggregation Facilities within the Workspace

Instructions for storing aggregate data are specified in a workspace object called an aggmap. The OLAP DML AGGREGATE command uses the aggmap to preaggregate the data. Any data that is not preaggregated is aggregated dynamically by the AGGREGATE function when the data is queried.

Choosing a balance between static and dynamic aggregation depends on many factors including disk space, available memory, and the nature and frequency of the queries that will run against the data. After weighing these factors, you may arrive at a percentage of the data to preaggregate.

Once you have determined the percentage of the data to preaggregate, you can use the Aggregate Advisor. These procedures analyze the distribution of dimension members within hierarchies and identify an optimal set of dimension members to preaggregate.

Example: Using the ADVISE_REL Procedure

Based on a precompute percentage that you specify, the ADVISE_REL procedure analyzes a family relation, which represents a dimension with all its hierarchical relationships, and returns a list of dimension members.

ADVISE_CUBE applies similar heuristics to each dimension in an aggmap for a cube.

See Also:

- "ADVISE_REL Procedure" on page 24-20
- ADVISE_CUBE Procedure on page 24-15

Example 24–2 uses the following sample Customer dimension to illustrate the ADVISE_REL procedure.

Sample Dimension: Customer in the Global Analytic Workspace

The Customer dimension in GLOBAL_AW.GLOBAL has two hierarchies: SHIPMENTS_ROLLUP with four levels, and MARKET_ROLLUP with three levels. The dimension has 106 members. This number includes all members at each level and all level names.

The members of the Customer dimension are integer keys whose text values are defined in long and short descriptions.

The following OLAP DML commands show information about the representation of the Customer dimension, which is in database standard form.

```
SQL>set serveroutput on
---- Number of members of Customer dimension
SQL>execute dbms_aw.execute('show statlen(customer)')
106

---- Hierarchies in Customer dimension;
SQL>execute dbms_aw.execute('rpr w 40 customer_hierlist');
CUSTOMER_HIERLIST
----------------------------------------
MARKET_ROLLUP
SHIPMENTS_ROLLUP

---- Levels in Customer dimension
SQL>execute dbms_aw.execute('rpr w 40 customer_levellist');
CUSTOMER_LEVELLIST
----------------------------------------
ALL_CUSTOMERS
REGION
WAREHOUSE
TOTAL_MARKET
MARKET_SEGMENT
ACCOUNT
SHIP_TO
---- Levels in each hierarchy from leaf to highest
SQL>execute dbms_aw.execute('report w 20 customer_hier_levels');

CUSTOMER_HIERL
IST           CUSTOMER_HIER_LEVELS
------------- --------------------
SHIPMENTS     SHIP_TO
              WAREHOUSE
              REGION
              TOTAL_CUSTOMER
MARKET_SEGMENT SHIP_TO
              ACCOUNT
              MARKET_SEGMENT
              TOTAL_MARKET

---- Parent relation showing parent-child relationships in the Customer dimension
---- Only show the last 20 members
SQL>execute dbms_aw.execute('limit customer to last 20');
SQL>execute dbms_aw.execute('rpr w 10 down customer w 20 customer_parentrel');
           -----------CUSTOMER_PARENTREL------------
           ------------CUSTOMER_HIERLIST------------
CUSTOMER      MARKET_ROLLUP       SHIPMENTS_ROLLUP
---------- -------------------- --------------------
103        44                   21
104        45                   21
105        45                   21
106        45                   21
7          NA                   NA
1          NA                   NA
8          NA                   1
9          NA                   1
10         NA                   1
```

```
11          NA                  8
12          NA                  10
13          NA                  9
14          NA                  9
15          NA                  8
16          NA                  9
17          NA                  8
18          NA                  8
19          NA                  9
20          NA                  9
21          NA                  10
---- Show text descriptions for the same twenty dimension members
SQL>execute dbms_aw.execute('report w 15 down customer w 35 across customer_hierlist:
<customer_short_description>');
ALL_LANGUAGES: AMERICAN_AMERICA
                ---------------------------CUSTOMER_HIERLIST---------------------------
                -----------MARKET_ROLLUP----------- ---------SHIPMENTS_ROLLUP----------
CUSTOMER            CUSTOMER_SHORT_DESCRIPTION          CUSTOMER_SHORT_DESCRIPTION
--------------- ---------------------------------- ----------------------------------
103             US Marine Svcs Washington          US Marine Svcs Washington
104             Warren Systems New York            Warren Systems New York
105             Warren Systems Philladelphia       Warren Systems Philladelphia
106             Warren Systems Boston              Warren Systems Boston
7               Total Market                       NA
1               NA                                 All Customers
8               NA                                 Asia Pacific
9               NA                                 Europe
10              NA                                 North America
11              NA                                 Australia
12              NA                                 Canada
13              NA                                 France
14              NA                                 Germany
15              NA                                 Hong Kong
16              NA                                 Italy
17              NA                                 Japan
18              NA                                 Singapore
19              NA                                 Spain
20              NA                                 United Kingdom
21              NA                                 United States
```

Example 24–2 ADVISE_REL: Suggested Preaggregation of the Customer Dimension

This example uses the GLOBAL Customer dimension described in Sample Dimension: Customer in the Global Analytic Workspace on page 24-9.

The following PL/SQL statements assume that you want to preaggregate 25% of the Customer dimension. ADVISE_REL returns the suggested set of members in a valueset.

```
SQL>set serveroutput on
SQL>execute dbms_aw.execute('aw attach global_aw.global');
SQL>execute dbms_aw.execute('define customer_preagg valueset customer');
SQL>execute dbms_aw.advise_rel('customer_parentrel', 'customer_preagg', 25);
SQL>execute dbms_aw.execute('show values(customer_preagg)');
31
2
4
5
6
7
1
8
9
20
21
```

The returned Customer members with their text descriptions, related levels, and related hierarchies, are shown as follows.

Customer Member	Description	Hierarchy	Level
31	Kosh Enterprises	MARKET_ROLLUP	ACCOUNT
2	Consulting	MARKET_ROLLUP	MARKET_SEGMENT
4	Government	MARKET_ROLLUP	MARKET_SEGMENT
5	Manufacturing	MARKET_ROLLUP	MARKET_SEGMENT
6	Reseller	MARKET_ROLLUP	MARKET_SEGMENT
7	TOTAL_MARKET	MARKET_ROLLUP	TOTAL_MARKET
1	ALL_CUSTOMERS	SHIPMENTS_ROLLUP	ALL_CUSTOMERS
8	Asia Pacific	SHIPMENTS_ROLLUP	REGION
9	Europe	SHIPMENTS_ROLLUP	REGION
20	United Kingdom	SHIPMENTS_ROLLUP	WAREHOUSE
21	United States	SHIPMENTS_ROLLUP	WAREHOUSE

Summary of DBMS_AW Subprograms

The following table describes the subprograms provided in DBMS_AW.

Table 24–1 DBMS_AW Subprograms

Subprogram	Description
ADD_DIMENSION_SOURCE Procedure on page 24-14	Populates a table type named DBMS_AW$_DIMENSION_SOURCES_T with information provided in its parameters about the dimensions of the cube.
ADVISE_CUBE Procedure on page 24-15	Suggests how to preaggregate a cube, based on a specified percentage of the cube's data.
ADVISE_DIMENSIONALITY Function on page 24-16	Returns a recommended composite definition for the cube and a recommended dimension order.
ADVISE_DIMENSIONALITY Procedure on page 24-18	Generates the OLAP DML commands for defining the recommended composite and measures in a cube.
ADVISE_REL Procedure on page 24-20	Suggests how to preaggregate a dimension, based on a specified percentage of the dimension's members.
ADVISE_SPARSITY Procedure on page 24-21	Analyzes a fact table for sparsity and populates a table with the results of its analysis.
AW_ATTACH Procedure on page 24-23	Attaches an analytic workspace to a session.
AW_COPY Procedure on page 24-24	Creates a new analytic workspace and populates it with the object definitions and data from another analytic workspace.
AW_CREATE Procedure on page 24-25	Creates a new, empty analytic workspace.
AW_DELETE on page 24-26	Deletes an analytic workspace
AW_DETACH Procedure on page 24-26	Detaches an analytic workspace from a session.
AW_RENAME Procedure on page 24-27	Changes the name of an analytic workspace.
AW_TABLESPACE Function on page 24-27	Returns the name of the tablespace in which a particular analytic workspace is stored.
AW_UPDATE Procedure on page 24-28	Saves changes made to an analytic workspace.
CONVERT Procedure on page 24-29	Converts an analytic workspace from 9*i* to 10*g* storage format.
EVAL_NUMBER Function on page 24-29	Returns the result of a numeric expression in an analytic workspace.
EVAL_TEXT Function on page 24-30	Returns the result of a text expression in an analytic workspace.
EXECUTE Procedure on page 24-31	Executes one or more OLAP DML commands. Input and output is limited to 4K. Typically used in an interactive session using an analytic workspace.
GETLOG Function on page 24-32	Returns the session log from the last execution of the INTERP or INTERPCLOB functions.
INFILE Procedure on page 24-32	Executes the OLAP DML commands specified in a file.

Table 24–1 (Cont.) DBMS_AW Subprograms

Subprogram	Description
INTERP Function on page 24-33	Executes one or more OLAP DML commands. Input is limited to 4K and output to 4G. Typically used in applications when the 4K limit on output for the `EXECUTE` procedure is too restrictive.
INTERPCLOB Function on page 24-34	Executes one or more OLAP DML commands. Input and output are limited to 4G. Typically used in applications when the 4K input limit of the `INTERP` function is too restrictive.
INTERP_SILENT Procedure on page 24-35	Executes one or more OLAP DML commands and suppresses the output. Input is limited to 4K and output to 4G.
OLAP_ON Function on page 24-35	Returns a boolean indicating whether or not the OLAP option is installed in the database.
OLAP_RUNNING Function on page 24-36	Returns a boolean indicating whether or not the OLAP option has been initialized in the current session.
PRINTLOG Procedure on page 24-36	Prints a session log returned by the `INTERP`, `INTERCLOB`, or `GETLOG` functions.
RUN Procedure on page 24-37	Executes one or more OLAP DML commands.
SHUTDOWN Procedure on page 24-39	Shuts down the current OLAP session.
SPARSITY_ADVICE_TABLE Procedure on page 24-39	Creates a table which the `ADVISE_SPARSITY` procedure will use to store the results of its analysis.
STARTUP Procedure on page 24-39	Starts an OLAP session without attaching a user-defined analytic workspace.

ADD_DIMENSION_SOURCE Procedure

The ADD_DIMENSION_SOURCE procedure populates a table type named DBMS_AW$_DIMENSION_SOURCES_T with information about the dimensions of a cube. This information is analyzed by the ADVISE_SPARSITY procedure.

Syntax

```
ADD_DIMENSION_SOURCE (
         dimname  IN      VARCHAR2,
         colname  IN      VARCHAR2,
         sources  IN OUT  dbms_aw$_dimension_sources_t,
         srcval   IN      VARCHAR2     DEFAULT NULL,
         dimtype  IN      NUMBER       DEFAULT NO_HIER,
         hiercols IN      columnlist_t DEFAULT NULL,
         partby   IN      NUMBER       DEFAULT PARTBY_DEFAULT);
```

Parameters

Table 24–2 ADD_DIMENSION_SOURCE Procedure Parameters

Parameter	Description
dimname	A name for the dimension. For clarity, use the logical name of the dimension in the analytic workspace.
colname	The name of the column in the fact table that maps to the dimension members for *dimname*.
sources	The name of an object (such as a PL/SQL variable) defined with a data type of DBMS_AW$_DIMENSION_SOURCES_T, which will be used to store the information provided by the other parameters.
srcval	The name of a dimension table, or a SQL statement that returns the columns that define the dimension. If this parameter is omitted, then *colname* is used.
dimtype	One of the following hierarchy types: DBMS_AW.HIER_LEVELS — Level-based hierarchy DBMS_AW.HIER_PARENTCHILD — Parent-child hierarchy DBMS_AW.MEASURE — Measure dimension DBMS_AW.NO_HIER — No hierarchy
hiercols	The names of the columns that define a hierarchy. For level-based hierarchies, list the base-level column first and the topmost-level column last. If the dimension has multiple hierarchies, choose the one you predict will be used the most frequently; only list the columns that define the levels of this one hierarchy. For parent-child hierarchies, list the child column first, then the parent column. For measure dimensions, list the columns in the fact table that will become dimension members.

Table 24–2 (Cont.) ADD_DIMENSION_SOURCE Procedure Parameters

Parameter	Description
partby	A keyword that controls partitioning. Use one of the following values: ■ DBMS_AW.PARTBY_DEFAULT Allow the Sparsity Advisor to determine whether or not partitioning is appropriate for this dimension. ■ DBMS_AW.PARTBY_NONE Do not allow partitioning on this dimension. ■ DBMS_AW.PARTBY_FORCE Force partitioning on this dimension. **Important**: Do not force partitioning on more than one dimension. ■ An integer value for the number of partitions you want created for this dimension.

Example

The following PL/SQL program fragment provides information about the TIME dimension for use by the Sparsity Advisor. The source data for the dimension is stored in a dimension table named TIME_DIM. Its primary key is named MONTH_ID, and the foreign key column in the fact table is also named MONTH_ID. The dimension hierarchy is level based as defined by the columns MONTH_ID, QUARTER_ID, and YEAR_ID.

The program declares a PL/SQL variable named DIMSOURCES with a table type of DBMS_AW$_DIMENSION_SOURCES_T to store the information.

```
DECLARE
     dimsources dbms_aw$_dimension_sources_t;
BEGIN
     dbms_aw.add_dimension_source('time', 'month_id', dimsources,
          'time_dim', dbms_aw.hier_levels,
          dbms_aw$_columnlist_t('month_id', 'quarter_id', 'year_id'));
                    .
                    .
                    .
END;
/
```

See Also

"Using the Sparsity Advisor" on page 24-4.

ADVISE_CUBE Procedure

The ADVISE_CUBE procedure helps you determine how to preaggregate a standard form cube in an analytic workspace. When you specify a percentage of the cube's data to preaggregate, ADVISE_CUBE recommends a set of members to preaggregate from each of the cube's dimensions.

The ADVISE_CUBE procedure takes an aggmap and a precompute percentage as input. The aggmap must have a precompute clause in each of its RELATION statements. The precompute clause must consist of a valueset. Based on the precompute percentage that you specify, ADVISE_CUBE returns a set of dimension members in each valueset.

Syntax

```
ADVISE_CUBE (
          aggmap_name             IN   VARCHAR2,
          precompute_percentage   IN   INTEGER DEFAULT 20,
          compressed              IN   BOOLEAN DEFAULT FALSE);
```

Parameters

Table 24–3 ADVISE_CUBE Procedure Parameters

Parameter	Description
aggmap_name	The name of an aggmap associated with the cube. Each RELATION statement in the aggmap must have a precompute clause containing a valueset. ADVISE_CUBE returns a list of dimension members in each valueset. If the valueset is not empty, ADVISE_CUBE deletes its contents before adding new values.
precompute_percentage	A percentage of the cube's data to preaggregate. The default is 20%.
compressed	Controls whether the advice is for a regular composite (FALSE) or a compressed composite (TRUE).

Example

This example illustrates the ADVISE_CUBE procedure with a cube called UNITS dimensioned by PRODUCT and TIME. ADVISE_CUBE returns the dimension combinations to include if you want to preaggregate 40% of the cube's data.

```
set serveroutput on
--- View valuesets
SQL>execute dbms_aw.execute('describe prodvals');
     DEFINE PRODVALS VALUESET PRODUCT
SQL>execute dbms_aw.execute('describe timevals');
     DEFINE TIMEVALS VALUESET TIME
--- View aggmap
SQL>execute dbms_aw.execute ('describe units_agg');
     DEFINE UNITS_AGG AGGMAP
          RELATION product_parentrel PRECOMPUTE (prodvals)
          RELATION time_parentrel PRECOMPUTE (timevals)
SQL>EXECUTE dbms_aw.advise_cube ('units_agg', 40);
----
---- The results are returned in the prodvals and timevals valuesets
```

See Also

"Using the Aggregate Advisor" on page 24-8.

ADVISE_DIMENSIONALITY Function

The ADVISE_DIMENSIONALITY function returns an OLAP DML definition of a composite dimension and the dimension order for variables in the cube, based on the sparsity recommendations generated by the ADVISE_SPARSITY procedure for a particular partition.

Syntax

```
ADVISE_DIMENSIONALITY (
          cubename   IN     VARCHAR2,
          sparsedfn  OUT    VARCHAR2
```

```
          sparsename IN     VARCHAR2 DEFAULT NULL,
          partnum    IN     NUMBER DEFAULT 1,
          advtable   IN     VARCHAR2 DEFAULT NULL)
     RETURN VARCHAR2;
```

Parameters

Table 24–4 ADVISE_DIMENSIONALITY Function Parameters

Parameter	Description
cubename	The same *cubename* value provided in the call to ADVISE_SPARSITY.
sparsedfn	The name of an object (such as a PL/SQL variable) in which the definition of the composite dimension will be stored.
sparsename	An object name for the composite. The default value is *cubename*.cp.
partnum	The number of a partition. By default, you see only the definition of the first partition.
advtable	The name of a table created by the SPARSITY_ADVICE_TABLE procedure for storing the results of analysis.

Example

The following PL/SQL program fragment defines two variables to store the recommendations returned by the ADVISE_DIMENSIONALITY function. SPARSEDIM stores the definition of the recommended composite, and DIMLIST stores the recommended dimension order of the cube.

```
DECLARE
     sparsedim VARCHAR2(500);
     dimlist VARCHAR2(500);
BEGIN
-- Calls to ADD_DIMENSION_SOURCE and ADVISE_SPARSITY omitted here
          .
          .
          .
dimlist := dbms_aw.advise_dimensionality('units_cube', sparsedim);
dbms_output.put_line('Sparse dimension:  ' || sparsedim);
dbms_output.put_line('Dimension list:  ' || dimlist);
END;
/
```

The program uses DBMS_OUTPUT.PUT_LINE to display the results of the analysis. The Sparsity Advisor recommends a composite dimension for the sparse dimensions, which are PRODUCT, CUSTOMER, and TIME. The recommended dimension order for UNITS_CUBE is CHANNEL followed by this composite.

```
Sparse dimension:  DEFINE units_cube.cp COMPOSITE <product customer time>
Dimension list:  channel units_cube.cp<product customer time>
```

The next example uses the Sparsity Advisor to evaluate the SALES table in the Sales History sample schema. A WHILE loop displays the recommendations for all partitions.

```
DECLARE
     dimlist VARCHAR2(500);
     sparsedim VARCHAR2(500);
     counter NUMBER(2) := 1;
```

```
     maxpart NUMBER(2);
BEGIN
-- Calls to ADD_DIMENSION_SOURCE and ADVISE_SPARSITY omitted here
          .
          .
          .

select max(partnum) into maxpart from sh_sparsity_advice;
WHILE counter <= maxpart LOOP
dimlist := dbms_aw.advise_dimensionality('sales_cube', sparsedim,
   'sales_cube_composite', counter,'sh_sparsity_advice');
dbms_output.put_line('Dimension list:  ' || dimlist);
dbms_output.put_line('Sparse dimension:  ' || sparsedim);
counter := counter+1;
END LOOP;
dbms_aw.advise_dimensionality(defs,'sales_cube', 'sales_cube_composite',
   'DECIMAL', 'sh_sparsity_advice');
dbms_output.put_line('Definitions:  ');
dbms_aw.printlog(defs);
END;
/
```

The Sparsity Advisor recommends 11 partitions; the first ten use the same composite. The last partition uses a different composite. (The SH_SPARSITY_ADVICE table shows that TIME_ID is dense in the last partition, whereas it is very sparse in the other partitions.)

```
Dimension list:  sales_cube_composite<time channel product promotion customer>
Sparse dimension:  DEFINE sales_cube_composite COMPOSITE COMPRESSED <time channel product promotion customer>
Dimension list:   sales_cube_composite<time channel product promotion customer>
Sparse dimension:  DEFINE sales_cube_composite COMPOSITE COMPRESSED <time channel product promotion customer>
                   .
                   .
                   .
Dimension list:  time sales_cube_composite<channel product promotion customer>
Sparse dimension:  DEFINE sales_cube_composite COMPOSITE COMPRESSED <channel product promotion customer>
```

See Also

"Using the Sparsity Advisor" on page 24-4.

ADVISE_DIMENSIONALITY Procedure

The ADVISE_DIMENSIONALITY procedure evaluates the information provided by the ADVISE_SPARSITY procedure and generates the OLAP DML commands for defining a composite and a variable in the analytic workspace.

Syntax

```
ADVISE_DIMENSIONALITY (
          output     OUT    CLOB,
          cubename   IN     VARCHAR2,
          sparsename IN     VARCHAR2 DEFAULT NULL,
          dtype      IN     VARCHAR2 DEFAULT 'NUMBER',
          advtable   IN     VARCHAR2 DEFAULT NULL);
```

Parameters

Table 24–5 ADVISE_DIMENSIONALITY Procedure Parameters

Parameter	Description
output	The name of an object (such as a PL/SQL variable) in which the recommendations of the procedure will be stored.
cubename	The same *cubename* value provided in the call to ADVISE_SPARSITY.
sparsename	An object name for the sample composite. The default value is *cubename*.cp.
dtype	The OLAP DML data type of the sample variable.
advtable	The name of the table created by the SPARSITY_ADVICE_TABLE procedure in which the results of the analysis are stored.

Example

The following PL/SQL program fragment defines a variable named DEFS to store the recommended definitions.

```
DECLARE
     defs CLOB;
BEGIN
-- Calls to ADD_DIMENSION_SOURCE and ADVISE_SPARSITY omitted here
          .
          .
          .
dbms_aw.advise_dimensionality(defs, 'units_cube_measure_stored',
     'units_cube_composite', 'DECIMAL');
dbms_output.put_line('Definitions:  ');
dbms_aw.printlog(defs);
END;
/
```

The program uses the DBMS_OUTPUT.PUT_LINE and DBMS_AW.PRINTLOG procedures to display the recommended object definitions.

```
Definitions:
DEFINE units_cube.cp COMPOSITE <product customer time>
DEFINE units_cube NUMBER VARIABLE <channel units_cube.cp<product customer time>>
```

In contrast to the Global schema, which is small and dense, the Sales cube in the Sales History sample schema is large and very sparse, and the Sparsity Advisor recommends 11 partitions. The following excerpt shows some of the additional OLAP DML definitions for defining a partition template and moving the TIME dimension members to the various partitions.

```
Definitions:
DEFINE sales_cube_composite_p1 COMPOSITE COMPRESSED <time channel product promotion customer>
DEFINE sales_cube_composite_p2 COMPOSITE COMPRESSED <time channel product promotion customer>
DEFINE sales_cube_composite_p3 COMPOSITE COMPRESSED <time channel product promotion customer>
DEFINE sales_cube_composite_p4 COMPOSITE COMPRESSED <time channel product promotion customer>
DEFINE sales_cube_composite_p5 COMPOSITE COMPRESSED <time channel product promotion customer>
DEFINE sales_cube_composite_p6 COMPOSITE COMPRESSED <time channel product promotion customer>
DEFINE sales_cube_composite_p7 COMPOSITE COMPRESSED <time channel product promotion customer>
DEFINE sales_cube_composite_p8 COMPOSITE COMPRESSED <time channel product promotion customer>
DEFINE sales_cube_composite_p9 COMPOSITE COMPRESSED <time channel product promotion customer>
DEFINE sales_cube_composite_p10 COMPOSITE COMPRESSED <time channel product promotion customer>
```

```
DEFINE sales_cube_composite_p11 COMPOSITE <channel product promotion customer>
DEFINE sales_cube_pt PARTITION TEMPLATE <time channel product promotion customer> -
  PARTITION BY LIST (time) -
   (PARTITION p1 VALUES () <sales_cube_composite_p1<>> -
    PARTITION p2 VALUES () <sales_cube_composite_p2<>> -
    PARTITION p3 VALUES () <sales_cube_composite_p3<>> -
    PARTITION p4 VALUES () <sales_cube_composite_p4<>> -
    PARTITION p5 VALUES () <sales_cube_composite_p5<>> -
    PARTITION p6 VALUES () <sales_cube_composite_p6<>> -
    PARTITION p7 VALUES () <sales_cube_composite_p7<>> -
    PARTITION p8 VALUES () <sales_cube_composite_p8<>> -
    PARTITION p9 VALUES () <sales_cube_composite_p9<>> -
    PARTITION p10 VALUES () <sales_cube_composite_p10<>> -
    PARTITION p11 VALUES () <time sales_cube_composite_p11<>>)
MAINTAIN sales_cube_pt MOVE TO PARTITION p1 -
   '06-JAN-98', '07-JAN-98', '14-JAN-98', '21-JAN-98', -
   '24-JAN-98', '28-JAN-98', '06-FEB-98', '07-FEB-98', -
   '08-FEB-98', '16-FEB-98', '21-FEB-98', '08-MAR-98', -
   '20-MAR-98', '03-JAN-98', '26-JAN-98', '27-JAN-98'
MAINTAIN sales_cube_pt MOVE TO PARTITION p1 -
   '31-JAN-98', '11-FEB-98', '12-FEB-98', '13-FEB-98', -
   '15-FEB-98', '17-FEB-98', '14-MAR-98', '18-MAR-98', -
   '26-MAR-98', '30-MAR-98', '05-JAN-98', '08-JAN-98', -
   '10-JAN-98', '16-JAN-98', '23-JAN-98', '01-FEB-98'
MAINTAIN sales_cube_pt MOVE TO PARTITION p1 -
   '14-FEB-98', '28-FEB-98', '05-MAR-98', '07-MAR-98', -
   '15-MAR-98', '19-MAR-98', '17-JAN-98', '18-JAN-98', -
   '22-JAN-98', '25-JAN-98', '03-FEB-98', '10-FEB-98', -
   '19-FEB-98', '22-FEB-98', '23-FEB-98', '26-FEB-98'
          .
          .
          .
```

See Also

"Using the Sparsity Advisor" on page 24-4.

ADVISE_REL Procedure

The ADVISE_REL procedure helps you determine how to preaggregate a standard form dimension in an analytic workspace. When you specify a percentage of the dimension to preaggregate, ADVISE_REL recommends a set of dimension members.

The ADVISE_REL procedure takes a family relation, a valueset, and a precompute percentage as input. The family relation is a standard form object that specifies the hierarchical relationships between the members of a dimension. The valueset must be defined from the dimension to be analyzed. Based on the precompute percentage that you specify, ADVISE_REL returns a set of dimension members in the valueset.

Syntax

```
ADVISE_REL (
          family_relation_name    IN   VARCHAR2,
          valueset_name           IN   VARCHAR2,
          precompute_percentage   IN   INTEGER DEFAULT 20,
          compressed              IN   BOOLEAN DEFAULT FALSE);
```

Parameters

Table 24–6 ADVISE_REL Procedure Parameters

Parameter	Description
family_relation_name	The name of a family relation, which specifies a dimension and the hierarchical relationships between the dimension members.
valueset_name	The name of a valueset to contain the results of the procedure. The valueset must be defined from the dimension in the family relation. If the valueset is not empty, ADVISE_REL deletes its contents before adding new values.
precompute_percentage	A percentage of the dimension to preaggregate. The default is 20%.
compressed	Controls whether the advice is for a regular composite (FALSE) or a compressed composite (TRUE).

See Also

"Using the Aggregate Advisor" on page 24-8.

ADVISE_SPARSITY Procedure

The ADVISE_SPARSITY procedure analyzes a fact table for sparsity using information about its dimensions provided by the ADD_DIMENSION_SOURCE procedure. It populates a table created by the SPARSITY_ADVICE_TABLE procedure with the results of its analysis.

Syntax

```
ADVISE_SPARSITY (
          fact       IN      VARCHAR2,
          cubename   IN      VARCHAR2,
          dimsources IN      dbms_aw$_dimension_sources_t,
          advmode    IN      BINARY_INTEGER DEFAULT ADVICE_DEFAULT,
          partby     IN      BINARY_INTEGER DEFAULT PARTBY_DEFAULT,
          advtable   IN      VARCHAR2 DEFAULT NULL);
```

Parameters

Table 24–7 ADVISE_SPARSITY Procedure Parameters

Parameter	Description
fact	The name of the source fact table.
cubename	A name for the facts being analyzed, such as the name of the logical cube in the analytic workspace.
dimsources	The name of the object type where the ADD_DIMENSION_SOURCE procedure has stored information about the cube's dimensions.
advmode	The level of advise you want to see. Select one of the following values: DBMS_AW.ADVICE_DEFAULT DBMS_AW.ADVICE_FAST DBMS_AW.ADVICE_FULL

Table 24–7 (Cont.) ADVISE_SPARSITY Procedure Parameters

Parameter	Description
partby	A keyword that controls partitioning. Use one of the following values: ■ DBMS_AW.PARTBY_DEFAULT Allow the Sparsity Advisor to determine whether or not partitioning is appropriate. ■ DBMS_AW.PARTBY_NONE Do not allow partitioning. ■ DBMS_AW.PARTBY_FORCE Force partitioning.
advtable	The name of a table created by the procedure for storing the results of analysis.

Output Description

Table 24–8 describes the information generated by ADVISE_SPARSITY.

Table 24–8 Output Column Descriptions

Column	Datatype	NULL	Description
CUBENAME	VARCHAR2(100)	NOT NULL	The values of *cubename* in calls to ADVISE_SPARSITY, typically the name of the logical cube.
FACT	VARCHAR2(4000)	NOT NULL	The values of *fact* in calls to ADVISE_SPARSITY; the name of the fact table that will provide the source data for one or more analytic workspace variables.
DIMENSION	VARCHAR2(100)	NOT NULL	The logical names of the cube's dimensions; the dimensions described in calls to ADVISE_DIMENSIONALITY.
DIMCOLUMN	VARCHAR2(100)		The names of dimension columns in *fact* (the source fact table), which relate to a dimension table.
DIMSOURCE	VARCHAR2(4000)		The names of the dimension tables.
MEMBERCOUNT	NUMBER(12,0)		The total number of dimension members at all levels.
LEAFCOUNT	NUMBER(12,0)		The number of dimension members at the leaf (or least aggregate) level.
ADVICE	VARCHAR2(10)	NOT NULL	The sparsity evaluation of the dimension: DENSE, SPARSE, or COMPRESSED.
POSITION	NUMBER(4,0)	NOT NULL	The recommended order of the dimensions.
DENSITY	NUMBER(11,8)		A number that provides an indication of sparsity relative to the other dimensions. The larger the number, the more sparse the dimension.
PARTNUM	NUMBER(6,0)	NOT NULL	The number of the partition described in the PARTBY and PARTTOPS columns. If partitioning is not recommended, then 1 is the maximum number of partitions.
PARTBY	CLOB		A list of all dimension members that should be stored in this partition. This list is truncated in SQL*Plus unless you significantly increase the size of the LONG setting.
PARTTOPS	CLOB		A list of top-level dimension members for this partition.

Example

The following PL/SQL program fragment analyzes the sparsity characteristics of the UNITS_HISTORY_FACT table.

```
DECLARE
     dimsources dbms_aw$_dimension_sources_t;
```

```
BEGIN
-- Calls to ADD_DIMENSION_SOURCE for each dimension in the cube
                .
                .
                .

   dbms_aw.advise_sparsity('units_history_fact', 'units_cube', dimsources,
        dbms_aw.advice_default);

END;
/
```

The following SELECT command displays the results of the analysis, which indicate that there is one denser dimension (CHANNEL) and three comparatively sparse dimensions (PRODUCT, CUSTOMER, and TIME).

```
SQL> SELECT fact, dimension, dimcolumn, membercount nmem, leafcount nleaf, advice, density
       FROM aw_sparsity_advice
       WHERE cubename='units_cube';

FACT                DIMENSION    DIMCOLUMN     NMEM  NLEAF ADVICE          DENSITY
------------------- ------------ ------------ ----- ------ ------------ ----------
units_history_fact  channel      channel_id       3      3 DENSE         .86545382
units_history_fact  product      item_id         36     36 SPARSE        .98706809
units_history_fact  customer     ship_to_id      61     62 SPARSE        .99257713
units_history_fact  time         month_id        96     80 SPARSE        .99415964
```

See Also

"Using the Sparsity Advisor" on page 24-4.

AW_ATTACH Procedure

The AW_ATTACH procedure attaches an analytic workspace to your SQL session so that you can access its contents. The analytic workspace remains attached until you explicitly detach it, or you end your session.

AW_ATTACH can also be used to create a new analytic workspace, but the AW_CREATE procedure is provided specifically for that purpose.

Syntax

```
AW_ATTACH (
          awname        IN VARCHAR2,
          forwrite      IN BOOLEAN DEFAULT FALSE,
          createaw      IN BOOLEAN DEFAULT FALSE,
          attargs       IN VARCHAR2 DEFAULT NULL,
          tablespace    IN VARCHAR2 DEFAULT NULL);
AW_ATTACH (
          schema        IN VARCHAR2,
          awname        IN VARCHAR2,
          forwrite      IN BOOLEAN DEFAULT FALSE,
          createaw      IN BOOLEAN DEFAULT FALSE,
          attargs       IN VARCHAR2 DEFAULT NULL,
          tablespace    IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 24–9 AW_ATTACH Procedure Parameters

Parameter	Description
schema	The schema that owns *awname*.
awname	The name of an existing analytic workspace, unless *createaw* is specified as TRUE. See the description of *createaw*.
forwrite	TRUE attaches the analytic workspace in read/write mode, giving you exclusive access and full administrative rights to the analytic workspace. FALSE attaches the analytic workspace in read-only mode.
createaw	TRUE creates an analytic workspace named *awname*. If *awname* already exists, then an error is generated. FALSE attaches an existing analytic workspace named *awname*.
attargs	Keywords for attaching an analytic workspace, such as FIRST or LAST, as described in the *Oracle OLAP DML Reference* under the AW command.

Example

The following command attaches an analytic workspace named GLOBAL in read/write mode.

```
SQL>execute dbms_aw.aw_attach('global', true);
```

The next command creates an analytic workspace named GLOBAL_PROGRAMS in the user's schema. GLOBAL_PROGRAMS is attached read/write as the last user-owned analytic workspace.

```
SQL>execute dbms_aw.aw_attach('global_programs', true, true, 'last');
```

This command attaches an analytic workspace named SH from the SH_AW schema in read-only mode.

```
SQL>execute dbms_aw.aw_attach('sh_aw', 'sh');
```

See Also

"Managing Analytic Workspaces" on page 24-1.

AW_COPY Procedure

The AW_COPY procedure copies the object definitions and data from one analytic workspace into a new analytic workspace.

AW_COPY detaches the original workspace and attaches the new workspace first with read/write access.

Syntax

```
AW_COPY (
          oldname          IN VARCHAR2,
          newname          IN VARCHAR2,
          tablespace       IN VARCHAR2 DEFAULT NULL,
          partnum          IN NUMBER DEFAULT 8);
```

Parameters

Table 24–10 AW_COPY Procedure Parameters

Parameter	Description
oldname	The name of an existing analytic workspace that contains object definitions. The workspace cannot be empty.
newname	A name for the new analytic workspace that is a copy of *oldname*.
tablespace	The name of a tablespace in which *newname* will be stored. If this parameter is omitted, then the analytic workspace is created in the user's default tablespace.
partnum	The number of partitions that will be created for the AW$*newname* table.

Example

The following command creates a new analytic workspace named DEMO and copies the contents of GLOBAL into it. The workspace is stored in a table named AW$DEMO, which has three partitions and is stored in the user's default tablespace.

```
SQL>execute dbms_aw.aw_copy('global', 'demo', null, 3);
```

See Also

"Managing Analytic Workspaces" on page 24-1.

AW_CREATE Procedure

The AW_CREATE procedure creates a new, empty analytic workspace and makes it the current workspace in your session.

The current workspace is first in the list of attached workspaces.

Syntax

```
AW_CREATE (
           awname        IN VARCHAR2 ,
           tablespace    IN VARCHAR2 DEFAULT NULL ,
           partnum       IN NUMBER DEFAULT 8 );
AW_CREATE (
           schema        IN VARCHAR2 ,
           awname        IN VARCHAR2 ,
           tablespace    IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 24–11 AW_CREATE Procedure Parameters

Parameter	Description
schema	The schema that owns *awname*.
awname	The name of a new analytic workspace. The name must comply with the naming requirements for a table in an Oracle database. This procedure creates a table named AW$*awname*, in which the analytic workspace is stored.
tablespace	The tablespace in which the analytic workspace will be created. If you omit this parameter, the analytic workspace is created in your default tablespace.
partnum	The number of partitions that will be created for the AW$*awname* table.

Example

The following command creates a new, empty analytic workspace named GLOBAL. The new analytic workspace is stored in a table named AW$GLOBAL with eight partitions in the user's default tablespace.

```
SQL>execute dbms_aw.aw_create('global');
```

The next command creates an analytic workspace named DEMO in the GLOBAL_AW schema. AW$DEMO will have two partitions and will be stored in the GLOBAL tablespace.

```
SQL>execute dbms_aw.aw_create('global_aw.demo', 'global', 2);
```

AW_DELETE

The AW_DELETE procedure deletes an existing analytic workspace.

Syntax

```
AW_DELETE (
    awname        IN VARCHAR2);
AW_DELETE (
    schema        IN VARCHAR2,
    awname        IN VARCHAR2);
```

Parameters

Table 24–12 AW_DELETE Procedure Parameters

Parameter	Description
schema	The schema that owns *awname*.
awname	The name of an existing analytic workspace that you want to delete along with all of its contents. You must be the owner of *awname* or have DBA rights to delete it, and it cannot currently be attached to your session. The AW$*awname* file is deleted from the database.

Example

The following command deletes the GLOBAL analytic workspace in the user's default schema.

```
SQL>execute dbms_aw.aw_delete('global');
```

AW_DETACH Procedure

The AW_DETACH procedure detaches an analytic workspace from your session so that its contents are no longer accessible. All changes that you have made since the last update are discarded. Refer to "AW_UPDATE Procedure" on page 24-28 for information about saving changes to an analytic workspace.

Syntax

```
AW_DETACH (
          awname        IN VARCHAR2);
AW_DETACH (
          schema        IN VARCHAR2,
          awname        IN VARCHAR2);
```

Parameters

Table 24–13 AW_DETACH Procedure Parameters

Parameter	Description
schema	The schema that owns *awname*.
awname	The name of an attached analytic workspace that you want to detach from your session.

Example

The following command detaches the GLOBAL analytic workspace.

```
SQL>execute dbms_aw.aw_detach('global');
```

AW_RENAME Procedure

The AW_RENAME procedure changes the name of an analytic workspace.

Syntax

```
AW_RENAME (
          oldname       IN VARCHAR2,
          newname       IN VARCHAR2 );
```

Parameters

Table 24–14 AW_RENAME Procedure Parameters

Parameter	Description
oldname	The current name of the analytic workspace. The analytic workspace cannot be attached to any session.
newname	The new name of the analytic workspace.

Example

The following command changes the name of the GLOBAL analytic workspace to DEMO.

```
SQL>execute dbms_aw.aw_rename('global', 'demo');
```

See Also

"Procedure: Convert an Analytic Workspace from 9i to10g Storage Format" on page 24-2.

AW_TABLESPACE Function

The AW_TABLESPACE function returns the name of the tablespace in which a particular analytic workspace is stored.

Syntax

```
AW_TABLESPACE (
          awname        IN VARCHAR2)
     RETURN VARCHAR2;
AW_TABLESPACE (
          schema        IN VARCHAR2,
          awname        IN VARCHAR2)
```

```
     RETURN VARCHAR2;
```

Returns

Name of a tablespace.

Parameters

Table 24–15 AW_TABLESPACE Function Parameters

Parameter	Description
schema	The schema that owns *awname*.
awname	The name of an analytic workspace.

Example

The following example shows the tablespace in which the GLOBAL analytic workspace is stored.

```
SQL> set serveroutput on
SQL> execute dbms_output.put_line('Global is stored in tablespace ' ||
     dbms_aw.aw_tablespace('GLOBAL_AW', 'GLOBAL'));
Global is stored in tablespace GLOBAL_DATA

PL/SQL procedure successfully completed.
```

AW_UPDATE Procedure

The AW_UPDATE procedure saves the changes made to an analytic workspace in its permanent database table. For the updated version of this table to be saved in the database, you must issue a SQL COMMIT statement before ending your session.

If you do not specify a workspace to update, AW_UPDATE updates all the user-defined workspaces that are currently attached with read/write access.

Syntax

```
AW_UPDATE (
          awname     IN VARCHAR2 DEFAULT NULL);
AW_UPDATE (
          schema     IN VARCHAR2 DEFAULT NULL,
          awname     IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 24–16 AW_UPDATE Procedure Parameters

Parameter	Description
schema	The schema that owns *awname*.
awname	Saves changes to *awname* by copying them to a table named AW$*awname*. If this parameter is omitted, then changes are saved for all analytic workspaces attached in read/write mode.

Example

The following command saves changes to the GLOBAL analytic workspace to a table named AW$GLOBAL.

```
SQL>execute dbms_aw.aw_update('global');
```

See Also

"Managing Analytic Workspaces" on page 24-1.

CONVERT Procedure

The CONVERT procedure converts an analytic workspace from Oracle9*i* to Oracle 10*g* storage format.

See "Converting an Analytic Workspace to Oracle 10g Storage Format" on page 24-2.

Syntax

```
CONVERT (
          original_aw      IN VARCHAR2,
          converted_aw     IN VARCHAR2,
          tablespace       IN NUMBER DEFAULT);
```

Parameters

Table 24–17 CONVERT Procedure Parameters

Parameter	Description
original_aw	The analytic workspace in 9*i* storage format.
converted_aw	The same analytic workspace in 10*g* storage format.
tablespace	The name of a tablespace in which the converted workspace will be stored. If this parameter is omitted, then the analytic workspace is created in the user's default tablespace.

Example

The following example shows how to convert a 9*i* compatible workspace called GLOBAL_AW to 10*g* storage format. The converted workspace must have the same name as the original workspace, because the fully-qualified names of objects in the workspace include the workspace name.

```
SQL>execute dbms_aw.rename ('global_aw', 'global_aw_temp');
SQL>execute dbms_aw.convert ('global_aw_temp', 'global_aw');
```

EVAL_NUMBER Function

The EVAL_NUMBER function evaluates a numeric expression in an analytic workspace and returns the resulting number.

You can specify the EVAL_NUMBER function in a SELECT from DUAL statement to return a numeric constant defined in an analytic workspace. Refer to the *Oracle Database SQL Reference* for information on selecting from the DUAL table.

Syntax

```
EVAL_NUMBER (
          olap_numeric_expression     IN    VARCHAR2)
     RETURN NUMBER;
```

Parameters

Table 24–18 EVAL_NUMBER Function Parameters

Parameter	Description
olap_numeric_ expression	An OLAP DML expression that evaluates to a number. Refer to the chapter on "Expressions" in the *Oracle OLAP DML Reference*

Example

The following example returns the value of the DECIMALS option in the current analytic workspace. The DECIMALS option controls the number of decimal places that are shown in numeric output. In this example, the value of DECIMALS is 2, which is the default.

```
SQL>set serveroutput on
SQL>select dbms_aw.eval_number('decimals') from dual;

     DBMS_AW.EVAL_NUMBER('DECIMALS')
     -------------------------------
                                   2
     1 row selected.
```

EVAL_TEXT Function

The EVAL_TEXT function evaluates a text expression in an analytic workspace and returns the resulting character string.

You can specify the EVAL_TEXT function in a SELECT from DUAL statement to return a character constant defined in an analytic workspace. Refer to the *Oracle Database SQL Reference* for information on selecting from the DUAL table.

Syntax

```
EVAL_TEXT (
          olap_text_expression     IN    VARCHAR2)
    RETURN VARCHAR2;
```

Parameters

Table 24–19 EVAL_TEXT Function Parameters

Parameter	Description
olap_text_expression	An OLAP DML expression that evaluates to a character string. Refer to the chapter on "Expressions" in the *Oracle OLAP DML Reference*

Example

The following example returns the value of the NLS_LANGUAGE option, which specifies the current language of the OLAP session. The value of NLS_LANGUAGE in this example is "AMERICAN".

```
SQL>set serveroutput on
SQL>select dbms_aw.eval_text('nls_language') from dual;

     DBMS_AW.EVAL_TEXT('NLS_LANGUAGE')
     AMERICAN
     1 row selected.
```

EXECUTE Procedure

The EXECUTE procedure executes one or more OLAP DML commands and directs the output to a printer buffer. It is typically used to manipulate analytic workspace data within an interactive SQL session. In contrast to the RUN Procedure, EXECUTE continues to process commands after it gets an error.

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

```
SET SERVEROUT ON
```

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

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

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

Syntax

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

Parameters

Table 24–20 EXECUTE Procedure Parameters

Parameter	Description
olap-commands	One or more OLAP DML commands separated by semicolons. See "Guidelines for Using Quotation Marks in OLAP DML Commands" on page 24-4.
text	Output from the OLAP engine in response to the OLAP commands.

Example

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

```
SQL> set serveroutput on

SQL> execute dbms_aw.execute('AW ATTACH xademo RW; DEFINE cost_pp FORMULA LAG(analytic_cube_
f.costs, 1, time, LEVELREL time_levelrel)');

     PL/SQL procedure successfully completed.

SQL> execute dbms_aw.execute('DESCRIBE cost_pp');

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

     PL/SQL procedure successfully completed.
```

The next example show how EXECUTE continues to process commands after encountering an error:

```
SQL> execute dbms_aw.execute('call nothing; colwidth=20');
```

```
BEGIN dbms_aw.execute('call nothing; colwidth=20'); END;

*
ERROR at line 1:
ORA-34492: Analytic workspace object NOTHING does not exist.
ORA-06512: at "SYS.DBMS_AW", line 90
ORA-06512: at "SYS.DBMS_AW", line 119
ORA-06512: at line 1


SQL> execute dbms_aw.execute('show colwidth');
20

PL/SQL procedure successfully completed.
```

GETLOG Function

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

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

Syntax

```
GETLOG()
          RETURN CLOB;
```

Returns

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

Example

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

```
SQL>set serverout on size 1000000
SQL>execute dbms_aw.printlog(dbms_aw.interp('AW ATTACH xademo; LISTNAMES AGGMAP'));
     2 AGGMAPs
     ----------------------------------------
     ANALYTIC_CUBE.AGGMAP.1
     SALES_MULTIKEY_CUBE.AGGMAP.1

     PL/SQL procedure successfully completed.


SQL>execute dbms_aw.printlog(dbms_aw.getlog());
     2 AGGMAPs
     ----------------------------------------
     ANALYTIC_CUBE.AGGMAP.1
     SALES_MULTIKEY_CUBE.AGGMAP.1

     PL/SQL procedure successfully completed.
```

INFILE Procedure

The INFILE procedure evaluates the OLAP DML commands in the specified file and executes them in the current analytic workspace.

Syntax

```
INFILE (
```

```
          filename     IN   VARCHAR2);
```

Parameters

Table 24–21 INFILE Procedure Parameters

Parameter	Description
filename	The name of a file containing OLAP DML commands. The file path must be specified in a current directory object for your OLAP session. Use the OLAP DML CDA command to identify or change the current directory object.

Example

The following example executes the OLAP DML commands specified in the file test_setup.tst. The directory path of the file is specified in the OLAP directory object called work_dir.

```
SQL>execute dbms_aw.execute('cda work_dir');
SQL>execute dbms_aw.infile('test_setup.tst');
```

INTERP Function

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

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

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

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

Syntax

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

Parameters

Table 24–22 INTERP Function Parameters

Parameter	Description
olap-commands	One or more OLAP DML commands separated by semi-colons. See "Guidelines for Using Quotation Marks in OLAP DML Commands" on page 24-4.

Returns

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

Example

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

```
SQL>set serverout on size 1000000
SQL> execute dbms_aw.printlog(dbms_aw.interp('AW ATTACH cloned; REPORT product'));
     PRODUCT
     --------------
     L1.TOTALPROD
     L2.ACCDIV
     L2.AUDIODIV
     L2.VIDEODIV
     L3.AUDIOCOMP
     L3.AUDIOTAPE
        .
        .
        .
     PL/SQL procedure successfully completed.
```

INTERPCLOB Function

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

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.

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

Syntax

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

Parameters

Table 24–23 INTERPCLOB Function Parameters

Parameter	Description
olap-commands	One or more OLAP DML commands separated by semi-colons. See "Guidelines for Using Quotation Marks in OLAP DML Commands" on page 24-4.

Returns

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

Example

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

```
SQL> set serverout on size 1000000
SQL> execute dbms_aw.printlog(dbms_aw.interpclob('AW CREATE electronics; IMPORT
ALL FROM EIF FILE ''dbs/electronics.eif'' DATA DFNS; DESCRIBE'));
```

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

INTERP_SILENT Procedure

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

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

Syntax

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

Parameters

Table 24–24 INTERP_SILENT Function Parameters

Parameter	Description
olap-commands	One or more OLAP DML commands separated by semi-colons. See "Guidelines for Using Quotation Marks in OLAP DML Commands" on page 24-4.

Example

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

```
SQL> execute dbms_aw.execute('AW ATTACH xademo');
     IMPORTANT: Analytic workspace XADEMO is read-only. Therefore, you will
     not be able to use the UPDATE command to save changes to it.

     PL/SQL procedure successfully completed.


SQL>execute dbms_aw.interp_silent('AW ATTACH xademo');

     PL/SQL procedure successfully completed.
```

OLAP_ON Function

The OLAP_ON function returns a boolean indicating whether or not the OLAP option is installed in the database.

Syntax

```
OLAP_ON ( )
    RETURN BOOLEAN;
```

Returns

The value of the OLAP parameter in the V$OPTION table.

OLAP_RUNNING Function

The OLAP_RUNNING function returns a boolean indicating whether or not the OLAP option has been initialized in the current session. Initialization occurs when you execute an OLAP DML command (either directly or by using an OLAP PL/SQL or Java package), query an analytic workspace, or execute the STARTUP Procedure.

Syntax

```
OLAP_RUNNING( )
    RETURN BOOLEAN;
```

Returns

TRUE if OLAP has been initialized in the current session, or FALSE if it has not.

Example

The following PL/SQL script tests whether the OLAP environment has been initialized, and starts it if not.

```
BEGIN
 IF DBMS_AW.OLAP_RUNNING() THEN
   DBMS_OUTPUT.PUT_LINE('OLAP is already running');
 ELSE
     DBMS_AW.STARTUP;
     IF DBMS_AW.OLAP_RUNNING() THEN
        DBMS_OUTPUT.PUT_LINE('OLAP started successfully');
    ELSE
       DBMS_OUTPUT.PUT_LINE('OLAP did not start. Is it installed?');
     END IF;
 END IF;
END;
/
```

PRINTLOG Procedure

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

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

```
SET SERVEROUT ON SIZE 1000000
```

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

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

Syntax

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

Parameters

Table 24–25 PRINTLOG Procedure Parameters

Parameter	Description
session-log	The log of a session.

Example

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

```
SQL>set serverout on size 1000000
SQL>execute dbms_aw.printlog(dbms_aw.interp('DESCRIBE analytic_cube_f.profit'));

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

     PL/SQL procedure successfully completed.
```

RUN Procedure

The RUN procedure executes one or more OLAP DML commands and directs the output to a printer buffer. It is typically used to manipulate analytic workspace data within an interactive SQL session. In contrast to the EXECUTE Procedure, RUN stops processing commands when it gets an error.

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

```
SET SERVEROUT ON
```

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

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

Syntax

```
RUN (
          olap_commands     IN      STRING,
          silent            IN      BOOLEAN DEFAULT FALSE);
RUN (
          olap_commands     IN      CLOB,
          silent            IN      BOOLEAN DEFAULT FALSE);
RUN (
          olap_commands     IN      STRING,
          output            OUT     STRING);
RUN (
          olap_commands     IN      STRING,
          output            IN OUT  CLOB);
RUN (
          olap_commands     IN      CLOB,
          output            OUT     STRING);
RUN (
```

```
    olap_commands     IN      CLOB,
    output            IN OUT  CLOB);
```

Parameters

Table 24–26 EXECUTE Procedure Parameters

Parameter	Description
olap-commands	One or more OLAP DML commands separated by semicolons. See "Guidelines for Using Quotation Marks in OLAP DML Commands" on page 24-4.
silent	A boolean value that signals whether the output from the OLAP DML commands should be suppressed. (Error messages from the OLAP engine are never suppressed, regardless of this setting.)
output	Output from the OLAP engine in response to the OLAP commands.

Example

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

```
SQL> set serveroutput on

SQL> execute dbms_aw.run('AW ATTACH xademo RW; DEFINE cost_pp FORMULA LAG(analytic_cube_
f.costs, 1, time, LEVELREL time_levelrel)');

     PL/SQL procedure successfully completed.

SQL> execute dbms_aw.run('DESCRIBE cost_pp');

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

     PL/SQL procedure successfully completed.
```

The next example shows how RUN stops executing commands after encountering an error.

```
SQL> execute dbms_aw.execute('show colwidth');
10

PL/SQL procedure successfully completed.

SQL> execute dbms_aw.run('call nothing; colwidth=20');
BEGIN dbms_aw.run('call nothing; colwidth=20'); END;

*
ERROR at line 1:
ORA-34492: Analytic workspace object NOTHING does not exist.
ORA-06512: at "SYS.DBMS_AW", line 55
ORA-06512: at "SYS.DBMS_AW", line 131
ORA-06512: at line 1

SQL> execute dbms_aw.execute('show colwidth');
10

PL/SQL procedure successfully completed.
```

SHUTDOWN Procedure

The SHUTDOWN procedure terminates the current OLAP session.

By default, the SHUTDOWN procedure terminates the session only if there are no outstanding changes to any of the attached read/write workspaces. If you want to terminate the session without updating the workspaces, specify the force parameter.

Syntax

```
SHUTDOWN (
          force     IN    BOOLEAN  DEFAULT NO);
```

Parameters

Table 24–27 SHUTDOWN Procedure Parameters

Parameter	Description
force	When YES, this parameter forces the OLAP session to shutdown even though one or more attached workspaces has not been updated. Default is NO.

SPARSITY_ADVICE_TABLE Procedure

The SPARSITY_ADVICE_TABLE procedure creates a table for storing the advice generated by the ADVISE_SPARSITY procedure.

Syntax

```
SPARSITY_ADVICE_TABLE (
          tblname   IN    VARCHAR2 DEFAULT);
```

Parameters

Table 24–28 SPARSITY_ADVICE_TABLE Procedure Parameters

Parameter	Description
tblname	The name of the table. The default name is AW_SPARSITY_ADVICE, which is created in your own schema.

Example

The following example creates a table named GLOBAL_SPARSITY_ADVICE.

```
execute dbms_aw.sparsity_advice_table('global_sparsity_advice');
```

See Also

ADVISE_SPARSITY Procedure on page 24-21 for a description of the columns in *tblname*.

"Using the Sparsity Advisor" on page 24-4.

STARTUP Procedure

The STARTUP procedure starts up an OLAP session without attaching any user-defined workspaces.

STARTUP initializes the OLAP processing environment and attaches the read-only EXPRESS workspace, which contains the program code for the OLAP engine.

Syntax

```
STARTUP ( );
```

25

DBMS_AW_XML

The DBMS_AW_XML package builds an analytic workspace based on a logical model described in an XML document. The XML can be created using the Oracle OLAP Analytic Workspace Java API.

This chapter includes the following topics:

Analytic Workspace Java API Overview

The Oracle OLAP Analytic Workspace API is a Java API for building and maintaining standard form analytic workspaces. The API provides classes for describing a logical cube, mapping the cube to a relational data source, and aggregating the cube's data. You can also use the API to specify complex solves, such as allocations and forecasts, and define custom measures and custom dimension members.

The Analytic Workspace API supports two deployment modes: It can be embedded in a Java application, or it can be used to generate XML that serializes the object model for execution by the EXECUTE function. The functionality of the API is identical whether executed from a Java client through JDBC or directly in the database through SQL.

The Analytic Workspace API does not use OLAP Catalog metadata.

Oracle OLAP XML Schema

The EXECUTE and EXECUTEFILE functions process XML that conforms to the Oracle OLAP XML schema defined in awxml.xsd. The XML generated by the Analytic Workspace Java API automatically conforms to awxml.xsd. You can also create your own XML and validate it against the Oracle OLAP XML schema.

Example 25–1 provides an excerpt from an XML document that conforms to the Oracle OLAP XML schema.

> **Tip:** You can obtain AWXML.xsd, as well as the latest version of the *Oracle OLAP Analytic Workspace Java API Reference* (Javadoc), from the Oracle Technology Network Web site:
>
> http://www.otn.oracle.com/products/bi/olap/olap.html

Example 25–1 Oracle OLAP XML Document

```
<AWXML version = '1.0' timestamp = 'Mon Feb 11 13:29:11 2002' >
<AWXML.content>
  <Create  Id="Action3">
    <ActiveObject >
      <AW  Name="GLOBAL_AW.GLOBAL" LongName="GLOBAL_AW.GLOBAL"
          ShortName="GLOBAL_AW.GLOBAL" PluralName="GLOBAL_AW.GLOBAL"
          Id="GLOBAL_AW.GLOBAL.AW" Schema="GLOBAL_AW" MetaDataFormat="10.2"
          DefaultLanguage="AMERICAN" Languages="AMERICAN">
        <Dimension  Name="TIME" LongName="AMERICAN::Time"
            ShortName="AMERICAN::Time" PluralName="AMERICAN::Time"
            Id="TIME.DIMENSION" Schema="GLOBAL_AW" isTime="true"
            isMeasure="false" UseNativeKey="true">
          <Attribute  Name="END_DATE" LongName="AMERICAN::END_DATE"
            ShortName="AMERICAN::END_DATE" PluralName="AMERICAN::END_DATE"
            Id="TIME.END_DATE.ATTRIBUTE" DataType="DATE"
            Classification="END_DATE" InstallAsRelation="false"
            IsDefaultOrder="false"/>
          <Attribute  Name="TIME_SPAN" LongName="AMERICAN::TIME_SPAN"
            ShortName="AMERICAN::TIME_SPAN" PluralName="AMERICAN::TIME_SPAN"
            Id="TIME.TIME_SPAN.ATTRIBUTE" DataType="INTEGER"
            Classification="TIME_SPAN" InstallAsRelation="false"
            IsDefaultOrder="false"/>
          <Attribute  Name="LONG_DESCRIPTION"
            LongName="AMERICAN::Long Description"
            ShortName="AMERICAN::Long Description"
            PluralName="AMERICAN::Long Descriptions"
            Id="TIME.LONG_DESCRIPTION.ATTRIBUTE" DataType="TEXT"
            Classification="MEMBER_LONG_DESCRIPTION" InstallAsRelation="false"
            IsDefaultOrder="false" IsMultiLingual="true"/>
                .
                .
                .
```

Summary of DBMS_AW_XML Subprograms

The following table describes the subprograms provided in `DBMS_AW_EXECUTE`.

Table 25–1 DBMS_AW_XML Subprograms

Subprogram	Description
EXECUTE Function on page 25-4	Creates all or part of a standard form analytic workspace from an XML document stored in a CLOB.
EXECUTEFILE Function on page 25-5	Creates all or part of a standard form analytic workspace from an XML document stored in a text file.

EXECUTE Function

The EXECUTE function builds an analytic workspace using XML that conforms to the Oracle OLAP XML schema. The XML is stored in a database object.

Syntax

```
EXECUTE (
          xml_input     IN     CLOB)
     RETURN VARCHAR2;
```

Parameters

Table 25–2 EXECUTE Function Parameters

Parameter	Description
xml_input	An XML document stored in a CLOB. The XML contains instructions for building or maintaining an analytic workspace. The XML describes a logical model, provides instructions for loading data from relational tables, and defines aggregation and other calculations to be performed on the data in the workspace.

Example

The following SQL program creates a CLOB and loads into it the contents of an XML file. It then creates an analytic workspace named GLOBAL in the GLOBAL_AW schema from the XML document in the CLOB.

```
--Use DBMS_LOB package to create a clob
   DECLARE
      clb CLOB;
      infile BFILE;
      dname varchar2(500);
   BEGIN

   -- Create a temporary clob
      DBMS_LOB.CREATETEMPORARY(clb, TRUE,10);

   -- Create a BFILE use BFILENAME function
   -- Use file GLOBAL.XML in the SCRIPTS directory object.
      infile := BFILENAME('SCRIPTS', 'GLOBAL.XML');

   -- Open the BFILE
      DBMS_LOB.fileopen(infile, dbms_lob.file_readonly);

   -- Load temporary clob from the BFILE
      DBMS_LOB.LOADFROMFILE(clb,infile,DBMS_LOB.LOBMAXSIZE, 1, 1);

   -- Close the BFILE
      DBMS_LOB.fileclose(infile);

   -- Create the GLOBAL analytic workspace
      DBMS_OUTPUT.PUT_LINE(DBMS_AW_XML.execute(clb));
      DBMS_AW.AW_UPDATE;
      COMMIT;

   -- Free the Temporary Clob
      DBMS_LOB.FREETEMPORARY(clb);
   EXCEPTION
```

```
  WHEN OTHERS
   THEN
    DBMS_OUTPUT.PUT_LINE(SQLERRM);
END;
/
```

EXECUTEFILE Function

The EXECUTEFILE function builds an analytic workspace using XML that conforms to the Oracle OLAP XML schema. The XML is stored in a text file.

Syntax

```
EXECUTEFILE (
          dirname    IN     VARCHAR2
          filename   IN     VARCHAR2)
     RETURN VARCHAR2;
```

Returns

The string SUCCESS if successful

Parameters

Table 25–3 EXECUTEFILE Function Parameters

Parameter	Description
dirname	A directory object that identifies the physical directory where *xml_file* is stored.
xml_file	The name of a text file containing an XML document. The XML contains instructions for building or maintaining an analytic workspace. The XML describes a logical model, provides instructions for loading data from relational tables, and defines aggregation and other calculations to be performed on the data in the workspace.

Example

The following EXECUTEFILE function generates a standard form analytic workspace from the XML statements stored in GLOBAL.XML, which is located in a directory identified by the SCRIPTS directory object. The DBMS_OUTPUT.PUT_LINE function displays the "Success" message returned by EXECUTEFILE.

```
SQL> execute dbms_output.put_line(dbms_aw_xml.executefile('SCRIPTS',
'GLOBAL.XML'));
Success
```

26

DBMS_AWM

The Analytic Workspace Manager package, DBMS_AWM, provides procedures for building and maintaining analytic workspaces.

> **Note:** You can access much of the functionality of the DBMS_AWM package through the graphical user interface of the Analytic Workspace Manager.

See Also:

- Chapter 1, "Creating Analytic Workspaces with DBMS_AWM"
- Chapter 2, "Creating OLAP Catalog Metadata with CWM2"

This chapter discusses the following topics:

Parameters of DBMS_AWM Subprograms

The parameters cube_name, dimension_name, measure_name, and level_name refer to the metadata entities in the OLAP Catalog that map to the **relational source cube**.

The parameters aw_cube_name or aw_dimension_name refer to the **target cube** or dimension within an analytic workspace.

Parameters with the suffix _spec refer to the named specifications for loading, aggregating, and optimizing a target cube in an analytic workspace.

> **See Also:** "Overview" on page 1-1 for definitions of the terms, "relational source cube", "multidimensional target cube", and "relational target cube".

DBMS_AWM parameters are summarized in Table 26–1.

Table 26–1 Parameters of DBMS_AWM Procedures

Parameter	Description
cube_owner	Owner of the OLAP Catalog cube associated with the relational source tables (star schema).
cube_name	Name of the OLAP Catalog cube associated with the relational source tables (star schema).
dimension_owner	Owner of the OLAP Catalog dimension associated with the source dimension lookup table.
dimension_name	Name of the OLAP Catalog dimension associated with the source dimension lookup table.
aw_owner	Owner of the analytic workspace. Also the owner of cubes and dimensions within the workspace.
aw_cube_name	Name of the target cube within an analytic workspace. For information on naming requirements, see Table 26–13, " CREATE_AWCUBE Procedure Parameters".
aw_dimension_name	Name of the target dimension within an analytic workspace. For information on naming requirements, see Table 26–18, " CREATE_AWDIMENSION Procedure Parameters".
dimension_load_spec	The name of a specification for loading an OLAP Catalog source dimension into a target dimension in an analytic workspace.
cube_load_spec	The name of a specification for loading an OLAP Catalog source cube into a target cube in an analytic workspace.
aggregation_spec	The name of a specification for creating the stored summaries for a target cube in an analytic workspace.
composite_spec	The name of a specification for defining composites and dimension order for a target cube in an analytic workspace.

Summary of DBMS_AWM Subprograms

Table 26–2 lists the DBMS_AWM subprograms in alphabetical order. Each subprogram is described in detail further in this chapter.

To see the DBMS_AWM subprograms listed by function, refer to "Understanding the DBMS_AWM Procedures" on page 1-5.

Table 26–2 DBMS_AWM Subprograms

Subprogram	Description
ADD_AWCOMP_SPEC_COMP_MEMBER Procedure on page 26-6	Adds a member to a composite in a composite specification.
ADD_AWCOMP_SPEC_MEMBER Procedure on page 26-7	Adds a member to a composite specification.
ADD_AWCUBEAGG_SPEC_LEVEL Procedure on page 26-8	Adds a level to an aggregation specification.
ADD_AWCUBEAGG_SPEC_MEASURE Procedure on page 26-8	Adds a measure to an aggregation specification.
ADD_AWCUBELOAD_SPEC_COMP Procedure on page 26-9	Adds a composite specification to a cube load specification.
ADD_AWCUBELOAD_SPEC_FILTER Procedure on page 26-10	Adds a WHERE clause to a cube load specification.
ADD_AWCUBELOAD_SPEC_MEASURE Procedure on page 26-11	Adds a measure to a cube load specification.
ADD_AWDIMLOAD_SPEC_FILTER Procedure on page 26-12	Adds a WHERE clause to a dimension load specification.
AGGREGATE_AWCUBE Procedure on page 26-13	Creates stored summaries for a cube in an analytic workspace.
CREATE_AWCOMP_SPEC Procedure on page 26-14	Creates a composite specification for a cube.
CREATE_AWCUBE Procedure on page 26-15	Creates containers within an analytic workspace to hold a cube defined in the OLAP Catalog.
CREATE_AWCUBE_ACCESS Procedure on page 26-17	Creates a script to enable relational access to a cube in an analytic workspace.
CREATE_AWCUBE_ACCESS_FULL Procedure on page 26-19	Enables relational access to a cube in an analytic workspace.
CREATE_AWCUBEAGG_SPEC Procedure on page 26-20	Creates an aggregation specification for a cube.
CREATE_AWCUBELOAD_SPEC Procedure on page 26-33	Creates a load specification for a cube.
CREATE_AWDIMENSION Procedure on page 26-22	Creates containers within an analytic workspace to hold a dimension defined in the OLAP Catalog.
CREATE_AWDIMENSION_ACCESS Procedure on page 26-24	Creates a script to enable relational access to a dimension in an analytic workspace.
CREATE_AWDIMENSION_ACCESS_FULL Procedure on page 26-26	Enables relational access to a dimension in an analytic workspace.

Table 26–2 (Cont.) DBMS_AWM Subprograms

Subprogram	Description
UPGRADE_AW_TO_10_2 Procedure on page 26-53	Converts an analytic workspace from 10.1.0.4 to 10.2 format.
CREATE_AWDIMLOAD_SPEC Procedure on page 26-27	Creates a load specification for a dimension.
CREATE_DYNAMIC_AW_ACCESS Procedure on page 26-28	Upgrades standard form metadata to the current release, which supports queries from the OLAP API without the need for relational views.
DELETE_AWCOMP_SPEC Procedure on page 26-28	Deletes a composite specification.
DELETE_AWCOMP_SPEC_MEMBER Procedure on page 26-29	Deletes a member of a composite specification.
DELETE_AWCUBE_ACCESS Procedure on page 26-29	Creates a script that deletes the enablement views and metadata for a cube in an analytic workspace.
DELETE_AWCUBE_ACCESS_ALL Procedure on page 26-30	Deletes the enablement views and metadata for a cube in an analytic workspace.
DELETE_AWCUBEAGG_SPEC Procedure on page 26-31	Deletes an aggregation specification.
DELETE_AWCUBEAGG_SPEC_LEVEL Procedure on page 26-31	Removes a level from an aggregation specification.
DELETE_AWCUBEAGG_SPEC_MEASURE Procedure on page 26-32	Removes a measure from an aggregation specification.
DELETE_AWCUBELOAD_SPEC Procedure on page 26-33	Deletes a cube load specification.
DELETE_AWCUBELOAD_SPEC_COMP Procedure on page 26-33	Removes a composite specification from a cube load specification.
DELETE_AWCUBELOAD_SPEC_FILTER Procedure on page 26-33	Removes a `WHERE` clause from a cube load specification.
DELETE_AWCUBELOAD_SPEC_MEASURE Procedure on page 26-34	Removes a measure from a cube load specification.
DELETE_AWDIMENSION_ACCESS Procedure on page 26-35	Creates a script that deletes the enablement views and metadata for a dimension in an analytic workspace.
DELETE_AWDIMENSION_ACCESS_ALL Procedure on page 26-35	Deletes the enablement views and metadata for a dimension in an analytic workspace.
DELETE_AWDIMLOAD_SPEC Procedure on page 26-36	Deletes a dimension load specification.
DELETE_AWDIMLOAD_SPEC_FILTER Procedure on page 26-37	Removes a `WHERE` clause from a dimension load specification.
REFRESH_AWCUBE Procedure on page 26-37	Loads the data and metadata of an OLAP Catalog source cube into a target cube in an analytic workspace.
REFRESH_AWCUBE_VIEW_NAME Procedure on page 26-39	Creates metadata in the analytic workspace to support user-defined enablement view names for a cube.
REFRESH_AWDIMENSION Procedure on page 26-39	Loads the data and metadata of an OLAP Catalog source dimension into a target dimension in an analytic workspace.

Table 26–2 (Cont.) DBMS_AWM Subprograms

Subprogram	Description
REFRESH_AWDIMENSION_VIEW_NAME Procedure on page 26-41	Creates metadata in the analytic workspace to support user-defined enablement view names for a dimension.
SET_AWCOMP_SPEC_CUBE Procedure on page 26-41	Changes the cube associated with a composite specification.
SET_AWCOMP_SPEC_MEMBER_NAME Procedure on page 26-42	Renames a member of a composite specification.
SET_AWCOMP_SPEC_MEMBER_POS Procedure on page 26-42	Changes the position of a member in a composite specification.
SET_AWCOMP_SPEC_MEMBER_SEG Procedure on page 26-44	Changes the segment size associated with a member of a composite specification.
SET_AWCOMP_SPEC_NAME Procedure on page 26-45	Renames a composite specification.
SET_AWCUBE_VIEW_NAME Procedure on page 26-45	Renames the relational views of an analytic workspace cube.
SET_AWCUBEAGG_SPEC_AGGOP Procedure on page 26-46	Specifies an aggregation operator for aggregating measures along a dimension of a cube.
SET_AWCUBELOAD_SPEC_CUBE Procedure on page 26-47	Changes the cube associated with a cube load specification.
SET_AWCUBELOAD_SPEC_LOADTYPE Procedure on page 26-47	Changes the type of a cube load specification.
SET_AWCUBELOAD_SPEC_NAME Procedure on page 26-48	Renames of a cube load specification.
SET_AWCUBELOAD_SPEC_PARAMETER Procedure on page 26-48	Sets parameters for a cube load specification.
SET_AWDIMENSION_VIEW_NAME Procedure on page 26-49	Renames the relational views of an analytic workspace dimension.
SET_AWDIMLOAD_SPEC_DIMENSION Procedure on page 26-50	Changes the dimension associated with a dimension load specification.
SET_AWDIMLOAD_SPEC_LOADTYPE Procedure on page 26-51	Changes the type of a dimension load specification.
SET_AWDIMLOAD_SPEC_NAME Procedure on page 26-51	Renames a dimension load specification.
SET_AWDIMLOAD_SPEC_PARAMETER Procedure on page 26-52	Sets a parameter for a dimension load specification.

ADD_AWCOMP_SPEC_COMP_MEMBER Procedure

This procedure adds a member to a composite in a composite specification. The member may be a dimension or it may be a nested composite.

Composite members must be added in order. If you want to reorder the members, you must drop and re-create the composite. Call DELETE_AWCOMP_SPEC_MEMBER and ADD_AWCOMP_SPEC_MEMBER.

Syntax

```
ADD_AWCOMP_SPEC_COMP_MEMBER (
               composite_spec         IN   VARCHAR2,
               cube_owner             IN   VARCHAR2,
               cube_name              IN   VARCHAR2,
               composite_name         IN   VARCHAR2,
               nested_member_name     IN   VARCHAR2,
               nested_member_type     IN   VARCHARs,
               dimension_owner        IN   VARCHAR2 DEFAULT NULL,
               dimension_name         IN   VARCHAR2 DEFAULT NULL);
```

Parameters

Table 26–3 ADD_AWCOMP_SPEC_COMP_MEMBER Procedure Parameters

Parameter	Description
composite_spec	Name of a composite specification for a cube.
cube_owner	Owner of the OLAP Catalog source cube.
cube_name	Name of the OLAP Catalog source cube.
composite_name	Name of a composite in the composite specification.
nested_member_name	Name of the member to add to the composite.
nested_member_type	Type of the new member. The type can be either 'DIMENSION' or 'COMPOSITE'.
dimension_owner	Owner of the OLAP Catalog source dimension to add to the composite. If the new member is a nested composite instead of a dimension, this parameter should be NULL (default).
dimension_name	Name of the OLAP Catalog source dimension to add to the composite. If the new member is a nested composite instead of a dimension, this parameter should be NULL (default).

Example

The following statements add a composite COMP1, consisting of the PRODUCT and GEOGRAPHY dimensions, to the composite specification AC_COMPSPEC.

```
execute DBMS_AWM.Create_AWComp_spec
          ('AC_COMPSPEC' ,'XADEMO' ,'ANALYTIC_CUBE');
execute DBMS_AWM.Add_AWComp_Spec_Member
          ('AC_COMPSPEC' ,'XADEMO' ,'ANALYTIC_CUBE' ,'COMP1'  ,'COMPOSITE');
execute DBMS_AWM.Add_AWComp_Spec_Comp_Member
          ('AC_COMPSPEC','XADEMO','ANALYTIC_CUBE', 'COMP1','PROD_COMP',
           'DIMENSION','XADEMO','PRODUCT');
execute DBMS_AWM.Add_AWComp_Spec_Comp_Member
          ('AC_COMPSPEC','XADEMO','ANALYTIC_CUBE', 'COMP1','GEOG_COMP',
           'DIMENSION','XADEMO','GEOGRAPHY');
```

See Also

- "Managing Sparse Data and Optimizing the Workspace Cube" on page 1-12
- DELETE_AWCOMP_SPEC_MEMBER Procedure on page 26-29
- ADD_AWCOMP_SPEC_MEMBER Procedure on page 26-7
- CREATE_AWCOMP_SPEC Procedure on page 26-14

ADD_AWCOMP_SPEC_MEMBER Procedure

This procedure adds a member to a composite specification. The members of a composite specification are composites and dimensions.

Syntax

```
ADD_AWCOMP_SPEC_MEMBER (
               composite_spec      IN   VARCHAR2,
               cube_owner          IN   VARCHAR2,
               cube_name           IN   VARCHAR2,
               member_name         IN   VARCHAR2,
               member_type         IN   VARCHAR2,
               dimension_owner     IN   VARCHAR2 DEFAULT NULL,
               diimension_name     IN   VARCHAR2 DEFAULT NULL);
```

Parameters

Table 26–4 ADD_AWCOMP_SPEC_MEMBER Procedure Parameters

Parameter	Description
composite_spec	Name of a composite specification for a cube.
cube_owner	Owner of the OLAP Catalog source cube.
cube_name	Name of the OLAP Catalog source cube.
member_name	Name of the member of the composite specification.
member_type	Type of the member. The type can be either 'DIMENSION' or 'COMPOSITE'.
dimension_owner	Owner of the OLAP Catalog source dimension to add to the composite specification. If the new member is a composite instead of a dimension, this parameter should be NULL (default).
dimension_name	Name of the OLAP Catalog source dimension to add to the composite specification. If the new member is a composite instead of a dimension, this parameter should be NULL (default).

Example

The following statements add the Time dimension and a composite called COMP1 to the composite specification AC_COMPSPEC.

```
execute DBMS_AWM.Add_AWComp_Spec_Member
          ('AC_COMPSPEC' ,'XADEMO' ,'ANALYTIC_CUBE' ,'TIMECOMP_MEMBER' ,
           'DIMENSION' ,'XADEMO' ,'TIME');
execute DBMS_AWM.Add_AWComp_Spec_Member
          ('AC_COMPSPEC' ,'XADEMO' ,'ANALYTIC_CUBE' ,'COMP1'  ,'COMPOSITE');
```

See Also

- "Managing Sparse Data and Optimizing the Workspace Cube" on page 1-12
- CREATE_AWCOMP_SPEC Procedure on page 26-14

ADD_AWCUBEAGG_SPEC_LEVEL Procedure

This procedure adds a level to an aggregation specification.

Syntax

```
ADD_AWCUBEAGG_SPEC_LEVEL (
               aggregation_spec      IN   VARCHAR2,
               aw_owner              IN   VARCHAR2,
               aw_name               IN   VARCHAR2,
               aw_cube_name          IN   VARCHAR2,
               aw_dimension_name     IN   VARCHAR2,
               aw_level_name         IN   VARCHAR2);
```

Parameters

Table 26–5 ADD_AWCUBEAGG_SPEC_LEVEL Procedure Parameters

Parameter	Description
aggregation_spec	Name of an aggregation specification for a cube in an analytic workspace.
aw_owner	Owner of the analytic workspace.
aw_name	Name of the analytic workspace.
aw_cube_name	Name of the cube within the analytic workspace.
aw_dimension_name	Name of a dimension of the cube.
aw_level_name	Name of a level of the dimension.

Example

The following statements add two levels of Product, one level of Channel, and one level of Time to the aggregation specification AC_AGGSPEC.

```
execute dbms_awm.add_awcubeagg_spec_level
          ('AC_AGGSPEC', 'MYSCHEMA', 'MYAW', 'AW_ANACUBE', 'AW_PROD', 'L3')
execute dbms_awm.add_awcubeagg_spec_level
          ('AC_AGGSPEC', 'MYSCHEMA', 'MYAW', 'AW_ANACUBE', 'AW_PROD', 'L2')
execute dbms_awm.add_awcubeagg_spec_level
          ('AC_AGGSPEC','MYSCHEMA','MYAW','AW_ANACUBE','AW_CHAN','STANDARD_2')
execute dbms_awm.add_awcubeagg_spec_level
          ('AC_AGGSPEC', 'MYSCHEMA', 'MYAW', 'AW_ANACUBE', 'AW_TIME', 'L2')
```

See Also

- "Aggregating the Data in an Analytic Workspace" on page 1-14
- CREATE_AWCUBEAGG_SPEC Procedure on page 26-20

ADD_AWCUBEAGG_SPEC_MEASURE Procedure

This procedure adds a measure to an aggregation specification.

Syntax

```
ADD_AWCUBEAGG_SPEC_MEASURE (
               aggregation_spec     IN   VARCHAR2,
               aw_owner             IN   VARCHAR2,
               aw_name              IN   VARCHAR2,
               aw_cube_name         IN   VARCHAR2,
               aw_measure_name      IN   VARCHAR2);
```

Parameters

Table 26–6 ADD_AWCUBEAGG_SPEC_MEASURE Procedure Parameters

Parameter	Description
aggregation_spec	Name of an aggregation specification for a cube in an analytic workspace.
aw_owner	Owner of the analytic workspace.
aw_name	Name of the analytic workspace.
aw_cube_name	Name of the cube within the analytic workspace.
aw_measure_name	Name of one of the measures of the cube.

Example

The following statements add the Costs and Quota measures to the aggregation specification for the cube AW_ANACUBE in the analytic workspace MYAW.

```
execute dbms_awm.add_awcubeagg_spec_measure
          ('AC_AGGSPEC', 'MYSCHEMA', 'MYAW', 'AW_ANACUBE', 'XXF.COSTS')
execute dbms_awm.add_awcubeagg_spec_measure
          ('AC_AGGSPEC', 'MYSCHEMA', 'MYAW', 'AW_ANACUBE', 'XXF.QUOTA')
```

See Also

- "Aggregating the Data in an Analytic Workspace" on page 1-14
- CREATE_AWCUBEAGG_SPEC Procedure on page 26-20

ADD_AWCUBELOAD_SPEC_COMP Procedure

This procedure adds a composite specification to a cube load specification.

Syntax

```
ADD_AWCUBELOAD_SPEC_COMP (
               cube_load_spec     IN   VARCHAR2,
               cube_owner         IN   VARCHAR2,
               cube_name          IN   VARCHAR2,
               composite_spec     IN   VARCHAR2);
```

Parameters

Table 26–7 ADD_AWCUBELOAD_SPEC_COMP Procedure Parameters

Parameter	Description
cube_load_spec	Name of a cube load specification.
cube_owner	Owner of the OLAP Catalog source cube.

Table 26–7 (Cont.) ADD_AWCUBELOAD_SPEC_COMP Procedure Parameters

Parameter	Description
cube_name	Name of the OLAP Catalog source cube.
composite_spec	Name of the composite specification to add to the cube load specification.

Example

The following statement adds the composite specification AC_COMPSPEC to the cube load specification AC_CUBELOADSPEC.

```
execute DBMS_AWM.add_AWCubeLoad_Spec_Comp
     ('AC_CUBELOADSPEC' ,'XADEMO', 'ANALYTIC_CUBE', 'AC_COMPSPEC');
```

See Also

- "Creating Cubes in the Analytic Workspace" on page 1-4
- CREATE_AWCUBELOAD_SPEC Procedure on page 26-21
- CREATE_AWCOMP_SPEC Procedure on page 26-14

ADD_AWCUBELOAD_SPEC_FILTER Procedure

This procedure adds a filter condition to a cube load specification. The filter is a SQL WHERE clause that will be used in the query against the source fact table.

Syntax

```
ADD_AWCUBELOAD_SPEC_FILTER (
               cube_load_spec       IN   VARCHAR2,
               cube_owner           IN   VARCHAR2,
               cube_name            IN   VARCHAR2,
               fact_table_owner     IN   VARCHAR2,
               fact_table_name      IN   VARCHAR2,
               where_clause         IN   VARCHAR2);
```

Parameters

Table 26–8 ADD_AWCUBELOAD_SPEC_FILTER Procedure Parameters

Parameter	Description
cube_load_spec	Name of a cube load specification.
cube_owner	Owner of the OLAP Catalog source cube.
cube_name	Name of the OLAP Catalog source cube.
fact_table_owner	Owner of the fact table that is mapped to the OLAP Catalog source cube.
fact_table_name	Name of the fact table that is mapped to the OLAP Catalog source cube
where_clause	A SQL WHERE clause that specifies which rows to load from the fact table.

Example

The following statements create a cube load specification called AC_CUBELOADSPEC2. When the target cube in the analytic workspace is refreshed with this specification, only sales figures less than 25 will be loaded.

```
execute dbms_awm.create_awcubeload_spec
          ('AC_CUBELOADSPEC2', 'XADEMO', 'ANALYTIC_CUBE', 'LOAD_DATA');
execute dbms_awm.add_awcubeload_spec_measure
          ('AC_CUBELOADSPEC2', 'XADEMO', 'ANALYTIC_CUBE', 'F.SALES',
           'AW_SALES', 'Sales');
execute dbms_awm.add_awcubeload_spec_filter
          ('AC_CUBELOADSPEC2', 'XADEMO', 'ANALYTIC_CUBE',
           'XADEMO', 'XADEMO_ANALYTIC_FACTS', '''SALES'' < 25');
```

See Also

- "Creating Cubes in the Analytic Workspace" on page 1-4
- CREATE_AWCUBELOAD_SPEC Procedure on page 26-21

ADD_AWCUBELOAD_SPEC_MEASURE Procedure

This procedure adds a measure to a cube load specification.

If you add one or more measures to a cube load specification, only those measures will be loaded. If you do not add measures to the cube load specification, then all the cube's measures will be loaded.

You can use this procedure to specify the target name of the measure, its display name, and its description in the analytic workspace. If you do not specify the target names, or if you do not call this procedure at all, the source names from the OLAP Catalog are used.

Syntax

```
ADD_AWCUBELOAD_SPEC_MEASURE (
               cube_load_spec              IN   VARCHAR2,
               cube_owner                  IN   VARCHAR2,
               cube_name                   IN   VARCHAR2,
               measure_name                IN   VARCHAR2,
               aw_measure_name             IN   VARCHAR2 DEFAULT NULL,
               aw_measure_display_name     IN   VARCHAR2 DEFAULT NULL,
               aw_measure_description      IN   VARCHAR2 DEFAULT NULL);
```

Parameters

Table 26–9 ADD_AWCUBELOAD_SPEC_MEASURE Procedure Parameters

Parameter	Description
cube_load_spec	Name of a cube load specification.
cube_owner	Owner of the OLAP Catalog source cube.
cube_name	Name of the OLAP Catalog source cube.
measure_name	Name of the OLAP Catalog source measure.
aw_measure_name	Name of the target measure in the analytic workspace. If you do not specify a name, the measure name from the OLAP Catalog is used.

Table 26–9 (Cont.) ADD_AWCUBELOAD_SPEC_MEASURE Procedure Parameters

Parameter	Description
aw_measure_display_name	Display name for the target measure in the analytic workspace. If you do not specify a display name, the display name for the measure in the OLAP Catalog is used.
aw_measure_description	Description for the target measure in the analytic workspace. If you do not specify a description, the description for the measure in the OLAP Catalog is used.

Example

The following statements create a cube load specification called AC_CUBELOADSPEC2. When the target cube in the analytic workspace is refreshed with this specification, only the sales measure will be loaded.

The target sales measure will have the logical name AW_SALES, and its description will be 'Sales'.

```
execute dbms_awm.create_awcubeload_spec
          ('AC_CUBELOADSPEC2', 'XADEMO', 'ANALYTIC_CUBE', 'LOAD_DATA');
execute dbms_awm.add_awcubeload_spec_measure
          ('AC_CUBELOADSPEC2', 'XADEMO', 'ANALYTIC_CUBE', 'F.SALES',
           'AW_SALES', 'Sales');
```

See Also

- CREATE_AWCUBELOAD_SPEC Procedure on page 26-21
- REFRESH_AWCUBE Procedure on page 26-37

ADD_AWDIMLOAD_SPEC_FILTER Procedure

This procedure adds a filter condition to a dimension load specification. The filter is a SQL WHERE clause that will be used in the query against the source dimension tables.

Syntax

```
ADD_AWDIMLOAD_SPEC_FILTER (
               dimension_load_spec       IN   VARCHAR2,
               dimension_owner           IN   VARCHAR2,
               dimension_name            IN   VARCHAR2,
               dimension_table_owner     IN   VARCHAR2,
               dimension_table_name      IN   VARCHAR2,
               where_clause              IN   VARCHAR2);
```

Parameters

Table 26–10 ADD_AWDIMLOAD_SPEC_FILTER Procedure Parameters

Parameter	Description
dimension_load_spec	Name of a dimension load specification.
dimension_owner	Owner of the OLAP Catalog source dimension.
dimension_name	Name of the OLAP Catalog source dimension.
dimension_table_owner	Owner of the dimension table that is mapped to the OLAP Catalog source dimension.

Table 26–10 (Cont.) ADD_AWDIMLOAD_SPEC_FILTER Procedure Parameters

Parameter	Description
dimension_table_name	Name of the dimension table that is mapped to the OLAP Catalog source dimension.
where_clause	A SQL WHERE clause that specifies which rows to load from the dimension table into an analytic workspace.

Example

The following statements create a load specification for the CHANNEL dimension in XADEMO. When the target dimension is refreshed with this specification, only the member DIRECT will be loaded.

```
execute dbms_awm.create_awdimload_spec
          ('CHAN_DIMLOADSPEC', 'XADEMO', 'CHANNEL', 'FULL_LOAD');
execute dbms_awm.add_awdimload_spec_filter
          ('CHAN_DIMLOADSPEC', 'XADEMO', 'CHANNEL', 'XADEMO',
          'XADEMO_CHANNEL', '''CHAN_STD_CHANNEL'' = ''DIRECT''' );
```

See Also

- "Creating Dimensions in the Analytic Workspace" on page 1-3
- CREATE_AWDIMLOAD_SPEC Procedure on page 26-27

AGGREGATE_AWCUBE Procedure

This procedure uses an aggregation specification to precompute and store aggregate data for a cube in an analytic workspace.

The REFRESH_AWCUBE procedure loads detail data and sets up the internal workspace structures that support dynamic aggregation. If you want to precompute and store summarized data for the cube, you must use the AGGREGATE_AWCUBE procedure.

You must rerun AGGREGATE_AWCUBE after every refresh to ensure that the stored summaries are consistent with the data.

AGGREGATE_AWCUBE executes an OLAP DML UPDATE command to save the changes in the analytic workspace. AGGREGATE_AWCUBE *does not* execute a SQL COMMIT.

Syntax

```
AGGREGATE_AWCUBE (
               aw_owner             IN   VARCHAR2,
               aw_name              IN   VARCHAR2,
               aw_cube_name         IN   VARCHAR2,
               aggregation_spec     IN   VARCHAR2);
```

Parameters

Table 26–11 AGGREGATE_AWCUBE Procedure Parameters

Parameter	Description
aw_owner	Owner of the analytic workspace.
aw_name	Name of the analytic workspace.
aw_cube_name	Name of the cube within the analytic workspace.
aggregation_spec	Name of an aggregation specification for the cube.

Example

The following statements create an aggregation plan AGG1 for the target cube AC2 in the analytic workspace MYSCHEMA.MYAW. The target cube was created from the source cube XADEMO.ANALYTIC_CUBE.

```
----  Create agg plan for analytic cube ---------------------------------------
----   with levels 2 and 3 of product, standard_2 of channel, and 2 of time ----
----   with measures costs and quota -------------------------------------------

execute dbms_awm.create_awcubeagg_spec
          ('AGG1', 'MYSCHEMA', 'MYAW', 'AC2')
execute dbms_awm.add_awcubeagg_spec_level
          ('AGG1', 'MYSCHEMA', 'MYAW', 'AC2', 'PRODUCT', 'L3')
execute dbms_awm.add_awcubeagg_spec_level
          ('AGG1', 'MYSCHEMA', 'MYAW', 'AC2', 'PRODUCT', 'L2')
execute dbms_awm.add_awcubeagg_spec_level
          ('AGG1', 'MYSCHEMA', 'MYAW', 'AC2', 'CHANNEL', 'STANDARD_2')
execute dbms_awm.add_awcubeagg_spec_level
          ('AGG1', 'MYSCHEMA', 'MYAW', 'AC2', 'TIME', 'L2')
execute dbms_awm.add_awcubeagg_spec_measure
          ('AGG1', 'MYSCHEMA', 'MYAW', 'AC2', 'XXF.COSTS')
execute dbms_awm.add_awcubeagg_spec_measure
          ('AGG1', 'MYSCHEMA', 'MYAW', 'AC2', 'XXF.QUOTA')
execute dbms_awm.aggregate_awcube('MYSCHEMA', 'MYAW', 'AC2', 'AGG1')
```

See Also

- "Aggregating the Data in an Analytic Workspace" on page 1-14
- "CREATE_AWCUBEAGG_SPEC Procedure" on page 26-20

CREATE_AWCOMP_SPEC Procedure

This procedure creates a **composite specification** for an OLAP Catalog source cube. The composite specification determines how sparse data will be stored in the target cube in an analytic workspace. It also determines the dimension order, which affects the efficiency of data loads and queries.

A **composite** is a list of dimension value combinations that provides an index into one or more sparse measures. Composites are named objects within an analytic workspace. Composites are defined and maintained with OLAP DML commands.

Members of a composite specification are composites (whose members are dimensions) and individual dimensions.

Syntax

```
CREATE_AWCOMP_SPEC (
               composite_spec     IN   VARCHAR2,
               cube_owner         IN   VARCHAR2,
               cube_name          IN   VARCHAR2);
```

Parameters

Table 26–12 CREATE_AWCOMP_SPEC Procedure Parameters

Parameter	Description
composite_spec	Name of a composite specification for a cube.
cube_owner	Owner of the OLAP Catalog source cube.
cube_name	Name of the OLAP Catalog source cube.

Note

You can use the following procedures to modify an existing composite specification:

- SET_AWCOMP_SPEC_CUBE Procedure
- SET_AWCOMP_SPEC_MEMBER_NAME Procedure
- SET_AWCOMP_SPEC_MEMBER_POS Procedure
- SET_AWCOMP_SPEC_MEMBER_SEG Procedure
- SET_AWCOMP_SPEC_NAME Procedure

Example

The following statements create a composite specification for the ANALYTIC_CUBE in XADEMO. It consists of the Time dimension followed by a composite called COMP1.

```
execute DBMS_AWM.Create_AWComp_spec
          ('AC_COMPSPEC' ,'XADEMO' ,'ANALYTIC_CUBE');
execute DBMS_AWM.Add_AWComp_Spec_Member
          ('AC_COMPSPEC' ,'XADEMO' ,'ANALYTIC_CUBE' ,'TIMECOMP_MEMBER' ,
           'DIMENSION' ,'XADEMO' ,'TIME');
execute DBMS_AWM.Add_AWComp_Spec_Member
          ('AC_COMPSPEC' ,'XADEMO' ,'ANALYTIC_CUBE' ,'COMP1'  ,'COMPOSITE');
```

See Also

- "Managing Sparse Data and Optimizing the Workspace Cube" on page 1-12
- ADD_AWCOMP_SPEC_MEMBER Procedure on page 26-7
- ADD_AWCOMP_SPEC_COMP_MEMBER Procedure on page 26-6
- ADD_AWCUBELOAD_SPEC_COMP Procedure on page 26-9
- DEFINE COMPOSITE in the *Oracle OLAP DML Reference*

CREATE_AWCUBE Procedure

This procedure creates the multidimensional framework within an analytic workspace to hold a relational cube.

The relational cube, consisting of a star schema and OLAP Catalog metadata, is the source for the target multidimensional cube in the analytic workspace. Data and metadata are loaded from the source cube to the target cube by the REFRESH_AWCUBE procedure.

CREATE_AWCUBE executes an OLAP DML UPDATE command to save the changes in the analytic workspace. CREATE_AWCUBE *does not* execute a SQL COMMIT.

The multidimensional framework for the cube is in database standard form.

Note: Before executing CREATE_AWCUBE to create a new workspace cube, you must execute CREATE_AWDIMENSION for each of the cube's dimensions.

Syntax

```
CREATE_AWCUBE (
               cube_owner       IN   VARCHAR2,
               cube_name        IN   VARCHAR2,
               aw_owner         IN   VARCHAR2,
               aw_name          IN   VARCHAR2,
               aw_cube_name     IN   VARCHAR2   DEFAULT NULL);
```

Parameters

Table 26–13 CREATE_AWCUBE Procedure Parameters

Parameter	Description
cube_owner	Owner of the OLAP Catalog source cube.
cube_name	Name of the OLAP Catalog source cube.
aw_owner	Owner of the analytic workspace.
aw_name	Name of the analytic workspace.
aw_cube_name	Name for the target cube within the analytic workspace. If you specify a name for the cube in the analytic workspace, the name must conform to general object naming conventions for SQL, and it must be unique within the schema that owns the analytic workspace. To test uniqueness, use a statement like the following. `select owner, cube_name from all_olap2_cubes union all select aw_owner, aw_logical_name from all_olap2_aw_cubes;` Within the analytic workspace, you can generally reference the cube by its simple target cube name. However, database standard form also supports a full name for logical objects. For cubes, the full name is: *aw_owner.aw_cube_name*.CUBE

Example

The following statements create the structures for the XADEMO.ANALYTIC_CUBE in the analytic workspace MYSCHEMA.MYAW. The name of the cube in the workspace is AW_ANACUBE.

```
--- Create the dimensions in the analytic workspace ----

execute dbms_awm.create_awdimension
          ('XADEMO','CHANNEL','MYSCHEMA', 'MYAW', 'AW_CHAN');
execute dbms_awm.create_awdimension
          ('XADEMO','GEOGRAPHY','MYSCHEMA','MYAW', 'AW_GEOG');
execute dbms_awm.create_awdimension
          ('XADEMO','PRODUCT','MYSCHEMA', 'MYAW', 'AW_PROD');
execute dbms_awm.create_awdimension
          ('XADEMO','TIME','MYSCHEMA', 'MYAW', 'AW_TIME');

--- Create the cube in the analytic workspace ----
```

```
execute dbms_awm.create_awcube
          ('XADEMO', 'ANALYTIC_CUBE','MYSCHEMA', 'MYAW','AW_ANACUBE');
```

You can use statements like the following to verify that the cube has been created in the analytic workspace.

```
--- View the cube in the analytic workspace ----

execute dbms_aw.execute
          ('aw attach MYSCHEMA.MYAW');
execute dbms_aw.execute
          ('limit name to obj(property''AW$ROLE'') eq ''CUBEDEF''');
execute dbms_aw.execute
          ('report w 40 name');

NAME
----------------------------------------
AW_ANACUBE
```

Alternatively, you can query the Active Catalog to verify that the cube has been created.

```
select * from all_olap2_aw_cubes
           where owner in 'myschema' and
                 aw_name in 'myaw' and
                 aw_logical_name in 'aw_anacube';
```

See Also

- "Creating and Refreshing a Workspace Cube" on page 1-10
- CREATE_AWDIMENSION Procedure on page 26-22
- REFRESH_AWCUBE Procedure on page 26-37
- CREATE_AWCUBE_ACCESS Procedure on page 26-17
- Chapter 3, "Active Catalog Views"

CREATE_AWCUBE_ACCESS Procedure

This procedure generates a script that creates relational fact views of a cube in an analytic workspace. The views are in embedded total format. The script can optionally generate OLAP Catalog metadata that maps to the views of the workspace cube.

Relational views enable applications to query an analytic workspace using standard SQL. Relational views are not used by the OLAP API.

Both dimension views and fact views are required for relational access to the workspace cube. Use the `CREATE_AWDIMENSION_ACCESS` procedure to generate the scripts that create the dimension views.

To accomplish the cube enablement process in a single step, use the `CREATE_AWCUBE_ACCESS_FULL` procedure. This procedure both creates and runs the enablement script.

Syntax

```
CREATE_AWCUBE_ACCESS (
               aw_owner             IN   VARCHAR2,
               aw_name              IN   VARCHAR2,
               aw_cube_name         IN   VARCHAR2,
               access_type          IN   VARCHAR2,
```

```
script_directory     IN   VARCHAR2,
script_name          IN   VARCHAR2,
open_mode            IN   VARCHAR2,
caller               IN   VARCHAR2   DEFAULT NULL,
spreadsheet_mode     IN   VARCHAR2   DEFAULT 'YES',
auto_adt_mode        IN   VARCHAR2   DEFAULT 'NO');
```

Parameters

Table 26–14 CREATE_AWCUBE_ACCESS Procedure Parameters

Parameter	Description
aw_owner	Owner of the analytic workspace.
aw_name	Name of the analytic workspace.
aw_cube_name	Name of the cube in the analytic workspace.
access_type	Controls whether or not the script generates OLAP Catalog metadata for the views. Specify one of the following values: ■ 'SQL' does not generate metadata. ■ 'OLAP' generates metadata
script_directory	The directory that will contain the script. This may be either a directory object or a path set by the UTL_FILE_DIR parameter.
script_name	Name of the script file.
open_mode	One of the following modes for opening the script file: ■ 'W' overwrites any existing contents of the script file ■ 'A' appends the new script to the existing contents of the script file.
caller	This parameter was used in earlier releases to identify the caller of the procedure. It is not used in the current release. By default, this parameter is null. It also accepts the value, 'EXTERNAL'.
spreadsheet_mode	Whether or not to use a MODEL clause in the SELECT FROM OLAP_TABLE statement in the view definition. A SQL MODEL significantly improves the performance of queries that use OLAP_TABLE. By default, a MODEL clause is used. See Chapter 34, "OLAP_TABLE".
auto_adt_mode	Whether or not the abstract data types used by OLAP_TABLE are automatically generated at runtime. By default, the abstract data types are predefined and are *not* automatically generated by OLAP_TABLE. See Chapter 34, "OLAP_TABLE".

Example

The following statement creates an enablement script called aw_anacube_enable.sql in the /dat1/scripts directory. You can run the script to create fact views of the AW_ANACUBE cube in workspace XADEMO.MYAW. The script will also generate an OLAP Catalog cube called AW_ANACUBE that maps to the views.

```
execute dbms_awm.create_awcube_access
         ('XADEMO', 'MYAW', 'AW_ANACUBE', 'OLAP',
          '/dat1/scripts/', 'aw_anacube_enable.sql', 'w');
```

See Also

- "Enabling Relational Access" on page 1-17
- "CREATE_AWCUBE_ACCESS_FULL Procedure" on page 26-19

- "DELETE_AWCUBE_ACCESS Procedure" on page 26-29
- "SET_AWCUBE_VIEW_NAME Procedure" on page 26-45
- "CREATE_AWDIMENSION_ACCESS Procedure" on page 26-24
- "REFRESH_AWCUBE Procedure" on page 26-37
- Chapter 34, "OLAP_TABLE"

CREATE_AWCUBE_ACCESS_FULL Procedure

This procedure accomplishes the entire process of enabling a workspace cube for relational access. Like CREATE_AWCUBE_ACCESS it produces an enablement script. However it does not write the script to a file. Instead it writes the script to temporary memory and runs the script.

The resulting views and metadata are identical to those created by the enablement scripts produced by CREATE_AWCUBE_ACCESS.

Relational views enable applications to query an analytic workspace using standard SQL. Relational views are not used by the OLAP API.

Syntax

```
CREATE_AWCUBE_ACCESS_FULL (
              run_id               IN   NUMBER,
              aw_owner             IN   VARCHAR2,
              aw_name              IN   VARCHAR2,
              aw_cube_name         IN   VARCHAR2,
              access_type          IN   VARCHAR2,
              spreadsheet_mode     IN   VARCHAR2   DEFAULT 'YES',
              auto_adt_mode        IN   VARCHAR2   DEFAULT 'NO');
```

Parameters

Table 26–15 CREATE_AWCUBE_ACCESS_FULL Procedure Parameters

Parameter	Description
run_id	An assigned slot in a global temporary table for holding the record associated with this operation. In most cases, simply specify "1".
aw_owner	Owner of the analytic workspace.
aw_name	Name of the analytic workspace.
aw_cube_name	Name of the cube in the analytic workspace.
access_type	Controls whether or not to generate OLAP Catalog metadata in addition to the enablement views. Specify one of the following values: ■ 'SQL' does not generate metadata ■ 'OLAP' generates metadata
spreadsheet_mode	Whether or not to use a MODEL clause in the SELECT FROM OLAP_TABLE statement in the view definition. A SQL MODEL significantly improves the performance of queries that use OLAP_TABLE. By default, a MODEL clause is used. See Chapter 34, "OLAP_TABLE".
auto_adt_mode	Whether or not the abstract data types used by OLAP_TABLE are automatically generated at runtime. By default, the abstract data types are predefined and are *not* automatically generated by OLAP_TABLE. See Chapter 34, "OLAP_TABLE".

See Also

- "Enabling Relational Access" on page 1-17
- "CREATE_AWCUBE_ACCESS Procedure" on page 26-17
- "REFRESH_AWCUBE Procedure" on page 26-37
- Chapter 34, "OLAP_TABLE"

CREATE_AWCUBEAGG_SPEC Procedure

This procedure creates an **aggregation specification** for an OLAP Catalog cube. The aggregation specification determines the summary data that will be stored with the target cube in the analytic workspace.

The aggregation specification determines which of the cube's levels will be pre-summarized. You can aggregate all of the cube's measures to these levels, or you can choose individual measures. All of the measures are aggregated to the same levels.

Any levels that are not pre-aggregated will be aggregated dynamically as they are queried. Determining which data to preaggregate will involve an evaluation of storage and memory constraints and typical client queries. If you do not provide an aggregation specification, no summaries will be stored and all aggregation will be performed on demand.

An aggregation specification uses the aggregation subsystem of the OLAP DML. This includes the AGGREGATE command, aggregation maps, and related functionality.

Syntax

```
CREATE_AWCUBEAGG_SPEC (
               aggregation_spec     IN   VARCHAR2,
               aw_owner             IN   VARCHAR2,
               aw_name              IN   VARCHAR2,
               aw_cube_name         IN   VARCHAR2);
```

Parameters

Table 26–16 CREATE_AWCUBEAGG_SPEC Procedure Parameters

Parameter	Description
aggregation_spec	Name of an aggregation specification for a cube in an analytic workspace.
aw_owner	Owner of the analytic workspace.
aw_name	Name of the analytic workspace.
aw_cube_name	Name of the cube in the analytic workspace.

Note

You can use the following procedure to modify an existing aggregation specification: SET_AWCUBEAGG_SPEC_AGGOP Procedure

Example

The following statements create an aggregation specification for the target cube AW_ANACUBE in the analytic workspace MYSCHEMA.MYAW. It specifies that the Costs and Sales measures should include stored totals for the third level of PRODUCT, the STANDARD_2 level of CHANNEL, and the second level of TIME.

```
execute dbms_awm.create_awcubeagg_spec
          ('AC_AGGSPEC', 'MYSCHEMA', 'MYAW', 'AW_ANACUBE');
execute dbms_awm.add_awcubeagg_spec_level
          ('AC_AGGSPEC', 'MYSCHEMA', 'MYAW', 'AW_ANACUBE', 'AW_PROD', 'L3');
execute dbms_awm.add_awcubeagg_spec_level
          ('AC_AGGSPEC', 'MYSCHEMA', 'MYAW', 'AW_ANACUBE', 'AW_CHAN',
           'STANDARD_2');
execute dbms_awm.add_awcubeagg_spec_level
          ('AC_AGGSPEC', 'MYSCHEMA', 'MYAW', 'AW_ANACUBE', 'AW_TIME', 'L2');
execute dbms_awm.add_awcubeagg_spec_measure
          ('AC_AGGSPEC', 'MYSCHEMA', 'MYAW', 'AW_ANACUBE', 'XXF.COSTS');
execute dbms_awm.add_awcubeagg_spec_measure
          ('AC_AGGSPEC', 'MYSCHEMA', 'MYAW', 'AW_ANACUBE', 'XXF.SALES');
```

See Also

- "Aggregating the Data in an Analytic Workspace" on page 1-14
- ADD_AWCUBEAGG_SPEC_LEVEL Procedure on page 26-8
- ADD_AWCUBEAGG_SPEC_MEASURE Procedure on page 26-8
- "AGGREGATE_AWCUBE Procedure" on page 26-13
- AGGREGATE Command in the *Oracle OLAP DML Reference*

CREATE_AWCUBELOAD_SPEC Procedure

This procedure creates a **load specification** for an OLAP Catalog cube. The load specification determines how the cube's data will be loaded from the relational fact table into an analytic workspace by the REFRESH_AWCUBE procedure.

A cube load specification defines a load type, which indicates whether the data or only the load instructions should be loaded into the analytic workspace. The load instructions are OLAP DML programs. If you choose to load only the instructions, you can run these programs to perform the data load at a later time.

A separate specification created by CREATE_AWCOMP_SPEC can be associated with a cube load specification. This specification specifies dimension order and determines how sparse data will be stored within the analytic workspace.

Syntax

```
CREATE_AWCUBELOAD_SPEC (
               cube_load_spec     IN   VARCHAR2,
               cube_owner         IN   VARCHAR2,
               cube_name          IN   VARCHAR2,
               load_type          IN   VARCHAR2);
```

Parameters

Table 26–17 CREATE_AWCUBELOAD_SPEC Procedure Parameters

Parameter	Description
cube_load_spec	Name of a cube load specification.
cube_owner	Owner of the OLAP Catalog source cube.

Table 26–17 (Cont.) CREATE_AWCUBELOAD_SPEC Procedure Parameters

Parameter	Description
cube_name	Name of the OLAP Catalog source cube.
load_type	'LOAD_DATA' -- Load the data and metadata for an OLAP Catalog cube into the analytic workspace target cube.
	'LOAD_PROGRAM' -- This argument is no longer used.

Note

You can use the following procedures to modify an existing cube load specification:

- SET_AWCUBELOAD_SPEC_CUBE Procedure
- SET_AWCUBELOAD_SPEC_LOADTYPE Procedure
- SET_AWCUBELOAD_SPEC_NAME Procedure
- SET_AWCUBELOAD_SPEC_PARAMETER Procedure

Example

The following statement creates a cube load specification for the source cube XADEMO.ANALYTIC_CUBE. The load specification is used to refresh the target cube AW_ANACUBE in MYSCHEMA.MYAW.

```
execute dbms_awm.create_awcubeload_spec
          ('AC_CUBELOADSPEC', 'XADEMO', 'ANALYTIC_CUBE', 'LOAD_DATA');
execute dbms_awm.refresh_awcube
          ('MYSCHEMA', 'MYAW', 'AW_ANACUBE', 'AC_CUBELOADSPEC');
```

See Also

- "Creating Cubes in the Analytic Workspace" on page 1-4
- ADD_AWCUBELOAD_SPEC_COMP Procedure on page 26-9
- REFRESH_AWCUBE Procedure on page 26-37

CREATE_AWDIMENSION Procedure

CREATE_AWDIMENSION uses a source dimension in the OLAP Catalog to create the standard form metadata for a target dimension in an analytic workspace. The dimension members and attribute values are loaded by the REFRESH_AWDIMENSION procedure.

CREATE_AWDIMENSION executes an OLAP DML UPDATE command to save the changes in the analytic workspace. CREATE_AWDIMENSION *does not* execute a SQL COMMIT.

Note: Before executing CREATE_AWCUBE to create a new workspace cube, you must execute CREATE_AWDIMENSION for each of the cube's dimensions.

The workspace must already exist before the first call to CREATE_AWDIMENSION.

Syntax

```
CREATE_AWDIMENSION (
               dimension_owner       IN   VARCHAR2,
```

```
dimension_name        IN   VARCHAR2,
aw_owner              IN   VARCHAR2,
aw_name               IN   VARCHAR2,
aw_dimension_name     IN   VARCHAR2   DEFAULT NULL),
```

Parameters

Table 26–18 CREATE_AWDIMENSION Procedure Parameters

Parameter	Description
dimension_owner	Owner of the OLAP Catalog source dimension.
dimension_name	Name of the OLAP Catalog source dimension.
aw_owner	Owner of the analytic workspace.
aw_name	Name of the analytic workspace.

aw_dimension_name	Name for the target dimension within the analytic workspace.
	If you specify a name for the dimension in the analytic workspace, the name must conform to general object naming conventions for SQL, and it must be unique within the schema that owns the analytic workspace. To test uniqueness, use a statement like the following.
	`select owner, dimension_name`
	`  from all_olap2_dimensions`
	`  union all`
	`select aw_owner, aw_logical_name`
	`  from all_olap2_aw_dimensions;`
	Within the analytic workspace, you can generally reference the dimension by its simple target dimension name. However, database standard form also supports a full name for logical objects. For dimensions, the full name is:
	aw_owner.aw_dimension_name.DIMENSION

Example

The following statements create analytic workspace dimensions for CHANNEL, GEOGRAPHY, PRODUCT, TIME, and DIVISION in the workspace MYAW in the XADEMO schema.

```
execute dbms_awm.create_awdimension
          ('XADEMO','CHANNEL','MYSCHEMA', 'MYAW', 'AW_CHAN');
execute dbms_awm.create_awdimension
          ('XADEMO','GEOGRAPHY','MYSCHEMA','MYAW', 'AW_GEOG');
execute dbms_awm.create_awdimension
          ('XADEMO','PRODUCT','MYSCHEMA', 'MYAW', 'AW_PROD');
execute dbms_awm.create_awdimension
          ('XADEMO','TIME','MYSCHEMA', 'MYAW', 'AW_TIME');
execute dbms_awm.create_awdimension
          ('XADEMO','DIVISION','MYSCHEMA', 'MYAW', 'AW_DIV');
```

You can use statements like the following to verify that the dimensions have been created in the analytic workspace.

```
execute dbms_aw.execute
          ('aw attach MYSCHEMA.MYAW');
execute dbms_aw.execute
          ('limit name to obj(property''AW$ROLE'') eq ''DIMDEF''');
execute dbms_aw.execute
          ('report w 40 name');
```

```
NAME
----------------------------------------
AW_CHAN
AW_GEOG
AW_PROD
AW_TIME
AW_DIV
```

Alternatively, you can query the Active Catalog to verify that the dimensions have been created.

```
select * from all_olap2_aw_dimensions
          where aw_owner in 'myschema' and aw_name in 'myaw';
```

See Also

- "Creating and Refreshing a Workspace Dimension" on page 1-9
- REFRESH_AWDIMENSION Procedure on page 26-39
- CREATE_AWDIMENSION_ACCESS Procedure on page 26-24
- CREATE_AWCUBE Procedure on page 26-15
- Chapter 3, "Active Catalog Views"

CREATE_AWDIMENSION_ACCESS Procedure

This procedure generates a script that creates relational views of a dimension in an analytic workspace. The views are in the embedded total format. The script can optionally generate OLAP Catalog metadata that maps to the views of the workspace dimension.

Relational views enable applications to query an analytic workspace using standard SQL. Relational views are not used by the OLAP API.

Both fact views and dimension views are required for relational access to a workspace cube. Use the `CREATE_AWCUBE_ACCESS` procedure to generate the scripts that create the fact views.

To accomplish the dimension enablement process in a single step, use the `CREATE_AWDIMENSION_ACCESS_FULL` procedure. This procedure both creates and runs the enablement script.

Syntax

```
CREATE_AWDIMENSION_ACCESS (
               aw_owner             IN   VARCHAR2,
               aw_name              IN   VARCHAR2,
               aw_dimension_name    IN   VARCHAR2,
               access_type          IN   VARCHAR2,
               script_directory     IN   VARCHAR2,
               script_name          IN   VARCHAR2,
               open_mode            IN   VARCHAR2,
               caller               IN   VARCHAR2   DEFAULT NULL,
               spreadsheet_mode     IN   VARCHAR2   DEFAULT 'YES',
               auto_adt_mode        IN   VARCHAR2   DEFAULT 'NO');
```

Parameters

Table 26–19 CREATE_AWDIMENSION_ACCESS Procedure Parameters

Parameter	Description
aw_owner	Owner of the analytic workspace.
aw_name	Name of the analytic workspace.
aw_dimension_name	Name of the dimension in the analytic workspace.
access_type	Controls whether or not the script generates OLAP Catalog metadata for the views. Specify one of the following values: ■ 'SQL' does not generate metadata. ■ 'OLAP' generates metadata
script_directory	The directory that will contain the script. This may be either a directory object or a path set by the UTL_FILE_DIR parameter.
script_name	Name of the script file.
open_mode	One of the following modes for opening the script file: ■ 'W' overwrites any existing contents of the script file ■ 'A' appends the new script to the existing contents of the script file.
caller	This parameter was used in earlier releases to identify the caller of the procedure. It is not used in the current release. By default, this parameter is null. It also accepts the value, 'EXTERNAL'.
spreadsheet_mode	Whether or not to use a MODEL clause in the SELECT FROM OLAP_TABLE statement in the view definition. A SQL MODEL significantly improves the performance of queries that use OLAP_TABLE. By default, a MODEL clause is used. See Chapter 34, "OLAP_TABLE".
auto_adt_mode	Whether or not the abstract data types used by OLAP_TABLE are automatically generated at runtime. By default, the abstract data types are predefined and are *not* automatically generated by OLAP_TABLE. See Chapter 34, "OLAP_TABLE".

Example

The following statement creates an enablement script called aw_prod_enable in the /dat1/scripts directory. You can run the script to create views of the AW_PROD dimension in workspace XADEMO.MYAW. The script will also generate an OLAP Catalog dimension called AW_PROD that maps to the view.

```
execute dbms_awm.create_awdimension_access
         ('XADEMO', 'MYAW', 'AW_PROD', 'OLAP',
          '/dat1/scripts/', 'aw_prod_enable', 'w');
```

See Also

- "Enabling Relational Access" on page 1-17
- "DELETE_AWDIMENSION_ACCESS Procedure" on page 26-35
- "SET_AWDIMENSION_VIEW_NAME Procedure" on page 26-49
- Chapter 34, "OLAP_TABLE"

CREATE_AWDIMENSION_ACCESS_FULL Procedure

This procedure accomplishes the entire process of enabling a workspace dimension for relational access. Like CREATE_AWDIMENSION_ACCESS it produces an enablement script. However it does not write the script to a file. Instead it writes the script to temporary memory and runs the script.

The resulting views and metadata are identical to those created by the enablement scripts created by CREATE_AWDIMENSION_ACCESS.

Relational views enable applications to query an analytic workspace using standard SQL. Relational views are not used by the OLAP API.

Syntax

```
CREATE_AWDIMENSION_ACCESS_FULL (
               run_id                IN   NUMBER,
               aw_owner              IN   VARCHAR2,
               aw_name               IN   VARCHAR2,
               aw_dimension_name     IN   VARCHAR2,
               access_type           IN   VARCHAR2,
               spreadsheet_mode      IN   VARCHAR2   DEFAULT 'YES',
               auto_adt_mode         IN   VARCHAR2   DEFAULT 'NO');
```

Parameters

Table 26–20 CREATE_AWDIMENSION_ACCESS_FULL Procedure Parameters

Parameter	Description
run_id	An assigned slot in a global temporary table for holding the record associated with this operation. In most cases, simply specify "1".
aw_owner	Owner of the analytic workspace.
aw_name	Name of the analytic workspace.
aw_dimension_name	Name of the dimension in the analytic workspace.
access_type	Controls whether or not to generate OLAP Catalog metadata in addition to the enablement views. Specify one of the following values: ■ 'SQL' does not generate metadata ■ 'OLAP' generates metadata
spreadsheet_mode	Whether or not to use a MODEL clause in the SELECT FROM OLAP_TABLE statement in the view definition. A SQL MODEL significantly improves the performance of queries that use OLAP_TABLE. By default, a MODEL clause is used. See Chapter 34, "OLAP_TABLE".
auto_adt_mode	Whether or not the abstract data types used by OLAP_TABLE are automatically generated at runtime. By default, the abstract data types are predefined and are *not* automatically generated by OLAP_TABLE. See Chapter 34, "OLAP_TABLE".

See Also

- "Enabling Relational Access" on page 1-17
- "CREATE_AWDIMENSION_ACCESS Procedure" on page 26-24
- "REFRESH_AWDIMENSION Procedure" on page 26-39
- Chapter 34, "OLAP_TABLE"

CREATE_AWDIMLOAD_SPEC Procedure

This procedure creates a **load specification** for an OLAP Catalog dimension. The load specification determines how the dimension will be loaded from relational dimension tables into an analytic workspace by the REFRESH_AWDIMENSION procedure.

If you refresh a dimension without a load specification, only new dimension members are loaded.

Syntax

```
CREATE_AWDIMLOAD_SPEC (
               dimension_load_spec     IN   VARCHAR2,
               dimension_owner         IN   VARCHAR2,
               dimension_name          IN   VARCHAR2,
               load_type               IN   VARCHAR2);
```

Parameters

Table 26–21 CREATE_AWDIMLOAD_SPEC Procedure Parameters

Parameter	Description
dimension_load_spec	Name of the load specification. You can use the SET_AWDIMLOAD_SPEC_NAME procedure to alter the name.
dimension_owner	Owner of the OLAP Catalog source dimension.
dimension_name	Name of the OLAP Catalog source dimension.
load_type	Specify one of the following: 'FULL_LOAD_ADDITIONS_ONLY' -- Only new dimension members will be loaded when the dimension is refreshed. (Default) 'FULL_LOAD' -- All dimension members in the workspace will be deleted, then all the members of the source dimension will be loaded.

Note

You can use the following procedures to modify an existing dimension load specification:

- SET_AWDIMLOAD_SPEC_DIMENSION Procedure
- SET_AWDIMLOAD_SPEC_LOADTYPE Procedure
- SET_AWDIMLOAD_SPEC_NAME Procedure
- SET_AWDIMLOAD_SPEC_PARAMETER Procedure

Example

The following statements create a load specification for the XADEMO.CHANNEL source dimension and use it to load the target dimension AW_CHAN in the analytic workspace MYSCHEMA.MYAW. The load specification includes a filter condition (WHERE clause) that causes only the dimension member 'DIRECT' to be loaded.

```
execute dbms_awm.create_awdimload_spec
          ('CHAN_DIMLOADSPEC', 'XADEMO', 'CHANNEL', 'FULL_LOAD');
execute dbms_awm.add_awdimload_spec_filter
          ('CHAN_DIMLOADSPEC', 'XADEMO', 'CHANNEL', 'XADEMO',
          'XADEMO_CHANNEL', '''CHAN_STD_CHANNEL'' = ''DIRECT''' );
execute dbms_awm.refresh_awdimension
```

```
('MYSCHEMA', 'MYAW', 'AW_CHAN', 'CHAN_DIMLOADSPEC');
```

See Also

- "Creating Dimensions in the Analytic Workspace" on page 1-3
- REFRESH_AWDIMENSION Procedure on page 26-39

CREATE_DYNAMIC_AW_ACCESS Procedure

This procedure upgrades standard form metadata created in a previous release of the Oracle Database to the standard form used in the current release. The workspace must already be in 10*g* storage format before the metadata can be upgraded.

Current standard form metadata supports direct queries by the OLAP API without the need for relational views, abstract data types, or OLAP Catalog metadata.

If you do not call CREATE_DYNAMIC_AW_ACCESS, the first DBMS_AWM procedure that you call will attempt to upgrade the metadata.

Syntax

```
CREATE_DYNAMIC_AW_ACCESS (
               aw_owner          IN   VARCHAR2,
               aw_name           IN   VARCHAR2);
```

Parameters

Table 26–22 CREATE_DYNAMIC_AW_ACCESS Procedure Parameters

Parameter	Description
aw_owner	Owner of the analytic workspace.
aw_name	Name of the analytic workspace.

Example

The following statement upgrades the standard form metadata in the GLOBAL_AW analytic workspace in the GLOBAL schema.

```
execute dbms_awm.create_dynamic_aw_access('global', 'global_aw');
```

See Also

- "Converting an Analytic Workspace to Oracle 10g Storage Format" on page 24-2
- "Enabling Access by the OLAP API" on page 1-17

DELETE_AWCOMP_SPEC Procedure

This procedure deletes a composite specification.

Syntax

```
DELETE_AWCOMP_SPEC (
               composite_spec    IN   VARCHAR2,
               cube_owner        IN   VARCHAR2,
               cube_name         IN   VARCHAR2);
```

Parameters

Table 26–23 DELETE_AWCOMP_SPEC Procedure Parameters

Parameter	Description
composite_spec	Name of a composite specification for a cube.
cube_owner	Owner of the OLAP Catalog source cube.
cube_name	Name of the OLAP Catalog source cube.

See Also

CREATE_AWCOMP_SPEC Procedure on page 26-14

DELETE_AWCOMP_SPEC_MEMBER Procedure

This procedure removes a member of a composite specification. The member can be either a dimension or composite.

Syntax

```
DELETE_AWCOMP_SPEC_MEMBER (
               composite_spec     IN   VARCHAR2,
               cube_owner         IN   VARCHAR2,
               cube_name          IN   VARCHAR2,
               member_name        IN   VARCHAR2);
```

Parameters

Table 26–24 DELETE_AWCOMP_SPEC_MEMBER Procedure Parameters

Parameter	Description
composite_spec	Name of a composite specification for a cube.
cube_owner	Owner of the OLAP Catalog source cube.
cube_name	Name of the OLAP Catalog source cube.
member_name	Name of the dimension or composite to delete.

See Also

ADD_AWCOMP_SPEC_MEMBER Procedure on page 26-7

DELETE_AWCUBE_ACCESS Procedure

This procedure generates a script that you can run to drop the views and OLAP Catalog metadata associated with a workspace cube. The script does not delete the enablement metadata that is stored in the analytic workspace.

If you drop the workspace cube or the workspace itself, you should run this procedure to clean up the associated enablement views and metadata.

You do not need to run this procedure if you are creating a new generation of enablement views and metadata. The enablement process itself drops the previous generation before creating the new views and metadata.

Syntax

```
DELETE_AWCUBE_ACCESS (
               aw_owner              IN   VARCHAR2,
```

```
                aw_name              IN   VARCHAR2,
                aw_cube_name         IN   VARCHAR2,
                access_type          IN   VARCHAR2,
                script_directory     IN   VARCHAR2,
                script_name          IN   VARCHAR2,
                open_mode            IN   VARCHAR2);
```

Parameters

Table 26–25 DELETE_AWCUBE_ACCESS Procedure Parameters

Parameter	Description
aw_owner	Owner of the analytic workspace.
aw_name	Name of the analytic workspace.
aw_cube_name	Name of the cube in the analytic workspace.
access_type	Specifies whether or not OLAP Catalog metadata exists for the views: ■ 'SQL' No metadata exists. ■ 'OLAP' OLAP Catalog metadata exists
script_directory	The directory that will contain the script. This may be either a directory object or a path set by the UTL_FILE_DIR parameter.
script_name	Name of the script file.
open_mode	One of the following modes for opening the script file: ■ 'W' overwrites any existing contents of the script file ■ 'A' appends the new script to the existing contents of the script file.

See Also

- "Enabling Relational Access" on page 1-17
- "CREATE_AWCUBE_ACCESS Procedure" on page 26-17
- "CREATE_AWCUBE_ACCESS_FULL Procedure" on page 26-19
- "SET_AWCUBE_VIEW_NAME Procedure" on page 26-45

DELETE_AWCUBE_ACCESS_ALL Procedure

This procedure deletes all the enablement views and metadata for a cube. It writes a script to a temporary location in memory and runs the script.

Syntax

```
DELETE_AWCUBE_ACCESS_ALL (
                run_id           IN   NUMBER,
                aw_owner         IN   VARCHAR2,
                aw_name          IN   VARCHAR2,
                aw_cube_name     IN   VARCHAR2,
                access_type      IN   VARCHAR2);
```

Parameters

Table 26–26 DELETE_AWCUBE_ACCESS_ALL Procedure Parameters

Parameter	Description
run_id	An assigned slot in a global temporary table for holding the record associated with this operation. In most cases, simply specify "1".
aw_owner	Owner of the analytic workspace.
aw_name	Name of the analytic workspace.
aw_cube_name	Name of the cube in the analytic workspace.
access_type	Controls whether or not to generate OLAP Catalog metadata in addition to the enablement views. Specify one of the following values: ■ 'SQL' does not generate metadata ■ 'OLAP' generates metadata

See Also

- "Enabling Relational Access" on page 1-17
- "CREATE_AWCUBE_ACCESS_FULL Procedure" on page 26-19

DELETE_AWCUBEAGG_SPEC Procedure

This procedure deletes an aggregation specification.

Syntax

```
DELETE_AWCUBEAGG_SPEC (
               aggregation_spec     IN   VARCHAR2,
               aw_owner             IN   VARCHAR2,
               aw_name              IN   VARCHAR2,
               aw_cube_name         IN   VARCHAR2);
```

Parameters

Table 26–27 DELETE_AWCUBEAGG_SPEC Procedure Parameters

Parameter	Description
aggregation_spec	Name of an aggregation specification for a cube in an analytic workspace.
aw_owner	Owner of the analytic workspace.
aw_name	Name of the analytic workspace.
aw_cube_name	Name of the cube in the analytic workspace.

See Also

CREATE_AWCUBEAGG_SPEC Procedure on page 26-20

DELETE_AWCUBEAGG_SPEC_LEVEL Procedure

This procedure removes a level from an aggregation specification.

Syntax

```
DELETE_AWCUBEAGG_SPEC_LEVEL (
```

```
                aggregation_spec       IN   VARCHAR2,
                aw_owner               IN   VARCHAR2,
                aw_name                IN   VARCHAR2,
                aw_cube_name           IN   VARCHAR2,
                aw_dimension_name      IN   VARCHAR2,
                aw_level_name          IN   VARCHAR2);
```

Parameters

Table 26–28 DELETE_AWCUBEAGG_SPEC_LEVEL Procedure Parameters

Parameter	Description
aggregation_spec	Name of an aggregation specification for a cube in an analytic workspace.
aw_owner	Owner of the analytic workspace.
aw_name	Name of the analytic workspace.
aw_cube_name	Name of the cube in the analytic workspace.
aw_dimension_name	Name of a dimension of the cube.
aw_level_name	Name of a level of the dimension.

See Also

ADD_AWCUBEAGG_SPEC_LEVEL Procedure on page 26-8

DELETE_AWCUBEAGG_SPEC_MEASURE Procedure

This procedure removes a measure from an aggregation specification.

Syntax

```
DELETE_AWCUBEAGG_SPEC_MEASURE (
                aggregation_spec     IN   VARCHAR2,
                aw_owner             IN   VARCHAR2,
                aw_name              IN   VARCHAR2,
                aw_cube_name         IN   VARCHAR2,
                aw_measure_name      IN   VARCHAR2);
```

Parameters

Table 26–29 DELETE_AWCUBEAGG_SPEC_MEASURE Procedure Parameters

Parameter	Description
aggregation_spec	Name of an aggregation specification for a cube in an analytic workspace.
aw_owner	Owner of the analytic workspace.
aw_name	Name of the analytic workspace.
aw_cube_name	Name of target cube in the analytic workspace.
aw_measure_name	Name of the measure to remove.

See Also

ADD_AWCUBEAGG_SPEC_MEASURE Procedure on page 26-8

DELETE_AWCUBELOAD_SPEC Procedure

This procedure deletes a cube load specification.

Syntax

```
DELETE_AWCUBELOAD_SPEC (
               cube_load_spec     IN   VARCHAR2,
               cube_owner         IN   VARCHAR2,
               cube_name          IN   VARCHAR2);
```

Parameters

Table 26–30 DELETE_AWCUBELOAD_SPEC Procedure Parameters

Parameter	Description
cube_load_spec	Name of a cube load specification.
cube_owner	Owner of the OLAP Catalog source cube.
cube_name	Name of the OLAP Catalog source cube.

See Also

CREATE_AWCUBELOAD_SPEC Procedure on page 26-21

DELETE_AWCUBELOAD_SPEC_COMP Procedure

This procedure removes a composite specification from a cube load specification.

Syntax

```
DELETE_AWCUBELOAD_SPEC_COMP (
               cube_load_spec     IN   VARCHAR2,
               cube_owner         IN   VARCHAR2,
               cube_name          IN   VARCHAR2,
               composite_spec     IN   VARCHAR2);
```

Parameters

Table 26–31 DELETE_AWCUBELOAD_SPEC_COMP Procedure Parameters

Parameter	Description
cube_load_spec	Name of a cube load specification.
cube_owner	Owner of the OLAP Catalog source cube.
cube_name	Name of the OLAP Catalog source cube.
composite_spec	Name of the composite specification to delete.

See Also

ADD_AWCUBELOAD_SPEC_COMP Procedure on page 26-9

DELETE_AWCUBELOAD_SPEC_FILTER Procedure

This procedure removes the filter condition (WHERE clause) from a cube load specification.

Syntax

```
DELETE_AWCUBELOAD_SPEC_FILTER (
               cube_load_spec       IN   VARCHAR2,
               cube_owner           IN   VARCHAR2,
               cube_name            IN   VARCHAR2,
               fact_table_owner     IN   VARCHAR2,
               fact_table_name      IN   VARCHAR2);
```

Parameters

Table 26–32 DELETE_AWCUBELOAD_SPEC_FILTER Procedure Parameters

Parameter	Description
cube_load_spec	Name of a cube load specification.
cube_owner	Owner of the OLAP Catalog source cube.
cube_name	Name of the OLAP Catalog source cube.
fact_table_owner	Owner of the fact table that is mapped to this OLAP Catalog source cube.
fact_table_name	Name of the fact table that is mapped to this OLAP Catalog source cube

See Also

ADD_AWCUBELOAD_SPEC_FILTER Procedure on page 26-10

DELETE_AWCUBELOAD_SPEC_MEASURE Procedure

This procedure removes a measure from a cube load specification.

Syntax

```
DELETE_AWCUBELOAD_SPEC_MEASURE (
               cube_load_spec    IN   VARCHAR2,
               cube_owner        IN   VARCHAR2,
               cube_name         IN   VARCHAR2,
               measure_name      IN   VARCHAR2);
```

Parameters

Table 26–33 DELETE_AWCUBELOAD_SPEC_MEASURE Procedure Parameters

Parameter	Description
cube_load_spec	Name of a cube load specification.
cube_owner	Owner of the OLAP Catalog source cube.
cube_name	Name of the OLAP Catalog source cube.
measure_name	Name of the measure to delete.

See Also

"ADD_AWCUBELOAD_SPEC_MEASURE Procedure" on page 26-11

DELETE_AWDIMENSION_ACCESS Procedure

This procedure generates a script that you can run to drop the views and OLAP Catalog metadata associated with a workspace dimension. The script does not delete the enablement metadata that is stored in the analytic workspace.

If you drop the workspace dimension or the workspace itself, you should run this procedure to clean up the associated enablement views and metadata.

You do not need to run this procedure if you are creating a new generation of enablement views and metadata. The enablement process itself drops the previous generation before creating the new views and metadata.

Syntax

```
DELETE_AWDIMENSION_ACCESS (
                aw_owner              IN   VARCHAR2,
                aw_name               IN   VARCHAR2,
                aw_dimension_name     IN   VARCHAR2,
                access_type           IN   VARCHAR2,
                script_directory      IN   VARCHAR2,
                script_name           IN   VARCHAR2,
                open_mode             IN   VARCHAR2);
```

Parameters

Table 26–34 DELETE_AWDIMENSION_ACCESS Procedure Parameters

Parameter	Description
aw_owner	Analytic workspace owner
aw_name	Analytic workspace name
aw_dimension_name	Analytic workspace dimension name.
access_type	Specifies whether or not OLAP Catalog metadata exists for the views: ■ 'SQL' No metadata exists. ■ 'OLAP' OLAP Catalog metadata exists
script_directory	The directory that will contain the script. This may be either a directory object or a path set by the UTL_FILE_DIR parameter.
script_name	Name of the script file.
open_mode	One of the following modes for opening the script file: ■ 'W' overwrites any existing contents of the script file ■ 'A' appends the new script to the existing contents of the script file.

See Also

- "CREATE_AWDIMENSION_ACCESS Procedure" on page 26-24
- "CREATE_AWCUBE_ACCESS_FULL Procedure" on page 26-19
- "SET_AWDIMENSION_VIEW_NAME Procedure" on page 26-49
- "Enabling Relational Access" on page 1-17

DELETE_AWDIMENSION_ACCESS_ALL Procedure

This procedure deletes all the enablement views and metadata for a dimension. It writes a script to a temporary location in memory and runs the script.

Syntax

```
DELETE_AWDIMENSION_ACCESS_ALL (
              run_id                IN   NUMBER,
              aw_owner              IN   VARCHAR2,
              aw_name               IN   VARCHAR2,
              aw_dimension_name     IN   VARCHAR2,
              access_type           IN   VARCHAR2);
```

Parameters

Table 26–35 DELETE_AWDIMENSION_ACCESS_ALL Procedure Parameters

Parameter	Description
run_id	An assigned slot in a global temporary table for holding the record associated with this operation. In most cases, simply specify "1".
aw_owner	Owner of the analytic workspace.
aw_name	Name of the analytic workspace.
aw_dimension_name	Name of the dimension in the analytic workspace.
access_type	Controls whether or not to generate OLAP Catalog metadata in addition to the enablement views. Specify one of the following values: ■ 'SQL' does not generate metadata ■ 'OLAP' generates metadata

See Also

- "Enabling Relational Access" on page 1-17
- "CREATE_AWDIMENSION_ACCESS_FULL Procedure" on page 26-26

DELETE_AWDIMLOAD_SPEC Procedure

This procedure deletes a dimension load specification.

Syntax

```
DELETE_AWDIMLOAD_SPEC (
              dimension_load_spec     IN   VARCHAR2,
              dimension_owner         IN   VARCHAR2,
              dimension_name          IN   VARCHAR2);
```

Parameters

Table 26–36 DELETE_AWDIMLOAD_SPEC Procedure Parameters

Parameter	Description
dimension_load_spec	Name of a dimension load specification.
dimension_owner	Owner of the OLAP Catalog source dimension.
dimension_name	Name of the OLAP Catalog source dimension.

See Also

CREATE_AWDIMLOAD_SPEC Procedure on page 26-27

DELETE_AWDIMLOAD_SPEC_FILTER Procedure

This procedure removes the filter condition (WHERE clause) from a dimension load specification.

Syntax

```
DELETE_AWDIMLOAD_SPEC_FILTER (
               dimension_load_spec       IN   VARCHAR2,
               dimension_owner           IN   VARCHAR2,
               dimension_name            IN   VARCHAR2,
               dimension_table_owner     IN   VARCHAR2,
               dimension_table_name      IN   VARCHAR2);
```

Parameters

Table 26–37 DELETE_AWDIMLOAD_SPEC_FILTER Procedure Parameters

Parameter	Description
dimension_load_spec	Name of a dimension load specification.
dimension_owner	Owner of the OLAP Catalog source dimension.
dimension_name	Name of the OLAP Catalog source dimension.
dimension_table_owner	Owner of the dimension table that is mapped to the OLAP Catalog source dimension.
dimension_table_name	Name of the dimension table that is mapped to the OLAP Catalog source dimension.

See Also

ADD_AWDIMLOAD_SPEC_FILTER Procedure on page 26-12

REFRESH_AWCUBE Procedure

This procedure loads data and metadata from an OLAP Catalog source cube into a target cube in an analytic workspace.

REFRESH_AWCUBE executes an OLAP DML UPDATE command to save the changes in the analytic workspace. REFRESH_AWCUBE *does not* execute a SQL COMMIT.

You can include a cube load specification to determine how the cube's data will be refreshed. The cube load specification determines whether to load the data or only the load program for execution at a later time. The cube load specification may include a composite specification, which determines dimension order and handling of sparse data.

If you do not include a load specification, all the data is loaded. If you do not include a composite specification, the dimensions are ordered with Time as the fastest-varying followed by a composite of all the other dimensions. The dimensions in the composite are ordered in descending order according to size (number of dimension members).

Unless the load specification for the cube identifies individual measures (ADD_AWCUBELOAD_SPEC_MEASURE), all of the cube's measures are loaded into the workspace. Unless the load specification for the cube includes a filter condition (a WHERE clause on the fact table), all the measures' data is loaded into the workspace.

Before the first call to REFRESH_AWCUBE, you must call REFRESH_AWDIMENSION for each of the cube's dimensions. Before refreshing a cube that already contains data, you must refresh any of its dimensions that have changed since the last refresh.

Syntax

```
REFRESH_AWCUBE (
               aw_owner           IN   VARCHAR2,
               aw_name            IN   VARCHAR2,
               aw_cube_name       IN   VARCHAR2,
               cube_load_spec     IN   VARCHAR2 DEFAULT NULL);
```

Parameters

Table 26–38 REFRESH_AWCUBE Procedure Parameters

Parameter	Description
aw_owner	Owner of the analytic workspace.
aw_name	Name of the analytic workspace.
aw_cube_name	Name of the target cube in the analytic workspace.
cube_load_spec	Name of the cube load specification. If you do not include a load specification, all the fact data is loaded (default).

Note

All the OLAP Catalog metadata that defines the logical cube, including its dimensionality, measures, and descriptions, is refreshed whenever you refresh the workspace cube. The cube's data is refreshed according to the load specification. For more information, see "Creating and Refreshing a Workspace Cube" on page 1-10

For information about the relationship between the refresh and aggregation processes, see "Aggregating the Data in an Analytic Workspace" on page 1-14.

Example

The following statements create the target cube AW_ANACUBE from the source cube XADEMO.ANALYTIC_CUBE. They refresh all of target cube's dimensions, then they create a load specification and refresh the target cube's data.

```
-- create cube, cube load spec, and refresh
execute dbms_awm.create_awcube
          ('XADEMO', 'ANALYTIC_CUBE','MYSCHEMA', 'MYAW','AW_ANACUBE');
execute dbms_awm.create_awcubeload_spec
          ('AC_CUBELOADSPEC', 'XADEMO', 'ANALYTIC_CUBE', 'LOAD_DATA')
execute dbms_awm.refresh_awdimension
          ('MYSCHEMA', 'MYAW', 'AW_CHAN');
execute dbms_awm.refresh_awdimension
          ('MYSCHEMA', 'MYAW', 'AW_PROD');
execute dbms_awm.refresh_awdimension
          ('MYSCHEMA', 'MYAW', 'AW_GEOG');
execute dbms_awm.refresh_awdimension
          ('MYSCHEMA', 'MYAW', 'AW_TIME');
execute dbms_awm.refresh_awcube
          ('MYSCHEMA', 'MYAW', 'AW_ANACUBE', 'AC_CUBELOADSPEC')
```

See Also

- "Creating and Refreshing a Workspace Cube" on page 1-10
- "CREATE_AWCUBE Procedure" on page 26-15
- "REFRESH_AWCUBE Procedure" on page 26-37
- "CREATE_AWCOMP_SPEC Procedure" on page 26-14

- "CREATE_AWCUBE_ACCESS Procedure" on page 26-17

REFRESH_AWCUBE_VIEW_NAME Procedure

This procedure creates metadata in the analytic workspace to support user-defined names for the enablement views of a cube. Call SET_AWCUBE_VIEW_NAME to specify the view names.

Syntax

```
REFRESH_AWCUBE_VIEW_NAME (
                aw_owner              IN   VARCHAR2,
                aw_name               IN   VARCHAR2,
                aw_cube_name          IN   VARCHAR2);
```

Parameters

Table 26–39 REFRESH_AWCUBE_VIEW_NAME Procedure Parameters

Parameter	Description
aw_owner	Analytic workspace owner.
aw_name	Analytic workspace name.
aw_cube_name	Analytic workspace cube name.

Note

For details about enablement view names, see "Specifying Names for Fact Views" on page 1-20.

See Also

- "Enabling Relational Access" on page 1-17
- "SET_AWCUBE_VIEW_NAME Procedure" on page 26-45

REFRESH_AWDIMENSION Procedure

This procedure loads the dimension members and attribute values from an OLAP Catalog source dimension into a target dimension in an analytic workspace.

REFRESH_AWDIMENSION executes an OLAP DML UPDATE command to save the changes in the analytic workspace. REFRESH_AWDIMENSION *does not* execute a SQL COMMIT.

You can include a dimension load specification to determine how the dimension's members will be refreshed in the workspace. If you do not include a load specification, all dimension members are selected for loading, but only new members are actually added to the target dimension.

You can select individual dimension members to load from the source tables by specifying a filter condition (a WHERE clause on the dimension table).

Before the first call to REFRESH_AWCUBE, you must call REFRESH_AWDIMENSION for each of the cube's dimensions. On all subsequent cube refreshes, you only need to call REFRESH_AWDIMENSION if changes have been made to the source dimensions, for example if new time periods have been added to a time dimension.

Syntax

```
REFRESH_AWDIMENSION (
               aw_owner                IN   VARCHAR2,
               aw_name                 IN   VARCHAR2,
               aw_dimension_name       IN   VARCHAR2,
               dimension_load_spec     IN   VARCHAR2 DEFAULT NULL);
```

Parameters

Table 26–40 REFRESH_AWDIMENSION Procedure Parameters

Parameter	Description
aw_owner	Owner of the analytic workspace.
aw_name	Name of the analytic workspace.
aw_dimension_name	Name of the target dimension within the analytic workspace.
dimension_load_spec	Name of a dimension load specification. If you do not include a load specification, new members are appended to the target dimension (default)

Note

All the OLAP Catalog metadata that defines the logical dimension, including its levels, hierarchies, attributes, and descriptions, is refreshed whenever you refresh the workspace dimension. The dimension's data is refreshed according to the load specification. For more information, see "Creating and Refreshing a Workspace Dimension" on page 1-9

Example

The following statements refresh the dimensions of the XADEMO.ANALYTIC_CUBE source cube in the analytic workspace MYSCHEMA.MYAW.

```
-- Create dimension load specs and refresh dimensions

-- CHANNEL dimension
execute dbms_awm.create_awdimload_spec
          ('CHAN_DIMLOADSPEC', 'XADEMO', 'CHANNEL', 'FULL_LOAD');
execute dbms_awm.add_awdimload_spec_filter
          ('CHAN_DIMLOADSPEC', 'XADEMO', 'CHANNEL', 'XADEMO',
          'XADEMO_CHANNEL', '''CHAN_STD_CHANNEL'' = ''DIRECT''' );
execute dbms_awm.refresh_awdimension
          ('MYSCHEMA', 'MYAW', 'AW_CHAN', 'CHAN_DIMLOADSPEC');

-- PRODUCT dimension
execute dbms_awm.create_awdimload_spec
          ('PROD_DIMLOADSPEC', 'XADEMO', 'PRODUCT', 'FULL_LOAD');
execute dbms_awm.Set_AWDimLoad_Spec_Parameter
          ('PROD_DIMLOADSPEC', 'XADEMO', 'PRODUCT', 'UNIQUE_RDBMS_KEY', 'YES');
execute dbms_awm.refresh_awdimension
          ('MYSCHEMA', 'MYAW', 'AW_PROD', 'PROD_DIMLOADSPEC');

-- GEOGRAPHY dimension
execute dbms_awm.create_awdimload_spec
          ('GEOG_DIMLOADSPEC', 'XADEMO', 'GEOGRAPHY', 'FULL_LOAD');
execute dbms_awm.refresh_awdimension
          ('MYSCHEMA', 'MYAW', 'AW_GEOG', 'GEOG_DIMLOADSPEC');
```

```
-- TIME dimension
execute dbms_awm.create_awdimload_spec
          ('TIME_DIMLOADSPEC', 'XADEMO', 'TIME', 'FULL_LOAD');
execute dbms_awm.refresh_awdimension
          ('MYSCHEMA', 'MYAW', 'AW_TIME', 'TIME_DIMLOADSPEC');
```

See Also

- "Creating and Refreshing a Workspace Dimension" on page 1-9
- CREATE_AWDIMENSION Procedure on page 26-22
- "CREATE_AWDIMLOAD_SPEC Procedure" on page 26-27
- "CREATE_AWDIMENSION_ACCESS Procedure" on page 26-24

REFRESH_AWDIMENSION_VIEW_NAME Procedure

This procedure creates metadata in the analytic workspace to support user-defined names for the enablement views of a cube. Call `SET_AWDIMENSION_VIEW_NAME` to specify the view names.

Syntax

```
REFRESH_AWDIMENSION_VIEW_NAME (
               aw_owner                  IN   VARCHAR2,
               aw_name                   IN   VARCHAR2,
               aw_dimension_name         IN   VARCHAR2);
```

Parameters

Table 26–41 REFRESH_AWDIMENSION_VIEW_NAME Procedure Parameters

Parameter	Description
`aw_owner`	Analytic workspace owner.
`aw_name`	Analytic workspace name.
`aw_dimension_name`	Analytic workspace dimension name.

Note

For details about enablement view names, see "Specifying Names for Dimension Views" on page 1-20.

See Also

- "Enabling Relational Access" on page 1-17
- "SET_AWDIMENSION_VIEW_NAME Procedure" on page 26-49

SET_AWCOMP_SPEC_CUBE Procedure

This procedure associates a composite specification with a different cube.

Syntax

```
SET_AWCOMP_SPEC_CUBE (
               composite_spec      IN   VARCHAR2,
               old_cube_owner      IN   VARCHAR2,
               old_cube_name       IN   VARCHAR2,
               new_cube_owner      IN   VARCHAR2,
```

```
                    new_cube_name       IN   VARCHAR2);
```

Parameters

Table 26–42 SET_AWCOMP_SPEC_CUBE Procedure Parameters

Parameter	Description
composite_spec	Name of a composite specification.
old_cube_owner	Owner of the old OLAP Catalog source cube.
old_cube_name	Name of the old OLAP Catalog source cube.
new_cube_owner	Owner of the new OLAP Catalog source cube.
new_cube_name	Name of the new OLAP Catalog source cube.

See Also

- "Managing Sparse Data and Optimizing the Workspace Cube" on page 1-12
- CREATE_AWCOMP_SPEC Procedure on page 26-14

SET_AWCOMP_SPEC_MEMBER_NAME Procedure

This procedure changes the name of a member of a composite specification. The member may be either a dimension or a composite.

Syntax

```
SET_AWCOMP_SPEC_MEMBER_NAME (
               composite_spec      IN   VARCHAR2,
               cube_owner          IN   VARCHAR2,
               cube_name           IN   VARCHAR2,
               old_member_name     IN   VARCHAR2,
               new_member_name     IN   VARCHAR2);
```

Parameters

Table 26–43 SET_AWCOMP_SPEC_MEMBER_NAME Procedure Parameters

Parameter	Description
composite_spec	Name of a composite specification for a cube.
cube_owner	Owner of the OLAP Catalog source cube.
cube_name	Name of the OLAP Catalog source cube.
old_member_name	Old member name. Either a dimension or a composite.
new_member_name	New member name.

See Also

- "Managing Sparse Data and Optimizing the Workspace Cube" on page 1-12
- CREATE_AWCOMP_SPEC Procedure on page 26-14

SET_AWCOMP_SPEC_MEMBER_POS Procedure

This procedure sets the position of a member of a composite specification. The member can be either a dimension or a composite.

Syntax

```
SET_AWCOMP_SPEC_MEMBER_POS (
               composite_spec      IN   VARCHAR2,
               cube_owner          IN   VARCHAR2,
               cube_name           IN   VARCHAR2,
               member_name         IN   VARCHAR2,
               member_position     IN   NUMBER);
```

Parameters

Table 26–44 SET_AWCOMP_SPEC_MEMBER_POS Procedure Parameters

Parameter	Description
composite_spec	Name of a composite specification for a cube.
cube_owner	Owner of the OLAP Catalog source cube.
cube_name	Name of the OLAP Catalog source cube.
member_name	Member of the composite specification. Either a dimension or a composite.
member_position	Position of the member within the composite specification.

Example

The following statements create a composite specification for the ANALYTIC_CUBE in XADEMO. It includes two members: a time dimension called TIMECOMP_MEMBER and a composite called COMP1.

```
---- The logical members of the specification are:
---             <TIME COMP1<PRODUCT, GEOGRAPHY>.
-------------------------------------------------------
execute DBMS_AWM.Create_AWComp_spec
          ('AC_COMPSPEC' ,'XADEMO' ,'ANALYTIC_CUBE');
execute DBMS_AWM.Add_AWComp_Spec_Member
          ('AC_COMPSPEC' ,'XADEMO' ,'ANALYTIC_CUBE' ,'TIMECOMP_MEMBER' ,
           'DIMENSION' ,'XADEMO' ,'TIME');
execute DBMS_AWM.Add_AWComp_Spec_Member
          ('AC_COMPSPEC' ,'XADEMO' ,'ANALYTIC_CUBE' ,'COMP1'  ,'COMPOSITE');
execute DBMS_AWM.Add_AWComp_Spec_Comp_Member
          ('AC_COMPSPEC','XADEMO','ANALYTIC_CUBE', 'COMP1','PROD_COMP',
           'DIMENSION','XADEMO','PRODUCT');
execute DBMS_AWM.Add_AWComp_Spec_Comp_Member
          ('AC_COMPSPEC','XADEMO','ANALYTIC_CUBE', 'COMP1','GEOG_COMP',
           'DIMENSION','XADEMO','GEOGRAPHY');

---- With the following statement, the logical members of the specification
---- are reordered as follows.
---             <COMP1<PRODUCT, GEOGRAPHY> TIME>.
-------------------------------------------------------

execute DBMS_AWM.Set_AWComp_Spec_Member_Pos
          ('AC_COMPSPEC' ,'XADEMO' ,'ANALYTIC_CUBE' ,'COMP1'  ,1);
```

See Also

- "Managing Sparse Data and Optimizing the Workspace Cube" on page 1-12
- CREATE_AWCOMP_SPEC Procedure on page 26-14

SET_AWCOMP_SPEC_MEMBER_SEG Procedure

This procedure sets the segment size for a member of a composite specification. A member is either a dimension or a composite.

A segment is an internal buffer used by the OLAP engine for storing data. The size of segments affects the performance of data loads and queries against the data.

Syntax

```
SET_AWCOMP_SPEC_MEMBER_SEG (
               composite_spec      IN   VARCHAR2,
               cube_owner          IN   VARCHAR2,
               cube_name           IN   VARCHAR2,
               member_name         IN   VARCHAR2,
               member_segwidth     IN   NUMBER DEFAULT NULL);
```

Parameters

Table 26–45 SET_AWCOMP_SPEC_MEMBER_SEG Procedure Parameters

Parameter	Description
composite_spec	Name of a composite specification.
cube_owner	Owner of the OLAP Catalog source cube.
cube_name	Name of the OLAP Catalog source cube.
member_name	Name of the dimension or composite.
member_segwidth	Segment size associated with a dimension or composite. If you do not specify a segment size for a dimension, the value is the maximum size of the dimension (number of dimension members). If you do not specify a segment size for a composite, the value is 10 million.

Example

The following statements set the segment size for the time dimension to zero (the default setting in the analytic workspace) and the segment size for the COMP1 composite to 10,000,000.

```
 execute DBMS_AWM.Create_AWComp_spec
          ('AC_COMPSPEC' ,'XADEMO' ,'ANALYTIC_CUBE');
execute DBMS_AWM.Add_AWComp_Spec_Member
          ('AC_COMPSPEC' ,'XADEMO' ,'ANALYTIC_CUBE' ,'TIME_DIM'  ,
           'DIMENSION' ,'XADEMO' ,'time');
execute DBMS_AWM.Add_AWComp_Spec_Member
          ('AC_COMPSPEC' ,'XADEMO' ,'ANALYTIC_CUBE' ,'COMP1'  ,'COMPOSITE');
execute DBMS_AWM.Add_AWComp_Spec_Comp_Member
          ('AC_COMPSPEC', 'XADEMO', 'ANALYTIC_CUBE', 'COMP1', 'COMP1_PROD',
           'DIMENSION', 'XADEMO', 'product');
execute DBMS_AWM.Add_AWComp_Spec_Comp_Member
          ('AC_COMPSPEC', 'XADEMO', 'ANALYTIC_CUBE', 'COMP1', 'COMP1_GEOG',
           'DIMENSION', 'XADEMO', 'geography');
execute DBMS_AWM.Set_AWComp_Spec_Member_Seg
          ('AC_COMPSPEC' , 'XADEMO', 'ANALYTIC_CUBE', 'TIME_DIM', 0);
execute DBMS_AWM.Set_AWComp_Spec_Member_Seg
          ('AC_COMPSPEC' , 'XADEMO', 'ANALYTIC_CUBE', 'COMP1', NULL);
```

See Also

- "Managing Sparse Data and Optimizing the Workspace Cube" on page 1-12

- In *Oracle OLAP DML Reference*, search for "segment width"
- CREATE_AWCOMP_SPEC Procedure on page 26-14

SET_AWCOMP_SPEC_NAME Procedure

This procedure renames a composite specification.

Syntax

```
SET_AWCOMP_SPEC_NAME (
             old_composite_spec        IN   VARCHAR2,
             cube_owner                IN   VARCHAR2,
             cube_name                 IN   VARCHAR2,
             new_composite_spec        IN   VARCHAR2);
```

Parameters

Table 26–46 SET_AWCOMP_SPEC_NAME Procedure Parameters

Parameter	Description
old_composite_spec	Old name of a composite specification for a cube.
cube_owner	Owner of the OLAP Catalog source cube.
cube_name	Name of the OLAP Catalog source cube.
new_composite_spec	New name of the composite specification.

See Also

- "Managing Sparse Data and Optimizing the Workspace Cube" on page 1-12
- CREATE_AWCOMP_SPEC Procedure on page 26-14

SET_AWCUBE_VIEW_NAME Procedure

This procedure renames the relational views of an analytic workspace cube. The names are stored in the analytic workspace and instantiated when you generate and run new enablement scripts.

Syntax

```
SET_AWCUBE_VIEW_NAME (
               aw_owner                  IN   VARCHAR2,
               aw_name                   IN   VARCHAR2,
               aw_cube_name              IN   VARCHAR2,
               hierarchy_combo_number    IN   NUMBER,
               view_name                 IN   VARCHAR2);
```

Parameters

Table 26–47 SET_AWCUBE_VIEW_NAME Procedure Parameters

Parameter	Description
aw_owner	Analytic workspace owner.
aw_name	Analytic workspace name.
aw_cube_name	Analytic workspace cube name.

Table 26–47 (Cont.) SET_AWCUBE_VIEW_NAME Procedure Parameters

Parameter	Description
hierarchy_combo_number	Number of the hierarchy combination.
view_name	Name for the fact view for this hierarchy combination.

Note

For details about enablement view names, see "Specifying Names for Fact Views" on page 1-20.

See Also

- "Enabling Relational Access" on page 1-17
- "CREATE_AWCUBE_ACCESS Procedure" on page 26-17
- "DELETE_AWCUBE_ACCESS Procedure" on page 26-29
- "REFRESH_AWCUBE_VIEW_NAME Procedure" on page 26-39

SET_AWCUBEAGG_SPEC_AGGOP Procedure

This procedure sets the operator for aggregation along one of the dimensions in an aggregation specification.

You can specify any aggregation operator that can be used with the OLAP DML RELATION command. The default operator is addition (SUM). You can use this procedure to override the aggregation operator associated with the source cube in the OLAP Catalog.

> **Note:** The DBMS_AWM package currently does not support weighted aggregation operators. For example, if the OLAP Catalog specifies a weighted sum or weighted average for aggregation along one of the cube's dimensions, it is converted to the scalar equivalent (sum or average) in the analytic workspace. Weighted operators specified by SET_AWCUBEAGG_SPEC_AGGOP are similarly converted.

Syntax

```
SET_AWCUBEAGG_SPEC_AGGOP (
                aggregation_spec         IN   VARCHAR2,
                aw_owner                 IN   VARCHAR2,
                aw_name                  IN   VARCHAR2,
                aw_cube_name             IN   VARCHAR2,
                aw_measure_name          IN   VARCHAR2,
                aw_dimension_name        IN   VARCHAR2,
                aggregation_operator     IN   VARCHAR2);
```

Parameters

Table 26–48 SET_AWCUBEAGG_SPEC_AGGOP Procedure Parameters

Parameter	Description
aggregation_spec	Name of the aggregation specification in the analytic workspace.
aw_owner	Owner of the analytic workspace.
aw_name	Name of the analytic workspace.

Table 26–48 (Cont.) SET_AWCUBEAGG_SPEC_AGGOP Procedure Parameters

Parameter	Description
aw_cube_name	Name of the target cube in the analytic workspace.
aw_measure_name	Name of a measure to aggregate.
aw_dimension_name	Name of a dimension of the cube.
aggregation_operator	Aggregation operator for aggregation along this dimension. See Table 1–10, " Aggregation Operators".

Note

See "Aggregating the Data in an Analytic Workspace" on page 1-14 for details on aggregation methods supported in the OLAP Catalog and in the analytic workspace.

See Also

- "Aggregating the Data in an Analytic Workspace" on page 1-14
- CREATE_AWCUBEAGG_SPEC Procedure on page 26-20
- RELATION command entry in Oracle9i OLAP DML Reference help
- Chapter on Aggregation in *Oracle OLAP DML Reference*

SET_AWCUBELOAD_SPEC_CUBE Procedure

This procedure associates a cube load specification with a different cube.

Syntax

```
SET_AWCUBELOAD_SPEC_CUBE (
             cube_load_spec     IN   VARCHAR2,
             old_cube_owner     IN   VARCHAR2,
             old_cube_name      IN   VARCHAR2,
             new_cube_owner     IN   VARCHAR2,
             new_cube_name      IN   VARCHAR2);
```

Parameters

Table 26–49 SET_AWCUBELOAD_SPEC_CUBE Procedure Parameters

Parameter	Description
cube_load_spec	Name of a cube load specification.
old_cube_owner	Owner of the old OLAP Catalog source cube.
old_cube_name	Name of the old OLAP Catalog source cube.
new_cube_owner	Owner of the new OLAP Catalog source cube.
new_cube_name	Name of the new OLAP Catalog source cube.

See Also

CREATE_AWCUBELOAD_SPEC Procedure on page 26-21

SET_AWCUBELOAD_SPEC_LOADTYPE Procedure

This procedure resets the load type for a cube load specification. The load type indicates how data will be loaded into the analytic workspace.

Syntax

```
SET_AWCUBELOAD_SPEC_LOADTYPE (
               cube_load_spec     IN   VARCHAR2,
               cube_owner         IN   VARCHAR2,
               cube_name          IN   VARCHAR2,
               load_type          IN   VARCHAR2);
```

Parameters

Table 26–50 SET_AWCUBELOAD_SPEC_LOADTYPE Procedure Parameters

Parameter	Description
cube_load_spec	Name of a load specification for a cube.
cube_owner	Owner of the OLAP Catalog source cube.
cube_name	Name of the OLAP Catalog source cube.
load_type	'LOAD_DATA' -- Load the data and metadata for an OLAP Catalog cube into the analytic workspace target cube. 'LOAD_PROGRAM' -- This argument is no longer used.

See Also

CREATE_AWCUBELOAD_SPEC Procedure on page 26-21

SET_AWCUBELOAD_SPEC_NAME Procedure

This procedure renames a cube load specification.

Syntax

```
SET_AWCUBELOAD_SPEC_NAME (
               old_cube_load_spec     IN   VARCHAR2,
               cube_owner             IN   VARCHAR2,
               cube_name              IN   VARCHAR2,
               new_cube_load_spec     IN   VARCHAR2);
```

Parameters

Table 26–51 SET_AWCUBELOAD_SPEC_NAME Procedure Parameters

Parameter	Description
old_cube_load_spec	Old name of a cube load specification.
cube_owner	Owner of the OLAP Catalog source cube.
cube_name	Name of the OLAP Catalog source cube.
new_cube_load_spec	New name of the cube load specification.

See Also

CREATE_AWCUBELOAD_SPEC Procedure on page 26-21

SET_AWCUBELOAD_SPEC_PARAMETER Procedure

This procedure sets parameters for a cube load specification.

Syntax

```
SET_AWCUBELOAD_SPEC_PARAMETER (
               cube_load_spec     IN   VARCHAR2,
               cube_owner         IN   VARCHAR2,
               cube_name          IN   VARCHAR2,
               parameter_name     IN   VARCHAR2,
               parameter_value    IN   VARCHAR2 DEFAULT NULL);
```

Parameters

Table 26–52 SET_AWCUBELOAD_SPEC_PARAMETER Procedure Parameters

Parameter	Description
cube_load_spec	Name of a cube load specification.
cube_owner	Owner of the OLAP Catalog source cube.
cube_name	Name of the OLAP Catalog source cube.
parameter_name	'DISPLAY_NAME' -- Whether to use the OLAP Catalog source cube name or the target cube display name as the display name for the target cube in the analytic workspace.
parameter_value	Value of DISPLAY_NAME is the display name for the target cube in the analytic workspace. If you do not specify this parameter, the display name for the source cube in the OLAP Catalog will be used as the display name for the target cube in the analytic workspace.

Example

The following statement specifies a target cube display name for the AC_CUBELOADSPEC cube load specification.

```
execute dbms_awm.set_awcubeload_spec_parameter
          ('AC_CUBELOADSPEC', 'XADEMO', 'ANALYTIC_CUBE',
           'DISPLAY_NAME', 'My AW Analytic Cube Display Name')
```

See Also

CREATE_AWCUBELOAD_SPEC Procedure on page 26-21

SET_AWDIMENSION_VIEW_NAME Procedure

This procedure renames the relational views of an analytic workspace dimension. The names are stored in the analytic workspace and instantiated when you generate and run new enablement scripts.

Syntax

```
SET_AWDIMENSION_VIEW_NAME (
               aw_owner              IN   VARCHAR2,
               aw_name               IN   VARCHAR2,
               aw_dimension_name     IN   VARCHAR2,
               hierarchy_name        IN   VARCHAR2,
               view_name             IN   VARCHAR2);
```

Parameters

Table 26–53 SET_AWDIMENSION_VIEW_NAME Procedure Parameters

Parameter	Description
aw_owner	Analytic workspace owner
aw_name	Analytic workspace name
aw_dimension_name	Analytic workspace dimension name
hierarchy_name	Analytic workspace hierarchy name
view_name	Name for the view of the dimension hierarchy.

Note

For details about enablement view names, see "Specifying Names for Dimension Views" on page 1-20.

See Also

- "Enabling Relational Access" on page 1-17
- "CREATE_AWDIMENSION_ACCESS Procedure" on page 26-24
- "DELETE_AWDIMENSION_ACCESS Procedure" on page 26-35
- "REFRESH_AWDIMENSION_VIEW_NAME Procedure" on page 26-41

SET_AWDIMLOAD_SPEC_DIMENSION Procedure

This procedure associates a dimension load specification with a different dimension.

Syntax

```
SET_AWDIMLOAD_SPEC_DIMENSION (
                dimension_load_spec     IN   VARCHAR2,
                old_dimension_owner     IN   VARCHAR2,
                old_dimension_name      IN   VARCHAR2,
                new_dimension_owner     IN   VARCHAR2,
                new_dimension_name      IN   VARCHAR2);
```

Parameters

Table 26–54 SET_AWDIMLOAD_SPEC_DIMENSION Procedure Parameters

Parameter	Description
dimension_load_spec	Name of a dimension load specification.
old_dimension_owner	Owner of the old OLAP Catalog source dimension.
old_dimension_name	Name of the old OLAP Catalog source dimension.
new_dimension_owner	Owner of the new OLAP Catalog source dimension.
new_dimension_name	Name of the new OLAP Catalog source dimension.

See Also

CREATE_AWDIMLOAD_SPEC Procedure on page 26-27

SET_AWDIMLOAD_SPEC_LOADTYPE Procedure

This procedure resets the load type for a dimension load specification. The load type indicates how dimension members will be loaded into the analytic workspace.

By default only new members are loaded when the dimension is refreshed.

Syntax

```
SET_AWDIMLOAD_SPEC_LOADTYPE (
               dimension_load_spec     IN   VARCHAR2,
               dimension_owner         IN   VARCHAR2,
               dimension_name          IN   VARCHAR2,
               load_type               IN   VARCHAR2);
```

Parameters

Table 26–55 SET_AWDIMLOAD_SPEC_LOADTYPE Procedure Parameters

Parameter	Description
dimension_load_spec	Name of a dimension load specification.
dimension_owner	Owner of the OLAP Catalog source dimension.
dimension_name	Name of the OLAP Catalog source dimension.
load_type	Specify one of the following: 'FULL_LOAD_ADDITIONS_ONLY' -- Only new dimension members will be loaded when the dimension is refreshed. (Default) 'FULL_LOAD' -- When the dimension is refreshed, all dimension members in the workspace will be deleted, then all the members of the source dimension will be loaded.

See Also

CREATE_AWDIMLOAD_SPEC Procedure on page 26-27

SET_AWDIMLOAD_SPEC_NAME Procedure

This procedure renames a dimension load specification.

Syntax

```
SET_AWDIMLOAD_SPEC_NAME (
               old_dimension_load_spec     IN   VARCHAR2,
               dimension_owner             IN   VARCHAR2,
               dimension_name              IN   VARCHAR2,
               new_dimension_load_spec     IN   VARCHAR2);
```

Parameters

Table 26–56 SET_AWDIMLOAD_SPEC_NAME Procedure Parameters

Parameter	Description
old_dimension_load_spec	Old name of the dimension load specification.
dimension_owner	Owner of the OLAP Catalog source dimension.

Table 26–56 (Cont.) SET_AWDIMLOAD_SPEC_NAME Procedure Parameters

Parameter	Description
dimension_name	Name of the OLAP Catalog source dimension.
new_dimension_load_spec	New name for the dimension load specification.

See Also

CREATE_AWDIMLOAD_SPEC Procedure on page 26-27

SET_AWDIMLOAD_SPEC_PARAMETER Procedure

This procedure sets parameters for a dimension load specification.

Syntax

```
SET_AWDIMLOAD_SPEC_PARAMETER (
               dimension_load_spec     IN   VARCHAR2,
               dimension_owner         IN   VARCHAR2,
               dimension_name          IN   VARCHAR2,
               parameter_name          IN   VARCHAR2,
               parameter_value         IN   VARCHAR2 DEFAULT NULL);
```

Parameters

Table 26–57 SET_AWDIMLOAD_SPEC_PARAMETER Procedure Parameters

Parameter	Description
dimension_load_spec	Name of a dimension load specification.
dimension_owner	Owner of the OLAP Catalog source dimension.
dimension_name	Name of the OLAP Catalog source dimension.

Table 26–57 (Cont.) SET_AWDIMLOAD_SPEC_PARAMETER Procedure Parameters

Parameter	Description
parameter_name	One of the following: 'UNIQUE_RDBMS_KEY' -- Whether or not the members of this dimension are unique across all levels in the source tables. 'DISPLAY_NAME' -- Display name for the target dimension in the analytic workspace. 'P_DISPLAY_NAME' -- Plural display name for the target dimension in the analytic workspace.
parameter_value	Values of UNIQUE_RDBMS_KEY can be either 'YES' or 'NO'. The default is 'NO'. NO -- Dimension member names are not unique across levels in the RDBMS tables. The corresponding dimension member names in the analytic workspace include the level name as a prefix. (Default) YES -- Dimension member names are unique across levels in the RDBMS tables. The corresponding dimension member names in the analytic workspace have the same names as in the source relational dimension. Value of DISPLAY_NAME is the display name for the target dimension in the analytic workspace. If you do not specify this parameter, the display name for the source dimension in the OLAP Catalog will be used as the display name for the target dimension in the analytic workspace. Value of P_DISPLAY_NAME is the plural display name for the target dimension in the analytic workspace. If you do not specify this parameter, the plural display name for the source dimension in the OLAP Catalog will be used as the plural display name for the target dimension in the analytic workspace.

Example

The following statements set parameters for the product dimension in the load specification PROD_LOADSPEC. These parameters prevent level prefixes on dimension member names, and they specify a display name and plural display name for the target dimension.

```
execute dbms_awm.Set_AWDimLoad_Spec_Parameter
          ('PROD_LOADSPEC', 'XADEMO', 'PRODUCT', 'UNIQUE_RDBMS_KEY', 'YES')
execute dbms_awm.Set_AWDimLoad_Spec_Parameter
          ('PROD_LOADSPEC', 'XADEMO', 'PRODUCT', 'DISPLAY_NAME',
           'My AW Product Display Name')
execute dbms_awm.Set_AWDimLoad_Spec_Parameter
          ('PROD_LOADSPEC', 'XADEMO', 'PRODUCT', 'P_DISPLAY_NAME',
           'My AW Product Plural Display Name')
```

See Also

CREATE_AWDIMLOAD_SPEC Procedure on page 26-27

UPGRADE_AW_TO_10_2 Procedure

This procedure upgrades an analytic workspace from 10.1.0.4 to 10.2. It first converts the database format, if necessary, then converts the standard form metadata.

Syntax

```
UPGRADE_AW_TO_10_2 (
               aw_owner     IN   VARCHAR2,
```

```
                    aw_name                 IN   VARCHAR2;
```

Parameters

Table 26–58 UPGRADE_AW_TO_10_2 Parameters

Parameter	Description
aw_owner	Owner of the analytic workspace.
aw_name	Name of the analytic workspace.

Example

The following SQL command upgrades an analytic workspace named GLOBAL, owned by GLOBAL_AW, from 10.1.0.2 to 10.2.

```
execute dbms_awm.upgrade_aw_to_10_2('global_aw', 'global');
```

27

DBMS_ODM

The OLAP Data Management package, DBMS_ODM, provides procedures for creating materialized views specific to the requirements of the OLAP API.

> **See Also:** *Oracle OLAP Application Developer's Guide* for information on summary management for Oracle OLAP

This chapter includes the following topics:

Materialized Views for the OLAP API

Summary management for relational warehouses is managed by the query rewrite facility in the Oracle Database. Query rewrite enables a query to fetch aggregate data from materialized views rather than recomputing the aggregates at runtime.

When the OLAP API queries a warehouse stored in relational tables, it uses query rewrite whenever possible. However, the OLAP API can only use query rewrite when the materialized views have a specific format. The procedures in the DBMS_ODM package create materialized views that satisfy the requirements of the OLAP API.

When the source data is stored in an analytic workspace, materialized views are not used. The native multidimensional structures within analytic workspaces support both stored summarization and run-time aggregation. You can use the DBMS_AWM package, Analytic Workspace Manager, or the OLAP Analytic Workspace Java API to move your data from a star schema to an analytic workspace.

Materialized Views Created by DBMS_OMDM

The DBMS_ODM package creates a set of materialized views based on a cube defined in the OLAP Catalog. The cube must be mapped to a star schema with a single fact table containing only lowest level data.

Scripts generated by DBMS_ODM procedures create the following materialized views:

- A dimension materialized view for each hierarchy of each of the cube's dimensions
- A single fact materialized view, created with GROUP BY GROUPING SETS syntax, for the cube's measures

Generating the Grouping Sets

A grouping set identifies a unique combination of levels. With grouping sets, you can summarize your data symmetrically, for example sales at the month level across all levels of geography, or you can summarize it asymmetrically, for example sales at the month level for cities and at the quarter level for states.

The DBMS_ODM package provides two ways of calculating the grouping sets included in the fact materialized view. You can execute a single procedure that automatically calculates the grouping sets. Or you can manually choose the grouping sets.

Automatically Calculate the Grouping Sets

To automatically calculate the grouping sets, execute the CREATESTDFACTMV procedure with one of the following options:

- Fully materialize the cube. Include all level combinations in the materialized view.
- Partially materialize the cube. Include a subset of the level combinations in the materialized view.
- Materialize the cube using a percentage of the cube's level combinations.

The first two options summarize the data symmetrically. The third option typically produces asymmetrical summarization.

Manually Calculate the Grouping Sets

To manually calculate the grouping sets:

1. Execute the CREATEDIMLEVTUPLE procedure to list all the levels in the cube. Choose the levels to include in the grouping sets.
2. Execute the CREATECUBELEVELTUPLE procedure to create a table containing all the combinations of the levels you chose in the previous step. Edit the table to choose the level combinations (grouping sets) to include in the fact materialized view.

When you manually choose the grouping sets, you can specify either symmetrical or asymmetrical summarization.

Aggregation Operators

Addition is the default aggregation method used in the materialized views. If you want to use a different aggregation method, you must specify it in the OLAP Catalog metadata for each of the cube's dimensions. The same aggregation method must be specified for each dimension, otherwise DBMS_ODM uses addition.

You can use Enterprise Manager or the CWM2_OLAP_CUBE package to specify the aggregation method. See "SET_AGGREGATION_OPERATOR Procedure" on page 9-6.

Example: Automatically Generate the Minimum Grouping Sets for a Cube

This example shows how to automatically generate a minimum set of materialized views for the cube UNITS_CUBE in the GLOBAL schema. This cube has dimensions for Channel, Customer, Product, and Time. The Customer dimension has two hierarchies, which share the same lowest level.

The dimensions of the UNITS_CUBE are described in Table 27–1. The levels in each hierarchy are listed from lowest (the "leaf" level) to highest (the most aggregate). The position of a level in a hierarchy determines whether it is among the minimum

grouping sets. For the rules for creating the minimum grouping sets, refer to "MINIMUM Grouping Sets" on page 27-12.

Table 27–1 Dimensions of GLOBAL.UNITS_CUBE

Dimension	Hierarchy	Levels
CHANNEL	CHANNEL_ROLLUP	Channel
		All_Channels
CUSTOMER	MARKET_ROLLUP	Ship_to
		Account
		Market_Segment
		Total_Market
	SHIPMENTS_ROLLUP	Ship_to
		Warehouse
		Region
		All_Customers
PRODUCT	PRODUCT_ROLLUP	Item
		Family
		Class
		Total_Product
TIME	CALENDAR	Month
		Quarter
		Year

To generate the materialized views:

1. Identify a scripts directory. The directory can be specified in the UTL_FILE_DIR initialization parameter, or you can define a directory object with statements like the following.

```
CREATE OR REPLACE DIRECTORY GLOBALDIR AS '/users/global/scripts';
GRANT ALL ON DIRECTORY GLOBALDIR TO PUBLIC;
```

2. Generate the scripts for the dimension materialized views. The following statements create the scripts chanmv.sql, custmv.sql, prodmv.sql, and timemv.sql in the /users/global/scripts directory.

```
exec dbms_odm.createdimmv_gs
      ('global', 'channel', 'chanmv.sql', 'GLOBALDIR');
exec dbms_odm.createdimmv_gs
      ('global', 'customer', 'custmv.sql', 'GLOBALDIR');
exec dbms_odm.createdimmv_gs
      ('global', 'product', 'prodmv.sql', 'GLOBALDIR');
exec dbms_odm.createdimmv_gs
      ('global', 'time', 'timemv.sql', 'GLOBALDIR');
```

3. Run these scripts to create the dimension materialized views. The scripts will create one materialized view for the CHANNEL dimension, one for the PRODUCT dimension, one for the TIME dimension, and one for each of the two hierarchies of the CUSTOMER dimension.

```
@/users/global/scripts/chanmv
@/users/global/scripts/custmv
@/users/global/scripts/prodmv
```

```
@/users/global/scripts/timemv
```

4. Once you have created the dimension materialized views, execute the following procedure to create a script for the fact materialized view.

```
exec dbms_odm.createstdfactmv
        ('global', 'units_cube', 'units_cube_mv.sql', 'GLOBALDIR',
          false, 'MINIMUM');
```

   This statement creates a script called `units_cube_mv.sql` in the directory `/users/global/scripts`.

5. Run the script to create the fact materialized view.

```
@/users/global/scripts/units_cube_mv
```

The script creates a materialized view with the grouping sets identified in Table 27–2.

Table 27–2 Minimum Grouping Sets for Units Cube

CHANNEL_DIM	CUSTOMER_DIM	PRODUCT_DIM	TIME_DIM
CHANNEL	SHIP_TO (MARKET_ROLLUP hierarchy)	ITEM	QUARTER
CHANNEL	SHIP_TO (MARKET_ROLLUP hierarchy)	ITEM	YEAR
CHANNEL	SHIP_TO (MARKET_ROLLUP hierarchy)	FAMILY	MONTH
CHANNEL	SHIP_TO (MARKET_ROLLUP hierarchy)	TOTAL_PRODUCT	MONTH
CHANNEL	SHIP_TO (SHIPMENTS_ROLLUP hierarchy)	ITEM	QUARTER
CHANNEL	SHIP_TO (SHIPMENTS_ROLLUP hierarchy)	ITEM	YEAR
CHANNEL	SHIP_TO (SHIPMENTS_ROLLUP hierarchy)	FAMILY	MONTH
CHANNEL	SHIP_TO (SHIPMENTS_ROLLUP hierarchy)	TOTAL_PRODUCT	MONTH
CHANNEL	ACCOUNT	ITEM	MONTH
CHANNEL	TOTAL_MARKET	ITEM	MONTH
CHANNEL	ALL_CUSTOMERS	ITEM	MONTH
ALL_CHANNELS	SHIP_TO (MARKET_ROLLUP hierarchy)	ITEM	MONTH
ALL_CHANNELS	SHIP_TO (SHIPMENTS_ROLLUP hierarchy)	ITEM	MONTH
ALL_CHANNELS	ACCOUNT	FAMILY	QUARTER
ALL_CHANNELS	WAREHOUSE	FAMILY	QUARTER
ALL_CHANNELS	TOTAL_MARKET	TOTAL_PRODUCT	YEAR
ALL_CHANNELS	ALL_CUSTOMERS	TOTAL_PRODUCT	YEAR

Example: Manually Choose the Grouping Sets for a Cube

This example creates materialized views for the cube PRICE_CUBE in the GLOBAL schema.

This cube contains unit costs and unit prices for different products over time. The dimensions are PRODUCT, with levels for products, families of products, classes of products, and totals, and TIME with levels for months, quarters, and years.

You want to summarize product families by month and product classes by quarter.

1. Identify a scripts directory. The directory can be specified in the UTL_FILE_DIR initialization parameter, or you can define a directory object with a statement like the following.

```
CREATE OR REPLACE DIRECTORY GLOBALDIR AS '/users/global/scripts';
GRANT ALL ON DIRECTORY GLOBALDIR TO PUBLIC;
```

2. Generate the scripts for the dimension materialized views. The following statements create the scripts prodmv.sql and timemv.sql in the /users/global/scripts directory.

```
exec dbms_odm.createdimmv_gs
      ('global', 'product', 'prodmv.sql', 'GLOBALDIR');
exec dbms_odm.createdimmv_gs
      ('global', 'time', 'timemv.sql', 'GLOBALDIR');
```

3. Run these scripts to create the dimension materialized views. The scripts will create one materialized view for the PRODUCT dimension and one for the TIME dimension.

4. Create the table of dimension levels for the fact materialized view.

```
exec dbms_odm.createdimlevtuple('global', 'price_cube');
```

   The table of levels, sys.olaptablevels, is a temporary table specific to your session. It lists all the levels in PRICE_CUBE. You can view the table as follows.

```
select * from sys.olaptablevels;

SCHEMA_NAME DIMENSION_NAME DIMENSION_OWNER CUBE_NAME    LEVEL_NAME SELECTED
----------- -------------- --------------- ----------   ---------- --------
GLOBAL      TIME           GLOBAL          PRICE_CUBE   YEAR            1
GLOBAL      TIME           GLOBAL          PRICE_CUBE   QUARTER         1
GLOBAL      TIME           GLOBAL          PRICE_CUBE   MONTH           1
GLOBAL      PRODUCT        GLOBAL          PRICE_CUBE   TOTAL_PRODUCT   1
GLOBAL      PRODUCT        GLOBAL          PRICE_CUBE   CLASS           1
GLOBAL      PRODUCT        GLOBAL          PRICE_CUBE   FAMILY          1
GLOBAL      PRODUCT        GLOBAL          PRICE_CUBE   ITEM            1
```

   All the levels are initially selected with "1" in the SELECTED column.

5. Since you want the materialized view to include only product families by month and product classes by quarter, you can deselect all other levels. You could edit the table with a statement like the following.

```
update SYS.OLAPTABLEVELS set selected = 0
   where LEVEL_NAME in ('ITEM','TOTAL_PRODUCT', 'YEAR');
select * from sys.olaptablevels;

SCHEMA_NAME DIMENSION_NAME DIMENSION_OWNER CUBE_NAME    LEVEL_NAME SELECTED
----------- -------------- --------------- ----------   ---------- --------
GLOBAL      TIME           GLOBAL          PRICE_CUBE   YEAR            0
GLOBAL      TIME           GLOBAL          PRICE_CUBE   QUARTER         1
GLOBAL      TIME           GLOBAL          PRICE_CUBE   MONTH           1
GLOBAL      PRODUCT        GLOBAL          PRICE_CUBE   TOTAL_PRODUCT   0
GLOBAL      PRODUCT        GLOBAL          PRICE_CUBE   CLASS           1
GLOBAL      PRODUCT        GLOBAL          PRICE_CUBE   FAMILY          1
GLOBAL      PRODUCT        GLOBAL          PRICE_CUBE   ITEM            0
```

6. Next create the table sys.olaptableveltuples. This table, which is also a session-specific temporary table, contains all the possible combinations of the levels that you selected in the previous step. Each combination of levels, or

grouping set, has an identification number. All the grouping sets are initially selected with "1" in the SELECTED column.

```
exec dbms_odm.createcubeleveltuple('global','price_cube');
select ID, SCHEMA_NAME, CUBE_NAME, DIMENSION_NAME, DIMENSION_OWNER,
       LEVEL_NAME, SELECTED
       from sys.olaptableveltuples;

ID SCHEMA_NAME CUBE_NAME   DIMENSION_NAME DIMENSION_OWNER LEVEL_NAME SELECTED
-- ----------- ---------   -------------- --------------- ---------- --------
1   GLOBAL     PRICE_CUBE  PRODUCT        GLOBAL          FAMILY      1
2   GLOBAL     PRICE_CUBE  PRODUCT        GLOBAL          CLASS       1
3   GLOBAL     PRICE_CUBE  PRODUCT        GLOBAL          FAMILY      1
4   GLOBAL     PRICE_CUBE  PRODUCT        GLOBAL          CLASS       1
1   GLOBAL     PRICE_CUBE  TIME           GLOBAL          MONTH       1
2   GLOBAL     PRICE_CUBE  TIME           GLOBAL          MONTH       1
3   GLOBAL     PRICE_CUBE  TIME           GLOBAL          QUARTER     1
4   GLOBAL     PRICE_CUBE  TIME           GLOBAL          QUARTER     1
```

There are four grouping sets numbered 1, 2, 3, and 4. Each grouping set identifies a unique combination of the levels Quarter and Month in the TIME dimension and CLASS and FAMILY in the PRODUCT dimension.

7. Since you want the materialized view to include only product families by month and product classes by quarter, you can deselect the other level combinations. You could edit the sys.olaptableveltuples table with a statement like the following.

```
update sys.olaptableveltuples set selected = 0
    where ID in ('2', '3');

select ID, SCHEMA_NAME, CUBE_NAME, DIMENSION_NAME, DIMENSION_OWNER,
       LEVEL_NAME, SELECTED
       from sys.olaptableveltuples where SELECTED = '1';

ID SCHEMA_NAME CUBE_NAME  DIMENSION_NAME  DIMENSION_OWNER LEVEL_NAME SELECTED
-- ----------- ---------  --------------  --------------- ---------- --------
1 GLOBAL       PRICE_CUBE PRODUCT         GLOBAL           FAMILY      1
4 GLOBAL       PRICE_CUBE PRODUCT         GLOBAL           CLASS       1
1 GLOBAL       PRICE_CUBE TIME            GLOBAL           MONTH       1
4 GLOBAL       PRICE_CUBE TIME            GLOBAL           QUARTER     1
```

8. To create the script that will generate the fact materialized view, run the CREATEFACTMV_GS procedure.

```
exec dbms_odm.createfactmv_gs
      ('global','price_cube',
       'price_cost_mv.sql','GLOBALDIR',TRUE);
```

The grouping sets specified in the CREATE MATERIALIZED VIEW statement for the cube are:

```
GROUP BY GROUPING SETS (

(TIME_DIM.YEAR_ID, TIME_DIM.QUARTER_ID, TIME_DIM.MONTH_ID,
 PRODUCT_DIM.TOTAL_PRODUCT_ID, PRODUCT_DIM.CLASS_ID, PRODUCT_DIM.FAMILY_ID),

(TIME_DIM.YEAR_ID, TIME_DIM.QUARTER_ID,
 PRODUCT_DIM.TOTAL_PRODUCT_ID, PRODUCT_DIM.CLASS_ID) )
```

9. Go to the `users/global/scripts` directory and run the `price_cost_mv` script to create the fact materialized view.

Summary of DBMS_ODM Subprograms

Table 27–3 DBMS_ODM Subprograms

Subprogram	Description
CREATECUBELEVELTUPLE Procedure on page 27-9	Creates a table of level combinations to be included in the materialized view for a cube.
CREATEDIMLEVTUPLE Procedure on page 27-9	Creates a table of levels to be included in the materialized view for a cube.
CREATEDIMMV_GS Procedure on page 27-10	Generates a script that creates a materialized view for each hierarchy of a dimension.
CREATEFACTMV_GS Procedure on page 27-11	Generates a script that creates a materialized view for the fact table associated with a cube. The materialized view includes individual level combinations that you have previously specified.
CREATESTDFACTMV Procedure on page 27-11	Generates a script that creates a materialized view for the fact table associated with a cube. The materialized view is automatically constructed according to general instructions that you provide.

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. By default, all level 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.

Use this procedure to manually specify the grouping sets for the fact table.

Before calling this procedure, call `CREATEDIMLEVTUPLE` to create the table of levels for the cube.

Syntax

```
CREATECUBELEVELTUPLE (
               cube_owner     IN   VARCHAR2,
               cube_name      IN   VARCHAR2);
```

Parameters

Table 27–4 CREATECUBELEVELTUPLE Procedure Parameters

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

See Also

"Example: Manually Choose the Grouping Sets for a Cube" on page 27-4

CREATEDIMLEVTUPLE Procedure

This procedure creates the table `sys.olaptablevels`, which lists all the levels of all 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.

Use this procedure to manually specify the grouping sets for the fact table.

After calling this procedure, call `CREATECUBELEVELTUPLE` to create the table of level combinations (level tuples) for the cube.

Syntax

```
CREATEDIMLEVTUPLE (
               cube_owner    IN varchar2,
               cube_name     IN varchar2);
```

Parameters

Table 27–5 CREATEDIMLEVTUPLE Procedure Parameters

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

See Also

"Example: Manually Choose the Grouping Sets for a Cube" on page 27-4

CREATEDIMMV_GS Procedure

This procedure generates a script that creates a materialized view for each hierarchy of a dimension. You must call this procedure for each dimension of a cube.

The process of creating the dimension materialized views is the same whether you generate the fact materialized view automatically or manually.

> **Note:** This procedure is overloaded, so that it is backwardly compatible with earlier versions that did not include the `partitioning` parameter.

Syntax

```
CREATEDIMMV_GS (
               dimension_owner    IN   VARCHAR2,
               dimension_name     IN   VARCHAR2,
               output_file        IN   VARCHAR2,
               output_path        IN   VARCHAR2,
               partitioning       IN   BOOLEAN,
               tablespace_mv      IN   VARCHAR2 DEFAULT NULL,
               tablespace_index   IN   VARCHAR2 DEFAULT NULL);
```

Parameters

Table 27–6 CREATEDIMMV_GS Procedure Parameters

Parameter	Description
dimension_owner	Owner of the dimension.
`dimension_name`	Name of the dimension.
`output_file`	File name for the output script.
`output_path`	Directory path where `output_file` will be created. This may be either a directory object or a path set by the `UTL_FILE_DIR` parameter.
`partitioning`	`TRUE` turns on 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.

See Also

"Example: Automatically Generate the Minimum Grouping Sets for a Cube" on page 27-4

"Example: Manually Choose the Grouping Sets for a Cube" on page 27-4

CREATEFACTMV_GS Procedure

This procedure generates a script that creates a materialized view for the fact table associated with a cube.

Use this procedure to manually specify the grouping sets for the fact table.

Prior to calling this procedure, you must call `CREATEDIMLEVTUPLE` and `CREATECUBELEVELTUPLE` to create the `sys.olaptableveltuples` table. The materialized view will include all level combinations selected in the `sys.olaptableveltuples` table.

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 27–7 CREATEFACTMV_GS Procedure Parameters

Parameter	Description
`cube_owner`	Owner of the cube.
`cube_name`	Name of the cube.
`output_file`	File name for the output script.
`output_path`	Directory path where `output_file` will be created.This may be either a directory object or a path set by the `UTL_FILE_DIR` parameter.
`partitioning`	`TRUE` turns on 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.

See Also

"Manually Calculate the Grouping Sets" on page 27-2

"Example: Manually Choose the Grouping Sets for a Cube" on page 27-4

CREATESTDFACTMV Procedure

This procedure generates a script that creates a materialized view for the fact table associated with a cube.

This procedure automatically generates and updates the tables of levels and level tuples. If you want to edit these tables yourself, you must use the `CREATEDIMLEVTUPLE`, `CREATECUBELEVELTUPLE`, and `CREATEFACTMV_GS` procedures.

Syntax

```
CREATESTDFACTMV (
               cube_owner              IN   VARCHAR2,
               cube_name               IN   VARCHAR2,
               outfile                 IN   VARCHAR2,
               outfile_path            IN   VARCHAR2,
               partitioning            IN   BOOLEAN,
               materialization_level   IN   VARCHAR2,
               materialization pct     IN   NUMBER DEFAULT NULL,
               tablespace_mv           IN   VARCHAR2 DEFAULT NULL,
               tablespace_index        IN   VARCHAR2 DEFAULT NULL);
```

Parameters

Table 27–8 CREATESTDFACTMV Procedure Parameters

Parameter	Description
cube_owner	Owner of the cube.
cube_name	Name of the cube.
outfile	File name for the output script.
outfile_path	Directory path where output_file will be created.This may be either a directory object or a path set by the UTL_FILE_DIR parameter.
partitioning	TRUE turns on partitioning; FALSE turns it off.
materialization_level	The level of materialization. This parameter identifies the level combinations that will be included in the materialized view. Specify one of the following values: ■ FULL — Fully materialize the cube's data. Include every level combination in the materialized view. ■ MINIMUM — Minimally materialize the cube's data. See "MINIMUM Grouping Sets" on page 27-12. ■ PERCENT — Materialize the cube's data based on a percentage of the cube's level combinations. For example, consider a cube that has two dimensions with three levels and one dimension with four levels. This cube has 36 possible level combinations (3*3*4). If you choose to materialize the cube by 30%, then 12 level combinations will be included in the materialized view.
materialization_pct	The percentage of level combinations to materialize. Specify this parameter only if you have specified PERCENT for the *materialization_level*.
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.

MINIMUM Grouping Sets

If you choose minimal materialization, your fact materialized view will contain a grouping set for each of the following hierarchy combinations:

- The top level of each hierarchy
- One level above the lowest of each hierarchy

- Top level of one hierarchy and the lowest level of all other hierarchies.
- One level above the lowest of one hierarchy and the lowest level of all other hierarchies.

See Also

"Automatically Calculate the Grouping Sets" on page 27-2

"Example: Automatically Generate the Minimum Grouping Sets for a Cube" on page 27-2

28

OLAP_API_SESSION_INIT

The OLAP_API_SESSION_INIT package provides procedures for maintaining a table of initialization parameters for the OLAP API.

This chapter contains the following topics:

Initialization Parameters for the OLAP API

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.

Note: This package is owned by the SYS user. You must explicitly be granted execution rights before you can use it.

Viewing the Configuration Table

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.

ALL_OLAP_ALTER_SESSION View

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 28–1 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

Summary of OLAP_API_SESSION_INIT Subprograms

The following table describes the subprograms provided in `OLAP_API_SESSION_INIT`.

Table 28–2 OLAP_API_SESSION_INIT Subprograms

Subprogram	Description
ADD_ALTER_SESSION Procedure on page 28-4	Specifies an `ALTER SESSION` parameter for OLAP API users with a particular database role.
CLEAN_ALTER_SESSION Procedure on page 28-4	Removes orphaned data, that is, any `ALTER SESSION` parameters for roles that are no longer defined in the database.
DELETE_ALTER_SESSION Procedure on page 28-4	Removes a previously defined `ALTER SESSION` parameter for OLAP API users with a particular database role.

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 28–3 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.

Example

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 TEST
-----------------   --------------------------------------------------
OLAP_DBA            ALTER SESSION 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();
```

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 28–4 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.

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');
```

29

OLAP_CONDITION

OLAP_CONDITION is a SQL function that dynamically executes an OLAP DML command during a query of an analytic workspace.

See Also:

- Chapter 34, "OLAP_TABLE".
- *Oracle OLAP DML Reference* for information on analytic workspace objects and the syntax of individual OLAP DML commands.

This chapter includes the following topics:

- OLAP_CONDITION Overview
- OLAP_CONDITION Examples
- OLAP_CONDITION Syntax

OLAP_CONDITION Overview

OLAP_CONDITION modifies an analytic workspace within the context of a SELECT FROM OLAP_TABLE statement. You can specify OLAP_CONDITION like other Oracle functions, typically in the WHERE clause.

You can use OLAP_CONDITION to set an option, execute a LIMIT command, execute an OLAP model or forecast, or run a program. The changes made to the workspace can be transitory or they can persist in your session upon completion of the query.

Entry Points in the Limit Map

Parameters of OLAP_CONDITION identify an invocation of OLAP_TABLE, specify an entry point in the limit map, and provide the OLAP DML command to be executed at that entry point.

The target limit map must include a ROW2CELL column. OLAP_CONDITION uses this column to identify an instance of OLAP_TABLE. Within that instance OLAP_CONDITION executes the OLAP DML command at one of three possible entry points. The entry point that you specify will determine whether the condition affects the data returned by the query and whether the condition remains in effect upon completion of the query.

OLAP_CONDITION can be triggered at any of the following points:

- Before the status of the dimensions in the limit map is saved (which occurs before the result set is calculated).

- After the result set has been calculated and before it is fetched. (Default)
- After the result set has been fetched and the status of the dimensions in the limit map has been restored.

The entry points are described in detail in Table 29–2, " Entry Points for OLAP_CONDITION in the OLAP_TABLE Limit Map".

Dynamically Modifying a Workspace during a Query

There are several mechanisms for modifying an analytic workspace on the fly during the execution of OLAP_TABLE. In addition to OLAP_CONDITION, you can use syntax supported by the OLAP_TABLE function itself: The PREDMLCMD and POSTDMLCMD clauses in the limit map, as well as the *olap_command* parameter. OLAP_CONDITION has the advantage of portability, since it is not embedded within OLAP_TABLE, and versatility, since it can be applied at different entry points.

OLAP_TABLE saves the status of dimensions in the limit map before executing the LIMIT commands that generate the result set for the query. After the data is fetched, OLAP_TABLE restores the status of the dimensions. You can specify a PREDMLCMD clause in the limit map to cause an OLAP DML command to execute before the dimension status is saved. Modifications resulting from the PREDMLCMD clause remain in the workspace after execution of OLAP_TABLE, unless reversed with a POSTDMLCMD clause. For more information, see "Limit Map Parameter" on page 34-15.

The *olap_command* parameter of OLAP_TABLE specifies an OLAP DML command that executes immediately before the result set is fetched. In some circumstances, the *olap_command* parameter may contain an OLAP DML FETCH command, which itself manages the fetch. Limits set by the *olap_command* parameter are only in effect during the execution of OLAP_TABLE. For more information, see "OLAP Command Parameter" on page 34-13.

OLAP_CONDITION Examples

Several sample queries using OLAP_CONDITION are shown in Example 29–2. These examples use the PRICE_CUBE in the GLOBAL analytic workspace. The cube has a time dimension, a product dimension, and measures for unit cost and unit price.

> **See Also:** "OLAP_CONDITION Syntax" on page 29-6 for complete descriptions of the syntax used in these examples.

The examples are based on a view called unit_cost_price_view. The SQL for creating this view is shown in Example 29–1. For information about creating views of analytic workspaces, see "OLAP_TABLE Overview" on page 34-1.

Example 29–1 View of PRICE_CUBE in GLOBAL Analytic Workspace

```
-- Create the logical row
SQL>CREATE TYPE unit_cost_price_row AS OBJECT (
            aw_unit_cost          NUMBER,
            aw_unit_price         NUMBER,
            aw_product            VARCHAR2(50),
            aw_product_gid        NUMBER(10),
            aw_time               VARCHAR2(20),
            aw_time_gid           NUMBER(10),
            r2c                   RAW(32));

-- Create the logical table
```

```
SQL>CREATE TYPE unit_cost_price_table AS TABLE OF unit_cost_price_row;

-- Create the view
SQL>CREATE OR REPLACE VIEW unit_cost_price_view AS
    SELECT aw_unit_cost, aw_unit_price, aw_product, aw_product_gid,
           aw_time, aw_time_gid, r2c
      FROM TABLE(OLAP_TABLE(
         'global DURATION SESSION',
         'unit_cost_price_table',
         '',
         'MEASURE aw_unit_cost  FROM price_cube_unit_cost
          MEASURE aw_unit_price FROM price_cube_unit_price
          DIMENSION product WITH
             HIERARCHY product_parentrel
                INHIERARCHY product_inhier
                GID aw_product_gid FROM product_gid
             ATTRIBUTE aw_product FROM product_short_description
          DIMENSION time WITH
             HIERARCHY time_parentrel
                INHIERARCHY time_inhier
                GID aw_time_gid FROM time_gid
             ATTRIBUTE aw_time FROM time_short_description
          ROW2CELL r2c'));

-- query the view
SQL>SELECT * FROM unit_cost_price_view
             WHERE aw_product = 'Hardware'
             AND aw_time in ('2000', '2001', '2002', '2003')
             ORDER BY aw_time;

AW_UNIT_COST AW_UNIT_PRICE AW_PRODUCT AW_PRODUCT_GID AW_TIME AW_TIME_GID R2C
------------ ------------- ---------- -------------- ------- ----------- -----
   211680.12    224713.71  Hardware                3 2000              3  00...
   195591.60    207513.16  Hardware                3 2001              3  00...
   184413.05    194773.78  Hardware                3 2002              3  00...
    73457.31     77275.06  Hardware                3 2003              3  00...
```

Example 29–2 Queries of UNIT_COST_PRICE_VIEW Using OLAP_CONDITION

The queries in this example use OLAP_CONDITION to modify the query of UNIT_COST_PRICE_VIEW in Example 29–1. In each query, OLAP_CONDITION uses a different entry point to limit the TIME dimension to the year 2000.

In the first query, OLAP_CONDIITON uses entry point 0. The limited data is returned by OLAP_TABLE, and the limit remains in effect in the analytic workspace.

```
SQL>SELECT * FROM unit_cost_price_view
             WHERE aw_product = 'Hardware'
             AND aw_time in ('2000', '2001', '2002', '2003')
             AND OLAP_CONDITION(r2c,
                       'limit time to time_short_description eq ''2000''', 0)=1
             ORDER BY aw_time;

AW_UNIT_COST AW_UNIT_PRICE AW_PRODUCT AW_PRODUCT_GID AW_TIME AW_TIME_GID R2C
------------ ------------- ---------- -------------- ------- ----------- -----
   211680.12    224713.71  Hardware                3 2000              3  00...

--Check status in the analytic workspace
SQL>exec dbms_aw.execute('rpr time_short_description');

TIME     TIME_SHORT_DESCRIPTION
```

```
----     ----------------------
  3      2000

-- Reset status
SQL>exec dbms_aw.execute('allstat');
```

In the next query, OLAP_CONDIITON uses entry point 1. The limited data is returned by OLAP_TABLE, but the limit does not remain in effect in the analytic workspace.

Note that the third parameter is not required in this case, since entry point 1 is the default.

```
SQL>SELECT * FROM unit_cost_price_view
             WHERE aw_product = 'Hardware'
             AND aw_time in ('2000', '2001', '2002', '2003')
             AND OLAP_CONDITION(r2c,
                       'limit time to time_short_description eq ''2000''', 1)=1
             ORDER BY aw_time;

AW_UNIT_COST AW_UNIT_PRICE AW_PRODUCT AW_PRODUCT_GID AW_TIME AW_TIME_GID R2C
------------ ------------- ---------- -------------- ------- ----------- -----
   211680.12    224713.71  Hardware               3 2000              3  00...

--Check status in the analytic workspace
SQL>exec dbms_aw.execute('rpr time_short_description');

TIME     TIME_SHORT_DESCRIPTION
----     ----------------------
 19      Jan-98
 20      Feb-98
 21      Mar-98
 22      Apr-98
.
.
.
  1      1998
  2      1999
  3      2000
  4      2001
 85      2002
102      2003
119      2004

-- Reset status
SQL>exec dbms_aw.execute('allstat');
```

In the final query, OLAP_CONDIITON uses entry point 2. The limit does not affect the data returned by OLAP_TABLE, but the limit remains in effect in the analytic workspace.

```
SQL>SELECT * FROM unit_cost_price_view
             WHERE aw_product = 'Hardware'
             AND aw_time in ('2000', '2001', '2002', '2003')
             AND OLAP_CONDITION(r2c,
                       'limit time to time_short_description eq ''2000''', 2)=1
             ORDER BY aw_time;

AW_UNIT_COST AW_UNIT_PRICE AW_PRODUCT AW_PRODUCT_GID AW_TIME AW_TIME_GID R2C
------------ ------------- ---------- -------------- ------- ----------- -----
   211680.12    224713.71  Hardware               3 2000              3  00...
   195591.60    207513.16  Hardware               3 2001              3  00...
```

```
   184413.05    194773.78  Hardware                3 2002             3  00...
    73457.31     77275.06  Hardware                3 2003             3  00...

--Check status in the analytic workspace
SQL>exec dbms_aw.execute('rpr time_short_description');

TIME     TIME_SHORT_DESCRIPTION
----     ----------------------
  3      2000
```

OLAP_CONDITION Syntax

The OLAP_CONDITION function executes an OLAP DML command at one of three entry points in the limit map used in a call to OLAP_TABLE.

Syntax

```
OLAP_CONDITION(
          r2c          IN   RAW(32),
          expression   IN   VARCHAR2,
          event        IN   NUMBER DEFAULT 1);
     RETURN NUMBER;
```

Parameters

Table 29–1 OLAP_CONDITION Function Parameters

Parameter	Description
r2c	The name of a column specified by a ROW2CELL clause in the limit map. This parameter is used by OLAP_CONDITION to identify a particular invocation of OLAP_TABLE. The ROW2CELL column is used in the processing of the single-row functions. (See Chapter 30, "OLAP_EXPRESSION") OLAP_CONDITION simply uses it as an identifier. For information on creating a ROW2CELL column, see "Limit Map Parameter" on page 34-15.
expression	A single OLAP DML command to be executed within the context of the OLAP_TABLE function identified by the *r2c* parameter. For information on the OLAP DML, see the *Oracle OLAP DML Reference*.
event	The event during OLAP_TABLE processing that will trigger the execution of the OLAP DML command specified by the *expression* parameter. This parameter can have the value 0, 1, or 2, as described in Table 29–2

Returns

The number 1 to indicate a successful invocation of OLAP_CONDITION.

Note

The entry points for OLAP_CONDITION are described in Table 29–2. Refer to "Order of Processing in OLAP_TABLE" on page 34-21 to determine where each entry point occurs.

Table 29–2 Entry Points for OLAP_CONDITION in the OLAP_TABLE Limit Map

Entry Point	Description
0	Execute the OLAP DML command after the PREDMLCMD clause of the limit map is processed and before the status of the dimensions in the limit map is saved. The entry point is between steps 1 and 2 in "Order of Processing in OLAP_TABLE" on page 34-21. If OLAP_CONDITION limits any of the dimensions in the limit map, the limits remain in the workspace after the execution of OLAP_TABLE (unless a command in the POSTDMLCMD clause of the limit map changes the status).

Table 29–2 (Cont.) Entry Points for OLAP_CONDITION in the OLAP_TABLE Limit Map

Entry Point	Description
1	Execute the OLAP DML command after the conditions of the `WHERE` clause are satisfied and before the data is fetched. (Default.) The entry point is between steps 4 and 5 in "Order of Processing in OLAP_TABLE" on page 34-21. If an OLAP DML command (other than `FETCH`) is specified in the *olap_command* parameter of `OLAP_TABLE`, it is executed after `OLAP_CONDITION` and before the data is fetched. (The use of a `FETCH` command in the *olap_command* parameter, or in `OLAP_CONDITION` itself, is not generally recommended. See "Using FETCH in the olap_command Parameter" on page 34-14.) If `OLAP_CONDITION` limits any of the dimensions in the limit map, the limits remain in effect for the duration of the query only.
2	Execute the OLAP DML command after the data is fetched and the status of dimensions in the limit map has been restored. The entry point is after step 8 in "Order of Processing in OLAP_TABLE" on page 34-21. If `OLAP_CONDITION` limits any dimensions, the limits remain in the analytic workspace after the query completes.

Example

See "OLAP_CONDITION Examples" on page 29-2.

30

OLAP_EXPRESSION

OLAP_EXPRESSION is a SQL function that dynamically executes a single-row numeric function in an analytic workspace and returns the results.

See Also:

- *Oracle OLAP Application Developer's Guide* for information about using OLAP_EXPRESSION to create custom measures.
- *Oracle OLAP DML Reference* for information on analytic workspace objects and the syntax of individual OLAP DML commands.
- Chapter 31, "OLAP_EXPRESSION_BOOL"
- Chapter 32, "OLAP_EXPRESSION_DATE"
- Chapter 33, "OLAP_EXPRESSION_TEXT"
- Chapter 29, "OLAP_CONDITION"
- Chapter 34, "OLAP_TABLE"

This chapter includes the following topics:

- OLAP_EXPRESSION Overview
- OLAP_EXPRESSION Examples
- OLAP_EXPRESSION Syntax

OLAP_EXPRESSION Overview

OLAP_EXPRESSION acts as a numeric single-row function within the context of a SELECT FROM OLAP_TABLE statement. You can specify OLAP_EXPRESSION in the same way you specify other Oracle single-row functions, notably in the select list, WHERE , and ORDER BY clauses.

Single-Row Functions

Single-row functions return a single result row for every row of a queried table or view. Oracle supports a number of predefined single-row functions, for example COS, LOG, and ROUND which return numeric data, and UPPER and LOWER which return character data. For more information on single-row functions, refer to the *Oracle Database SQL Reference*.

The OLAP single-row functions, OLAP_EXPRESSION and its variants for text, date, and boolean data, return the result of an OLAP DML expression that you specify. The OLAP DML supports a rich syntax for specifying computations ranging from simple

arithmetic expressions to statistical, financial, and time-series operations. You can use OLAP_EXPRESSION to dynamically perform any valid numeric expression within an analytic workspace and retrieve its results. For more information on OLAP DML expressions, refer to the *Oracle OLAP DML Reference*.

OLAP_EXPRESSION and OLAP_TABLE

OLAP_TABLE uses a limit map to present the multidimensional data from an analytic workspace in tabular form. The limit map specifies the columns of the logical table. When an OLAP_EXPRESSION function is specified in the select list of the query, OLAP_TABLE generates additional columns for the results of the function.

To use OLAP_EXPRESSION, you must specify a ROW2CELL clause in the limit map used by OLAP_TABLE. ROW2CELL identifies a RAW column that OLAP_TABLE populates with information used by the OLAP single-row functions.

> **See Also:** "Limit Maps" on page 34-1 and "Limit Map: ROW2CELL Clause" on page 34-20

OLAP_EXPRESSION Examples

The following script was used to create the view unit_cost_price_view, which is used in Example 30–1 and Example 30–2 to illustrate the use of OLAP_EXPRESSION. For information about creating views of analytic workspaces, see "OLAP_TABLE Overview" on page 34-1.

Sample View: GLOBAL.UNIT_COST_PRICE_VIEW

```
-- Create the logical row
CREATE TYPE unit_cost_price_row AS OBJECT (
            aw_unit_cost          NUMBER,
            aw_unit_price         NUMBER,
            aw_product            VARCHAR2(50),
            aw_time               VARCHAR2(20),
            r2c                   RAW(32));
/
-- Create the logical table
CREATE TYPE unit_cost_price_table AS TABLE OF unit_cost_price_row;
/
-- Create the view
CREATE OR REPLACE VIEW unit_cost_price_view AS
    SELECT aw_unit_cost, aw_unit_price, aw_product, aw_time, r2c
      FROM TABLE(OLAP_TABLE(
         'global DURATION SESSION',
         'unit_cost_price_table',
         '',
         'MEASURE aw_unit_cost  FROM price_cube_unit_cost
          MEASURE aw_unit_price FROM price_cube_unit_price
          DIMENSION product WITH
             HIERARCHY product_parentrel
                INHIERARCHY product_inhier
             ATTRIBUTE aw_product FROM product_short_description
          DIMENSION time WITH
             HIERARCHY time_parentrel
                INHIERARCHY time_inhier
             ATTRIBUTE aw_time FROM time_short_description
          ROW2CELL r2c'));
/
```

The following query shows some of the aggregate data in the view.

```
SQL>SELECT * FROM unit_cost_price_view
             WHERE aw_product = 'Hardware'
             AND aw_time in ('2000', '2001', '2002', '2003')
             ORDER BY aw_time;

AW_UNIT_COST    AW_UNIT_PRICE    AW_PRODUCT    AW_TIME    R2C
------------    -------------    ----------    -------    -----
   211680.12        224713.71      Hardware       2000    00...
   195591.60        207513.16      Hardware       2001    00...
   184413.05        194773.78      Hardware       2002    00...
    73457.31         77275.06      Hardware       2003    00...
```

Example 30–1 OLAP_EXPRESSION: Time Series Function in a WHERE Clause

This example uses the view described in "Sample View: GLOBAL.UNIT_COST_PRICE_VIEW" on page 30-2.

The following `SELECT` statement calculates an expression with an alias of `PERIODAGO`, and limits the result set to calculated values greater than 50,000. The calculation uses the `LAG` function to return the value of the previous time period.

```
SQL>SELECT aw_time time, aw_unit_cost unit_cost,
            OLAP_EXPRESSION(r2c,
                            'LAG(price_cube_unit_cost, 1, time,
                                 LEVELREL time_levelrel)') periodago
        FROM unit_cost_price_view
        WHERE aw_product = 'Hardware'
        AND OLAP_EXPRESSION(r2c,
                            'LAG(price_cube_unit_cost, 1, time,
                                 LEVELREL time_levelrel)') > 50000;
```

This `SELECT` statement produces these results.

```
TIME                 UNIT_COST  PERIODAGO
-------------------- ---------- ----------
2003                  73457.31  184413.05
2004                             73457.31
1999                  231095.4  162526.92
2000                 211680.12   231095.4
2001                  195591.6  211680.12
2002                 184413.05   195591.6
Q2-99                 57587.34   57856.76
Q3-99                 59464.25   57587.34
Q4-99                 56187.05   59464.25
Q1-00                 53982.32   56187.05
Q2-00                 53629.74   53982.32
Q3-00                 53010.65   53629.74
Q4-00                 51057.41   53010.65
Q1-01                 49691.22   51057.41
```

Example 30–2 OLAP_EXPRESSION: Numeric Calculation in an ORDER BY CLause

This example uses the view described in "Sample View: GLOBAL.UNIT_COST_PRICE_VIEW" on page 30-2.

This example subtracts costs from price, and gives this expression an alias of `MARKUP`. The rows are ordered by markup from highest to lowest.

```
SQL>SELECT aw_time time, aw_unit_cost unit_cost, aw_unit_price unit_price,
            OLAP_EXPRESSION(r2c,
                        'PRICE_CUBE_UNIT_PRICE - PRICE_CUBE_UNIT_COST') markup
     FROM unit_cost_price_view
```

```
WHERE aw_product = 'Hardware'
AND aw_time in ('1998', '1999', '2000', '2001')
ORDER BY OLAP_EXPRESSION(r2c,
                'PRICE_CUBE_UNIT_PRICE - PRICE_CUBE_UNIT_COST') DESC;
```

This SELECT statement produces these results.

```
TIME                 UNIT_COST   UNIT_PRICE  MARKUP
-------------------- ----------  ----------  ---------
1999                 231095.40   245412.91   14317.51
2000                 211680.12   224713.71   13033.59
2001                 195591.60   207513.16   11921.56
1998                 162526.92   173094.41   10567.49
```

OLAP_EXPRESSION Syntax

The OLAP_EXPRESSION function dynamically executes an OLAP DML numeric expression within the context of an OLAP_TABLE function. In addition to returning a custom measure, OLAP_EXPRESSION can be used in the WHERE and ORDER BY clauses to modify the result set of the query of the analytic workspace.

Syntax

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

Parameters

Table 30–1 OLAP_EXPRESSION Function Parameters

Parameter	Description
r2c	The name of a column specified by a ROW2CELL clause in the limit map. OLAP_TABLE populates this column with information used by the OLAP single-row functions, including OLAP_EXPRESSION. See "Limit Map Parameter" on page 34-15.
numeric_ expression	An OLAP DML expression that returns a numeric result. Search for "expressions" in the *Oracle OLAP DML Reference*. See also "Guidelines for Using Quotation Marks in OLAP DML Commands" on page 24-4.

Returns

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

OLAP_EXPRESSION returns numeric data. To return text, boolean, or date data, use the OLAP_EXPRESSION_TEXT, OLAP_EXPRESSION_BOOL, or OLAP_EXPRESSION_ DATE functions.

Example

See "OLAP_EXPRESSION Examples" on page 30-2.

31

OLAP_EXPRESSION_BOOL

OLAP_EXPRESSION_BOOL is a SQL function that dynamically executes a single-row boolean function in an analytic workspace and returns the results.

See Also: Chapter 30, "OLAP_EXPRESSION"

This chapter includes the following topics:

- OLAP_EXPRESSION_BOOL Overview
- OLAP_EXPRESSION_BOOL Example
- OLAP_EXPRESSION_BOOL Syntax

OLAP_EXPRESSION_BOOL Overview

OLAP_EXPRESSION_BOOL acts as a boolean single-row function within the context of a SELECT FROM OLAP_TABLE statement. You can specify OLAP_EXPRESSION_BOOL in the same way you specify other Oracle single-row functions, notably in the select list and WHERE clauses.

Single-Row Functions

Single-row functions return a single result row for every row of a queried table or view. Oracle supports a number of predefined single-row functions, for example COS, LOG, and ROUND which return numeric data, and UPPER and LOWER which return character data. For more information on single-row functions, refer to the *Oracle Database SQL Reference*.

The OLAP single-row functions, OLAP_EXPRESSION and its variants for text, date, and boolean data, return the result of an OLAP DML expression that you specify. The OLAP DML supports a rich syntax for specifying computations ranging from simple arithmetic expressions to statistical, financial, and time-series operations.

You can use OLAP_EXPRESSION_BOOL to dynamically perform any valid boolean expression within an analytic workspace and retrieve its results. For more information on boolean expressions in the OLAP DML, search for "boolean expression" in the *Oracle OLAP DML Reference*.

OLAP_EXPRESSION_BOOL and OLAP_TABLE

OLAP_TABLE uses a limit map to present the multidimensional data from an analytic workspace in tabular form. The limit map specifies the columns of the logical table. When an OLAP_EXPRESSION_BOOL function is specified in the select list of the query, OLAP_TABLE generates an additional column for the results of the function.

To use OLAP_EXPRESSION_BOOL, you must specify a ROW2CELL clause in the limit map used by OLAP_TABLE. ROW2CELL identifies a RAW column that OLAP_TABLE populates with information used by the OLAP single-row functions.

> **See Also:** "Limit Maps" on page 34-1 and "Limit Map: ROW2CELL Clause" on page 34-20

OLAP_EXPRESSION_BOOL Example

The following script was used to create the view awunits_view, which is used in Example 31–1 to illustrate the use of OLAP_EXPRESSION_BOOL.

> **See Also:** See "OLAP_TABLE Overview" on page 34-1 for information about creating views of analytic workspaces.

Sample View: GLOBAL_AW.AWUNITS_VIEW

```
-- Create the logical row
CREATE TYPE awunits_row AS OBJECT (
             awtime                  VARCHAR2(12),
             awcustomer              VARCHAR2(30),
             awproduct               VARCHAR2(30),
             awchannel               VARCHAR2(30),
             awunits                 NUMBER(16),
             r2c                     RAW(32));
/
-- Create the logical table
CREATE TYPE awunits_table AS TABLE OF awunits_row;
/
-- Create the view
CREATE OR REPLACE VIEW awunits_view AS
   SELECT awunits,
          awtime, awcustomer, awproduct, awchannel, r2c
      FROM TABLE(OLAP_TABLE(
         'global_aw.globalaw DURATION SESSION',
         'awunits_table',
         '',
         'MEASURE awunits FROM units_cube_aw_units_aw
          DIMENSION awtime FROM time_aw WITH
             HIERARCHY time_aw_parentrel
          DIMENSION awcustomer FROM customer_aw WITH
             HIERARCHY customer_aw_parentrel
                       (customer_aw_hierlist ''MARKET_ROLLUP_AW'')
                INHIERARCHY customer_aw_inhier
          DIMENSION awproduct FROM product_aw WITH
             HIERARCHY product_aw_parentrel
          DIMENSION channel_aw WITH
             HIERARCHY channel_aw_parentrel
             ATTRIBUTE  awchannel FROM channel_aw_short_description
          ROW2CELL r2c'))
      WHERE awunits IS NOT NULL;
/
```

The following query shows some of the aggregate data in the view. For all products in all markets during the year 2001, it shows the number of units sold through each channel.

```
SQL> SELECT awchannel, awunits FROM awunits_view
     WHERE    awproduct = '1'
     AND      awcustomer = '7'
     AND      awtime = '4';
```

```
AWCHANNEL            AWUNITS
---------            -------
All Channels         415392
Direct Sales         43783
Catalog              315737
Internet             55872
```

The following statements show the descriptions of the Product, Customer, and Time dimension members used in the query.

```
SQL>execute dbms_aw.execute('limit product_aw to ''1''');
SQL>execute dbms_aw.execute('rpr product_aw_short_description');

PRODUCT_AW                PRODUCT_AW_SHORT_DESCRIPTION
--------------- --------------------------------------------------
1               Total Product

SQL>execute dbms_aw.execute('limit customer_aw to ''7''');
SQL>execute dbms_aw.execute('rpr customer_aw_short_description');

CUSTOMER_AW               CUSTOMER_AW_SHORT_DESCRIPTION
--------------- --------------------------------------------------
7               Total Market

SQL>execute dbms_aw.execute('limit time_aw to ''4''');
SQL>execute dbms_aw.execute('rpr time_aw_short_description');

TIME_AW                    TIME_AW_SHORT_DESCRIPTION
--------------- --------------------------------------------------
4               2001
```

Example 31–1 OLAP_EXPRESSION_BOOL Function in a SELECT List

This example uses the view described in "Sample View: GLOBAL_AW.AWUNITS_VIEW" on page 31-2.

The following `SELECT` statement calculates an expression with an alias of `lowest_units`, which indicates whether or not the number of units of each product was less than 500.

```
SQL>SELECT awproduct products,
       olap_expression_bool(r2c, 'units_cube_aw_units_aw le 500') lowest_units
          FROM awunits_view
               WHERE     awproduct > 39
               AND       awproduct < 46
               AND       awcustomer = '7'
               AND       awchannel = 'Internet'
               AND       awtime = '4';

PRODUCTS        LOWEST_UNITS
--------------- ------------
40              0
41              1
42              1
43              1
44              1
45              0
```

This query shows that products 41-44 all had less than 500 units. These products are the documentation sets in German, French, Spanish, and Italian. The selected products are shown as follows.

```
SQL>execute dbms_aw.execute
           ('limit product_aw to product_aw gt 39 and product_aw lt 46');
SQL>execute dbms_aw.execute('rpr product_aw_short_description');

PRODUCT_AW                 PRODUCT_AW_SHORT_DESCRIPTION
-------------- --------------------------------------------------
40             O/S Documentation Set - English
41             O/S Documentation Set - German
42             O/S Documentation Set - French
43             O/S Documentation Set - Spanish
44             O/S Documentation Set - Italian
45             O/S Documentation Set - Kanji
```

OLAP_EXPRESSION_BOOL Syntax

The OLAP_EXPRESSION_BOOL function dynamically executes an OLAP DML boolean expression within the context of an OLAP_TABLE function.

Syntax

```
OLAP_EXPRESSION_BOOL(
          r2c                 IN   RAW(32),
          boolean_expression  IN   VARCHAR2)
     RETURN NUMBER;
```

Parameters

Table 31–1 OLAP_EXPRESSION_BOOL Function Parameters

Parameter	Description
r2c	The name of a column populated by a ROW2CELL clause in a call to OLAP_TABLE. ROW2CELL is a component of a limit map parameter of the OLAP_TABLE function. See "Limit Map Parameter" on page 34-15.
boolean_ expression	A boolean calculation that will be performed in the analytic workspace. Search for "boolean expression" in the *Oracle OLAP DML Reference*. See also "Guidelines for Using Quotation Marks in OLAP DML Commands" on page 24-4.

Returns

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

OLAP_EXPRESSION_BOOL returns boolean data. To return numeric, date, or text data, use the OLAP_EXPRESSION, OLAP_EXPRESSION_DATE, or OLAP_EXPRESSION_ TEXT functions.

Example

Refer to "OLAP_EXPRESSION" on page 30-1 for more examples of OLAP single-row functions.

32

OLAP_EXPRESSION_DATE

OLAP_EXPRESSION_DATE is a SQL function that dynamically executes a single-row date function in an analytic workspace and returns the results.

See Also: Chapter 30, "OLAP_EXPRESSION"

This chapter includes the following topics:

- OLAP_EXPRESSION_DATE Overview
- OLAP_EXPRESSION_DATE Syntax

OLAP_EXPRESSION_DATE Overview

OLAP_EXPRESSION_DATE acts as a single-row function within the context of a SELECT FROM OLAP_TABLE statement. You can specify OLAP_EXPRESSION_DATE in the same way you specify other Oracle single-row functions, notably in the select list and WHERE and ORDER BY clauses.

Single-Row Functions

Single-row functions return a single result row for every row of a queried table or view. Oracle supports a number of predefined single-row functions, for example COS, LOG, and ROUND which return numeric data, and UPPER and LOWER which return character data. For more information on single-row functions, refer to the *Oracle Database SQL Reference*.

The OLAP single-row functions, OLAP_EXPRESSION and its variants for text, date, and boolean data, return the result of an OLAP DML expression that you specify. The OLAP DML supports a rich syntax for specifying computations ranging from simple arithmetic expressions to statistical, financial, and time-series operations.

You can use OLAP_EXPRESSION_DATE to dynamically calculate any valid date expression within an analytic workspace and retrieve its results. For more information on date expressions in the OLAP DML, search for "working with dates in text expressions" and DATEFORMAT in the *Oracle OLAP DML Reference*.

OLAP_EXPRESSION_DATE and OLAP_TABLE

OLAP_TABLE uses a limit map to present the multidimensional data from an analytic workspace in tabular form. The limit map specifies the columns of the logical table. When an OLAP_EXPRESSION_DATE function is specified in the select list of the query, OLAP_TABLE generates an additional column for the results of the function.

To use OLAP_EXPRESSION_DATE, you must specify a ROW2CELL clause in the limit map used by OLAP_TABLE. ROW2CELL identifies a RAW column that OLAP_TABLE populates with information used by the OLAP single-row functions.

> **See Also:** "Limit Maps" on page 34-1 and "Limit Map: ROW2CELL Clause" on page 34-20

OLAP_EXPRESSION_DATE Syntax

The OLAP_EXPRESSION_DATE function dynamically executes an OLAP DML date expression within the context of an OLAP_TABLE function.

Syntax

```
OLAP_EXPRESSION_DATE(
          r2c               IN   RAW(32),
          date_expression   IN   VARCHAR2)
     RETURN NUMBER;
```

Parameters

Table 32–1 OLAP_EXPRESSION_DATE Function Parameters

Parameter	Description
r2c	The name of a column populated by a ROW2CELL clause in a call to OLAP_TABLE. ROW2CELL is a component of a limit map parameter of the OLAP_TABLE function. See "Limit Map Parameter" on page 34-15.
date_ expression	A date expression in the analytic workspace. Search for "working with dates in text expressions" and DATEFORMAT in the *Oracle OLAP DML Reference*. See also "Guidelines for Using Quotation Marks in OLAP DML Commands" on page 24-4.

Returns

An evaluation of *date_expression* for each row of the table object returned by the OLAP_ TABLE function.

OLAP_EXPRESSION_DATE returns date data. To return numeric, boolean, or text data, use the OLAP_EXPRESSION, OLAP_EXPRESSION_BOOL, or OLAP_EXPRESSION_ TEXT functions.

Example

Refer to "OLAP_EXPRESSION Examples" on page 30-2 and "OLAP_EXPRESSION_ BOOL Example" on page 31-2 for examples of OLAP single-row functions.

33

OLAP_EXPRESSION_TEXT

OLAP_EXPRESSION_TEXT is a SQL function that dynamically executes a single-row character function in an analytic workspace and returns the results.

See Also: Chapter 30, "OLAP_EXPRESSION"

This chapter includes the following topics:

- OLAP_EXPRESSION_TEXT Overview
- OLAP_EXPRESSION_TEXT Syntax

OLAP_EXPRESSION_TEXT Overview

OLAP_EXPRESSION_TEXT acts as a single-row character function within the context of a SELECT FROM OLAP_TABLE statement. You can specify OLAP_EXPRESSION_TEXT in the same way you specify other Oracle single-row functions, notably in the select list and WHERE and ORDER BY clauses.

Single-Row Functions

Single-row functions return a single result row for every row of a queried table or view. Oracle supports a number of predefined single-row functions, for example COS, LOG, and ROUND which return numeric data, and UPPER and LOWER which return character data. For more information on single-row functions, refer to the *Oracle Database SQL Reference*.

The OLAP single-row functions, OLAP_EXPRESSION and its variants for text, date, and boolean data, return the result of an OLAP DML expression that you specify. The OLAP DML supports a rich syntax for specifying computations ranging from simple arithmetic expressions to statistical, financial, and time-series operations.

You can use OLAP_EXPRESSION_TEXT to dynamically calculate any valid text expression within an analytic workspace and retrieve its results. For more information on text expressions in the OLAP DML, search for "text expression" in the *Oracle OLAP DML Reference*.

OLAP_EXPRESSION_TEXT and OLAP_TABLE

OLAP_TABLE uses a limit map to present the multidimensional data from an analytic workspace in tabular form. The limit map specifies the columns of the logical table. When an OLAP_EXPRESSION_TEXT function is specified in the select list of the query, OLAP_TABLE generates an additional column for the results of the function.

To use OLAP_EXPRESSION_TEXT, you must specify a ROW2CELL clause in the limit map used by OLAP_TABLE. ROW2CELL identifies a RAW column that OLAP_TABLE populates with information used by the OLAP single-row functions.

See Also: "Limit Maps" on page 34-1 and "Limit Map: ROW2CELL Clause" on page 34-20

OLAP_EXPRESSION_TEXT Syntax

The OLAP_EXPRESSION_TEXT function dynamically executes an OLAP DML text expression within the context of an OLAP_TABLE function.

Syntax

```
OLAP_EXPRESSION_TEXT(
          r2c               IN   RAW(32),
          text_expression   IN   VARCHAR2)
     RETURN NUMBER;
```

Parameters

Table 33–1 OLAP_EXPRESSION_TEXT Function Parameters

Parameter	Description
r2c	The name of a column populated by a ROW2CELL clause in a call to OLAP_TABLE. ROW2CELL is a component of a limit map parameter of the OLAP_TABLE function. See "Limit Map Parameter" on page 34-15.
text_ expression	A text expression in the analytic workspace. Search for "text expression" in the *Oracle OLAP DML Reference*. See also "Guidelines for Using Quotation Marks in OLAP DML Commands" on page 24-4.

Returns

An evaluation of *text_expression* for each row of the table object returned by the OLAP_ TABLE function.

OLAP_EXPRESSION_TEXT returns character data. To return numeric, boolean, or date data, use the OLAP_EXPRESSION, OLAP_EXPRESSION_BOOL, or OLAP_ EXPRESSION_DATE functions.

Example

Refer to "OLAP_EXPRESSION Examples" on page 30-2 and "OLAP_EXPRESSION_ BOOL Example" on page 31-2 for examples of OLAP single-row functions.

34

OLAP_TABLE

OLAP_TABLE is a SQL function that extracts multidimensional data from an analytic workspace and presents it in the two-dimensional format of a relational table.

See Also:

- *Oracle OLAP Application Developer's Guide*
- *Oracle OLAP DML Reference*
- Chapter 29, "OLAP_CONDITION"
- Chapter 30, "OLAP_EXPRESSION"

This chapter contains the following topics:

OLAP_TABLE Overview

OLAP_TABLE is the fundamental mechanism in the Database for querying an analytic workspace. Within a SQL statement, you can specify an OLAP_TABLE function call wherever you would provide the name of a table or view.

OLAP_TABLE returns a table of objects that can be joined to relational tables and views, and to other tables of objects populated by OLAP_TABLE.

OLAP_TABLE is used internally by the tools and APIs that access analytic workspaces. For example, Analytic Workspace Manager, the Active Catalog views, the OLAP Java APIs, and the DBMS_AW package all use OLAP_TABLE to obtain data and other information from analytic workspaces.

Note: The OLAP tools and APIs that use OLAP_TABLE require database standard form, but OLAP_TABLE itself does not use standard form metadata. See the *Oracle OLAP Application Developer's Guide* for information on standard form.

Limit Maps

OLAP_TABLE uses a **limit map** to map dimensions and measures defined in an analytic workspace to columns in a logical table. The limit map combines with the

WHERE clause of a SQL SELECT statement to generate a series of OLAP DML LIMIT commands that are executed in the analytic workspace.

OLAP_TABLE can use a limit map in conjunction with a predefined logical table, or it can use the information in a limit map to dynamically generate a logical table at runtime.

See Also: "Limit Map Parameter" on page 34-15.

Logical Tables

The logical table populated by OLAP_TABLE is actually a table type whose rows are user-defined object types, also known as **Abstract Data Types** or **ADTs**.

A user-defined object type is composed of attributes, which are equivalent to the columns of a table. The basic syntax for defining a row is as follows.

```
CREATE TYPE object_name AS OBJECT (
   attribute1      datatype,
   attribute2      datatype,
   attributen      datatype);
```

A table type is a collection of object types; this collection is equivalent to the rows of a table. The basic syntax for creating a table type is as follows.

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

See Also:

- *Oracle Database Application Developer's Guide - Object-Relational Features* for information about object types
- "Create Type" in the *Oracle Database SQL Reference*

Using OLAP_TABLE With Predefined ADTs

You can predefine the table of objects or generate it dynamically. When you create the table type in advance, it is available in the database for use by any invocation of OLAP_TABLE. Queries that use predefined objects typically perform better than queries that dynamically generate the objects.

Example 34–1 shows how to create a view of an analytic workspace using predefined ADTs.

Example 34–1 Template for Creating a View Using Predefined ADTs

```
SET ECHO ON
SET SERVEROUT ON

DROP TYPE table_obj;
DROP TYPE row_obj;

CREATE TYPE row_obj AS OBJECT (
            column_first     datatype,
            column_next      datatype,
            column_n         datatype);
/
CREATE TYPE table_obj AS TABLE OF row_obj;
/
CREATE OR REPLACE VIEW view_name AS
   SELECT column_first, column_next, column_n
      FROM TABLE(OLAP_TABLE(
```

```
        'analytic_workspace',
        'table_obj',
        'olap_command',
        'limit_map'));
/
COMMIT;
/
GRANT SELECT ON view_name TO PUBLIC;
```

Example 34–2 uses OLAP_TABLE with a predefined table type to create a relational view of the TIME dimension in the GLOBAL analytic workspace of the GLOBAL_AW schema. The first parameter in the OLAP_TABLE call is the name of the analytic workspace. The second is the name of the predefined table type. The forth is the limit map that specifies how to map the workspace dimension to the columns of the predefined table type. The third parameter is not specified.

Example 34–2 Sample View of the TIME Dimension Using Predefined ADTs

```
CREATE TYPE time_cal_row AS OBJECT (
           time_id          varchar2(32),
           cal_short_label  varchar2(32),
           cal_end_date     date,
           cal_timespan     number(6));

CREATE TYPE time_cal_table AS TABLE OF time_cal_row;

CREATE OR REPLACE VIEW time_cal_view AS
   SELECT time_id, cal_short_label, cal_end_date, cal_timespan
      FROM TABLE(OLAP_TABLE(
         'global_aw.global duration session',
         'time_cal_table',
          '',
         'DIMENSION time_id from time with
            HIERARCHY time_parentrel
               INHIERARCHY time_inhier
            ATTRIBUTE cal_short_label from time_short_description
            ATTRIBUTE cal_end_date    from time_end_date
            ATTRIBUTE cal_timespan    from time_time_span'));
```

Using OLAP_TABLE With Automatic ADTs

If you do not supply the name of a table type as an argument, OLAP_TABLE uses information in the limit map to generate the logical table automatically. In this case, the table type is only available at runtime within the context of the calling SQL SELECT statement.

Example 34–3 shows how to create a view of an analytic workspace using automatic ADTs.

Example 34–3 Template for Creating a View Using Automatic ADTs

```
SET ECHO ON
SET SERVEROUT ON

CREATE OR REPLACE VIEW view_name AS
   SELECT column_first, column_next, column_n
      FROM TABLE(OLAP_TABLE(
         'analytic_workspace',
         '',
         'olap_command',
```

```
        'limit_map'));
/
COMMIT;
/
GRANT SELECT ON view_name TO PUBLIC;
```

Example 34–4 creates the same view produced by Example 34–2, but it automatically generates the ADTs instead of using a predefined table type. It uses AS clauses in the limit map to specify the data types of the target columns.

Example 34–4 View of the TIME Dimension Using Automatic ADTs

```
CREATE OR REPLACE VIEW time_cal_view AS
   SELECT time_id, cal_short_label, cal_end_date, cal_timespan
      FROM TABLE(OLAP_TABLE(
        'global_aw.global duration session',
        null,
        null,
        'DIMENSION time_id AS varchar2(32) FROM time WITH
           HIERARCHY time_parentrel
              INHIERARCHY time_inhier
           ATTRIBUTE cal_short_label AS VARCHAR2(32) from time_short_description
           ATTRIBUTE cal_end_date AS DATE            from time_end_date
           ATTRIBUTE cal_timespan AS NUMBER(6)       from time_time_span'));
```

When automatically generating ADTs, OLAP_TABLE uses default relational data types for the target columns unless you override them with AS clauses in the limit map. The default data type conversions used by OLAP_TABLE are described in Table 34–1.

Table 34–1 Default Data Type Conversions

Analytic Workspace Data Type	SQL Data Type
ID	CHAR(8)
TEXT	VARCHAR2(4000)
TEXT*(n)*	VARCHAR2*(n)*
NTEXT	NVARCHAR2(4000)
NTEXT*(n)*	NVARCHAR2*(n)*
NUMBER	NUMBER
NUMBER*(p,s)*	NUMBER*(p,s)*
LONGINTEGER	NUMBER(19)
INTEGER	NUMBER(10)
SHORTINTEGER	NUMBER(5)
INTEGER WIDTH 1	NUMBER(3)
BOOLEAN	NUMBER(1)
DECIMAL	BINARY_DOUBLE
SHORTDECIMAL	BINARY_FLOAT
DATE	DATE
DAY, WEEK, MONTH, QUARTER, YEAR	DATE
DATETIME	TIMESTAMP

Table 34–1 (Cont.) Default Data Type Conversions

Analytic Workspace Data Type	SQL Data Type
COMPOSITE	VARCHAR2(4000)
Other	VARCHAR2(4000)

Using a MODEL Clause

You can specify a MODEL clause in a SELECT FROM OLAP_TABLE statement to significantly improve query performance. The MODEL clause causes OLAP_TABLE to use an internal optimization.

You can use the following syntax to maximize the performance advantage of the MODEL clause with OLAP_TABLE. This is the recommended syntax for views of analytic workspaces.

```
SELECT column_first, column_next, column_n
   FROM TABLE(OLAP_TABLE(
      'analytic_workspace',
      'table_obj',
      'olap_command',
      'limit_map'))
   MODEL
      DIMENSION BY(dimensions, gids)
      MEASURES(measures, attributes, rowtocell)
      RULES UPDATE SEQUENTIAL ORDER();
```

The MODEL clause must include DIMENSION BY and MEASURES subclauses that specify the columns in the table object. DIMENSION BY should list all the dimensions, as defined in the limit map. The list should include the GID columns for applications that use the OLAP API or BI Beans. MEASURES should list all the measures, attributes, ROW2CELL columns, and any other columns excluded from the DIMENSION BY list.

A MODEL clause lets you view the results of a query as a multidimensional array and specify calculations (rules) to perform on individual cells and ranges of cells within the array. You can specify calculation rules in the MODEL clause with OLAP_TABLE, but they will affect response time. If you wish to obtain the full benefit of the performance optimization, you should specify UPDATE and SEQUENTIAL ORDER in the RULES clause.

The UPDATE keyword indicates that you are not adding any custom members in the DIMENSION BY clause. If you do not include this keyword, the SQL WHERE clauses for measures will be discarded, which can significantly degrade performance.

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

See Also:

- *Oracle Database SQL Reference* and *Oracle Database Data Warehousing Guide* for more information on SQL models.
- *Oracle OLAP Application Developer's Guide* for examples of OLAP_TABLE queries that include a MODEL clause.

OLAP_TABLE Examples

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

> **See Also:** "OLAP_TABLE Syntax" on page 34-12 for complete descriptions of the syntax used in these examples.

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

> **Note:** The examples in this section use predefined ADTs. You could modify them to use automatic ADTs. See "Using OLAP_TABLE With Automatic ADTs" on page 34-3.
>
> The examples in this section do not include a MODEL clause. In general, you *should* specify a MODEL clause for performance reasons, as described in "Using a MODEL Clause" on page 34-5.

Example: Creating Views of Embedded Total Dimensions

Example 34–5 shows the PL/SQL script used to create an embedded total view of the TIME dimension in the GLOBAL analytic workspace. This view is similar to the view in Example 34–2, but it specifies both a Calendar and a Fiscal hierarchy, and it includes HATTRIBUTE subclauses for hierarchy-specific End Date attributes.

The INHIERARCHY subclause identifies a valueset in the analytic workspace that lists all the dimension members in each hierarchy of a dimension. OLAP_TABLE saves the status of all dimensions in the limit map that have INHIERARCHY subclauses during the processing of the limit map. See "Order of Processing in OLAP_TABLE" on page 34-21.

Example 34–5 Script for an Embedded Total Dimension View Using OLAP_TABLE

```
CREATE TYPE awtime_row AS OBJECT (
             awtime_id               VARCHAR2(12),
             awtime_short_label      VARCHAR2(12),
             awtime_cal_end_date     DATE,
             awtime_fis_end_date     DATE);
/
CREATE TYPE awtime_table AS TABLE OF awtime_row;
/
CREATE OR REPLACE VIEW awtime_view AS
   SELECT awtime_id, awtime_short_label,
          awtime_cal_end_date, awtime_fis_end_date
      FROM TABLE(OLAP_TABLE(
         'global DURATION SESSION',
         'awtime_table',
         '',
         'DIMENSION awtime_id FROM time WITH
             HIERARCHY time_parentrel
                (time_hierlist ''CALENDAR'')
                INHIERARCHY time_inhier
                HATTRIBUTE awtime_cal_end_date  FROM time_cal_end_date
             HIERARCHY time_parentrel
```

```
                (time_hierlist ''FISCAL'')
                INHIERARCHY time_inhier
                HATTRIBUTE awtime_fis_end_date  FROM time_fis_end_date
          ATTRIBUTE awtime_short_label  FROM time_short_description'));
/
SQL>SELECT * FROM awtime_view;

AWTIME_ID  AWTIME_SHORT_LABEL  AWTIME_CAL_END_DATE  AWTIME_FIS_END_DATE
---------  ------------------  -------------------  -------------------
19         Jan-98              31-JAN-98            31-JAN-98
20         Feb-98              28-FEB-98            28-FEB-98
21         Mar-98              31-MAR-98            31-MAR-98
22         Apr-98              30-APR-98            30-APR-98
23         May-98              31-MAY-98            31-MAY-98
24         Jun-98              30-JUN-98            30-JUN-98
.
.
.
.
98         Q1-03               31-MAR-03            30-SEP-03
99         Q2-03               30-JUN-03            31-DEC-03
1          1998                31-DEC-98            30-JUN-99
102        2003                31-DEC-03            30-JUN-04
119        2004                31-DEC-04            30-JUN-05
2          1999                31-DEC-99            30-JUN-00
3          2000                31-DEC-00            30-JUN-01
4          2001                31-DEC-01            30-JUN-02
85         2002                31-DEC-02            30-JUN-03
```

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

Example: Creating Views of Embedded Total Measures

In a star schema, a separate measure view is needed with columns that can be joined to each of the dimension views. Example 34–6 shows the PL/SQL script used to create a measure view with a column populated by a `ROW2CELL` clause to support custom measures.

See Also: "Limit Map: ROW2CELL Clause" on page 34-20 for information on `ROW2CELL`.

Example 34–6 Script for a Measure View Using OLAP_TABLE

```
CREATE TYPE awunits_row AS OBJECT (
            awtime                  VARCHAR2(12),
            awcustomer              VARCHAR2(30),
            awproduct               VARCHAR2(30),
            awchannel               VARCHAR2(30),
            awunits                 NUMBER(16),
            r2c                     RAW(32));
/
CREATE TYPE awunits_table AS TABLE OF awunits_row;
/
CREATE OR REPLACE VIEW awunits_view AS
   SELECT awunits,
          awtime, awcustomer, awproduct, awchannel, r2c
     FROM TABLE(OLAP_TABLE(
```

```
          'global DURATION SESSION',
          'awunits_table',
          '',
          'MEASURE awunits FROM units_cube_units
           DIMENSION awtime FROM time WITH
              HIERARCHY time_parentrel
           DIMENSION awcustomer FROM customer WITH
              HIERARCHY customer_parentrel
                        (customer_hierlist ''MARKET_ROLLUP'')
                 INHIERARCHY customer_inhier
           DIMENSION awproduct FROM product WITH
              HIERARCHY product_parentrel
           DIMENSION channel WITH
              HIERARCHY channel_parentrel
              ATTRIBUTE  awchannel FROM channel_short_description
           ROW2CELL r2c'))
       WHERE awunits IS NOT NULL;

SQL>SELECT awchannel, awunits FROM awunits_view
     WHERE    awproduct = '1'
     AND      awcustomer = '7'
     AND      awtime = '4';

AWCHANNEL            AWUNITS
---------            -------
All Channels         415392
Direct Sales         43783
Catalog              315737
Internet             55872
```

Example: Creating Views in Rollup Form

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

> **See Also:** "Limit Map: DIMENSION Clause: WITH HIERARCHY Subclause" on page 34-18 for information on `FAMILYREL`.

Example 34–7 shows the PL/SQL script used to create a rollup view of the `PRODUCT` dimension. It shows a dimension view to highlight the differences in the syntax of the limit map from the one used for the embedded total form, as shown in Example 34–5, "Script for an Embedded Total Dimension View Using OLAP_TABLE". Note that the target columns for these levels are listed in the `FAMILYREL` clause from most aggregate (`CLASS`) to least aggregate (`ITEM`), which is the order they are listed in the level list dimension. The family relation returns four columns. The most aggregate level (all products) is omitted from the view by mapping it to null.

Example 34–8 shows the alternate syntax for the `FAMILYREL` clause, which uses QDRs to identify exactly which columns will be mapped from the family relation.

The limit maps in Example 34–7 and Example 34–8 generate identical views.

Example 34–7 Script for a Rollup View of Products Using OLAP_TABLE

```
CREATE TYPE awproduct_row AS OBJECT (
            class       VARCHAR2(50),
            family      VARCHAR2(50),
            item        VARCHAR2(50));
/
CREATE TYPE awproduct_table AS TABLE OF awproduct_row;
```

```
/
CREATE OR REPLACE VIEW awproduct_view AS
   SELECT class, family, item
      FROM TABLE(OLAP_TABLE(
         'global DURATION QUERY',
         'awproduct_table',
         '',
         'DIMENSION product WITH
            HIERARCHY product_parentrel
               FAMILYREL null, class, family, item
                  FROM product_familyrel USING product_levellist
                  LABEL product_short_description'));

SQL> SELECT * FROM awproduct_view
             ORDER BY class, family, item;

CLASS            FAMILY             ITEM
--------------   ----------------   ------------------------
Hardware          CD-ROM             Envoy External 6X CD-ROM
Hardware          CD-ROM             Envoy External 8X CD-ROM
Hardware          CD-ROM             External 6X CD-ROM
Hardware          CD-ROM             External 8X CD-ROM
Hardware          CD-ROM             Internal 6X CD-ROM
Hardware          CD-ROM             Internal 8X CD-ROM
Hardware          CD-ROM
Hardware          Desktop PCs        Sentinel Financial
Hardware          Desktop PCs        Sentinel Multimedia
.
.
.
Software/Other   Operating Systems  Unix/Windows 1-user pack
Software/Other   Operating Systems  Unix/Windows 5-user pack
Software/Other   Operating Systems
Software/Other
```

Example 34–8 Script Using QDRs in the FAMILYREL Clause of OLAP_TABLE

```
CREATE OR REPLACE TYPE awproduct_row AS OBJECT (
            class       VARCHAR2(50),
            family      VARCHAR2(50),
            item        VARCHAR2(50));
/
CREATE TYPE awproduct_table AS TABLE OF awproduct_row;
/
CREATE OR REPLACE VIEW awproduct_view AS
   SELECT class, family, item
      FROM TABLE(OLAP_TABLE(
         'global DURATION QUERY',
         'awproduct_table',
         '',
         'DIMENSION product WITH
            HIERARCHY product_parentrel
               FAMILYREL class, family, item FROM
                  product_familyrel(product_levellist ''CLASS''),
                  product_familyrel(product_levellist ''FAMILY''),
                  product_familyrel(product_levellist ''ITEM'')
                  LABEL product_short_description'));

SQL> SELECT * FROM awproduct_view
             ORDER BY by class, family, item;
```

```
CLASS            FAMILY             ITEM
--------------   ----------------   ------------------------
Hardware          CD-ROM             Envoy External 6X CD-ROM
Hardware          CD-ROM             Envoy External 8X CD-ROM
Hardware          CD-ROM             External 6X CD-ROM
Hardware          CD-ROM             External 8X CD-ROM
Hardware          CD-ROM             Internal 6X CD-ROM
Hardware          CD-ROM             Internal 8X CD-ROM
Hardware          CD-ROM
Hardware          Desktop PCs        Sentinel Financial
Hardware          Desktop PCs        Sentinel Multimedia
.
.
.
Software/Other   Operating Systems  Unix/Windows 1-user pack
Software/Other   Operating Systems  Unix/Windows 5-user pack
Software/Other   Operating Systems
Software/Other
```

Using OLAP_TABLE with the FETCH Command

Oracle Express Server applications that are being revised for use with Oracle Database can use an OLAP DML FETCH command instead of a limit map to map workspace objects to relational columns.

The FETCH command is supplied in the third parameter of OLAP_TABLE, which specifies a single OLAP DML command. See "OLAP Command Parameter" on page 34-13.

The script shown in Example 34–9 fetches data from two variables (SALES and COST) in the GLOBAL analytic workspace, and calculates two custom measures (COST_PRIOR_PERIOD and PROFIT). This example also shows the use of OLAP_TABLE directly by an application, without creating a view.

> **Important:** The FETCH statement in Example 34–9 is formatted with indentation for readability. In reality, the entire FETCH statement must be entered on one line, without line breaks or continuation characters.

Example 34–9 Script Using FETCH with OLAP_TABLE

```
CREATE TYPE measure_row AS OBJECT (
            time                    VARCHAR2(20),
            geography               VARCHAR2(30),
            product                 VARCHAR2(30),
            channel                 VARCHAR2(30),
            sales                   NUMBER(16),
            cost                    NUMBER(16),
            cost_prior_period       NUMBER(16),
            profit                  NUMBER(16));
/
CREATE TYPE measure_table AS TABLE OF measure_row;
/
SELECT time, geography, product, channel,
       sales, cost, cost_prior_period, profit
          FROM TABLE(OLAP_TABLE(
             'xademo DURATION SESSION',
             'measure_table',
             'FETCH time, geography, product, channel, analytic_cube_f.sales,
                    analytic_cube_f.costs,
                    LAG(analytic_cube_f.costs, 1, time, LEVELREL time_member_levelrel),
```

```
                                analytic_cube_f.sales - analytic_cube_f.costs',
                      ''))
                   WHERE channel =    'STANDARD_2.TOTALCHANNEL' AND
                         product =    'L1.TOTALPROD' AND
                         geography =  'L1.WORLD'
                   ORDER BY time;
```

This SQL `SELECT` statement returns the following result set:

```
TIME      GEOGRAPHY PRODUCT      CHANNEL                SALES       COST    COST_PRIOR_PERIOD    PROFIT
--------- --------- ------------ ---------------------- --------- --------- ------------------ --------
L1.1996   L1.WORLD  L1.TOTALPROD STANDARD_2.TOTALCHANNEL 118247112   2490243                    115756869
L1.1997   L1.WORLD  L1.TOTALPROD STANDARD_2.TOTALCHANNEL  46412113   1078031          2490243   45334082
L2.Q1.96  L1.WORLD  L1.TOTALPROD STANDARD_2.TOTALCHANNEL  26084848    560379                     25524469
L2.Q1.97  L1.WORLD  L1.TOTALPROD STANDARD_2.TOTALCHANNEL  26501765    615399           560379    25886367
L2.Q2.96  L1.WORLD  L1.TOTALPROD STANDARD_2.TOTALCHANNEL  30468054    649004           615399    29819049
L2.Q2.97  L1.WORLD  L1.TOTALPROD STANDARD_2.TOTALCHANNEL  19910347    462632           649004    19447715
L2.Q3.96  L1.WORLD  L1.TOTALPROD STANDARD_2.TOTALCHANNEL  27781702    582693           462632    27199009
L2.Q4.96  L1.WORLD  L1.TOTALPROD STANDARD_2.TOTALCHANNEL  33912508    698166           582693    33214342
L3.APR96  L1.WORLD  L1.TOTALPROD STANDARD_2.TOTALCHANNEL   8859808    188851                      8670957
                                                 .
                                                 .
                                                 .
27 rows selected.
```

OLAP_TABLE Syntax

The OLAP_TABLE function returns multidimensional data in an analytic workspace as a logical table.

The order in which OLAP_TABLE processes information specified in its input parameters is described in "Order of Processing in OLAP_TABLE" on page 34-21.

Syntax

```
OLAP_TABLE(
        analytic_workspace    IN   VARCHAR2,
        table_object          IN   VARCHAR2,
        olap_command          IN   VARCHAR2,
        limit_map1            IN   VARCHAR2,
        limit_map2            IN   VARCHAR2,
              .
              .
              .
        limit_map8            IN   VARCHAR2)
     RETURN TYPE;
```

Parameters

Table 34–2 OLAP_TABLE Function Parameters

Parameter	Description
analytic_workspace	The name of the analytic workspace with the source data. This parameter also specifies how to attach the workspace to your session. See "Analytic Workspace Parameter" on page 34-12.
table_object	The name of a table of objects that has been defined to structure the multidimensional data in tabular form. See "Table Object Parameter" on page 34-13.
olap_command	An optional OLAP DML command. See "OLAP Command Parameter" on page 34-13.
limit_map1...8	A keyword-based map that identifies the source objects in the analytic workspace and the target columns in a table of objects. You can define up to eight limit maps in order to circumvent the 4000 byte VARCHAR2 limit. The limit maps are concatenated. Be sure to include a space character if needed between the strings. See "Limit Map Parameter" on page 34-15.

Returns

A table type whose rows are objects (ADTs) that identify the selected workspace data. See "Logical Tables" on page 34-2.

Analytic Workspace Parameter

The first parameter of the OLAP_TABLE function provides the name of the analytic workspace where the source data is stored. It also specifies how long the analytic workspace will be attached to your OLAP session, which opens on your first call to OLAP_TABLE.

This parameter is always required by OLAP_TABLE.

The syntax of this parameter is:

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

For example:

```
'olapuser.xademo DURATION SESSION'
```

owner

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

QUERY

Attaches an analytic workspace for the duration of a single query. Use `QUERY` only when you need to see updates to the analytic workspace made in other sessions.

SESSION

`SESSION` attaches an analytic workspace and keeps it attached at the end of the query. It provides better performance than `QUERY` because it keeps the OLAP session open. This performance difference is significant when the function is called without either a *table_object* parameter or `AS` clauses in the limit map; in this case, the `OLAP_TABLE` function must determine the appropriate table definition. See "Using OLAP_TABLE With Automatic ADTs" on page 34-3.

Table Object Parameter

The second parameter identifies the name of a predefined table of objects, as described in "Using OLAP_TABLE With Predefined ADTs" on page 34-2.

This parameter is optional. Omit this parameter if you are using automatic ADTs.

The syntax of this parameter is:

```
'table_name'
```

For example:

```
'product_dim_tbl'
```

When you specify the *table_name* parameter, the column data types for the returned data are predefined. In this case you cannot use `AS` clauses in the limit map.

When you omit the *table_name* parameter, the column data types for the returned data are generated at runtime. You can either provide the target data types with `AS` clauses in the limit map, or you can use the default data types described in Table 34–1, " Default Data Type Conversions". See "Using OLAP_TABLE With Automatic ADTs" on page 34-3.

OLAP Command Parameter

The third parameter of the `OLAP_TABLE` function is a single OLAP DML command. If you want to execute more than one command, then you must create a program in your analytic workspace and call the program in this parameter. The power and flexibility of this parameter comes from its ability to process virtually any data manipulation commands available in the OLAP DML.

The order in which `OLAP_TABLE` processes the *olap_command* parameter is specified in "Order of Processing in OLAP_TABLE" on page 34-21.

The syntax of this parameter is:

```
'olap_command'
```

There are two distinct ways of using the *olap_command* parameter:

- To make changes in the workspace session immediately before the data is fetched (after all the limits have been applied)
- To specify the source data directly instead of using a limit map

Both methods are described in the following sections.

Using olap_command with a Limit Map

You may want your application to modify the analytic workspace on the fly during the execution of OLAP_TABLE.

A common use of the *olap_command* parameter is to limit one or more dimensions. If you limit any of the dimensions that have INHIERARCHY clauses in the limit map, then the status of those dimensions is changed only during execution of this call to OLAP_TABLE; the limits do not affect the rest of your OLAP session. However, other commands (for example, commands that limit dimensions *not* referenced with INHIERARCHY clauses) can affect your session.

If you want a limit on a dimension in the limit map to stay in effect for the rest of your session, and not just during the command, specify it in the PREDMLCMD clause of the limit map or specify an OLAP_CONDITION function in the SQL SELECT statement.

The following is an example of a LIMIT command in the *olap_command* parameter.

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

> **See Also:** Chapter 29, "OLAP_CONDITION".

Using FETCH in the olap_command Parameter

If you specify an OLAP DML FETCH command in the *olap_command* parameter, OLAP_TABLE uses it, instead of the instructions in the limit map, to fetch the source data for the table object. Because of this usage, the *olap_command* parameter is sometimes referred to as the **data map**. In general, you should not specify a limit map if you specify a FETCH command.

> **Note:** Normally, you should only use the FETCH command with OLAP_TABLE if you are upgrading an Express application that used the FETCH command for SNAPI. If you are upgrading, note that the full syntax is the same in Oracle as in Express 6.3. You can use the same FETCH commands in OLAP_TABLE that you used previously in SNAPI. The syntax of the FETCH command is documented in the *Oracle OLAP DML Reference*

FETCH specifies explicitly how analytic workspace data is mapped to a table object. The basic syntax is:

```
FETCH expression...
```

Enter one expression for each target column, listing the expressions in the same order they appear in the row definition. Separate expressions with spaces or commas.You must enter the entire statement on one line, without line breaks or continuation marks of any type.

See Also: "Using OLAP_TABLE with the FETCH Command" on page 34-10.

Limit Map Parameter

The fourth (and last) parameter of the OLAP_TABLE function maps workspace objects to relational columns and identifies the role of each one. See "Limit Maps" on page 34-1.

The limit map can also specify special instructions to be executed by OLAP_TABLE. For example: It can cause an OLAP DML command to execute before or after the limit map is processed; it can specify a ROW2CELL column for the OLAP_CONDITION and OLAP_EXPRESSION functions. (See Chapter 29 and Chapter 30.)

The order in which OLAP_TABLE processes information in the limit map is specified in "Order of Processing in OLAP_TABLE" on page 34-21.

The limit map parameter is generally a required parameter. It can only be omitted when you specify a FETCH command in the *olap_command* parameter. See "OLAP Command Parameter" on page 34-13.

You can supply the entire text of the limit map as a parameter to OLAP_TABLE, or you can store all or part of the limit map in a text variable in the analytic workspace and reference it using ampersand substitution. For example, the following OLAP_TABLE query uses a limit map stored in a variable called limitmapvar in the GLOBAL analytic workspace of the GLOBAL_AW schema.

```
SELECT * FROM TABLE(OLAP_TABLE(
          'global_aw.global DURATION SESSION',
          '',
          '',
          '&(global_aw.global!limitmapvar)');
```

If you supply the limit map as text within the call to OLAP_TABLE, then it has a maximum length of 4000 characters, which is imposed by PL/SQL. If you store the limit map in the analytic workspace, then the limit map has no maximum length.

The syntax of the limit map has numerous clauses, primarily for defining dimension hierarchies. Pay close attention to the presence or absence of commas, since syntax errors will prevent your limit map from being parsed. The syntax of the limit map is summarized in Example 34–10. Individual syntax components are described in the following sections.

Note: Several objects must be predefined within the workspace to support the mapping of dimension hierarchies in the limit map. These objects are already defined in standard form workspaces. If the workspace does not conform to standard form, you may need to prepare the workspace by defining objects such as:

- a **parent relation**, which identifies the parent of each dimension member within a hierarchy.
- a **hierarchy dimension**, which lists the hierarchies of a dimension.
- an **inhierarchy** variable or valueset, which specifies which dimension members belong to each level of a hierarchy.
- a **grouping ID** variable, which identifies the depth within a hierarchy of each dimension member.
- a **family relation**, which provides the full parentage of each dimension member in a hierarchy.
- a **level dimension**, which lists the levels of a dimension.

Instructions for creating these workspace objects are provided in the *Oracle OLAP Application Developer's Guide*.

Example 34–10 Syntax of an OLAP_TABLE Limit Map

```
'[MEASURE column [AS datatype] FROM {measure | AW_EXPR expression}]
          .
          .
 DIMENSION [column [AS datatype] FROM] dimension
    [WITH
       [HIERARCHY [column [AS datatype] FROM] parent_relation
          [(hierarchy_dimension ''hierarchy_name'')]
          [INHIERARCHY inhierarchy_obj]
          [GID column [AS datatype] FROM gid_variable]
          [PARENTGID column [AS datatype] FROM gid_variable]
          [FAMILYREL column1 [AS datatype],
                     column2 [AS datatype],
                      ... columnn [AS datatype]
                     FROM {expression1, expression2, ... expressionn |
                           family_relation USING level_dimension }
                     [LABEL label_variable]]
          [HATTRIBUTE column [AS datatype] FROM hier_attribute_variable]
           .
           .
       ]
       [ATTRIBUTE column [AS datatype] FROM attribute_variable]
           .
           .
    ]
 [ROW2CELL column]
 [LOOP composite_dimension]
 [PREDMLCMD olap_command]
 [POSTDMLCMD olap_command]'
```

Where:

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

datatype is the data type of *column*.

measure is a measure in the analytic workspace.

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

dimension is a dimension in the analytic workspace.

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

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

hierarchy_name is a member of *hierarchy_dimension*.

inhierarchy_obj is a variable or valueset in the analytic workspace that identifies which dimension members are in each level of the hierarchy.

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

family_relation is a self-relation that provides the full parentage of each dimension member in the hierarchy.

level_dimension is a dimension in the analytic workspace that contains the names of the levels for the hierarchy.

label_variable is a variable in the analytic workspace that contains descriptive text values for *dimension*.

hier_attribute_variable is a variable in the analytic workspace that contains attribute values for *hierarchy_name*.

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

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

olap_command is an OLAP DML command.

Limit Map: MEASURE Clause

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

```
MEASURE column [AS datatype] FROM {measure | AW_EXPR expression}
```

The AS subclause specifies the data type of the target column. You can specify an AS subclause when the table of objects has not been predefined. See "Using OLAP_TABLE With Automatic ADTs" on page 34-3.

In the FROM subclause, you can either specify the name of a workspace measure or an OLAP expression that evaluates to a measure. For example:

```
AW_EXPR analytic_cube_sales - analytic_cube_cost
or
AW_EXPR LOGDIF(analytic_cube_sales, 1, time, LEVELREL time.lvlrel)
```

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

Limit Map: DIMENSION Clause

The DIMENSION clause identifies a dimension or conjoint in the analytic workspace that dimensions one or more measures or attributes, or provides the dimension members for one or more hierarchies in the limit map.

```
DIMENSION [column [AS datatype] FROM] dimension ....
```

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

For a description of the AS subclause, see "Limit Map: MEASURE Clause" on page 34-17.

Every limit map should have at least one DIMENSION clause. If the limit map contains MEASURE clauses, then it should also contain a single DIMENSION clause for each dimension of the measures, unless a dimension is being limited to a single value. If the measures are dimensioned by a composite, then you must identify each dimension in the composite with a DIMENSION clause. For the best performance when fetching a large result set, identify the composite in a LOOP clause. See "Limit Map: LOOP Clause" on page 34-20.

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

Limit Map: WITH Subclause for Dimension Hierarchies and Attributes

The WITH subclause introduces a HIERARCHY or ATTRIBUTE subclause. If you do not specify hierarchies or attributes, then omit the WITH keyword. If you specify both hierarchies and attributes, then precede them with a single WITH keyword. The syntax of the WITH clause is included in Example 34–10, "Syntax of an OLAP_TABLE Limit Map". It is shown without the rest of the limit map syntax in Example 34–11.

Example 34–11 WITH Subclause of Limit Map DIMENSION Clause

```
[WITH
   [HIERARCHY [column [AS datatype] FROM] parent_relation
       [(hierarchy_dimension ''hierarchy_name'')]
       [INHIERARCHY inhierarchy_obj]
       [GID column [AS datatype] FROM gid_variable]
       [PARENTGID column [AS datatype] FROM gid_variable]
       [FAMILYREL column1 [AS datatype],
                  column2 [AS datatype],
                  ... columnn [AS datatype]
                  FROM {expression1, expression2,... expressionn |
                        family_relation USING level_dimension}
                  [LABEL label_variable]]
         [HATTRIBUTE column [AS datatype] FROM hier_attribute_variable]
   ...
   ]
   [ATTRIBUTE column [AS datatype] FROM attribute_variable]
   ...
]
```

Limit Map: DIMENSION Clause: WITH HIERARCHY Subclause

The HIERARCHY subclause identifies the parent self-relation in the analytic workspace that defines the hierarchies for the dimension.

```
HIERARCHY [column [AS datatype] FROM] parent_relation
          [(hierarchy_dimension ''hierarchy_name'')]...
```

For a description of the column subclause, see "Limit Map: DIMENSION Clause" on page 34-17.

If the dimension has more than one hierarchy, specify a *hierarchy_dimension* phrase. *hierarchy_dimension* identifies a dimension in the analytic workspace which holds the names of the hierarchies for this dimension. *hierarchy_name* is a member of *hierarchy_*

dimension. The hierarchy dimension is limited to *hierarchy_name* for all workspace objects that are referenced in subsequent subclauses for this hierarchy (that is, INHIERARCHY, GID, PARENTGID, FAMILYREL, and HATTRIBUTE).

To include multiple hierarchies for the dimension, specify a HIERARCHY subclause for each one.

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

The keywords in the HIERARCHY subclause are described as follows:

- INHIERARCHY *inhierarchy_obj*

  The INHIERARCHY subclause identifies a boolean variable or a valueset in the analytic workspace that identifies the dimension members in each level of the hierarchy. It is required when there are members of the dimension that are omitted from the hierarchy. It is good practice to include an INHIERARCHY subclause, because OLAP_TABLE saves the status of all dimensions with INHIERARCHY subclauses during the processing of the limit map.

- GID *column* [AS *datatype*] FROM *gid_variable*

  The GID subclause maps an integer variable in the analytic workspace, which contains the grouping ID for each dimension member, to a column in the target table. The grouping ID variable is populated by the OLAP DML GROUPINGID command.

  For a description of the AS subclause, see "Limit Map: MEASURE Clause" on page 34-17.

  The GID subclause is required for Java applications that use the OLAP API.

- PARENTGID *column* [AS *datatype*] FROM *gid_variable*

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

  For a description of the AS subclause, see "Limit Map: MEASURE Clause" on page 34-17.

  The PARENTGID clause is recommended for Java applications that use the OLAP API.

- FAMILYREL *column1* [AS *datatype*], *column2* [AS *datatype*],
  ... *column*n [AS *datatype*]
  FROM {*expression1*, *expression2*, ... *expression*n |
  family_relation USING *level_dimension* }
  [LABEL *label_variable*]

  The FAMILYREL subclause is used primarily to map a family relation in the analytic workspace to multiple columns in the target table. List the columns in the order of *level_dimension* (a dimension in the analytic workspace that holds the names of all the levels for the dimension). If you do not want a particular level included, then specify null for the target column. For a description of the AS subclause, see "Limit Map: MEASURE Clause" on page 34-17.

  The tabular data resulting from a FAMILYREL clause is in **rollup form**, in which each level of the hierarchy is represented in a separate column, and the full parentage of each dimension member is identified within the row. See "Example: Creating Views in Rollup Form" on page 34-8.

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

You can use multiple FAMILYREL clauses for each hierarchy.

- HATTRIBUTE *column* [AS *datatype*] FROM *hier_attribute_variable*

  The HATTRIBUTE subclause maps a hierarchy-specific attribute variable, dimensioned by *hierarchy_dimension* in the analytic workspace, to a column in the target table.

Limit Map: DIMENSION Clause: WITH ATTRIBUTE Subclause

The ATTRIBUTE subclause maps an attribute variable in the analytic workspace to a column in the target table.

```
ATTRIBUTE column [AS datatype] FROM attribute_variable
```

If *attribute_variable* has multiple dimensions, then values are mapped for all members of *dimension*, but only for the first member in the current status of additional dimensions. For example, if your attributes have a language dimension, then you must set the status of that dimension to a particular language. You can set the status of dimensions in a PREDMLCMD clause. See "Limit Map: PREDMLCMD Clause" on page 34-20.

Limit Map: ROW2CELL Clause

The ROW2CELL clause creates a RAW column, between 16 and 32 characters wide, in the target table and populates it with information that is used by the OLAP_EXPRESSION functions. The OLAP_CONDITION function also uses the ROW2CELL column. Specify a ROW2CELL column when creating a view that will be used by these functions. See Chapter 29 and Chapter 30.

```
ROW2CELL column
```

Limit Map: LOOP Clause

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

```
LOOP sparse_dimension
```

Limit Map: PREDMLCMD Clause

The PREDMLCMD clause specifies an OLAP DML command that is executed before the data is fetched from the analytic workspace into the target table. It can be used, for example, to execute an OLAP model or forecast whose results will be fetched into the table. The results of the command are in effect during execution of the limit map, and continue into your session after execution of OLAP_TABLE is complete. See "Order of Processing in OLAP_TABLE" on page 34-21.

```
PREDMLCMD olap_command
```

Limit Map: POSTDMLCMD Clause

The POSTDMLCMD clauses specifies an OLAP DML command that is executed after the data is fetched from the analytic workspace into the target table. It can be used, for example, to delete objects or data that were created by commands in the PREDMLCMD clause, or to restore the dimension status that was changed in a PREDMLCMD clause. See "Order of Processing in OLAP_TABLE" on page 34-21.

```
POSTDMLCMD olap_command
```

Order of Processing in OLAP_TABLE

The following list identifies the order in which the OLAP_TABLE function processes instructions in the limit map that can change the status of dimensions in the analytic workspace.

1. Execute any OLAP DML command specified in the PREDMLCMD parameter of the limit map.
2. Save the current status of all dimensions in the limit map so that it can be restored later (PUSH status).
3. Keep in status only those dimension members specified by INHIERARCHY subclauses in the limit map (LIMIT KEEP).
4. Within the status set during step 3, keep only those dimension members that satisfy the WHERE clause of the SQL SELECT statement containing the OLAP_TABLE function (LIMIT KEEP).
5. Execute any OLAP DML command specified in the *olap_command* parameter of the OLAP_TABLE function. (If *olap_command* includes a FETCH, fetch the data.)
6. Fetch the data (unless a FETCH command was specified in the *olap_command* parameter).
7. Restore the status of all dimensions in the limit map (POP status).
8. Execute any OLAP DML command specified in the POSTDMLCMD parameter of the limit map.

Index

A

C

D

E

F

G

H

I

J

L

R

S

T

U

V

W

X

Y