### Oracle® OLAP

Reference 10*g* Release 2 (10.2.0.3) **B14350-02** 

September 2006



Oracle OLAP Reference, 10g Release 2 (10.2.0.3)

B14350-02

Copyright © 2003, 2006, Oracle. All rights reserved.

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software--Restricted Rights (June 1987). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and we disclaim liability for any damages caused by such use of the Programs.

Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

The Programs may provide links to Web sites and access to content, products, and services from third parties. Oracle is not responsible for the availability of, or any content provided on, third-party Web sites. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

# Contents

Preface	vii
Audience	vii
Documentation Accessibility	vi
Related Documents	viii
Conventions	viii
Documentation Accessibility Related Documents	vii viii

# 1 Active Catalog Views

Understanding the Active Catalog	1-1
Summary of Active Catalog Views	1-1
ALL_OLAP2_AWS	1-2
ALL_OLAP2_AW_ATTRIBUTES	1-2
ALL_OLAP2_AW_CATALOGS	1-3
ALL_OLAP2_AW_CATALOG_MEASURES	1-3
ALL_OLAP2_AW_CUBES	1-3
ALL_OLAP2_AW_CUBE_AGG_LVL	1-4
ALL_OLAP2_AW_CUBE_AGG_MEAS	1-4
ALL_OLAP2_AW_CUBE_AGG_OP	1-4
ALL_OLAP2_AW_CUBE_AGG_SPECS	1-5
ALL_OLAP2_AW_CUBE_DIM_USES	1-5
ALL_OLAP2_AW_CUBE_MEASURES	1-5
ALL_OLAP2_AW_DIMENSIONS	1-6
ALL_OLAP2_AW_DIM_HIER_LVL_ORD	1-6
ALL_OLAP2_AW_DIM_LEVELS	1-7
ALL_OLAP2_AW_PHYS_OBJ	
ALL_OLAP2_AW_PHYS_OBJ_PROP	1-7

# 2 OLAP Dynamic Performance Views

V\$ Tables for OLAP	2-1
Summary of OLAP Dynamic Performance Views	2-2
V\$AW_AGGREGATE_OP	2-2
V\$AW_ALLOCATE_OP	2-3
V\$AW_CALC	2-3
V\$AW_LONGOPS	2-4
V\$AW_OLAP	2-5
V\$AW_SESSION_INFO	2-6

# 3 DBMS\_AW

Managing Analytic Workspaces	. 3-1
Converting an Analytic Workspace to Oracle 10g Storage Format	. 3-2
Embedding OLAP DML in SQL Statements	3-3
Methods for Executing OLAP DML Commands	3-3
Guidelines for Using Quotation Marks in OLAP DML Commands	3-4
Using the Sparsity Advisor	3-4
Data Storage Options in Analytic Workspaces	3-4
Selecting the Best Data Storage Method	3-5
Using the Sparsity Advisor	3-5
Example: Evaluating Sparsity in the GLOBAL Schema	3-6
Using the Aggregate Advisor	3-8
Aggregation Facilities within the Workspace	3-8
Example: Using the ADVISE_REL Procedure	
Summary of DBMS_AW Subprograms	3-12
ADD_DIMENSION_SOURCE Procedure	3-14
ADVISE_CUBE Procedure	3-16
ADVISE_DIMENSIONALITY Function	3-18
ADVISE_DIMENSIONALITY Procedure	3-20
ADVISE_PARTITIONING_DIMENSION Function	3-22
ADVISE_PARTITIONING_LEVEL Function	3-23
ADVISE_REL Procedure	3-24
ADVISE_SPARSITY Procedure	3-25
AW_ATTACH Procedure	3-28
AW_COPY Procedure	3-30
AW_CREATE Procedure	3-31
AW_DELETE	3-32
AW_DETACH Procedure	3-33
AW_RENAME Procedure	3-34
AW_TABLESPACE Function	3-35
AW_UPDATE Procedure	3-36
CONVERT Procedure	3-37
EVAL_NUMBER Function	3-38
EVAL_TEXT Function	3-39
EXECUTE Procedure	3-40
GETLOG Function	3-42
INFILE Procedure	3-43
INTERP Function	3-44
INTERPCLOB Function	
INTERP_SILENT Procedure	3-46
OLAP_ON Function	3-47
OLAP_RUNNING Function	3-48
PRINTLOG Procedure	3-49
RUN Procedure	3-50
SHUTDOWN Procedure	3-52
SPARSITY_ADVICE_TABLE Procedure	3-53
STARTUP Procedure	3-54

### 4 DBMS\_AW\_XML

Analytic Workspace Java API Overview	4-1
Oracle OLAP XML Schema	4-1
Summary of DBMS_AW_XML Subprograms	4-3
EXECUTE Function	4-4
EXECUTEFILE Function	4-6

## 5 OLAP\_API\_SESSION\_INIT

Initialization Parameters for the OLAP API	5-1
Viewing the Configuration Table	5-1
ALL_OLAP_ALTER_SESSION View	5-1
Summary of OLAP_API_SESSION_INIT Subprograms	5-3
ADD_ALTER_SESSION Procedure	5-4
CLEAN_ALTER_SESSION Procedure	5-5
DELETE_ALTER_SESSION Procedure	5-6

## 6 OLAP\_CONDITION

OLAP_CONDITION Overview	6-1
Entry Points in the Limit Map	6-1
Dynamically Modifying a Workspace during a Query	6-2
OLAP_CONDITION Examples	6-2
OLAP_CONDITION Syntax	6-6

## 7 OLAP\_EXPRESSION

OLAP_EXPRESSION Overview	7-1
Single-Row Functions	7-1
OLAP_EXPRESSION and OLAP_TABLE	7-2
OLAP_EXPRESSION Examples	7-2
OLAP_EXPRESSION Syntax	7-5

## 8 OLAP\_EXPRESSION\_BOOL

OLAP_EXPRESSION_BOOL Overview	8-1
Single-Row Functions	8-1
OLAP_EXPRESSION_BOOL and OLAP_TABLE	8-1
OLAP_EXPRESSION_BOOL Example	8-2
OLAP_EXPRESSION_BOOL Syntax	8-5

## 9 OLAP\_EXPRESSION\_DATE

OLAP_EXPRESSION_DATE Overview	9-1
Single-Row Functions	9-1
OLAP_EXPRESSION_DATE and OLAP_TABLE	9-1
OLAP_EXPRESSION_DATE Syntax	9-3

## 10 OLAP\_EXPRESSION\_TEXT

OLAP_EXPRESSION_TEXT Overview	10-1
Single-Row Functions	10-1
OLAP_EXPRESSION_TEXT and OLAP_TABLE	10-1
OLAP_EXPRESSION_TEXT Syntax	10-3

## 11 OLAP\_TABLE

OLAP_TABLE Overview 11-	-1
Limit Maps 11-	-1
Logical Tables 11-	-2
Using OLAP_TABLE With Predefined ADTs 11-	-2
Using OLAP_TABLE With Automatic ADTs 11-	-3
Using a MODEL Clause 11-	-5
OLAP_TABLE Examples 11-	-5
Example: Creating Views of Embedded Total Dimensions 11-	-6
Example: Creating Views of Embedded Total Measures 11-	-7
Example: Creating Views in Rollup Form 11-	-8
Using OLAP_TABLE with the FETCH Command 11-1	0
OLAP_TABLE Syntax 11-1	2
Analytic Workspace Parameter 11-1	3
Table Object Parameter   11-1	4
OLAP Command Parameter 11-1	5
Limit Map Parameter 11-1	7
Order of Processing in OLAP_TABLE 11-2	23

## 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
- Documentation Accessibility
- Related Documents
- Conventions

## Audience

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

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

## **Documentation Accessibility**

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

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

#### Accessibility of Code Examples in Documentation

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

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

### Accessibility of Links to External Web Sites in Documentation

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

### **TTY Access to Oracle Support Services**

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

## **Related Documents**

For more information see these Oracle resources:

Oracle OLAP Application Developer's Guide

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

• Oracle OLAP DML Reference

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

• Oracle OLAP Developer's Guide to the OLAP API

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

Oracle OLAP Java API Reference

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

Oracle OLAP Analytic Workspace Java API Reference

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

Oracle Database PL/SQL User's Guide and Reference

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

## **Conventions**

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.

Convention	Meaning
monospace	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

# **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 within analytic workspaces. Views of this metadata are commonly referred to as the **Active Catalog**, because they are populated with information that is automatically generated within analytic workspaces.

Active Catalog views 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.

PUBLIC Synonym	Description
ALL_OLAP2_AWS	Lists the analytic workspaces.
ALL_OLAP2_AW_ATTRIBUTES	List of dimension attributes in analytic workspaces.
ALL_OLAP2_AW_CATALOGS	Lists the measure folders in analytic workspaces.
ALL_OLAP2_AW_CATALOG_ MEASURES	Lists the measures in the measure folders.
ALL_OLAP2_AW_CUBES	List of cubes in analytic workspaces.
ALL_OLAP2_AW_CUBE_AGG_LVL	List of levels in aggregation plans in analytic workspaces.
ALL_OLAP2_AW_CUBE_AGG_MEAS	List of measures in aggregation plans in analytic workspaces.
ALL_OLAP2_AW_CUBE_AGG_OP	List of aggregation operators in aggregation plans in analytic workspaces.
ALL_OLAP2_AW_CUBE_AGG_SPECS	List of aggregation plans in analytic workspaces.

Table 1–1 Active Catalog Views

 Table 1–1 (Cont.) Active Catalog Views

PUBLIC Synonym	Description
ALL_OLAP2_AW_CUBE_DIM_USES	List of cubes with their associated dimensions in analytic workspaces.
ALL_OLAP2_AW_CUBE_MEASURES	List of cubes with their associated measures in analytic workspaces.
ALL_OLAP2_AW_DIMENSIONS	List of dimensions in analytic workspaces.
ALL_OLAP2_AW_DIM_HIER_LVL_ORD	List of hierarchical levels in analytic workspaces.
ALL_OLAP2_AW_DIM_LEVELS	List of levels in analytic workspaces.
ALL_OLAP2_AW_PHYS_OBJ	List of standard form objects in analytic workspaces.
ALL_OLAP2_AW_PHYS_OBJ_PROP	List of properties associated with standard form objects in analytic workspaces.

## ALL\_OLAP2\_AWS

ALL\_OLAP2\_AWS provides a list of all the analytic workspaces accessible to the current user. This includes both standard form and non-standard analytic workspaces.

Column	Datatype	NULL	Description
OWNER	VARCHAR2(30)		Owner of the analytic workspace.
AW	VARCHAR2(30)		Name of the analytic workspace.
AW_NUMBER	NUMBER		Unique identifier for the analytic workspace.
AW_VERSION	VARCHAR2(4)		The version of Oracle Database in which the analytic workspace was created. If the version is 10.1 or higher, the workspace is in 10g storage format. Earlier versions are in 9 <i>i</i> format.
SF_VERSION	CHAR(8)		The version of 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. See Table 12–1, " Reserved Dimension Attributes".
SOURCE_OWNER	VARCHAR2(4000)		Owner of the source attribute in the OLAP Catalog (Oracle9 <i>i</i> metadata).

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

# ALL\_OLAP2\_AW\_CATALOGS

 ${\tt ALL\_OLAP2\_AW\_CATALOGS}\ lists\ the\ measure\ folders\ in\ analytic\ workspaces.$ 

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>i</i> metadata).
SOURCE_NAME	VARCHAR2(4000)		Name of the source cube in the OLAP Catalog (Oracle9 <i>i</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 workspace level of the dimension. This level is in the aggregation specification.

# ALL\_OLAP2\_AW\_CUBE\_AGG\_MEAS

ALL\_OLAP2\_AW\_CUBE\_AGG\_MEAS lists the measures in aggregation specifications in standard form analytic workspaces.

Column	Datatype	NULL	Description
AW_OWNER	VARCHAR2(30)		Owner of the analytic workspace.
AW_NAME	VARCHAR2(30)		Name of the analytic workspace.
AW_CUBE_NAME	VARCHAR2(90)		Name of a cube in the analytic workspace.
AW_AGGSPEC_NAME	VARCHAR2(4000)		Name of an aggregation specification for the cube.
AW_MEASURE_NAME	VARCHAR2(4000)		Name of a workspace measure of the cube. This measure is in the aggregation specification

# ALL\_OLAP2\_AW\_CUBE\_AGG\_OP

ALL\_OLAP2\_AW\_CUBE\_AGG\_OP lists the aggregation operators in aggregation specifications in standard form analytic workspaces.

Column	Datatype	NULL	Description
AW_OWNER	VARCHAR2(30)		Owner of the analytic workspace.
AW_NAME	VARCHAR2(30)		Name of the analytic workspace.
AW_CUBE_NAME	VARCHAR2(90)		Name of a cube in the analytic workspace.

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. See Table 1–10, " Aggregation Operators" for a list of valid operators.

## ALL\_OLAP2\_AW\_CUBE\_AGG\_SPECS

ALL\_OLAP2\_AW\_CUBE\_AGG\_SPECS lists the aggregation specifications in standard form analytic workspaces.

Column	Datatype	NULL	Description
AW_OWNER	VARCHAR2(30)		Owner of the analytic workspace.
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>i</i> metadata).
DIMENSION_SOURCE_NAME	VARCHAR2(4000)		Name of the source dimension in the OLAP Catalog (Oracle9 <i>i</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.

Column	Datatype	NULL	Description
AW_PHYSICAL_OBJECT	VARCHAR2(4000)		Standard form name of the measure.
MEASURE_SOURCE_NAME	VARCHAR2(4000)		Name of the source measure in the OLAP Catalog (Oracle9 <i>i</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 with the OLAP DML AGGREGATE command. The value is YES if the measure is implemented as an OLAP variable or if its underlying storage is a variable. For example, the measure could be implemented as a formula whose value is stored in a variable.

# ALL\_OLAP2\_AW\_DIMENSIONS

ALL\_OLAP2\_AW\_DIMENSIONS lists the dimensions in standard form analytic workspaces.

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>i</i> metadata).
SOURCE_NAME	VARCHAR2(4000)		Name of the source dimension in the OLAP Catalog (Oracle9 <i>i</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

<code>ALL\_OLAP2\_AW\_DIM\_LEVELS</code> 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 with the OLAP DML, including: dimensions, relations, variables, formulas, composites, and valuesets.
AW_OBJECT_DATATYPE	VARCHAR2(4000)		Data type of the standard form object. The data type may be any of the native types supported by the OLAP DML, including text, boolean, or integer, or it may be a defined type specific to standard form.

# ALL\_OLAP2\_AW\_PHYS\_OBJ\_PROP

<code>ALL\_OLAP2\_AW\_PHYS\_OBJ\_PROP</code> 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.

# **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
- Summary of OLAP Dynamic Performance Views
- V\$AW\_AGGREGATE\_OP
- V\$AW\_ALLOCATE\_OP
- V\$AW\_CALC
- V\$AW\_LONGOPS
- V\$AW\_OLAP
- V\$AW\_SESSION\_INFO

### V\$ Tables for OLAP

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

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

The SYS user owns the V\$ tables. In addition, any user with the SELECT CATALOG role can access the tables. The system creates views from these tables and creates public synonyms for the views. The views are also owned by SYS, but the DBA can grant access to them to a wider range of users.

The names of the OLAP V\$ tables begin with V\$AW. The view names also begin with V\$AW. The following sample SQL\*Plus session shows the list of OLAP system tables.

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

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

Table 2–1 OLAP Fixed Views	Table 2–1	OLAP	Fixed	Views
----------------------------	-----------	------	-------	-------

## V\$AW\_AGGREGATE\_OP

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

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

## V\$AW\_ALLOCATE\_OP

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

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

## V\$AW\_CALC

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

### **OLAP Caches**

Because OLAP queries tend to be iterative, the same data is typically queried repeatedly during a session. The caches provide much faster access to data that has already been calculated during a session than would be possible if the data had to be recalculated for each query.

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

Oracle OLAP uses the following caches:

- Aggregate cache. An internal cache used by the aggregation subsystem during querying. It stores the children of a given dimension member, such as Q1-04, Q2-04, Q3-04, and Q4-04 as the children of 2004.
- Session cache. Oracle OLAP maintains a cache for each session for storing the results of calculations. When the session ends, the contents of the cache are discarded.
- Page pool. A cache allocated from the User Global Area (UGA), which Oracle OLAP maintains for the session. The page pool is associated with a particular session and caches records from all the analytic workspaces attached in that session. If the page pool becomes too full, then Oracle OLAP writes some of the pages to the database cache. When an UPDATE command is issued in the OLAP DML, the changed pages associated with that analytic workspace are written to the permanent LOB, using temporary segments as the staging area for streaming the data to disk. The size of the page pool is controlled by the OLAP\_PAGE\_POOL initialization parameter.
- **Database cache**. The larger cache maintained by the Oracle RDBMS for the database instance.

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

### **Dynamic Aggregation**

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

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

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

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.

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

# V\$AW\_LONGOPS

 $\tt V\$  Longops provides status information about active SQL cursors initiated in the OLAP DML.

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

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

# 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
- Embedding OLAP DML in SQL Statements
- Using the Sparsity Advisor
- Using the Aggregate Advisor
- Summary of DBMS\_AW Subprograms

## 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 10g Storage Format

Analytic workspaces are stored in tables within the database. The storage format for Oracle 10g 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 SCRIPTS directory object.

```
SQL>execute dbms_aw.execute ('aw attach ''awname''');
SQL>execute dbms_aw.execute ('allstat');
SQL>execute dbms_aw.execute ('cda scripts');
SQL>execute dbms_aw.execute ('export all to eif file ''filename''');
```

In Oracle 10g SQL\*Plus, create a new workspace with the same name and schema, and import the EIF file from the SCRIPTS directory.

```
SQL>execute dbms_aw.execute ('aw create awname');
SQL>execute dbms_aw.execute ('cda scripts');
SQL>execute dbms_aw.execute ('import all from eif file ''filename''');
SQL>execute dbms_aw.execute ('update');
```

You can also use Oracle export and import utilities to move the entire schema, including the analytic workspaces to an Oracle 10g database. See Oracle Database Utilities and Oracle Database Upgrade Guide.

### Embedding OLAP DML in SQL Statements

With the DBMS\_AW package you can perform the full range of OLAP processing within analytic workspaces. You can import data from legacy workspaces, relational tables, or flat files. You can define OLAP objects and perform complex calculations.

**Note:** If you use the DBMS\_AW package to create analytic workspaces from scratch, you may not be able to use OLAP utilities, such as Analytic Workspace Manager and the DBMS\_AW Aggregate Advisor, which require standard form.

### Methods for Executing OLAP DML Commands

The DBMS\_AW package provides several procedures for executing ad hoc OLAP DML commands. Using the EXECUTE or INTERP\_SILENT procedures or the INTERP or INTERCLOB functions, you can execute a single OLAP DML command or a series of commands separated by semicolons.

Which procedures you use will depend on how you want to direct output and on the size of the input and output buffers. For example, the EXECUTE procedure directs

output to a printer buffer, the INTERP\_SILENT procedure suppresses output, and the INTERP function returns the session log.

The DBMS\_AW package also provides functions for evaluating OLAP expressions. The EVAL\_TEXT function returns the result of a text expression, and EVAL\_NUMBER returns the result of a numeric expression.

**See Also:** *Oracle OLAP DML Reference* for complete information about OLAP DML expressions.

Do not confuse the DBMS\_AW functions EVAL\_NUMBER and EVAL\_ TEXT with the SQL function OLAP\_EXPRESSION. See Chapter 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?	Туре
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
```

```
CONNECT global/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<pre>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<pre>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.

SELECT fact, dimension, dimcolumn, membercount nmem, leafcount nleaf, advice, density from aw\_sparsity\_advice WHERE cubename='units\_cube';

DENSITY
.46182
.94827
.97031
.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 3-24
- 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('rpr w 40 customer hierlist');
CUSTOMER HIERLIST
-----
MARKET ROLLUP
SHIPMENTS ROLLUP
---- Levels in Customer dimension
SQL>execute dbms_aw.execute('rpr w 40 customer_levellist');
CUSTOMER LEVELLIST
-----
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
            CUSTOMER_HIER_LEVELS
IST
-----
SHIPMENTS SHIP TO
            WAREHOUSE
            REGION
            TOTAL CUSTOMER
MARKET SEGMENT SHIP TO
            ACCOUNT
            MARKET SEGMENT
            TOTAL_MARKET
---- Parent relation showing parent-child relationships in the Customer dimension
---- Only show the last 20 members
SQL>execute dbms aw.execute('limit customer to last 20');
SQL>execute dbms_aw.execute('rpr w 10 down customer w 20 customer_parentrel');
         -----CUSTOMER PARENTREL------
         -----CUSTOMER HIERLIST------
CUSTOMER MARKET_ROLLUP SHIPMENTS_ROLLUP
     44
                         21
103
104
       45
                          21
      45
45
105
                          21
106
                          21
7
       NA
                          NA
       NA
1
                          NA
       NA
8
                          1
9
        NA
                          1
       NA
10
                          1
      NA
11
                          8
12
       NA
                          10
13
       NA
                          9
14
       NA
                          9
15
       NA
                          8
16
       NA
                          9
       NA
17
                          8
18
        NA
                          8
19
        NA
                          9
       NA
20
                          9
       NA
21
                          10
```

<customer_short< th=""><th>ns_aw.execute('report w 15 down custc :_description&gt;'); AMERICAN AMERICA</th><th>mer w 35 across customer_hierlist:</th></customer_short<>	ns_aw.execute('report w 15 down custc :_description>'); AMERICAN AMERICA	mer w 35 across customer_hierlist:
-		HIERLIST
		SHIPMENTS ROLLUP
	CUSTOMER_SHORT_DESCRIPTION	
103	US Marine Svcs Washington	
104		
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

---- Show text descriptions for the same twenty dimension members

### 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
```

Customer			
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

The returned Customer members with their text descriptions, related levels, and related hierarchies, are shown as follows.

# Summary of DBMS\_AW Subprograms

The following table describes the subprograms provided in DBMS\_AW.

 Table 3–1
 DBMS\_AW Subprograms

Table 3-1 DBMS_AW Subp	rograms
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 3-24	Suggests how to preaggregate a dimension, based on a specified percentage of the dimension's members.
ADVISE_SPARSITY Procedure on page 3-25	Analyzes a fact table for sparsity and populates a table with the results of its analysis.
AW_ATTACH Procedure on page 3-28	Attaches an analytic workspace to a session.
AW_COPY Procedure on page 3-30	Creates a new analytic workspace and populates it with the object definitions and data from another analytic workspace.
AW_CREATE Procedure on page 3-31	Creates a new, empty analytic workspace.
AW_DELETE on page 3-32	Deletes an analytic workspace
AW_DETACH Procedure on page 3-33	Detaches an analytic workspace from a session.
AW_RENAME Procedure on page 3-34	Changes the name of an analytic workspace.
AW_TABLESPACE Function on page 3-35	Returns the name of the tablespace in which a particular analytic workspace is stored.
AW_UPDATE Procedure on page 3-36	Saves changes made to an analytic workspace.
CONVERT Procedure on page 3-37	Converts an analytic workspace from 9 <i>i</i> to 10 <i>g</i> storage format.
EVAL_NUMBER Function on page 3-38	Returns the result of a numeric expression in an analytic workspace.
EVAL_TEXT Function on page 3-39	Returns the result of a text expression in an analytic workspace.
EXECUTE Procedure on page 3-40	Executes one or more OLAP DML commands. Input and output is limited to 4K. Typically used in an interactive session using an analytic workspace.
Subprogram	Description
---	---
GETLOG Function on page 3-42	Returns the session log from the last execution of the INTERP or INTERPCLOB functions.
INFILE Procedure on page 3-43	Executes the OLAP DML commands specified in a file.
INTERP Function on page 3-44	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 3-45	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 3-46	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 3-47	Returns a boolean indicating whether or not the OLAP option is installed in the database.
OLAP_RUNNING Function on page 3-48	Returns a boolean indicating whether or not the OLAP option has been initialized in the current session.
PRINTLOG Procedure on page 3-49	Prints a session log returned by the INTERP, INTERCLOB, or GETLOG functions.
RUN Procedure on page 3-50	Executes one or more OLAP DML commands.
SHUTDOWN Procedure on page 3-52	Shuts down the current OLAP session.
SPARSITY_ADVICE_TABLE Procedure on page 3-53	Creates a table which the ADVISE_SPARSITY procedure will use to store the results of its analysis.
STARTUP Procedure on page 3-54	Starts an OLAP session without attaching a user-defined analytic workspace.

Table 3–1 (Cont.) DBMS_AW Subprograms
---------------------------------------

# 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_SOUR	CE (		
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 A	ADD_DIMENSION_	SOURCE Procedure Pal	rameters
-------------	----------------	----------------------	----------

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 <i>dimname</i> .		
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 <i>colname</i> is used.		
dimtype	One of the following hierarchy types:		
	DBMS_AW.HIER_LEVELS DBMS_AW.HIER_PARENTCHILD DBMS_AW.MEASURE DBMS_AW.NO_HIER	Level-based hierarchy Parent-child hierarchy Measure dimension No hierarchy	
hiercols	The names of the columns that de	efine 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.		

Parameter	Description
partby	A keyword that controls partitioning. Use one of the following values:
	<ul> <li>DBMS_AW.PARTBY_DEFAULT Allow the Sparsity Advisor to determine whether or not partitioning is appropriate for this dimension.</li> </ul>
	<ul> <li>DBMS_AW.PARTBY_NONE Do not allow partitioning on this dimension.</li> </ul>
	<ul> <li>DBMS_AW.PARTBY_FORCE Force partitioning on this dimension.</li> </ul>
	<b>Important</b> : Do not force partitioning on more than one dimension.
	<ul> <li>An integer value for the number of partitions you want created for this dimension.</li> </ul>

Table 3–2 (Cont.) ADD\_DIMENSION\_SOURCE Procedure Parameters

## 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

See Also

## 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 DE	EFAULT 20,
compressed IN BOOLEAN DE	EFAULT 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.

# 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 <i>cubename</i> value provided in the call to ADVISE
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 <i>cubename</i> .cp.
partnum	The number of a partition. By default, you see only the definition of the first partition.
advtable	The name of a table created by the SPARSITY_ADVICE_TABLE procedure for storing the results of analysis.

## Example

The following PL/SQL program fragment defines two variables to store the recommendations returned by the ADVISE\_DIMENSIONALITY function. SPARSEDIM stores the definition of the recommended composite, and DIMLIST stores the recommended dimension order of the cube.

## 

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.cpproduct 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

## 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	C OUT	CLOB,		
cubena	ame IN	VARCHAR2	,	
sparse	ename IN	VARCHAR2	DEFAULT	NULL,
dtype	IN	VARCHAR2	DEFAULT	'NUMBER',
advtab	ole IN	VARCHAR2	DEFAULT	NULL);

#### Parameters

Table 3–5	ADVISE_DIMENSIONALITY Procedure Pa	arameters
-----------	------------------------------------	-----------

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 <i>cubename</i> value provided in the call to ADVISE
sparsename	An object name for the sample composite. The default value is <i>cubename</i> . 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.

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 pl1 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 pl1 VALUES () <time sales cube composite pl1<>>)
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

## 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 AE	DVISE PARTITIONING	DIMENSION Function Parameters
--------------	--------------------	-------------------------------

Parameter	Description
cubename	The same <i>cubename</i> value provided in the call to ADVISE
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.

## 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

# 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**

Parameter	Description
cubename	The same <i>cubename</i> value provided in the call to ADVISE
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.

Table 3–7	ADVISE_PARTITIONING_	LEVEL Function Parameters
-----------	----------------------	---------------------------

#### 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

# 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**

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:					
	<ul> <li>DBMS_AW. PARTBY_DEFAULT Allow the Sparsity Advisor to determine whether or not partitioning is appropriate.</li> </ul>					
	<ul> <li>DBMS_AW.PARTBY_NONE Do not allow partitioning.</li> </ul>					
	<ul> <li>DBMS_AW.PARTBY_FORCE Force partitioning.</li> </ul>					
advtable	The name of a table created by the procedure for storing the results of analysis.					

Table 3–9 ADVISE\_SPARSITY Procedure Parameters

## **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 <i>cubename</i> in calls to ADVISE_SPARSITY, typically the name of the logical cube.
FACT	VARCHAR2(4000)	NOT NULL	The values of <i>fact</i> 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 <i>fact</i> (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

/

The following SELECT command displays the results of the analysis, which indicate that there is one denser dimension (CHANNEL) and three comparatively sparse dimensions (PRODUCT, CUSTOMER, and TIME).

SQL> SELECT fact, dimension, dimcolumn, membercount nmem, leafcount nleaf, advice, density
 FROM aw\_sparsity\_advice
 WHERE cubename='units\_cube';

FACT	DIMENSION	DIMCOLUMN	NMEM	NLEAF	ADVICE	DENSITY
units_history_fact	channel	channel_id	3	3	DENSE	.86545382
units_history_fact	product	item_id	36	36	SPARSE	.98706809
units_history_fact	customer	ship_to_id	61	62	SPARSE	.99257713
units_history_fact	time	month_id	96	80	SPARSE	.99415964

## See Also

# 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 <i>awname</i> .
awname	The name of an existing analytic workspace, unless <i>createaw</i> is specified as TRUE. See the description of <i>createaw</i> .
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 <i>awname</i> . If <i>awname</i> already exists, then an error is generated. FALSE attaches an existing analytic workspace named <i>awname</i> .
attargs	Keywords for attaching an analytic workspace, such as FIRST or LAST, as described in the <i>Oracle OLAP DML Reference</i> under the AW command.

#### Example

The following command attaches an analytic workspace named GLOBAL in read/write mode.

SQL>execute dbms\_aw.aw\_attach('global', true);

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

SQL>execute dbms\_aw.aw\_attach('global\_programs', true, true, 'last');

This command attaches an analytic workspace named SH from the SH\_AW schema in read-only mode.

SQL>execute dbms\_aw.aw\_attach('sh\_aw', 'sh');

See Also

"Managing Analytic Workspaces" on page 3-1.

# **AW\_COPY Procedure**

The AW\_COPY procedure copies the object definitions and data from one analytic workspace into a new analytic workspace.

 $\tt AW\_COPY$  detaches the original workspace and attaches the new workspace first with read/write access.

## Syntax

AW\_COPY (

τ.	(	
	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 <i>oldname</i> .
tablespace	The name of a tablespace in which <i>newname</i> 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\$ <i>newname</i> table.

## Example

The following command creates a new analytic workspace named DEMO and copies the contents of GLOBAL into it. The workspace is stored in a table named AW\$DEMO, which has three partitions and is stored in the user's default tablespace.

SQL>execute dbms\_aw.aw\_copy('global', 'demo', null, 3);

## See Also

"Managing Analytic Workspaces" on page 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 DE	EFAULT 8	);
AW_CREATE (					
	schema	IN	VARCHAR2	,	
	awname	IN	VARCHAR2	,	
	tablespace	IN	VARCHAR2	DEFAULT	NULL);

## **Parameters**

Parameter	Description
schema	The schema that owns <i>awname</i> .
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\$ <i>awname</i> , 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\$ <i>awname</i> table.

Table 3–13 AW\_CREATE Procedure Parameters

## Example

The following command creates a new, empty analytic workspace named GLOBAL. The new analytic workspace is stored in a table named AW\$GLOBAL with eight partitions in the user's default tablespace.

SQL>execute dbms\_aw.aw\_create('global');

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

SQL>execute dbms\_aw.aw\_create('global\_aw.demo', 'global', 2);

# AW\_DELETE

The AW\_DELETE procedure deletes an existing analytic workspace.

## Syntax

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

# Parameters

Table 3–14	AW_DELETE Procedure Parameters
------------	--------------------------------

Parameter	Description
schema	The schema that owns <i>awname</i> .
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 <i>awname</i> or have DBA rights to delete it, and it cannot currently be attached to your session. The AW\$ <i>awname</i> file is deleted from the database.

## Example

The following command deletes the GLOBAL analytic workspace in the user's default schema.

SQL>execute dbms\_aw.aw\_delete('global');

# **AW\_DETACH Procedure**

The AW\_DETACH procedure detaches an analytic workspace from your session so that its contents are no longer accessible. All changes that you have made since the last update are discarded. Refer to "AW\_UPDATE Procedure" on page 3-36 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 <i>awname</i> .
awname	The name of an attached analytic workspace that you want to detach from your session.

## Example

The following command detaches the GLOBAL analytic workspace.

SQL>execute dbms\_aw.aw\_detach('global');

# **AW\_RENAME** Procedure

The AW\_RENAME procedure changes the name of an analytic workspace.

## Syntax

AW_RENAME	(		
	oldname	IN	VARCHAR2,
	newname	IN	VARCHAR2 );

## Parameters

Table 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 command changes the name of the GLOBAL analytic workspace to DEMO.

SQL>execute dbms\_aw.aw\_rename('global', 'demo');

## See Also

"Procedure: Convert an Analytic Workspace 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;	

#### Returns

Name of a tablespace.

#### **Parameters**

Table 3–17 AW\_TABLESPACE Function Parameters

Parameter	Description
schema	The schema that owns <i>awname</i> .
awname	The name of an analytic workspace.

## Example

The following example shows the tablespace in which the GLOBAL analytic workspace is stored.

```
SQL> set serveroutput on
SQL> execute dbms_output.put_line('Global is stored in tablespace ' ||
        dbms_aw.aw_tablespace('GLOBAL_AW', 'GLOBAL'));
Global is stored in tablespace GLOBAL_DATA
```

PL/SQL procedure successfully completed.

# **AW\_UPDATE** Procedure

The AW\_UPDATE procedure saves the changes made to an analytic workspace in its permanent database table. For the updated version of this table to be saved in the database, you must issue a SQL COMMIT statement before ending your session.

If you do not specify a workspace to update, AW\_UPDATE updates all the user-defined workspaces that are currently attached with read/write access.

#### Syntax

AW_UPDATE	(				
	awname	IN	VARCHAR2	DEFAULT	NULL);
AW_UPDATE	(				
	schema	IN	VARCHAR2	DEFAULT	NULL,
	awname	IN	VARCHAR2	DEFAULT	NULL);

## **Parameters**

Table 3–18 AW\_UPDATE Procedure Parameters

Parameter	Description
schema	The schema that owns <i>awname</i> .
awname	Saves changes to <i>awname</i> by copying them to a table named AW\$ <i>awname</i> . If this parameter is omitted, then changes are saved for all analytic workspaces attached in read/write mode.

## Example

The following command saves changes to the GLOBAL analytic workspace to a table named AW\$GLOBAL.

SQL>execute dbms\_aw.aw\_update('global');

## See Also

"Managing Analytic Workspaces" on page 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 D	EFAULT);

## **Parameters**

Table 3–19 CONVERT Procedure Parameters

Parameter	Description
original_aw	The analytic workspace in 9 <i>i</i> storage format.
converted_aw	The same analytic workspace in 10g 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\_aw');

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_aw', 'global_aw_temp');
SQL>execute dbms_aw.convert ('global_aw_temp', 'global_aw');
SQL>execute dbms_aw.delete ('global_aw_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

## **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 <i>Oracle OLAP DML Reference</i>

## Example

The following example returns the value of the DECIMALS option in the current analytic workspace. The DECIMALS option controls the number of decimal places that are shown in numeric output. In this example, the value of DECIMALS is 2, which is the default.

```
SQL>set serveroutput on
SQL>select dbms_aw.eval_number('decimals') from dual;
```

```
DBMS_AW.EVAL_NUMBER('DECIMALS')
1 row selected.
```

# **EVAL\_TEXT** Function

The EVAL\_TEXT function evaluates a text expression in an analytic workspace and returns the resulting character string.

You can specify the EVAL\_TEXT function in a SELECT from DUAL statement to return a character constant defined in an analytic workspace. Refer to the *Oracle Database SQL Reference* for information on selecting from the DUAL table.

## Syntax

## **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	

## Example

The following example returns the value of the NLS\_LANGUAGE option, which specifies the current language of the OLAP session. The value of NLS\_LANGUAGE in this example is "AMERICAN".

```
SQL>set serveroutput on
SQL>select dbms_aw.eval_text('nls_language') from dual;
```

DBMS\_AW.EVAL\_TEXT('NLS\_LANGUAGE') AMERICAN 1 row selected.

# **EXECUTE** Procedure

The EXECUTE procedure executes one or more OLAP DML commands and directs the output to a printer buffer. It is typically used to manipulate analytic workspace data within an interactive SQL session. In contrast to the RUN Procedure, EXECUTE continues to process commands after it gets an error.

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

SET SERVEROUT ON

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

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

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

#### Syntax

EXECUTE	(		
	olap_commands	IN	VARCHAR2
	text	OUT	VARCHAR2);

#### Parameters

Table 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 sample SQL\*Plus session attaches an analytic workspace named XADEMO, creates a formula named COST\_PP in XADEMO, and displays the new formula definition.

SQL> set serveroutput on

SQL> execute dbms\_aw.execute('AW ATTACH xademo RW; DEFINE cost\_pp FORMULA LAG(analytic\_cube\_ f.costs, 1, time, LEVELREL time levelrel)');

PL/SQL procedure successfully completed.

SQL> execute dbms aw.execute('DESCRIBE cost pp');

DEFINE COST\_PP FORMULA DECIMAL <CHANNEL GEOGRAPHY PRODUCT TIME> EQ lag(analytic\_cube\_f.costs, 1, time, levelrel time.levelrel)

PL/SQL procedure successfully completed.

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

SQL> execute dbms\_aw.execute('call nothing; colwidth=20'); BEGIN dbms\_aw.execute('call nothing; colwidth=20'); END; \* ERROR at line 1: ORA-34492: Analytic workspace object NOTHING does not exist. ORA-06512: at "SYS.DBMS\_AW", line 90 ORA-06512: at "SYS.DBMS\_AW", line 119 ORA-06512: at line 1 SQL> execute dbms\_aw.execute('show colwidth');

20

 $\ensuremath{\texttt{PL}}\xspace/\ensuremath{\texttt{SQL}}\xspace$  procedure successfully completed.

# **GETLOG Function**

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

To print the session log returned by this function, use the DBMS\_AW.PRINTLOG procedure.

## Syntax

GETLOG() RETURN CLOB;

#### Returns

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

#### Example

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

PL/SQL procedure successfully completed.

```
SQL>execute dbms_aw.printlog(dbms_aw.getlog());
    2 AGGMAPs
    ANALYTIC_CUBE.AGGMAP.1
    SALES_MULTIKEY_CUBE.AGGMAP.1
```

PL/SQL procedure successfully completed.

# **INFILE Procedure**

The INFILE procedure evaluates the OLAP DML commands in the specified file and executes them in the current analytic workspace.

## **Syntax**

INFILE (

filename IN VARCHAR2);

## **Parameters**

Table 3–23	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 specified in the file test\_setup.tst. The directory path of the file is specified in the OLAP directory object called work\_dir.

```
SQL>execute dbms_aw.execute('cda work_dir');
SQL>execute dbms_aw.infile('test_setup.tst');
```

# **INTERP** Function

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

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

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

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

#### Syntax

INTERP ( olap-commands IN VARCHAR2) RETURN CLOB;

#### Parameters

Table 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.

#### Returns

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

## Example

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

## **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**

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.

#### Returns

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

## Example

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

SQL> set serverout on size 1000000 SQL> execute dbms\_aw.printlog(dbms\_aw.interpclob('AW CREATE electronics; IMPORT ALL FROM EIF FILE ''dbs/electronics.eif'' DATA DFNS; DESCRIBE'));

DEFINE GEOGRAPHY DIMENSION TEXT WIDTH 12 LD Geography Dimension Values DEFINE PRODUCT DIMENSION TEXT WIDTH 12 LD Product Dimension Values DEFINE TIME DIMENSION TEXT WIDTH 12 LD Time Dimension Values DEFINE CHANNEL DIMENSION TEXT WIDTH 12 LD Channel Dimension Values

PL/SQL procedure successfully completed.

# **INTERP\_SILENT** Procedure

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

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

#### Syntax

INTERP\_SILENT ( olap-commands IN VARCHAR2);

## **Parameters**

Table 3–26         INTERP_SILENT Function Parameters	
Parameter	Description
olap-commands	One or more OLAP DML commands separated by semi-colons. See "Guidelines for Using Quotation Marks in OLAP DML Commands" on page 3-4.

#### Example

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

SQL> execute dbms\_aw.execute('AW ATTACH xademo'); IMPORTANT: Analytic workspace XADEMO is read-only. Therefore, you will not be able to use the UPDATE command to save changes to it.

PL/SQL procedure successfully completed.

SQL>execute dbms\_aw.interp\_silent('AW ATTACH xademo');

PL/SQL procedure successfully completed.

# **OLAP\_ON Function**

The  ${\tt OLAP\_ON}$  function returns a boolean indicating whether or not the OLAP option is installed in the database.

# Syntax

OLAP\_ON ( ) RETURN BOOLEAN;

## Returns

The value of the <code>OLAP</code> parameter in the <code>V\$OPTION</code> table.

# **OLAP\_RUNNING Function**

The OLAP\_RUNNING function returns a boolean indicating whether or not the OLAP option has been initialized in the current session. Initialization occurs when you execute an OLAP DML command (either directly or by using an OLAP PL/SQL or Java package), query an analytic workspace, or execute the STARTUP Procedure.

## Syntax

```
OLAP_RUNNING()
RETURN BOOLEAN;
```

## Returns

TRUE if OLAP has been initialized in the current session, or FALSE if it has not.

## Example

The following PL/SQL script tests whether the OLAP environment has been initialized, and starts it if not.

# BEGIN IF DBMS\_AW.OLAP\_RUNNING() THEN DBMS\_OUTPUT.PUT\_LINE('OLAP is already running'); ELSE DBMS\_AW.STARTUP;

```
DBMS_AW.STARTUP;
IF DBMS_AW.OLAP_RUNNING() THEN
DBMS_OUTPUT.PUT_LINE('OLAP started successfully');
ELSE
DBMS_OUTPUT.PUT_LINE('OLAP did not start. Is it installed?');
END IF;
END IF;
END;
/
```
## **PRINTLOG Procedure**

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

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

SET SERVEROUT ON SIZE 1000000

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

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

#### Syntax

PRINTLOG (

session-loq 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('DESCRIBE analytic\_cube\_f.profit'));

DEFINE ANALYTIC\_CUBE.F.PROFIT FORMULA DECIMAL <CHANNEL GEOGRAPHY PRODUCT TIME> EQ analytic\_cube.f.sales - analytic\_cube.f.costs

PL/SQL procedure successfully completed.

## **RUN Procedure**

The RUN procedure executes one or more OLAP DML commands and directs the output to a printer buffer. It is typically used to manipulate analytic workspace data within an interactive SQL session. In contrast to the EXECUTE Procedure, RUN stops processing commands when it gets an error.

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

SET SERVEROUT ON

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

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

#### Syntax

RUN	(			
		olap commands	IN	STRING,
		silent	IN	BOOLEAN DEFAULT FALSE);
RUN	(			
		olap_commands	IN	CLOB,
		silent	IN	BOOLEAN DEFAULT FALSE);
RUN	(			
		olap_commands	IN	STRING,
		output	OUT	STRING);
RUN	(	-		
		olap commands	IN	STRING,
		output	IN OUT	CLOB);
RUN	(	-		
		olap commands	IN	CLOB,
		output	OUT	STRING);
RUN	(	-		
	•	olap_commands	IN	CLOB,
		output	IN OUT	CLOB);
			001	/

#### **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> set serveroutput on SQL> execute dbms\_aw.run('AW ATTACH xademo RW; DEFINE cost\_pp FORMULA LAG(analytic\_cube\_ f.costs, 1, time, LEVELREL time\_levelrel)'); PL/SQL procedure successfully completed.

SQL> execute dbms\_aw.run('DESCRIBE cost\_pp');

DEFINE COST\_PP FORMULA DECIMAL <CHANNEL GEOGRAPHY PRODUCT TIME> EQ lag(analytic\_cube\_f.costs, 1, time, levelrel time.levelrel)

PL/SQL procedure successfully completed.

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

```
SQL> execute dbms_aw.execute('show colwidth');
10
PL/SQL procedure successfully completed.
```

- - -

SQL> execute dbms\_aw.run('call nothing; colwidth=20'); BEGIN dbms\_aw.run('call nothing; colwidth=20'); END;

```
ERROR at line 1:
ORA-34492: Analytic workspace object NOTHING does not exist.
ORA-06512: at "SYS.DBMS_AW", line 55
ORA-06512: at "SYS.DBMS_AW", line 131
ORA-06512: at line 1
SQL> execute dbms_aw.execute('show colwidth');
```

10

\*

PL/SQL procedure successfully completed.

# **SHUTDOWN Procedure**

The SHUTDOWN procedure terminates the current OLAP session.

By default, the SHUTDOWN procedure terminates the session only if there are no outstanding changes to any of the attached read/write workspaces. If you want to terminate the session without updating the workspaces, specify the force parameter.

## Syntax

SHUTDOWN ( force IN BOOLEAN DEFAULT NO);

## Parameters

Parameter	Description	
force	When YES, this parameter forces the OLAP session to shutdown even though one or more attached workspaces has not been updated. Default is NO.	

#### Table 3–29 SHUTDOWN Procedure Parameters

# 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**

Parameter	Description
tblname	The name of the table. The default name is AW_SPARSITY_ ADVICE, which is created in your own schema.

## Example

The following example creates a table named GLOBAL\_SPARSITY\_ADVICE.

execute dbms\_aw.sparsity\_advice\_table('global\_sparsity\_advice');

## See Also

ADVISE\_SPARSITY Procedure on page 3-25 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 ( );

# DBMS\_AW\_XML

The DBMS\_AW\_XML package builds an analytic workspace based on a logical model described in an XML document. The XML can be created using the Oracle OLAP Analytic Workspace Java API.

This chapter includes the following topics:

- Analytic Workspace Java API Overview
- Oracle OLAP XML Schema
- Summary of DBMS\_AW\_XML Subprograms

## Analytic Workspace Java API Overview

The Oracle OLAP Analytic Workspace API is a Java API for building and maintaining standard form analytic workspaces. The API provides classes for describing a logical cube, mapping the cube to a relational data source, and aggregating the cube's data. You can also use the API to specify complex solves, such as allocations and forecasts, and define custom measures and custom dimension members.

The Analytic Workspace API supports two deployment modes: It can be embedded in a Java application, or it can be used to generate XML that serializes the object model for execution by the EXECUTE function. The functionality of the API is identical whether executed from a Java client through JDBC or directly in the database through SQL.

The Analytic Workspace API does not use OLAP Catalog metadata.

# Oracle OLAP XML Schema

The EXECUTE and EXECUTEFILE functions process XML that conforms to the Oracle OLAP XML schema defined in awxml.xsd. The XML generated by the Analytic Workspace Java API automatically conforms to awxml.xsd. You can also create your own XML and validate it against the Oracle OLAP XML schema.

Example 4–1 provides an excerpt from an XML document that conforms to the Oracle OLAP XML schema.

**Tip:** You can obtain AWXML.xsd, as well as the latest version of the *Oracle OLAP Analytic Workspace Java API Reference* (Javadoc), from the Oracle Technology Network Web site:

http://www.oracle.com/technology/products/bi/olap/in
dex.html

#### Example 4–1 Oracle OLAP XML Document

```
<AWXML version = '1.0' timestamp = 'Mon Feb 11 13:29:11 2002' >
<AWXML.content>
 <Create Id="Action3">
   <ActiveObject >
      <AW Name="GLOBAL AW.GLOBAL" LongName="GLOBAL AW.GLOBAL"
          ShortName="GLOBAL AW.GLOBAL" PluralName="GLOBAL AW.GLOBAL"
          Id="GLOBAL AW.GLOBAL.AW" Schema="GLOBAL AW" MetaDataFormat="10.2"
          DefaultLanguage="AMERICAN" Languages="AMERICAN">
        <Dimension Name="TIME" LongName="AMERICAN::Time"</pre>
             ShortName="AMERICAN::Time" PluralName="AMERICAN::Time"
             Id="TIME.DIMENSION" Schema="GLOBAL_AW" isTime="true"
             isMeasure="false" UseNativeKey="true">
          <Attribute Name="END DATE" LongName="AMERICAN::END DATE"</pre>
             ShortName="AMERICAN::END DATE" PluralName="AMERICAN::END DATE"
             Id="TIME.END DATE.ATTRIBUTE" DataType="DATE"
             Classification="END DATE" InstallAsRelation="false"
             IsDefaultOrder="false"/>
          <Attribute Name="TIME SPAN" LongName="AMERICAN::TIME SPAN"</pre>
             ShortName="AMERICAN::TIME SPAN" PluralName="AMERICAN::TIME SPAN"
             Id="TIME.TIME SPAN.ATTRIBUTE" DataType="INTEGER"
             Classification="TIME_SPAN" InstallAsRelation="false"
             IsDefaultOrder="false"/>
          <Attribute Name="LONG DESCRIPTION"
             LongName="AMERICAN::Long Description"
             ShortName="AMERICAN::Long Description"
             PluralName="AMERICAN::Long Descriptions"
             Id="TIME.LONG DESCRIPTION.ATTRIBUTE" DataType="TEXT"
             Classification="MEMBER LONG DESCRIPTION" InstallAsRelation="false"
             IsDefaultOrder="false" IsMultiLingual="true"/>
```

# Summary of DBMS\_AW\_XML Subprograms

The following table describes the subprograms provided in DBMS\_AW\_EXECUTE.

Subprogram	Description	
EXECUTE Function on page 4-4	Creates all or part of a standard form analytic workspace from an XML document stored in a CLOB.	
EXECUTEFILE Function on page 4-6	Creates all or part of a standard form analytic workspace from an XML document stored in a text file.	

Table 4–1 DBMS\_AW\_XML Subprograms

## **EXECUTE** Function

The EXECUTE function builds an analytic workspace using XML that conforms to the Oracle OLAP XML schema. The XML is stored in a database object.

#### Syntax

#### **Parameters**

Table 4–2 EXECUTE Function Parameters

Parameter	Description
xml_input	An XML document stored in a CLOB. The XML contains instructions for building or maintaining an analytic workspace. The XML describes a logical model, provides instructions for loading data from relational tables, and defines aggregation and other calculations to be performed on the data in the workspace.

#### Example

The following SQL program creates a CLOB and loads into it the contents of an XML file. It then creates an analytic workspace named GLOBAL in the GLOBAL\_AW schema from the XML document in the CLOB.

```
--Use DBMS LOB package to create a clob
  DECLARE
     clb CLOB;
     infile BFILE;
     dname varchar2(500);
  BEGIN
  -- Create a temporary clob
     DBMS_LOB.CREATETEMPORARY(clb, TRUE,10);
  -- Create a BFILE use BFILENAME function
  -- Use file GLOBAL.XML in the SCRIPTS directory object.
     infile := BFILENAME('SCRIPTS', 'GLOBAL.XML');
  -- Open the BFILE
     DBMS LOB.fileopen(infile, dbms lob.file readonly);
  -- Load temporary clob from the BFILE
     DBMS LOB.LOADFROMFILE(clb, infile, DBMS LOB.LOBMAXSIZE, 1, 1);
  -- Close the BFILE
     DBMS LOB.fileclose(infile);
  -- Create the GLOBAL analytic workspace
     DBMS OUTPUT.PUT LINE(DBMS AW XML.execute(clb));
     DBMS AW.AW UPDATE;
     COMMIT;
  -- Free the Temporary Clob
     DBMS LOB.FREETEMPORARY(clb);
```

EXCEPTION

```
WHEN OTHERS
THEN
DBMS_OUTPUT.PUT_LINE(SQLERRM);
END;
/
```

# **EXECUTEFILE** Function

The EXECUTEFILE function builds an analytic workspace using XML that conforms to the Oracle OLAP XML schema. The XML is stored in a text file.

#### **Syntax**

EXECUTEFILE (			
d	irname	IN	VARCHAR2
filename		IN	VARCHAR2)
RETURN	VARCHAR2	2;	

#### Returns

The string SUCCESS if successful

#### Parameters

Parameter	Description	
dirname	A directory object that identifies the physical directory where <i>xml_file</i> is stored.	
xml_file	The name of a text file containing an XML document. The XML contains instructions for building or maintaining an analytic workspace. The XML describes a logical model, provides instructions for loading data from relational tables, and defines aggregation and other calculations to be performed on the data in the workspace.	

Table 4–3 EXECUTEFILE Function Parameters

## Example

The following EXECUTEFILE function generates a standard form analytic workspace from the XML statements stored in GLOBAL.XML, which is located in a directory identified by the SCRIPTS directory object. The DBMS\_OUTPUT.PUT\_LINE function displays the "Success" message returned by EXECUTEFILE.

SQL> execute dbms\_output.put\_line(dbms\_aw\_xml.executefile('SCRIPTS', 'GLOBAL.XML')); Success

# 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
- Viewing the Configuration Table
- Summary of OLAP\_API\_SESSION\_INIT Subprograms

## Initialization Parameters for the OLAP API

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

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

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

The information in the table can be queried through the ALL\_OLAP\_ALTER\_SESSION view alias, which is also described in this chapter.

**Note:** This package is owned by the SYS user. You must explicitly be granted execution rights before you can use it.

## Viewing the Configuration Table

ALL\_OLAP\_ALTER\_SESSION is the public synonym for V\$OLAP\_ALTER\_SESSION, which is a view for the OLAP\$ALTER\_SESSION table. The view and table are owned by the SYS user.

## ALL\_OLAP\_ALTER\_SESSION View

Each row of ALL\_OLAP\_ALTER\_SESSION identifies a role and a session initialization parameter. When a user opens a session using the OLAP API, the session is initialized using the parameters for roles granted to that user. For example, if there are rows for

the OLAP\_DBA role and the SELECT\_CATALOG\_ROLE, and a user has the OLAP\_DBA role, then the parameters for the OLAP\_DBA role will be set, but those for the SELECT\_CATALOG\_ROLE will be ignored.

Table 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  ${\tt OLAP\_API\_SESSION\_INIT}.$ 

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.

Table 5–2 OLAP\_API\_SESSION\_INIT Subprograms

# 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 call inserts a row in OLAP\$ALTER\_SESSION that turns on query rewrite for users with the OLAP DBA role.

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

The ALL\_OLAP\_ALTER\_SESSION view now contains the following row.

ROLE	CLAUSE TEST
OLAP_DBA	ALTER SESSION SET QUERY_REWRITE_ENABLED=TRUE

## **CLEAN\_ALTER\_SESSION Procedure**

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

## **Syntax**

CLEAN\_ALTER\_SESSION ();

## **Examples**

The following call deletes all orphaned rows.

call olap\_api\_session\_init.clean\_alter\_session();

## DELETE\_ALTER\_SESSION Procedure

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

#### Syntax

DELETE\_ALTER\_SESSION ( role\_name IN VARCHAR2, session\_parameter IN VARCHAR2);

#### Parameters

The role\_name and session\_parameter together uniquely identify a row in OLAP\$ALTER SESSION.

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.

Table 5–4 DELETE\_ALTER\_SESSION Procedure Parameters

#### Examples

The following call deletes a row in OLAP\$ALTER\_SESSION that contains a value of OLAP\_DBA in the first column and QUERY\_REWRITE\_ENABLED=TRUE in the second column.

call olap\_api\_session\_init.delete\_alter\_session(
 'OLAP\_DBA', 'SET\_QUERY\_REWRITE\_ENABLED=TRUE');

# **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 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',
         ۲۲,
         '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
_____
  211680.12224713.71Hardware32000300...195591.60207513.16Hardware32001300...184413.05194773.78Hardware32002300...73457.3177275.06Hardware32003300...
```

#### 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,
	<pre>- 'limit time to time short description eq ''2000''', 0)=1</pre>
	ORDER BY aw_time;
AW_UNIT_COST	AW_UNIT_PRICE AW_PRODUCT AW_PRODUCT_GID AW_TIME AW_TIME_GID R2C
211680.12	224713.71 Hardware 3 2000 3 00

```
--Check status in the analytic workspace
SQL>exec dbms_aw.execute('rpr time_short_description');
TIME TIME_SHORT_DESCRIPTION
---- 3 2000
-- Reset status
SQL>exec dbms_aw.execute('allstat');
```

In the next query, OLAP\_CONDIITON uses entry point 1. The limited data is returned by OLAP TABLE, but the limit does not remain in effect in the analytic workspace.

Note that the third parameter is not required in this case, since entry point 1 is the default.

```
SQL>SELECT * FROM unit_cost_price_view
          WHERE aw product = 'Hardware'
          AND aw time in ('2000', '2001', '2002', '2003')
          AND OLAP CONDITION (r2c,
                 'limit time to time_short_description eq ''2000''', 1)=1
          ORDER BY aw time;
AW_UNIT_COST AW_UNIT_PRICE AW_PRODUCT AW_PRODUCT GID AW_TIME AW_TIME GID R2C
211680.12 224713.71 Hardware 3 2000 3 00...
--Check status in the analytic workspace
SQL>exec dbms aw.execute('rpr time short description');
TIME TIME SHORT DESCRIPTION
----
      ------
19
     Jan-98
     Feb-98
20
21
     Mar-98
22
     Apr-98
.
     1998
 1
     1999
 2
 3
     2000
     2001
 4
85
      2002
102
      2003
119
      2004
-- Reset status
SQL>exec dbms aw.execute('allstat');
```

In the final query, OLAP\_CONDIITON uses entry point 2. The limit does not affect the data returned by OLAP\_TABLE, but the limit remains in effect in the analytic workspace.

AW\_UNIT\_COST AW\_UNIT\_PRICE AW\_PRODUCT AW\_PRODUCT\_GID AW\_TIME AW\_TIME\_GID R2C 211680.12224713.71Hardware32000300...195591.60207513.16Hardware32001300...184413.05194773.78Hardware32002300...73457.3177275.06Hardware32003300...

--Check status in the analytic workspace

SQL>exec dbms\_aw.execute('rpr time\_short\_description');

TIME\_SHORT\_DESCRIPTION TIME ----2000 -----

3

# **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**

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 <i>r</i> 2 <i>c</i> parameter. For information on the OLAP DML, see the <i>Oracle OLAP DML Reference</i> .
event	The event during OLAP_TABLE processing that will trigger the execution of the OLAP DML command specified by the <i>expression</i> parameter. This parameter can have the value 0, 1, or 2, as described in Table 6–2

#### Table 6–1 OLAP\_CONDITION Function Parameters

#### Returns

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 11-23 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 11-23.
	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).

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 11-23.
	If an OLAP DML command (other than FETCH) is specified in the <i>olap_command</i> parameter of OLAP_TABLE, it is executed after OLAP_CONDITION and before the data is fetched. (The use of a FETCH command in the <i>olap_command</i> 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 11-23.
	If OLAP_CONDITION limits any dimensions, the limits remain in the analytic workspace after the query completes.

#### Table 6–2 (Cont.) Entry Points for OLAP\_CONDITION in the OLAP\_TABLE Limit Map

## Example

See "OLAP\_CONDITION Examples" on page 6-2.

# **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 VARCHAR
                               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',
         ۰',
         '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_pri WHERE aw_product = AND aw_time in ('2 ORDER BY aw_time;	= 'Hardware'	'2002', '20	03')
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.

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.

#### 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.

#### Returns

An evaluation of *numeric\_expression* for each row of the table object returned by the OLAP\_TABLE function.

OLAP\_EXPRESSION returns numeric data. To return text, boolean, or date data, use the OLAP\_EXPRESSION\_TEXT, OLAP\_EXPRESSION\_BOOL, or OLAP\_EXPRESSION\_DATE functions.

#### Example

See "OLAP\_EXPRESSION Examples" on page 7-2.

# 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 (
             awtimeVARCHAR2(12),awcustomerVARCHAR2(30),awproductVARCHAR2(30),awchannelVARCHAR2(30),awunitsNUMBER(16),r2cRAW(32));
             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',
         ۰',
          'MEASURE awunits FROM units cube aw units aw
          DIMENSION awtime FROM time aw WITH
             HIERARCHY time aw parentrel
           DIMENSION awcustomer FROM customer_aw WITH
             HIERARCHY customer aw parentrel
                        (customer_aw_hierlist ''MARKET_ROLLUP AW'')
                 INHIERARCHY customer aw inhier
          DIMENSION awproduct FROM product aw WITH
             HIERARCHY product_aw_parentrel
           DIMENSION channel aw WITH
             HIERARCHY channel aw parentrel
             ATTRIBUTE awchannel FROM channel_aw_short_description
          ROW2CELL r2c'))
      WHERE awunits IS NOT NULL;
```

/

The following query shows some of the aggregate data in the view. For all products in all markets during the year 2001, it shows the number of units sold through each channel.

```
SQL> SELECT awchannel, awunits FROM awunits_view
WHERE awproduct = '1'
AND awcustomer = '7'
AND awtime = '4';
```

AWCHANNEL	AWUNITS
All Channels	415392
Direct Sales	43783
Catalog	315737
Internet	55872

The following statements show the descriptions of the Product, Customer, and Time dimension members used in the query.

SQL>execute dbms\_aw.execute('limit product\_aw to ''1'''); SQL>execute dbms\_aw.execute('rpr product\_aw\_short\_description');

PRODUCT\_AW\_SHORT\_DESCRIPTION PRODUCT\_AW \_\_\_\_\_ 1 Total Product SQL>execute dbms aw.execute('limit customer aw to ''7'''); SQL>execute dbms\_aw.execute('rpr customer\_aw\_short\_description'); CUSTOMER AW SHORT DESCRIPTION CUSTOMER AW \_\_\_\_\_ 7 Total Market SQL>execute dbms\_aw.execute('limit time\_aw to ''4'''); SQL>execute dbms\_aw.execute('rpr time\_aw\_short\_description'); TIME AW TIME\_AW\_SHORT\_DESCRIPTION \_\_\_\_\_ 4 2001

#### Example 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
            AND awchannel = 'Internet'
AND awtime = '4';
PRODUCTS LOWEST UNITS
-----
40
            0
            1
41
            1
42
            1
43
            1
44
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	<pre>dbms_aw.execute('rpr product_aw_short_description');</pre>	
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**

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 <i>Oracle OLAP DML Reference</i> . 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.

# 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

#### **Parameters**

rs
9

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 <i>Oracle OLAP DML Reference</i> . 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.

# 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

#### **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 <i>Oracle OLAP DML Reference</i> . See also "Guidelines for Using Quotation Marks in OLAP DML Commands" on page 3-4.			

Table 10–1 OLAP\_EXPRESSION\_TEXT Function Parameters

#### 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.

#### See Also:

- Oracle OLAP Application Developer's Guide
- Oracle OLAP DML Reference
- Chapter 6, "OLAP\_CONDITION"
- Chapter 7, "OLAP\_EXPRESSION"

This chapter contains the following topics:

- OLAP\_TABLE Overview
- OLAP\_TABLE Examples
- OLAP\_TABLE Syntax

# **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

```
'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',
         ۰۰,
         '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

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.

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)

Table 11–1 Default Data Type Conversions

## 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 11-23.

Example 11–5 Script for an Embedded Total Dimension View Using OLAP\_TABLE

```
CREATE TYPE awtime row AS OBJECT (
              awtime_idVARCHAR2(12),awtime_short_labelVARCHAR2(12),awtime_cal_end_dateDATE,awtime_fis_end_dateDATE);
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',
           ۰۰,
           'DIMENSION awtime id FROM time WITH
               HIERARCHY time parentrel
                  (time hierlist ''CALENDAR'')
                  INHIERARCHY time inhier
                  HATTRIBUTE awtime_cal_end_date FROM time_cal_end_date
               HIERARCHY time_parentrel
                   (time hierlist ''FISCAL'')
                  INHIERARCHY time inhier
                  HATTRIBUTE awtime_fis_end_date FROM time_fis_end_date
           ATTRIBUTE awtime short label FROM time short description'));
SQL>SELECT * FROM awtime view;
AWTIME_ID AWTIME_SHORT_LABEL AWTIME_CAL_END_DATE AWTIME_FIS_END_DATE
----- -----
19 Jan-98 31-JAN-98 31-JAN-98

        20
        Feb-98
        28-FEB-98

        21
        Mar-98
        31-MAR-98

        22
        Apr-98
        30-APR-98

        23
        May-98
        31-MAY-98

20
          Feb-98
                                                          28-FEB-98
                                 28-FEB-98
                                                     31-MAR-98
30-APR-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 (
            awtimeVARCHAR2(12),awcustomerVARCHAR2(30),awproductVARCHAR2(30),awchannelVARCHAR2(30),
             awunits
                                    NUMBER(16),
                                     RAW(32));
             r2c
CREATE TYPE awunits table AS TABLE OF awunits row;
CREATE OR REPLACE VIEW awunits_view AS
   SELECT awunits,
         awtime, awcustomer, awproduct, awchannel, r2c
      FROM TABLE (OLAP TABLE (
         'global DURATION SESSION',
         'awunits table',
         · · ,
         'MEASURE awunits FROM units cube units
          DIMENSION awtime FROM time WITH
             HIERARCHY time_parentrel
          DIMENSION awcustomer FROM customer WITH
             HIERARCHY customer parentrel
                       (customer hierlist ''MARKET_ROLLUP'')
                INHIERARCHY customer inhier
          DIMENSION awproduct FROM product WITH
            HIERARCHY product_parentrel
          DIMENSION channel WITH
             HIERARCHY channel parentrel
             ATTRIBUTE awchannel FROM channel short description
```

```
ROW2CELL r2c'))

WHERE awunits IS NOT NULL;

SQL>SELECT awchannel, awunits FROM awunits_view

WHERE awproduct = '1'

AND awcustomer = '7'

AND awtime = '4';

AWCHANNEL AWUNITS

------

All Channels 415392

Direct Sales 43783

Catalog 315737

Internet 55872
```

#### Example: Creating Views in Rollup Form

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

**See Also:** "Limit Map: DIMENSION Clause: WITH HIERARCHY Subclause" on page 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));
            item
/
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',
         ۰۰,
         'DIMENSION product WITH
            HIERARCHY product parentrel
               FAMILYREL null, class, family, item
                  FROM product familyrel USING product levellist
                  LABEL product_short_description'));
```

SQL>	SELECT	*	FROM awproduct_view	
			ORDER BY class, family, item;	

CLASS	FAMILY	ITEM
Hardware Hardware Hardware Hardware	CD-ROM CD-ROM CD-ROM CD-ROM	Envoy External 6X CD-ROM Envoy External 8X CD-ROM External 6X CD-ROM External 8X CD-ROM
Hardware Hardware Hardware Hardware	CD-ROM CD-ROM CD-ROM Desktop PCs	Internal 6X CD-ROM Internal 8X CD-ROM Sentinel Financial
Hardware	Desktop PCs	Sentinel Multimedia
Software/Other Software/Other Software/Other Software/Other	Operating Systems Operating Systems Operating Systems	Unix/Windows 1-user pack Unix/Windows 5-user pack

#### Example 11–8 Script Using QDRs in the FAMILYREL Clause of OLAP\_TABLE

```
CREATE OR REPLACE TYPE awproduct row AS OBJECT (
                 class VARCHAR2(50),
                             VARCHAR2(50),
                 family
                 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',
            ۰۰,
            'DIMENSION product WITH
               HIERARCHY product parentrel
                   FAMILYREL class, family, item FROM
                       product familyrel(product levellist ''CLASS''),
                       product_familyrel(product_levellist ''FAMILY''),
                       product_familyrel(product_levellist ''ITEM'')
                       LABEL product short description'));
SQL> SELECT * FROM awproduct view
                 ORDER BY by class, family, item;
CLASS
         FAMILY
                                       ITEM
-----
                                              -----
HardwareCD-ROMEnvoy External 6X CD-ROMHardwareCD-ROMEnvoy External 8X CD-ROMHardwareCD-ROMExternal 6X CD-ROMHardwareCD-ROMExternal 8X CD-ROMHardwareCD-ROMInternal 6X CD-ROMHardwareCD-ROMInternal 6X CD-ROMHardwareCD-ROMInternal 8X CD-ROMHardwareCD-ROMInternal 8X CD-ROMHardwareDesktop PCsSentinel FinancialHardwareDesktop PCsSentinel Multimedia
```

```
.
Software/Other Operating Systems Unix/Windows 1-user pack
Software/Other Operating Systems Unix/Windows 5-user pack
Software/Other Operating Systems
```

# 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 ( VARCHAR2(20), time VARCHAR2(30), geography VARCHAR2(30), product VARCHAR2(30), channel NUMBER(16), sales NUMBER(16), cost cost\_prior\_period NUMBER(16), NUMBER(16)); profit CREATE TYPE measure table AS TABLE OF measure row; / SELECT time, geography, product, channel, sales, cost, cost\_prior\_period, profit FROM TABLE (OLAP TABLE ( 'xademo DURATION SESSION', 'measure table', 'FETCH time, geography, product, channel, analytic cube f.sales, analytic cube f.costs, LAG(analytic\_cube\_f.costs, 1, time, LEVELREL time\_member\_levelrel), analytic cube f.sales - analytic cube f.costs', '')) WHERE channel = 'STANDARD\_2.TOTALCHANNEL' AND
product = 'L1.TOTALPROD' AND geography = 'L1.WORLD' ORDER BY time;

This SQL  $\ensuremath{\texttt{SELECT}}$  statement returns the following result set:

TIME (	GEOGRAPHY	PRODUCT	CHANNEL	SALES	COST	COST_PRIOR_PERIOD	PROFIT
L1.1996 I	L1.WORLD	L1.TOTALPROD	STANDARD_2.TOTALCHANNEL	118247112	2490243		115756869
L1.1997 I	L1.WORLD	L1.TOTALPROD	STANDARD_2.TOTALCHANNEL	46412113	1078031	2490243	45334082
L2.Q1.96 I	L1.WORLD	L1.TOTALPROD	STANDARD_2.TOTALCHANNEL	26084848	560379		25524469
L2.Q1.97 I	L1.WORLD	L1.TOTALPROD	STANDARD_2.TOTALCHANNEL	26501765	615399	560379	25886367
L2.Q2.96 I	L1.WORLD	L1.TOTALPROD	STANDARD_2.TOTALCHANNEL	30468054	649004	615399	29819049
L2.Q2.97 I	L1.WORLD	L1.TOTALPROD	STANDARD_2.TOTALCHANNEL	19910347	462632	649004	19447715
L2.Q3.96 I	L1.WORLD	L1.TOTALPROD	STANDARD_2.TOTALCHANNEL	27781702	582693	462632	27199009
L2.Q4.96 I	L1.WORLD	L1.TOTALPROD	STANDARD_2.TOTALCHANNEL	33912508	698166	582693	33214342
L3.APR96 I	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 11-23.

#### 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_map18	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 11-23.

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 11-23.

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.

```
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.

Instructions for creating these workspace objects are provided in the *Oracle OLAP Application Developer's Guide*.

#### 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]
   . . .
   1
   [ATTRIBUTE column [AS datatype] FROM attribute variable]
   . . .
1
```

#### 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],

... columnn [AS datatype] FROM {expression1, expression2, ... expressionn | family\_relation USING level\_dimension } [LABEL label\_variable]

The FAMILYREL subclause is used primarily to map a family relation in the analytic workspace to multiple columns in the target table. List the columns in the order of *level\_dimension* (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 11-23.

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 11-23.

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

#### Α

abstract data types, 11-2 automatic, 11-3 predefining, 11-2 Active Catalog, 1-1, 11-1 ADD\_DIMENSION\_SOURCE procedure, 3-14 ADT See abstract data types ADVISE\_CUBE procedure, 3-16 ADVISE\_DIMENSIONALITY function, 3-18 ADVISE\_DIMENSIONALITY procedure, 3-20 ADVISE\_REL procedure, 3-8, 3-24 ADVISE\_SPARSITY procedure, 3-25 Aggregate Advisor, 3-8 to 3-11 aggregate cache performance statistics, 2-3 aggregation in analytic workspaces, 3-8 to 3-11 operators, 2-2 aggregation specifications, 1-5 ALL\_OLAP2\_AW\_ATTRIBUTES view, 1-2 ALL\_OLAP2\_AW\_CUBE\_AGG\_LVL view, 1-4 ALL\_OLAP2\_AW\_CUBE\_AGG\_MEAS view, 1-4 ALL\_OLAP2\_AW\_CUBE\_AGG\_OP view, 1-4 ALL\_OLAP2\_AW\_CUBE\_AGG\_SPECS view, 1-5 ALL\_OLAP2\_AW\_CUBE\_DIM\_USES view, 1-5 ALL\_OLAP2\_AW\_CUBE\_MEASURES view, 1-5 ALL\_OLAP2\_AW\_CUBES view, 1-3 ALL\_OLAP2\_AW\_DIM\_HIER\_LVL\_ORD view, 1-6 ALL\_OLAP2\_AW\_DIM\_LEVELS view, 1-7 ALL\_OLAP2\_AW\_DIMENSIONS view, 1-6 ALL\_OLAP2\_AW\_MAP\_ATTR\_USE view (obsolete) See ALL\_OLAP2\_AW\_ATTRIBUTES view ALL\_OLAP2\_AW\_MAP\_DIM\_USE view (obsolete) See ALL\_OLAP2\_AW\_DIMENSIONS view ALL\_OLAP2\_AW\_MAP\_MEAS\_USE view (obsolete) See ALL\_OLAP2\_AW\_CUBE\_MEASURES view ALL\_OLAP2\_AWS view, 1-2 allocation operators, 2-3 ALTER SESSION commands, 5-1 analytic workspace management APIs, 4-1 Analytic Workspace Manager, 11-1 analytic workspace objects defining in XML, 4-1 analytic workspaces

accessing from SQL, 3-1 to 3-54 aggregation, 3-8 converting to 10g storage format, 1-2, 3-2 creating with OLAP Analytic Workspace Java API. 4-1 creating with XML, 4-1 importing from Oracle 9i, 3-2 list of, 1-2 performance counters, 2-5 see also database standard form storage format, 1-2, 3-2 AW\_ATTACH procedure, 3-28 AW\_COPY procedure, 3-30 AW\_CREATE procedure, 3-31 AW\_DELETE procedure, 3-32 AW\_DETACH procedure, 3-33 AW\_RENAME procedure, 3-34 AW\_TABLESPACE function, 3-35 AW\_UPDATE procedure, 3-36 AWXML see OLAP Analytic Workspace Java API AWXML.xsd, 4-1 AWXML.xsd schema, 4-1

## С

caches performance statistics, 2-3 composites (regular and compressed) defined, 3-4 CONVERT procedure, 3-37 cubes in analytic workspaces, 1-3 custom measures, 7-1, 9-1, 10-1 creating with OLAP Analytic Workspace Java API, 4-1 examples with OLAP\_EXPRESSION, 7-2 to 7-4

#### D

data type conversions, 11-4 database cache, 2-3 database initialization, 5-1 database standard form, 1-1, 3-3, 11-1 see also analytic workspaces version, 1-2

views of. 1-1 to 1-7 DBMS\_AW SPARSITY\_ADVICE\_TABLE procedure, 3-53 DBMS\_AW package, 3-1 to 3-54, 11-1 ADD\_DIMENSION\_SOURCE procedure, 3-14 ADVISE CUBE procedure, 3-16 ADVISE\_DIMENSIONALITY function, 3-18 ADVISE\_DIMENSIONALITY procedure, 3-20 ADVISE\_REL procedure, 3-24 ADVISE\_SPARSITY procedure, 3-25 AW\_ATTACH procedure, 3-28 AW\_COPY procedure, 3-30 AW\_CREATE procedure, 3-31 AW\_DELETE procedure, 3-32 AW\_DETACH procedure, 3-33 AW\_RENAME procedure, 3-34 AW\_TABLESPACE function, 3-35 AW\_UPDATE procedure, 3-36 CONVERT procedure, 3-37 EVAL\_NUMBER function, 3-38 EVAL\_TEXT function, 3-39 EXECUTE procedure, 3-40 GETLOG function, 3-42 INFILE procedure, 3-43 INTERP function, 3-44 INTERP\_SILENT function, 3-46 INTERPCLOB function, 3-45 OLAP\_ON function, 3-47, 3-48 PRINTLOG procedure, 3-49 RUN procedure, 3-50 SHUTDOWN procedure, 3-52 STARTUP procedure, 3-54 DBMS\_AW\$\_COLUMNLIST\_T table, 3-5 DBMS\_AW\$\_DIMENSION\_SOURCE\_T object type, 3-5 DBMS\_AW\$\_DIMENSION\_SOURCES\_T table type, 3-5 DBMS\_AW\_XML package, 4-1 dimensions in analytic workspaces, 1-5, 1-6 dynamic performance views, 2-1 to 2-6

## Ε

embedded-total dimension views, 11-6 embedded-total fact view, 11-7 EVAL\_NUMBER function, 3-38 EVAL\_TEXT function, 3-39 EXECUTE procedure, 3-40

# F

FETCH command (OLAP DML), 11-10, 11-15 fixed views, 2-1

# G

GETLOG function, 3-42

INFILE procedure, 3-43
initialization parameters, 5-1
init.ora file, 5-1
INTERACTIONEXECUTE function see DBMS\_AW\_XML package
INTERP function, 3-44
INTERP\_SILENT procedure, 3-46
INTERPCLOB function, 3-45

# J

Java API Analytic Workspace, 4-1

# L

levels in analytic workspaces, 1-4, 1-7 limit maps, 11-1, 11-15, 11-17 to 11-23 order of processing, 11-23 syntax, 11-17

#### Μ

measures in analytic workspaces, 1-4, 1-5 multidimensional data model Active Catalog, 1-1

# 0

object types automatic, 11-2 predefining, 11-2 syntax for creating, 11-2 OLAP Analytic Workspace Java API, 11-1 OLAP API, 11-1 optimization, 5-1 OLAP DML executing in SQL, 3-1 to 3-49, 6-1 to 6-7, 7-1 to 7-4, 8-1 to 8-5, 9-1 to 9-3, 10-1 to 10-3 quotation marks in, 3-4 OLAP performance views, 2-1 OLAP XML schema, 4-1 OLAP\_API\_SESSION\_INIT package, 5-1 to 5-6 OLAP\_CONDITION function, 6-1 to 6-7, 11-15 OLAP\_EXPRESSION function, 3-4, 7-1 to 7-5 OLAP\_EXPRESSION\_BOOL function, 8-1 to 8-5 OLAP\_EXPRESSION\_DATE function, 9-1 to 9-3 OLAP\_EXPRESSION\_TEXT function, 10-1 to 10-3 OLAP\_ON function, 3-47, 3-48 OLAP\_PAGE\_POOL\_SIZE parameter, 2-3 OLAP\_TABLE function, 11-1 to 11-23 custom measures, 7-5 data map parameter, 11-15 data type conversions, 11-4 examples, 11-5 FETCH command, 11-10, 11-15 limit map, 11-1, 11-15, 11-17 to 11-23

retrieving session log, 3-42 specifying a ROW2CELL column, 11-22 specifying an OLAP DML command, 11-12, 11-15, 11-22 specifying the analytic workspace, 11-12, 11-13 specifying the limit map, 11-12 specifying the logical table, 11-12, 11-14 with MODEL clause, 11-5 optimization OLAP API, 5-1 OLAP\_TABLE, 11-5 OUTFILE command affect on DBMS\_AW.EXECUTE, 3-40 affect on DBMS\_AW.RUN, 3-50

#### Ρ

page pool performance statistics, 2-3 performance counters, 2-1 to 2-6 PGA allocation, 2-3 print buffer, 3-40, 3-50 PRINTLOG procedure, 3-49

#### Q

quotation marks in OLAP DML, 3-4

# R

ROW2CELL column, 6-1, 6-6, 7-2, 8-2, 9-2, 11-22 RUN procedure, 3-50

# S

SERVEROUTPUT option, 3-40, 3-49, 3-50 session shutting down, 3-52 starting up, 3-54 session cache performance statistics, 2-3 session counters, 2-6 session logs printing, 3-49 retrieving, 3-42 session statistics, 2-5 SHUTDOWN procedure, 3-52 single-row functions, 7-1, 8-1, 9-1, 10-1 Sparsity Advisor, 3-4 to 3-7 SPARSITY\_ADVICE\_TABLE column descriptions, 3-25 SPARSITY\_ADVICE\_TABLE procedure, 3-53 SOL embedding OLAP commands, 3-1 to 3-49, 6-1 to 6-7, 7-1 to 7-4, 8-1 to 8-5, 9-1 to 9-3, 10-1 to 10-3 managing analytic workspaces, 3-1 to 3-54 standard form see database standard form STARTUP procedure, 3-54

#### Т

table type, 11-12, 11-14 automatic, 11-2 predefining, 11-2 syntax for creating, 11-2 transaction statistics, 2-6 tuples, 3-4

#### V

V\$AW\_AGGREGATE\_OP view, 2-2 V\$AW\_ALLOCATE\_OP view, 2-3 V\$AW\_CALC view, 2-3 V\$AW\_OLAP view, 2-5 V\$AW\_SESSION\_INFO view, 2-6 views Active Catalog, 1-1 creating embedded total dimensions, 11-6 creating embedded total measures, 11-7 creating rollup form, 11-8 objects in analytic workspaces, 1-1 to 1-7 template for creating with OLAP\_TABLE, 11-2, 11-3

# Χ

XML document example, 4-2