Oracle® OLAP

Reference

10*g* Release 2 (10.2)

B14350-03

October 2007

ORACLE®

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

B14350-03

Contents

3 DBMS_AW

4 DBMS_AW_XML

5 OLAP_API_SESSION_INIT

6 OLAP_CONDITION

7 OLAP_EXPRESSION

8 OLAP_EXPRESSION_BOOL

9 OLAP_EXPRESSION_DATE

10 OLAP_EXPRESSION_TEXT

11 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, 7 days a week. For TTY support, call 800.446.2398. Outside the United States, call +1.407.458.2479.

Related Documents

For more information 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.

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

Active Catalog Views

This chapter describes the relational views of standard form metadata in analytic workspaces.

This chapter discusses the following topics:

- Understanding the Active Catalog
- Summary of Active Catalog Views

Understanding the Active Catalog

OLAP processing depends on a data model composed of cubes, measures, dimensions, hierarchies, levels, and attributes. Standard form metadata defines the logical model and is stored in the analytic workspaces. Views of this metadata are commonly referred to as the **Active Catalog**, because they are populated automatically and always reflect the current state of the analytic workspace.

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

The Active Catalog views are named with the `ALL_OLAP2_AW` prefix.

Summary of Active Catalog Views

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

Table 1–1 Active Catalog Views

PUBLIC Synonym	Description
ALL_OLAP2_AW_ATTRIBUTES	List of dimension attributes in analytic workspaces
ALL_OLAP2_AW_CATALOG_MEASURES	Lists the measures in the measure folders
ALL_OLAP2_AW_CATALOGS	Lists the measure folders 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

Table 1–1 (Cont.) Active Catalog Views

PUBLIC Synonym	Description
ALL_OLAP2_AW_CUBES	List of cubes 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_DIMENSIONS	List of dimensions 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	Lists the 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 Oracle 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.
SF_VERSION	VARCHAR2(8)		The version of the Oracle database in which the standard form metadata was created

ALL_OLAP2_AW_ATTRIBUTES

ALL_OLAP2_AW_ATTRIBUTES lists the attributes 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_DIMENSION_NAME	VARCHAR2(4000)		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(4000)		Standard form name for the attribute in the analytic workspace
DISPLAY_NAME	VARCHAR2(4000)		Display name for the attribute
DESCRIPTION	VARCHAR2(4000)		Description of the attribute
ATTRIBUTE_TYPE	VARCHAR2(4000)		Type of attribute
SOURCE_OWNER	VARCHAR2(4000)		Owner of the source attribute in the OLAP Catalog (Oracle9*i* metadata)

Column	Datatype	NULL	Description
SOURCE_DIMENSION_NAME	VARCHAR2(4000)		Name of the source dimension in the OLAP Catalog (Oracle9*i* metadata)
SOURCE_NAME	VARCHAR2(4000)		Name of the source attribute in the OLAP Catalog (Oracle9*i* metadata)

ALL_OLAP2_AW_CATALOGS

ALL_OLAP2_AW_CATALOGS lists the measure folders 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
CATALOG_ID	NUMBER		Unique identifier for the measure folder
CATALOG_NAME	VARCHAR2(4000)		Name of the measure folder
PARENT_CATALOG_NAME	VARCHAR2(4000)		Name of the parent folder when CATALOG_NAME is a subfolder
DESCRIPTION	VARCHAR2(4000)		Description of the measure folder

ALL_OLAP2_AW_CATALOG_MEASURES

ALL_OLAP2_AW_CATALOG_MEASURES lists the measures in the measure folders.

Column	Datatype	NULL	Description
AW_OWNER	VARCHAR2(30)		Owner of the analytic workspace
AW_NAME	VARCHAR2(30)		Name of the analytic workspace
CATALOG_ID	NUMBER		Unique identifier for the measure folder
CATALOG_NAME	VARCHAR2(4000)		Name of the measure folder
ENTITY_OWNER	VARCHAR2(4000)		Owner of the cube
ENTITY_NAME	VARCHAR2(4000)		Name of the cube with the measure
CHILD_ENTITY_NAME	VARCHAR2(4000)		Name of the measure included in the folder

ALL_OLAP2_AW_CUBES

ALL_OLAP2_AW_CUBES lists the 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)		Logical name for the cube in the analytic workspace
AW_PHYSICAL_OBJECT	VARCHAR2(4000)		Standard form name for the cube in the analytic workspace

Column	Datatype	NULL	Description
SOURCE_OWNER	VARCHAR2(4000)		Owner of the source cube in the OLAP Catalog (Oracle9*i* metadata)
SOURCE_NAME	VARCHAR2(4000)		Name of the source cube in the OLAP Catalog (Oracle9*i* metadata)

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(4000)		Name of an aggregation specification for the cube
AW_DIMENSION_NAME	VARCHAR2(4000)		Name of a workspace dimension of the cube
AW_LEVEL_NAME	VARCHAR2(4000)		Name of a level of the dimension, which 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(4000)		Name of an aggregation specification for the cube
AW_MEASURE_NAME	VARCHAR2(4000)		Name of a measure of the cube, which 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

Column	Datatype	NULL	Description
AW_MEASURE_NAME	VARCHAR2		Name of a workspace measure to aggregate
AW_AGGSPEC_NAME	VARCHAR2(4000)		Name of an aggregation specification for the cube
AW_DIMENSION_NAME	VARCHAR2(4000)		Name of a workspace dimension of the cube
OPERATOR	VARCHAR2(4000)		Operator for aggregation along this dimension

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
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(4000)		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(4000)		Owner of a workspace dimension of the cube
DIMENSION_AW_NAME	VARCHAR2(4000)		Name of a workspace dimension of the cube
DIMENSION_SOURCE_OWNER	VARCHAR2(4000)		Owner of the source dimension in the OLAP Catalog (Oracle9*i* metadata)
DIMENSION_SOURCE_NAME	VARCHAR2(4000)		Name of the source dimension in the OLAP Catalog (Oracle9*i* metadata)

ALL_OLAP2_AW_CUBE_MEASURES

ALL_OLAP2_AW_CUBE_MEASURES lists the measures 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_CUBE_NAME	VARCHAR2(90)		Name of a cube in the analytic workspace
AW_MEASURE_NAME	VARCHAR2(4000)		Logical name of a measure of the cube
AW_PHYSICAL_OBJECT	VARCHAR2(4000)		Standard form name of the measure

Column	Datatype	NULL	Description
MEASURE_SOURCE_NAME	VARCHAR2(4000)		Name of the source measure in the OLAP Catalog (Oracle9*i* metadata)
DISPLAY_NAME	VARCHAR2(4000)		Display name for the measure in the analytic workspace
DESCRIPTION	VARCHAR2(4000)		Description of the measure in the analytic workspace
IS_AGGREGATEABLE	VARCHAR2(4000)		Whether or not this measure can be aggregated 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.

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(4000)		Standard form name of the dimension in the analytic workspace
SOURCE_OWNER	VARCHAR2(4000)		Owner of the source dimension in the OLAP Catalog (Oracle9*i* metadata)
SOURCE_NAME	VARCHAR2(4000)		Name of the source dimension in the OLAP Catalog (Oracle9*i* metadata)

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(4000)		Name of a hierarchy of the workspace dimension
IS_DEFAULT_HIER	VARCHAR2(4000)		Whether or not this hierarchy is the default hierarchy
AW_LEVEL_NAME	VARCHAR2(4000)		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(4000)		Name of a workspace level of the dimension
DISPLAY_NAME	VARCHAR2(4000)		Display name of the level
DESCRIPTION	VARCHAR2(4000)		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 within an analytic workspace, 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 analytic workspaces, 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(4000)		Name of a property of the standard form object
AW_PROP_VALUE	VARCHAR2(4000)		Value of the property

2

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 query lists the OLAP system tables.

```
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 2–1 briefly describes each OLAP dynamic performance view.

Table 2–1 OLAP Fixed Views

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

V$AW_AGGREGATE_OP

V$AW_AGGREGATE_OP lists the aggregation operators available in analytic workspaces. 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 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 analytic workspaces. 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 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, 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 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.

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 analytic workspaces.

A cursor can be initiated within an analytic workspace 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

3

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 to the Latest Storage Format

To convert an Oracle9*i* or an Oracle Database 10*g* Release 1 analytic workspace to Oracle 10*g* Release 2 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 SQL*Plus, convert the workspace to the current format:

```
SQL> EXECUTE dbms_aw.convert ('my_aw');
```

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 Oracle9*i* SQL*Plus, export the analytic workspace to the directory identified by the work_dir directory object.

```
SQL> EXECUTE dbms_aw.execute ('AW ATTACH ''awname''');
SQL> EXECUTE dbms_aw.execute ('ALLSTAT');
SQL> EXECUTE dbms_aw.execute ('CDA work_dir');
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 WORK_DIR directory.

```
SQL> EXECUTE dbms_aw.execute ('AW CREATE awname');
SQL> EXECUTE dbms_aw.execute ('CDA work_dir');
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 will 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 7, "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 3–1 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 3–1 Sparsity Advisor Script for GLOBAL

```
SET ECHO ON
SET LINESIZE 300
SET PAGESIZE 300
SET SERVEROUT ON FORMAT WRAPPED

-- Define and initialize an advice table named AW_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 3–1 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 AW_SPARSITY_ADVICE Table

This SQL `SELECT` statement shows some of the columns from the `AW_SPARSITY_ADVICE` table, which is the basis for the recommended OLAP DML object definitions.

```
SQL> SELECT fact, dimension, dimcolumn, membercount nmem, leafcount nleaf,
     advice, density
     FROM aw_sparsity_advice
     WHERE cubename='units_cube';
```

This query returns the following result set:

```
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 3-25
- ADVISE_CUBE Procedure on page 3-16

Example 3–2 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('REPORT W 40 customer_hierlist');
CUSTOMER_HIERLIST
----------------------------------------
MARKET_ROLLUP
SHIPMENTS_ROLLUP

---- Levels in Customer dimension
SQL> EXECUTE dbms_aw.execute('REPORT W 40 customer_levellist');

CUSTOMER_LEVELLIST
----------------------------------------
TOTAL_CUSTOMER
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('REPORT 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 3–2 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 3-8.

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	TOTAL_CUSTOMER	SHIPMENTS_ROLLUP	TOTAL_CUSTOMER
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 3–1 DBMS_AW Subprograms

Subprogram	Description
ADD_DIMENSION_SOURCE Procedure on page 3-14	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 3-16	Suggests how to preaggregate a cube, based on a specified percentage of the cube's data.
ADVISE_DIMENSIONALITY Function on page 3-18	Returns a recommended composite definition for the cube and a recommended dimension order.
ADVISE_DIMENSIONALITY Procedure on page 3-20	Generates the OLAP DML commands for defining the recommended composite and measures in a cube.
ADVISE_PARTITIONING_DIMENSION Function on page 3-22	Identifies the dimension that the Sparsity Advisor partitioned over.
ADVISE_PARTITIONING_LEVEL Function on page 3-23	Returns the level used by the Sparsity Advisor for partitioning over a dimension.
ADVISE_REL Procedure on page 3-25	Suggests how to preaggregate a dimension, based on a specified percentage of the dimension's members.
ADVISE_SPARSITY Procedure on page 3-26	Analyzes a fact table for sparsity and populates a table with the results of its analysis.
AW_ATTACH Procedure on page 3-29	Attaches an analytic workspace to a session.
AW_COPY Procedure on page 3-31	Creates a new analytic workspace and populates it with the object definitions and data from another analytic workspace.
AW_CREATE Procedure on page 3-32	Creates a new, empty analytic workspace.
AW_DELETE on page 3-33	Deletes an analytic workspace
AW_DETACH Procedure on page 3-34	Detaches an analytic workspace from a session.
AW_RENAME Procedure on page 3-35	Changes the name of an analytic workspace.
AW_TABLESPACE Function on page 3-36	Returns the name of the tablespace in which a particular analytic workspace is stored.
AW_UPDATE Procedure on page 3-37	Saves changes made to an analytic workspace.
CONVERT Procedure on page 3-38	Converts an analytic workspace from 9*i* to 10*g* storage format.
EVAL_NUMBER Function on page 3-39	Returns the result of a numeric expression in an analytic workspace.
EVAL_TEXT Function on page 3-40	Returns the result of a text expression in an analytic workspace.
EXECUTE Procedure on page 3-41	Executes one or more OLAP DML commands. Input and output is limited to 4K. Typically used in an interactive session using an analytic workspace.

Table 3–1 (Cont.) DBMS_AW Subprograms

Subprogram	Description
GETLOG Function on page 3-43	Returns the session log from the last execution of the INTERP or INTERPCLOB functions.
INFILE Procedure on page 3-44	Executes the OLAP DML commands specified in a file.
INTERP Function on page 3-45	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 3-46	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 3-47	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 3-48	Returns a boolean indicating whether or not the OLAP option is installed in the database.
OLAP_RUNNING Function on page 3-49	Returns a boolean indicating whether or not the OLAP option has been initialized in the current session.
PRINTLOG Procedure on page 3-50	Prints a session log returned by the INTERP, INTERCLOB, or GETLOG functions.
RUN Procedure on page 3-51	Executes one or more OLAP DML commands.
SHUTDOWN Procedure on page 3-53	Shuts down the current OLAP session.
SPARSITY_ADVICE_TABLE Procedure on page 3-54	Creates a table which the ADVISE_SPARSITY procedure will use to store the results of its analysis.
STARTUP Procedure on page 3-55	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 3–2 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 3–2 (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 3-4.

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 3–3 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.

```
SQL> 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 3-8.

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 3–4 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.

Return Values

OLAP DML commands for creating a cube.

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 3-4.

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 3–5 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 3-4.

ADVISE_PARTITIONING_DIMENSION Function

The ADVISE_PARTITIONING_DIMENSION function identifies the dimension that the Sparsity Advisor partitioned over, if any. It returns NULL when the Sparsity Advisor did not partition the cube.

Syntax

```
ADVISE_PARTITIONING_DIMENSION (
     cubename            IN      VARCHAR2,
     sources             IN      dbms_aw$_dimension_sources_t,
     advtable            IN      VARCHAR2 DEFAULT NULL )
RETURN VARCHAR2;
```

Parameters

Table 3–6 ADVISE_PARTITIONING_DIMENSION Function Parameters

Parameter	Description
cubename	The same *cubename* value provided in the call to ADVISE_SPARSITY.
sources	The name of an object (such as a PL/SQL variable) defined with a data type of DBMS_AW$_DIMENSION_SOURCES_T, which was populated by ADD_DIMENSION_SOURCE for use by ADVISE_SPARSITY.
advtable	The name of a table created by the SPARSITY_ADVICE_TABLE procedure for storing the results of analysis.

Return Values

The name of the partitioning dimension of the cube.

Example

The following program fragment shows the ADVISE_PARTITIONING_DIMENSION function being used to query the results after using the Sparsity Advisor.

```
DECLARE
     dimsources dbms_aw$_dimension_sources_t;
BEGIN
-- Calls to ADD_DIMENSION_SOURCE and ADVISE_SPARSITY omitted here
     .
     .
     .
dbms_output.put_line('Partitioning Dimension: ' ||
     dbms_aw.advise_partitioning_dimension('units_cube', dimsources,
    'aw_sparsity_advice'));
END;
/
```

The program uses DBMS_OUTPUT to display the partitioning dimension, which in this case is the TIME dimension.

```
Partitioning Dimension: time
```

See Also

"Using the Sparsity Advisor" on page 3-4.

ADVISE_PARTITIONING_LEVEL Function

The ADVISE_PARTITIONING_LEVEL function returns the level used by the Sparsity Advisor for partitioning over a dimension. It returns NULL if the Sparsity Advisor did not partition the cube, and raises an exception if the dimension hierarchy is not level-based.

Syntax

```
ADVISE_PARTITIONING_LEVEL (
     cubename            IN      VARCHAR2,
     sources             IN      dbms_aw$_dimension_sources_t,
     advtable            IN      VARCHAR2 DEFAULT NULL)
RETURN VARCHAR2;
```

Parameters

Table 3–7 ADVISE_PARTITIONING_LEVEL Function Parameters

Parameter	Description
cubename	The same *cubename* value provided in the call to ADVISE_SPARSITY.
sources	The name of an object (such as a PL/SQL variable) defined with a data type of DBMS_AW$_DIMENSION_SOURCES_T, which was populated by ADD_DIMENSION_SOURCE for use by ADVISE_SPARSITY.
advtable	The name of a table created by the SPARSITY_ADVICE_TABLE procedure for storing the results of analysis.

Return Values

The name of the partitioning level.

Example

The following program fragment shows the ADVISE_PARTITIONING_LEVEL function being used to query the results after using the Sparsity Advisor.

```
DECLARE
     dimsources dbms_aw$_dimension_sources_t;
BEGIN
-- Calls to ADD_DIMENSION_SOURCE and ADVISE_SPARSITY omitted here
      .
      .
      .
dbms_output.put_line('Partitioning Level: ' ||
     dbms_aw.advise_partitioning_level('units_cube', dimsources,
    'aw_sparsity_advice'));
END;
/
```

The program uses DBMS_OUTPUT to display the partitioning level, which in this case is YEAR.

```
Partitioning Level: year
```

See Also

"Using the Sparsity Advisor" on page 3-4.

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 3–8 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 3-8.

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 3–9 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
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 3–10 describes the information generated by ADVISE_SPARSITY.

Table 3–10 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).

```
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 3-4.

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 3–11 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-only mode.

```
SQL> EXECUTE dbms_aw.aw_attach('global');
```

The next command creates an analytic workspace named GLOBAL_FINANCE in the user's schema. GLOBAL_FINANCE is attached read/write as the last user-owned analytic workspace.

```
SQL> EXECUTE dbms_aw.aw_attach('global_finance', TRUE, TRUE, 'LAST');
```

This command attaches an analytic workspace named SALES_HISTORY from the SH_AW schema in read-only mode.

```
SQL> EXECUTE dbms_aw.aw_attach('sh_aw', 'sales_history');
```

See Also

"Managing Analytic Workspaces" on page 3-1.

AW_COPY Procedure

The AW_COPY procedure copies the object definitions and data from an attached 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 3–12 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 commands create a new analytic workspace named GLOBAL_TRACKING and copies the contents of GLOBAL into it. The workspace is stored in a table named AW$GLOBAL_TRACKING, which has three partitions and is stored in the user's default tablespace.

```
SQL> EXECUTE dbms_aw.aw_attach('global');
SQL> EXECUTE dbms_aw.aw_copy('global', 'global_tracking', NULL, 3);
```

See Also

"Managing Analytic Workspaces" on page 3-1.

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 3–13 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_FINANCE. The new analytic workspace is stored in a table named AW$GLOBAL_FINANCE with eight partitions in the user's default tablespace.

```
SQL> EXECUTE dbms_aw.aw_create('global_finance');
```

The next command creates an analytic workspace named DEMO in the GLOBAL schema. AW$DEMO will have two partitions and will be stored in the GLOBAL tablespace.

```
SQL> EXECUTE dbms_aw.aw_create('global.demo', 'global', 2);
```

AW_DELETE

The AW_DELETE procedure deletes an analytic workspace.

Syntax

```
AW_DELETE (
     awname              IN      VARCHAR2 );

AW_DELETE (
     schema              IN      VARCHAR2,
     awname              IN      VARCHAR2 );
```

Parameters

Table 3–14 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 SALES_DEMO analytic workspace in the user's default schema.

```
SQL> EXECUTE dbms_aw.aw_delete('sales_demo');
```

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 3-37 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 3–15 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_FINANCE analytic workspace.

```
SQL> EXECUTE dbms_aw.aw_detach('global_finance');
```

The next command detaches the SALES_HISTORY analytic workspace in the SH_AW schema.

```
SQL> EXECUTE dbms_aw.aw_detach('sh_aw', 'sales_history');
```

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 3–16 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 commands detach the DEMO analytic workspace and change its name to SALES_DEMO.

```
SQL> EXECUTE dbms_aw.aw_detach('demo');
SQL> EXECUTE dbms_aw.aw_rename('demo', 'sales_demo');
```

See Also

"Procedure: Convert an Analytic Workspace to the Latest Storage Format" on page 3-2.

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;
```

Parameters

Table 3–17 AW_TABLESPACE Function Parameters

Parameter	Description
schema	The schema that owns *awname*.
awname	The name of an analytic workspace.

Return Values

The name of a tablespace.

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('Sales History is stored in tablespace ' ||
     dbms_aw.aw_tablespace('sh_aw', 'sales_history'));
```

This command generates the following statement:

```
Sales History is stored in tablespace SH_AW
```

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 3–18 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 commands save changes to the GLOBAL analytic workspace from the temporary to the permanent tablespace, then commit the change to the database.

```
SQL> EXECUTE dbms_aw.aw_update('global');
SQL> COMMIT;
```

See Also

"Managing Analytic Workspaces" on page 3-1.

CONVERT Procedure

The CONVERT procedure converts an analytic workspace from Oracle9*i* or Oracle Database 10*g* Release 1 format to Oracle Database 10*g* Release 2 format.

See "Converting an Analytic Workspace to Oracle 10g Storage Format" on page 3-2.

Syntax

```
CONVERT (
     original_aw         IN      VARCHAR2 );

CONVERT (
     original_aw         IN      VARCHAR2,
     converted_aw        IN      VARCHAR2,
     tablespace          IN      NUMBER DEFAULT );
```

Parameters

Table 3–19 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

This example performs the conversion in a single step, using the analytic workspace as both the source and the target of the conversion.

```
SQL> EXECUTE dbms_aw.convert('global');
```

The next example performs the conversion in several steps. 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', 'global_temp');
SQL> EXECUTE dbms_aw.convert('global_temp', 'global');
SQL> EXECUTE dbms_aw.delete('global_temp');
```

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 3–20 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*

Return Values

The result of a numeric expression.

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.

```
SQL> SET serveroutput ON
SQL> SELECT dbms_aw.eval_number('decimals') "Decimals" FROM dual;
```

In this example, the value of DECIMALS is 2, which is the default.

```
  Decimals
----------
         2
```

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 3–21 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*

Return Values

The result of a text expression.

Example

The following example returns the value of the NLS_LANGUAGE option, which specifies the current language of the session.

```
SQL> SET serveroutput ON
SQL> SELECT dbms_aw.eval_text('nls_language') "Language" FROM dual;
```

The value of NLS_LANGUAGE in this example is AMERICAN.

```
Language
---------------
AMERICAN
```

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 3–22 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 3-4.
text	Output from the OLAP engine in response to the OLAP commands.

Example

The following example attaches the GLOBAL analytic workspace and shows the object definition of TIME.

```
SQL> EXECUTE dbms_aw.aw_attach('global');
SQL> EXECUTE dbms_aw.execute('DESCRIBE time');

DEFINE TIME DIMENSION TEXT
```

The next example shows how EXECUTE continues to process commands after encountering an error:

```
SQL> EXECUTE dbms_aw.execute('SHOW DECIMALS');
2

SQL> EXECUTE dbms_aw.execute('CALL nothing; DECIMALS=0');
BEGIN dbms_aw.execute('CALL nothing; DECIMALS=0'); END;

*
ERROR at line 1:
```

```
ORA-34492: Analytic workspace object NOTHING does not exist.
ORA-06512: at "SYS.DBMS_AW", line 93
ORA-06512: at "SYS.DBMS_AW", line 122
ORA-06512: at line 1

SQL> EXECUTE dbms_aw.execute('SHOW DECIMALS');
0
```

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;
```

Return Values

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> EXECUTE dbms_aw.printlog(dbms_aw.interp('AW ATTACH global; REPORT units_
cube'));

UNITS_CUBE
--------------
TIME
CUSTOMER
PRODUCT
CHANNEL

SQL> EXECUTE dbms_aw.printlog(dbms_aw.getlog());

UNITS_CUBE
--------------
TIME
CUSTOMER
PRODUCT
CHANNEL
```

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 3–23 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 in the finances.inf file. The location of the file is identified by the WORK_DIR database directory.

```
SQL> EXECUTE dbms_aw.infile('work_dir/finances.inf');
```

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 3–24 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 3-4.

Return Values

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

Example

The following sample SQL*Plus session attaches the GLOBAL analytic workspace and lists the members of UNITS_CUBE.

```
SQL> SET serverout ON
SQL> EXECUTE dbms_aw.printlog(dbms_aw.interp('AW ATTACH global; REPORT units_
cube'));

UNITS_CUBE
--------------
TIME
CUSTOMER
PRODUCT
CHANNEL
```

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 3–25 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 3-4.

Return Values

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 ATTACH global; DESCRIBE'));

DEFINE GEN_OBJ_ROLES DIMENSION TEXT

DEFINE GEN_AW_OBJS VARIABLE TEXT <GEN_OBJ_ROLES>

DEFINE ALL_DIMENSIONS DIMENSION TEXT

DEFINE DIM_OBJ_LIST DIMENSION TEXT

DEFINE DIM_AW_OBJS VARIABLE TEXT <ALL_DIMENSIONS DIM_OBJ_LIST>
     .
     .
     .
```

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 3–26 INTERP_SILENT Procedure 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 3-4.

Example

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

```
SQL> EXECUTE dbms_aw.execute('AW ATTACH global');
IMPORTANT: Analytic workspace GLOBAL is read-only. Therefore, you will not be able
to use the UPDATE command to save changes to it.

SQL> EXECUTE dbms_aw.interp_silent('AW ATTACH global');
```

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;
```

Return Values

The value of the OLAP parameter in the V$OPTION table, which is TRUE if OLAP has been installed in the database, and otherwise FALSE.

Example

The following PL/SQL code tests the value returned by OLAP_ON and returns a status message.

```
BEGIN
      IF dbms_aw.olap_on() = true
      THEN dbms_output.put_line('The OLAP option is installed');
   ELSE dbms_output.put_line('The OLAP option is not installed');
   END IF;
END;
/

The OLAP option is installed
```

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;
```

Return Values

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;
/

OLAP started successfully
```

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 3–27 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('REPORT W 30 all_dimensions'));

ALL_DIMENSIONS
------------------------------
TIME.DIMENSION
CUSTOMER.DIMENSION
PRODUCT.DIMENSION
CHANNEL.DIMENSION
```

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:

```
SQL> 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 3–28 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 3-4.
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> EXECUTE dbms_aw.run('DESCRIBE time');

DEFINE TIME DIMENSION TEXT
```

The next example shows how RUN stops executing commands after encountering an error.

```
SQL> EXECUTE dbms_aw.run('SHOW DECIMALS');
0

SQL> EXECUTE dbms_aw.run('CALL nothing; DECIMALS=4');
BEGIN dbms_aw.run('CALL nothing; DECIMALS=4'); END;

*
ERROR at line 1:
ORA-34492: Analytic workspace object NOTHING does not exist.
ORA-06512: at "SYS.DBMS_AW", line 58
ORA-06512: at "SYS.DBMS_AW", line 134
ORA-06512: at line 1

SQL> EXECUTE dbms_aw.run('SHOW DECIMALS');
0
```

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 FALSE );
```

Parameters

Table 3–29 SHUTDOWN Procedure Parameters

Parameter	Description
force	When TRUE, this parameter forces the OLAP session to shutdown even though one or more attached workspaces has not been updated. Default is FALSE.

Example

The following commands save all changes to the GLOBAL analytic workspace and close the user's OLAP session.

```
SQL> EXECUTE dbms_aw.aw_update('global_finance');
SQL> COMMIT;
SQL> EXECUTE dbms_aw.shutdown();
```

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 3–30 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.

```
SQL> EXECUTE dbms_aw.sparsity_advice_table('global_sparsity_advice');
```

See Also

ADVISE_SPARSITY Procedure on page 3-26 for a description of the columns in *tblname*.

"Using the Sparsity Advisor" on page 3-4.

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 ( );
```

Example

The following example starts an OLAP session.

```
SQL> EXECUTE dbms_aw.startup();
```

4

DBMS_AW_XML

DBMS_AW_XML creates an analytic workspace based on the descriptions of objects stored an XML document.

This chapter includes the following topics:

- Using DBMS_AW_XML
- Summary of DBMS_AW_XML Subprograms

Using DBMS_AW_XML

This section contains topics that relate to using the DBMS_AW_XML package.

Overview

You can define an analytic workspace containing dimensional data objects such as cubes and dimensions using either Analytic Workspace Manager or the OLAP API. The definitions of these dimensional objects are stored in the database as an XML document. You can save the XML document as a text file. You can then load and execute the XML file to re-create the dimensional objects, such as for backup or to copy the objects to a different system. The XML file is often called an XML template, or just a template.

You can use either Analytic Workspace Manager, the OLAP API, or DBMS_AW_XML to load and execute an XML file. The results are identical.

Providing the Context for Cubes and Dimensions

You can save the XML for an entire analytic workspace, for a single cube, or for a single dimension. When re-creating just a cube or dimension, you must provide the appropriate context. This context is an analytic workspace containing all of the metadata objects.

While Analytic Workspace Manager controls the context, DBMS_AW_XML does not. You must create an analytic workspace with the metadata before loading the XML for a cube or a dimension. It is not sufficient just to attach an empty analytic workspace. You must first use the READAWMETADATA function of DMBS_AW_XML.

Obtaining Diagnostic Information

If the procedures execute successfully but do not generate an analytic workspace, the problem may be in the XML template. You should always load a template into the same version and release of Oracle Database as the one used to generate the template.

The build will also fail if objects by the same name already exist in the schema. For example, you cannot create the Global analytic workspace with a dimension named PRODUCT in a schema that contains an analytic workspace named Test with a dimension named PRODUCT.

Error messages generated during the creation of the analytic workspace are stored in OLAPSYS.XML_LOAD_LOG. This table is publicly accessible. Use a query such as the following:

```
SQL> SELECT xml_message FROM olapsys.xml_load_log ORDER BY xml_date;
```

Summary of DBMS_AW_XML Subprograms

The following table describes the subprograms provided in DBMS_AW_EXECUTE.

Table 4–1 DBMS_AW_XML Subprograms

Subprogram	Description
EXECUTE Function on page 4-4	Creates all or part of an analytic workspace from an XML document stored in a CLOB.
EXECUTEFILE Function on page 4-6	Creates all or part of an analytic workspace from an XML document stored in a text file.
READAWMETADATA Function on page 4-7	Loads the definition of an analytic workspace into database memory.

EXECUTE Function

EXECUTE creates dimensional objects from an XML document. The XML is stored in a database object.

Syntax

```
EXECUTE (
     xml_input           IN      CLOB )
RETURN VARCHAR2;
```

Parameters

Table 4–2 EXECUTE Function Parameters

Parameter	Description
xml_input	An XML document stored in a CLOB.

Return Values

The string Success if successful

Example

The following SQL script 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. The DBMS_OUTPUT.PUT_LINE procedure is used to provide status messages during execution of the script.

You would typically use the EXECUTEFILE function to create an analytic workspace from an XML file. This example creates a CLOB from a file just for illustrating the EXECUTE function.

```
DECLARE
     clb CLOB;
     infile BFILE;

BEGIN

     dbms_output.put_line('Create a temporary clob');
     dbms_lob.createtemporary(clb, TRUE,10);

     dbms_output.put_line('Create a BFILE using BFILENAME function');
     infile := bfilename('WORK_DIR', 'GLOBAL.XML');

     dbms_output.put_line('Open the BFILE');
     dbms_lob.fileopen(infile, dbms_lob.file_readonly);

     dbms_output.put_line('Load Temporary Clob from the BFILE');
     dbms_lob.loadfromfile(clb,infile,dbms_lob.lobmaxsize,1,1);
     COMMIT;

     dbms_output.put_line('Close the BFILE');
     dbms_lob.fileclose(infile);

     dbms_output.put_line('Create the analytic workspace');
     dbms_output.put_line(dbms_aw_xml.execute(clb));
     COMMIT;
     dbms_aw.aw_update;
```

```
dbms_output.put_line('Free the temporary clob');
dbms_lob.freetemporary(clb);

EXCEPTION
     WHEN OTHERS
     THEN
          dbms_output.put_line(SQLERRM);
END;
/
```

The successful execution of this script generates the following messages:

```
Create a temporary clob
Create a BFILE using BFILENAME function
Open the BFILE
Load Temporary Clob from the BFILE
Close the BFILE
Create the analytic workspace
Success
Free the temporary clob
```

EXECUTEFILE Function

EXECUTEFILE creates dimensional objects from an XML document stored in a text file. This file is also called a template.

Syntax

```
EXECUTEFILE (
     dirname             IN      VARCHAR2,
     xml_file            IN      VARCHAR2 )
RETURN VARCHAR2;
```

Returns

The string Success if successful

Parameters

Table 4–3 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.

Example

The following EXECUTEFILE function generates an analytic workspace from the XML document stored in GLOBAL.XML, which is located in a directory identified by the WORK_DIR directory object. The DBMS_OUTPUT.PUT_LINE function displays the Success message returned by EXECUTEFILE.

```
SQL> SET serveroutput ON format wrapped
SQL> EXECUTE dbms_output.put_line(dbms_aw_xml.executefile('WORK_DIR',
'GLOBAL.XML'));
Success
```

READAWMETADATA Function

Loads the XML definition of an analytic workspace into database memory and transfers the definition to the client, so that it can construct the client-side model.

Syntax

```
READAWMETADATA (
     awname              IN      VARCHAR2,
     rights              IN      VARCHAR2 )
RETURN GENWSTRINGSEQUENCE;
```

Returns

An XML document that defines the analytic workspace

Parameters

Table 4–4 READAWMETADATA Function Parameters

Parameter	Description
awname	The name of the analytic workspace you want to create.
rights	The access rights to attach the analytic workspace: ■ RW: Read-write ■ RWX: Read-write with exclusive access ■ RO: Read only; the analytic workspace cannot be saved

Example

This example first loads the OLAP metadata into memory. It creates an analytic workspace named GLOBAL that contains this OLAP metadata but no dimensions or cubes. It then uses the EXECUTEFILE function to create the Product dimension from an XML template file.

```
DECLARE
     tmp  GENWSTRINGSEQUENCE;
BEGIN
     dbms_output.put_line('Read the metadata');
     tmp := dbms_aw_xml.readawmetadata('global', 'RW');
END;
/

DECLARE
     tmp2 VARCHAR2(100);
BEGIN
     dbms_output.put_line('Create the Global aw');
     dbms_aw.execute('AW CREATE global');
     dbms_output.put_line('Create the Product dimension');
     dbms_output.put_line(dbms_aw_xml.executefile('work_dir', 'product.xml'));
     COMMIT;
     dbms_aw.aw_update;
EXCEPTION
     WHEN OTHERS
     THEN
          dbms_output.put_line(SQLERRM);
END;
 /
```

The successful execution of this script generates the following messages:

```
Read the metadata
Create the Global aw
Create the Product dimension
Success
```

5

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. To use it, you must either be a privileged user or have explicit execution rights.

Viewing the Configuration Table

ALL_OLAP_ALTER_SESSION is the public synonym for V$OLAP_ALTER_SESSION, which is a view of the OLAP$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 ALL_OLAP_

ALTER_SESSION has rows for the OLAP_USER and the OLAP_DBA roles, then a user with the OLAP_USER role will only have the privileges set for the OLAP_USER role. The privileges for the OLAP_DBA role will be ignored.

Table 5–1 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 5–2 OLAP_API_SESSION_INIT Subprograms

Subprogram	Description
ADD_ALTER_SESSION Procedure on page 5-4	Specifies an `ALTER SESSION` parameter for OLAP API users with a particular database role.
CLEAN_ALTER_SESSION Procedure on page 5-5	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 5-6	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 5–3 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 example inserts a row in OLAP$ALTER_SESSION that turns on query rewrite for users with the OLAP_DBA role.

```
SQL> EXECUTE olap_api_session_init.add_alter_session('OLAP_DBA',
     'SET QUERY_REWRITE_ENABLED=TRUE');
Row inserted

SQL> SELECT * FROM all_olap_alter_session WHERE role='OLAP_DBA';

ROLE         CLAUSE_TEXT
------------ ------------------------------------------------------------
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 ();
```

Example

The following example deletes all orphaned rows.

```
SQL> EXECUTE 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 5–4 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.

```
SQL> EXECUTE olap_api_session_init.delete_alter_session('OLAP_DBA',
     'SET QUERY_REWRITE_ENABLED=TRUE');

SQL> SELECT * FROM all_olap_alter_session WHERE role='OLAP_DBA';

no rows selected
```

6

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 11, "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 6–2, " 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 11-17.

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 11-15.

OLAP_CONDITION Examples

Several sample queries using OLAP_CONDITION are shown in Example 6–2. 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 6-6 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 6–1. For information about creating views of analytic workspaces, see "OLAP_TABLE Overview" on page 11-1.

Example 6–1 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 6–2 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 6–1. 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> EXECUTE dbms_aw.execute('REPORT time_short_description');

TIME     TIME_SHORT_DESCRIPTION
----     ----------------------
  3      2000

-- Reset status
SQL> EXECUTE 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> EXECUTE dbms_aw.execute('REPORT 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> EXECUTE 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> EXECUTE dbms_aw.execute('REPORT 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 6–1 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 7, "OLAP_EXPRESSION") OLAP_CONDITION simply uses it as an identifier. For information on creating a ROW2CELL column, see "Limit Map Parameter" on page 11-17.
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 6–2

Return Values

The number 1 to indicate a successful invocation of OLAP_CONDITION.

Note

The entry points for OLAP_CONDITION are described in Table 6–2. Refer to "Order of Processing in OLAP_TABLE" on page 11-24 to determine where each entry point occurs.

Table 6–2 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 11-24. 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 6–2 (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 11-24. 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 11-15.) 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 11-24. If OLAP_CONDITION limits any dimensions, the limits remain in the analytic workspace after the query completes.

Example

See "OLAP_CONDITION Examples" on page 6-2.

7

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 8, "OLAP_EXPRESSION_BOOL"
- Chapter 9, "OLAP_EXPRESSION_DATE"
- Chapter 10, "OLAP_EXPRESSION_TEXT"
- Chapter 6, "OLAP_CONDITION"
- Chapter 11, "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 11-1 and "Limit Map: ROW2CELL Clause" on page 11-22

OLAP_EXPRESSION Examples

The following script was used to create the view unit_cost_price_view, which is used in Example 7–1 and Example 7–2 to illustrate the use of OLAP_EXPRESSION. For information about creating views of analytic workspaces, see "OLAP_TABLE Overview" on page 11-1.

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 7–1 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 7-2.

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 7–2 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 7-2.

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 7–1 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 11-17.
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 3-4.

Return Values

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 7-2.

8

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 7, "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 11-1 and "Limit Map: ROW2CELL Clause" on page 11-22

OLAP_EXPRESSION_BOOL Example

The following script was used to create the view awunits_view, which is used in Example 8–1 to illustrate the use of OLAP_EXPRESSION_BOOL.

> **See Also:** See "OLAP_TABLE Overview" on page 11-1 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('REPORT 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('REPORT 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('REPORT time_aw_short_description');

TIME_AW                    TIME_AW_SHORT_DESCRIPTION
-------------- --------------------------------------------------
4              2001
```

Example 8–1 OLAP_EXPRESSION_BOOL Function in a SELECT List

This example uses the view described in "Sample View: GLOBAL_AW.AWUNITS_VIEW" on page 8-2.

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('REPORT 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 8–1 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 11-17.
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 3-4.

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 7-1 for more examples of OLAP single-row functions.

9

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 7, "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 11-1 and "Limit Map: ROW2CELL Clause" on page 11-22

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 9–1 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 11-17.
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 3-4.

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 7-2 and "OLAP_EXPRESSION_ BOOL Example" on page 8-2 for examples of OLAP single-row functions.

10

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 7, "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 11-1 and "Limit Map: ROW2CELL Clause" on page 11-22

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 10–1 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 11-17.
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 3-4.

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 7-2 and "OLAP_EXPRESSION_BOOL Example" on page 8-2 for examples of OLAP single-row functions.

11

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.

This chapter contains the following topics:

See Also:

- Chapter 6, "OLAP_CONDITION"
- Chapter 7, "OLAP_EXPRESSION"

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.

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 11-17.

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 11–1 shows how to create a view of an analytic workspace using predefined ADTs.

Example 11–1 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 11–2 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 11–2 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 11–3 shows how to create a view of an analytic workspace using automatic ADTs.

Example 11–3 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 11–4 creates the same view produced by Example 11–2, 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 11–4 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 11–1.

Table 11–1 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
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.

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 11-12 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 11-3.

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 11-5.

Example: Creating Views of Embedded Total Dimensions

Example 11–5 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 11–2, 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 11-24.

Example 11–5 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'));
/
```

The following SELECT statement queries the view created by the script:

```
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 11–6 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 11-22 for information on ROW2CELL.

Example 11–6 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;
```

The following SELECT statement queries the view created by the script:

```
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 11-20 for information on FAMILYREL.

Example 11–7 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 11–5, "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 11–8 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 11–7 and Example 11–8 generate identical views.

Example 11–7 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'));
```

The following SELECT statement queries the view created by the script:

```
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 11–8 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'));
```

The following SELECT statement queries the view created by the script:

```
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 11-15.

The script shown in Example 11–9 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 11–9 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 11–9 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;
/
```

The following SELECT statement queries the view created by the script:

```
SQL> 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 11-24.

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 11–2 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 11-13.
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 11-14.
olap_command	An optional OLAP DML command. See "OLAP Command Parameter" on page 11-15.
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 11-17.

Returns

A table type whose rows are objects (ADTs) that identify the selected workspace data. See "Logical Tables" on page 11-2.

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 11-3.

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 11-2.

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 11–1, " Default Data Type Conversions". See "Using OLAP_TABLE With Automatic ADTs" on page 11-3.

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 11-24.

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 6, "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 11-10.

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 11-1.

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 6 and Chapter 7.)

The order in which OLAP_TABLE processes information in the limit map is specified in "Order of Processing in OLAP_TABLE" on page 11-24.

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 11-15.

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.

```
SQL> 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 11–10. 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.

Example 11–10 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 11-3.

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 11-19.

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 11-22.

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 11–10, "Syntax of an OLAP_TABLE Limit Map". It is shown without the rest of the limit map syntax in Example 11–11.

Example 11–11 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 11-19.

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 11-19.

  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 11-19.

  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 11-19.

  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 11-8.

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 11-22.

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 6 and Chapter 7.

```
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 11-24.

```
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 11-24.

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

I

L

M

O

P

Q

R

S

T

V

X