

Oracle® OLAP

Application Developer's Guide

10g Release 2 (10.2)

B14349-01

June 2005

Copyright © 2003, 2005, 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 Corporation, 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 Retek 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	xiii
Audience	xiii
Documentation Accessibility	xiii
Related Documents	xiv
Conventions	xiv
 What's New in Oracle OLAP Applications Development?	xv
Oracle Database 10g Release 10.2 Oracle OLAP	xv
Oracle Database 10g Release 10.1.0.4 Oracle OLAP	xv
 Part I Fundamentals	
 1 Overview	
OLAP Technology Within Oracle Database	1-1
Problems Maintaining Two Distinct Systems	1-1
Full Integration of Multidimensional Technology	1-2
Using OLAP to Answer Business Questions	1-2
Common Analytical Applications	1-3
Tools for Querying OLAP Data Stores	1-3
Formulating Queries	1-4
Creating Custom Measures	1-4
The Logical Dimensional Data Model	1-5
Logical Cubes	1-6
Logical Measures	1-6
Logical Dimensions	1-6
Logical Hierarchies and Levels	1-7
Logical Attributes	1-7
About Multidimensional Data Stores	1-7
Creating Analytic Workspaces	1-8
Summary Data	1-8
Deciding When to Use Analytic Workspaces	1-9
When to Use Analytic Workspaces	1-9
When to Use Relational Schemas	1-9
Structured and Unstructured Data Stores	1-10
Processing Analytic Queries	1-10

Creating Summary Data.....	1-10
How Analytic Workspaces Store Summary Data	1-10
How Relational Schemas Store Aggregate Data	1-11
Components of Oracle OLAP	1-11
OLAP Analytic Engine	1-11
Analytic Workspaces	1-11
Analytic Workspace Manager	1-12
OLAP Worksheet	1-12
SQL Interface to OLAP	1-12
OLAP DML	1-12
Analytic Workspace Java APIs.....	1-12
OLAP API.....	1-12
OLAP Catalog.....	1-13
Implementing an Analytic Workspace.....	1-13
Identifying Business Goals	1-13
Identifying Data Sources.....	1-13
Defining a Logical Model.....	1-14
Mapping, Loading, and Aggregating the Data.....	1-14
Generating Information-Rich Data	1-14
Implementing a Relational Data Warehouse for OLAP	1-14
Identifying Business Goals	1-14
Identifying Data Sources.....	1-15
Defining a Logical Model.....	1-15
Generating Summary Data	1-15
Upgrading Oracle Database 10g Release 1 Analytic Workspaces	1-15
Upgrading Oracle9i Analytic Workspaces.....	1-16
Upgrading the Physical Storage Format.....	1-16
Upgrading the Standard Form Metadata	1-17

2 The Sample Schema

Case Study Scenario.....	2-1
Reporting Requirements	2-2
Business Goals	2-2
Information Requirements.....	2-3
Business Analysis Questions	2-3
What products are profitable?.....	2-3
Who are our customers, and what and how are they buying?	2-3
What accounts are most profitable?	2-4
What is the performance of each distribution channel?.....	2-4
Is there still a seasonal variance to the business?	2-4
Summary of Information Requirements.....	2-4
Identifying Required Business Facts.....	2-5

Designing a Logical Data Model for Global Computing	2-5
Identifying Dimensions.....	2-5
Identifying Levels	2-6
Identifying Hierarchies	2-6
Identifying Stored Measures	2-6
The Global Schema	2-7

Part II Creating and Managing Analytic Workspaces

3 Creating an Analytic Workspace

Introduction to Analytic Workspace Manager.....	3-1
Model View	3-2
Object View	3-2
OLAP Worksheet	3-3
Getting Started with Analytic Workspace Manager.....	3-4
Installing Analytic Workspace Manager	3-4
Opening Analytic Workspace Manager	3-4
Defining a Database Connection.....	3-5
Opening a Database Connection.....	3-5
Identifying the Source Data	3-5
Schema Requirements	3-5
Star Schema.....	3-6
Snowflake Schema	3-7
Other	3-8
Making Transformations in Your Source Data.....	3-9
Choosing a Build Tool	3-10
Creating a Standard Form Workspace Using Analytic Workspace Manager	3-10
How Analytic Workspace Manager Saves Changes.....	3-10
Basic Steps for Creating a Standard Form Workspace	3-11
Adding Functionality to a Standard Form Analytic Workspace	3-11
Creating Logical Dimensions.....	3-12
Creating Dimensions	3-12
Defining a Time Dimension	3-12
Creating Unique Dimension Members.....	3-12
Opening the Create Dimension Dialog Box.....	3-13
Creating Levels.....	3-13
Creating Hierarchies.....	3-14
Creating Attributes	3-14
Automatically Defined Attributes	3-14
User Attributes	3-15
Creating Logical Cubes	3-15
Creating Cubes	3-15
Creating Measures	3-16
Creating Calculated Measures	3-16

Making Data Storage Decisions	3-17
What is Sparsity?	3-17
Sparsity Patterns	3-17
Physical Storage of Sparse Data	3-18
Manually Calculating Sparsity in a Cube	3-18
Ordering the Dimensions in a Cube	3-19
Partitioning Large Measures	3-19
Defining Rules for Summarizing Data	3-21
Basic Strategy for Summarizing Analytic Workspace Data	3-21
Selecting Levels to Aggregate in the Builds	3-21
Choosing Aggregation Methods	3-22
Mapping Logical Objects to Data Sources	3-22
Mapping Dimensions	3-23
Mapping Cubes	3-24
Maintaining the Data	3-25
Submitting Maintenance Tasks to the Oracle Job Queue	3-25
Managing Maintenance Jobs	3-26
Defining Measure Folders	3-26
Supporting Multiple Languages	3-26
Creating Calculation Plans	3-27
Case Study: Creating the Global Analytic Workspace	3-27
Defining the GLOBAL_AW User	3-27
Examining Sparsity Characteristics for GLOBAL	3-28
Identifying Levels for Precalculation	3-28
Creating the GLOBAL Analytic Workspace	3-29
Creating GLOBAL Dimensions and Attributes	3-29
Creating GLOBAL Cubes and Measures	3-30
Mapping the GLOBAL Logical Model to Data Sources	3-30
Loading and Aggregating the Data	3-31
Creating Calculated Measures	3-32
Creating a Measure Folder	3-33
Case Study: Creating the Sales History Analytic Workspace	3-33
Creating the SH Analytic Workspace	3-34
Defining Database Parameters	3-35
Defining Tablespaces for Sales History	3-35
Defining the SH_AW User	3-36
Defining the Logical Dimensions for Sales History	3-36
Defining TIMES_DIM	3-36
Defining CUSTOMERS_DIM	3-37
Defining PRODUCTS_DIM, CHANNELS_DIM, and PROMOTIONS_DIM	3-37
Defining the Logical Sales Cube for Sales History	3-37
About the Sparsity Advisor	3-38
Sample Program for Evaluating Sales History Tables	3-38
Interpreting the Results from the Sparsity Advisor	3-40
Maintaining Sales History	3-42

4 Predicting Future Performance

Creating a Forecast	4-1
Steps for Creating a Forecast	4-1
Creating the Forecast Time Periods	4-2
Defining a Measure for the Results	4-2
Defining Supporting Variables (Optional)	4-2
Developing a Forecast Program	4-2
Generating a Forecast	4-3
Aggregating the Forecast Data	4-3
Case Study: Forecasting Global Sales	4-4
Defining the Sales Forecast Measure for Global Sales	4-4
Defining a Variable for Seasonal Adjustment	4-4
Developing a Forecasting Program for Global Sales	4-5
Historical and Forecast Time Periods	4-5
The FORECAST_SALES Program	4-5
Generating the Global Sales Forecast	4-6
Aggregating the Sales Forecast Measure	4-7

5 Developing Java Applications for OLAP

Building Analytical Java Applications	5-1
About Java	5-1
The Java Solution for OLAP	5-2
Oracle Java Development Environment	5-2
Introducing OracleBI Beans	5-3
Metadata	5-3
Navigation	5-3
Formatting	5-4
Graphs	5-4
Crosstabs	5-4
Data Beans	5-4
Wizards	5-4
JSP Tag Library	5-5
Understanding the OLAP API	5-5
How the OLAP API Accesses Dimensional Data	5-6
Calculation Capabilities	5-6
Intelligent Caching	5-7
Managing Data Sources for OracleBI Beans and the OLAP API	5-7
Building Java Applications that Manage Analytic Workspaces	5-7

6 Administering Oracle OLAP

Administration Overview	6-1
Creating Tablespace for Analytic Workspaces	6-2
Creating an UNDO Tablespace	6-2
Creating a Permanent Tablespace for Analytic Workspaces	6-2
Creating a Temporary Tablespace for Analytic Workspaces	6-3
Querying the Size of an Analytic Workspace	6-4

Setting Up User Names	6-4
SQL Access For DBAs and Application Developers.....	6-4
SQL Access for Analysts	6-5
Access to Database Objects Using OracleBI Beans.....	6-5
Access to the Oracle JVM.....	6-5
Initialization Parameters for Oracle OLAP	6-6
Procedure: Setting System Parameters for OLAP	6-6
About the PGA_AGGREGATE_TARGET Setting	6-7
Initialization Parameters for OracleBI Beans	6-7
Permitting Access to External Files.....	6-7
Creating a Directory Object	6-8
Granting Access Rights to a Directory Object.....	6-8
Example: Creating and Using a Directory Object	6-8
Understanding Data Storage.....	6-9
Analytic Workspace Tables	6-9
System Tables and Views.....	6-10
Monitoring Performance.....	6-11
Copying and Backing Up Analytic Workspaces.....	6-12

Part III Creating a Relational Data Warehouse

7 Using the OLAP Catalog

Choosing a Method for Creating OLAP Catalog Metadata	7-1
For Source Data in a Basic Star or Snowflake Schema.....	7-1
For Dimension Tables with Complex Hierarchies	7-2
For Other Schema Configurations	7-3
Overview of the OLAP Catalog	7-3
OLAP Catalog Components	7-3
About CWM1.....	7-3
About CWM2.....	7-4
Steps for Creating OLAP Metadata	7-4
Creating Metadata Using Enterprise Manager Database Control	7-4
Procedure: Accessing OLAP Management	7-4
Defining Metadata for Dimension Tables	7-5
Information That You Supply for Dimensions.....	7-5
Time Dimension.....	7-5
Procedure: Defining a Logical Dimension in the OLAP Catalog	7-5
Defining Metadata for Fact Tables.....	7-5
Information That You Supply for Cubes.....	7-6
Procedure: Defining a Logical Cube in the OLAP Catalog	7-6
Case Study: Creating Metadata for the GLOBAL Star Schema	7-6
Defining a Logical Time Dimension for the Global Schema.....	7-6
Defining a Logical Price and Cost Cube for the Global Schema	7-8
Creating Metadata Using PL/SQL.....	7-9
CWM2 Packages for Creating OLAP Dimensions	7-9
CWM2 Packages for Creating Cubes	7-9
CWM2 Package for Mapping Metadata	7-9

CWM2 Package for Creating Level-Based Dimension Tables	7-9
CWM2 Packages for Classification and Validation	7-9
8 Materialized Views for the OLAP API	
Summary Management with Oracle OLAP	8-1
Overview and Requirements	8-2
Materialized Views Required for a Cube	8-2
Materialized Views and OLAP Metadata.....	8-2
A Dimension Materialized View	8-3
CREATE Materialized View for a Dimension Hierarchy	8-3
Bitmap Indexes for a Dimension Hierarchy.....	8-3
Statistics for a Dimension Hierarchy.....	8-4
A Fact Materialized View	8-4
CREATE Fact Materialized View.....	8-4
Bitmap Indexes for Fact Materialized Views	8-5
Statistics for Fact Materialized Views	8-5
Using the DBMS_ODM Package	8-5
Procedure: Automatically Generate the Materialized Views	8-6
Procedure: Manually Generate the Materialized Views	8-7
Example: Automatically Generate the Materialized Views for a Price Cube	8-7
Example: Manually Generate the Materialized Views for a Sales Cube	8-8
A Database Standard Form for Analytic Workspaces	
Overview of Database Standard Form	A-1
Terminology: Using Role Names to Identify Objects	A-2
Querying a Standard Form Analytic Workspace	A-2
Querying the Standard Form Catalogs	A-2
Querying Properties.....	A-3
Standard Form Implementation of the Logical Model	A-4
Relationships Among Logical Objects	A-4
Classes of Workspace Objects	A-4
Object Naming Conventions	A-5
Logical Names	A-5
Simple Logical Names and Full Names.....	A-5
Name Space Organization	A-6
Workspace Object Properties	A-6
System Properties on All Workspace Objects	A-6
Properties Specific to Implementation Class Objects.....	A-7
Role Property Values for Implementation Class Objects	A-7
Role Property Values for Catalogs Class Objects	A-8
Role Property Values for Features Class Objects.....	A-10
Role Property Values for Extensions Class Objects	A-10
Implementation Class Objects	A-11
Cube Objects	A-11
Cubedef Dimension	A-12

Measure Objects	A-13
Measuredef Object	A-13
Dimension Objects	A-14
Dimdef Dimension.....	A-15
Hierlist Dimension.....	A-16
Levellist Dimension	A-16
Member_Levelrel Relation	A-17
Member_Parentrel Relation.....	A-17
Hier_Levels Valueset.....	A-18
Attrdef Object	A-18
Catalogs Class Objects	A-19
Lists of Objects	A-20
ALL_CUBES Dimension	A-20
ALL_MEASURES Dimension	A-20
ALL_DIMENSIONS Dimension	A-20
ALL_HIERARCHIES Dimension	A-21
ALL_LEVELS Dimension	A-21
ALL_ATTRIBUTES Dimension	A-22
ALL_OBJECTS Dimension	A-22
Lists of Types and Languages	A-23
ALL_DESCTYPES Dimension.....	A-23
ALL_ATTRTYPES Dimension	A-23
ALL_LANGUAGES Dimension	A-23
Lists of Cube and Dimension Objects	A-24
CUBE_MEASURES Relation	A-24
DIM_HIERARCHIES Relation.....	A-24
DIM_LEVELS Relation.....	A-25
DIM_ATTRIBUTES Relation.....	A-25
Supporting Object Information	A-26
AW_NAMES Variable.....	A-26
Features Class Objects	A-27
ALL_DESCRIPTIONS Variable	A-27
DEFAULT_HIER Relation	A-27
VISIBLE Variable.....	A-28
Member_Inhier Valueset.....	A-28
Member_Createdby Variable	A-29
Member_Familyrel Relation	A-29
Member_Gid Variable	A-29
OBJ_CREATEDBY Variable.....	A-30
VERSION Variable.....	A-30
Extensions Class Objects	A-30

B Upgrading From Express Server

Administration.....	B-1
Management Tools.....	B-1
Authentication of Users.....	B-2
Data Transfer	B-2

Localization	B-2
Applications Support	B-3
Programming Environment.....	B-3
Communications	B-4
Metadata	B-4
Programming Language Changes	B-4
New Commands.....	B-4
Obsolete Commands.....	B-4
UPDATE and COMMIT	B-5
Transforming Oracle Express Databases to Standard Form.....	B-5
Who Should Use the Transformation Tool.....	B-5
What the Transformation Tool Does For You	B-6
What the Transformation Tool Does Not Do For You	B-6
Converting From Oracle Express Objects Metadata	B-7
Procedure: Converting From Oracle Express Objects to Standard Form	B-7
Populating Time Attributes	B-8
Sorting Time Dimension Members	B-9
Creating and Populating End Date and Time Span Attributes.....	B-9
Setting Properties on Time Objects	B-9
Revising the Load Programs	B-9
Example: Converting the XADEMO Database to Standard Form.....	B-10
Creating a Standard Form XADEMO Analytic Workspace	B-10
About the Time Dimension in XADEMO	B-12
Populating the XADEMO Time Attributes	B-13

Glossary

Index

Preface

The *Oracle OLAP Application Developer's Guide* explains how SQL and Java applications can extend their analytic processing capabilities by using the OLAP option in the Enterprise edition of the Oracle Database.

The preface contains these topics:

- [Audience](#)
- [Documentation Accessibility](#)
- [Related Documents](#)
- [Conventions](#)

Audience

This manual is intended for applications developers and DBAs who need to perform these tasks:

- Develop business intelligence applications
- Design and develop dimensional data stores (analytic workspaces)
- Administer Oracle Database with the OLAP option

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 the following manuals in the Oracle Database 10g documentation set:

- *Oracle OLAP Application Developer's Guide*
Explains how SQL and Java applications can extend their analytic processing capabilities by using Oracle OLAP in the Enterprise Edition of Oracle Database.
- *Oracle OLAP Reference*
Explains the syntax of PL/SQL packages and types and the column structure of views related to 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 online analytical processing 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 Java API for building and maintaining analytic workspaces.

Conventions

The following text conventions are used in this document:

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

What's New in Oracle OLAP Applications Development?

The following identifies some of the major changes from prior releases.

Oracle Database 10g Release 10.2 Oracle OLAP

Oracle OLAP in Oracle Database 10g Release 2 (10.2) provides numerous performance enhancements and extensions to the dimensional data model.

Enhanced Data Model in Analytic Workspace Manager

Analytic Workspace Manager 10.2 supports calculation plans and multiple languages. Compressed composites provide support for partial computation and non-additive operators.

See Also:

- [Chapter 1](#) for upgrade instructions
- [Chapter 3](#) for new features in Analytic Workspace Manager

Support for Transportable Tablespaces

Analytic workspaces are included with other database objects in transportable tablespaces.

See Also: [Chapter 6](#) for information about backing up analytic workspaces

Oracle Database 10g Release 10.1.0.4 Oracle OLAP

Oracle OLAP 10.1.0.4 provides a simpler approach to building and enabling analytic workspaces while introducing the more powerful analytic tools of the OLAP engine into the build process.

New Storage Format for Analytic Workspaces

Analytic workspaces are still stored in LOB tables in Oracle Database 10g, but in a different format that supports partitioning and multiple writers.

See Also:

- [Chapter 1](#) for upgrade instructions
- [Chapter 6](#) for a description of the storage format

New Model View in Analytic Workspace Manager

The Model View in Analytic Workspace Manager 10g enables you to define the logical model of your analytic workspace directly in database standard form. You no longer create logical models in the OLAP Catalog for building analytic workspaces. Analytic Workspace Manager supports a wider range of schema designs than the OLAP Catalog.

See Also: [Chapter 3](#) for instructions on using the Model View

Database Standard Form 10g

Analytic Workspace Manager and the PL/SQL DBMS_AWM package generate a new version of standard form metadata that supports the new features of Oracle Database 10g.

See Also: [Appendix A](#) for a description of standard form metadata.

Dynamic Enabling for the OLAP API and OracleBI Beans

The `SELECT` statements for the views of an analytic workspace are stored in the analytic workspace itself. Enablement no longer requires the creation of database objects.

Direct Metadata Access

The OLAP API and OracleBI Beans query the Active Catalog views, which display the database standard form metadata stored in analytic workspaces. Enablement no longer requires the creation of OLAP Catalog CWM2 metadata.

Part I

Fundamentals

Part I introduces basic concepts, tools, and capabilities of the OLAP option. By reading the chapters in this part, you will learn how the OLAP option works within Oracle Database. You will also get an introduction to the sample schema used in examples throughout this guide.

Part I contains the following chapters:

- [Chapter 1, "Overview"](#)
- [Chapter 2, "The Sample Schema"](#)

Overview

This chapter introduces the powerful analytic resources available in Oracle Database 10g installed with the OLAP option. It consists of the following topics:

- [OLAP Technology Within Oracle Database](#)
- [Using OLAP to Answer Business Questions](#)
- [Common Analytical Applications](#)
- [Tools for Querying OLAP Data Stores](#)
- [The Logical Dimensional Data Model](#)
- [About Multidimensional Data Stores](#)
- [Deciding When to Use Analytic Workspaces](#)
- [Components of Oracle OLAP](#)
- [Implementing an Analytic Workspace](#)
- [Implementing a Relational Data Warehouse for OLAP](#)
- [Upgrading Oracle Database 10g Release 1 Analytic Workspaces](#)
- [Upgrading Oracle9i Analytic Workspaces](#)

OLAP Technology Within Oracle Database

Multidimensional technology is now available within Oracle Database. Organizations no longer need to choose between a multidimensional OLAP database and a relational database. By integrating multidimensional tables and an analytic engine into the database, Oracle provides the power of multidimensional analysis along with the manageability, scalability, and reliability of Oracle Database.

Problems Maintaining Two Distinct Systems

The integration of multidimensional technology in a relational database is important because maintaining a standalone multidimensional database is costly. It requires additional hardware and DBAs who are skilled at using the specialized administrative tools of the multidimensional database. Moreover, standalone multidimensional databases require applications that use proprietary APIs. This severely limits the number of applications that can be run against them, not only because fewer applications are available in these APIs, but because all the data that they run on must be transferred from the relational database to the multidimensional database. These requirements often force enterprises into supporting two sets of query and reporting tools, one for the relational database and the other for the multidimensional database.

Full Integration of Multidimensional Technology

In contrast, the OLAP option is fully integrated into the Oracle Database. DBAs use the same tools to administer this option as they use to administer all other components of the database. The DBA can decide the best location for storing and calculating the data as part of optimizing the operations of the database. A single application can access both relational and multidimensional data.

SQL-based applications can now use pure SQL against information-rich relational views of multidimensional data provided by an OLAP-enabled Oracle Database. OLAP calculations can be queried using SQL, enabling application developers to leverage their investment in SQL while expanding the analytic sophistication of their software to include modeling, forecasting, and what-if analysis. Standard reporting applications can present the results of complex multidimensional calculations, while ad-hoc querying tools such as custom aggregate members and custom measures can expand the analyst's range of calculation functions.

Using OLAP to Answer Business Questions

Relational databases provide the online transactional processing (OLTP) that is essential for businesses to keep track of their affairs. Designed for efficient selection, storage, and retrieval of data, relational databases are ideal for housing gigabytes of detailed data.

The success of relational databases is apparent in their use to store information about an increasingly wide scope of activities. As a result, they contain a wealth of data that can yield critical information about a business. This information can provide a significant edge in an increasingly competitive marketplace.

The challenge is in deriving answers to business questions from the available data, so that decision makers at all levels can respond quickly to changes in the business climate.

A standard transactional query might ask, "When did order 84305 ship?" This query reflects the basic mechanics of doing business. It involves simple data selection and retrieval of one record (or, at most, several related records) identified by a unique order number. Any follow-up questions, such as which postal carrier was used and where was the order shipped to, can probably be answered by the same record. This record has a useful life span in the transactional world: it begins when a customer places the order and ends when the order is shipped and paid for. At this point, the record can be rolled off to an archive.

In contrast, a typical series of analytical queries might ask, "How do sales in the Pacific Rim for this quarter compare with sales a year ago? What can we predict for sales next quarter? What factors can we alter to improve the sales forecast? What happens if I change this number?"

These are not questions about doing business transactions, but about analyzing past performance and making decisions that will improve future performance, provide a more competitive edge, and thus enhance profitability. The analytic database provides the information needed by decision makers whose ability to set goals today is dependent on how well they can predict the future. Getting the answers to these questions involves single-row calculations, time series analysis, and access to aggregated historical and current data. This requires OLAP -- online analytical processing.

Common Analytical Applications

Here are a few examples of common applications that can use the OLAP option to realize valuable gains in functionality and performance:

- Planning applications enable organizations to predict outcomes. They generate new data using predictive analytical tools such as models, forecasts, aggregation, allocation, and scenario management. Some examples of this type of application are corporate budgeting and financial analyses, and demand planning systems.
- Budgeting and financial analysis systems enable organizations to analyze past performance, build revenue and spending plans, manage to attain profit goals, and model the effects of change on the financial plan. Management can determine spending and investment levels that are appropriate for the anticipated revenue and profit levels. Financial analysts can prepare alternative budgets and investment plans contingent on factors such as fluctuations in currency values.
- Demand planning systems enable organizations to predict market demand based on factors such as sales history, promotional plans, and pricing models. They can model different scenarios that forecast product demand and then determine appropriate manufacturing goals.

As this discussion highlights, the data processing required to answer analytical questions is fundamentally different from the data processing required to answer transactional questions. The users are different, their goals are different, their queries are different, and the type of data that they need is different. A relational data warehouse enhanced with the OLAP option provides the best environment for data analysis.

Tools for Querying OLAP Data Stores

Analysts can choose between two query and analysis tools for selecting, viewing, and analyzing the data:

- **OracleBI Discoverer Plus OLAP** is a full featured tool for business analysis that provides a variety of presentation options.

Discoverer Plus OLAP provides various wizards to guide power users through the entire process of building and publishing sophisticated reports containing crosstabs and graphs. They can choose from multiple layout options to create a visual representation of their query results. They can create queries, drill, pivot, slice and dice data, add analytic calculations, graph the data, share results with other users, and export their Discoverer reports in various data formats. Discoverer reports can also be published in dashboards where other users can access them from their browsers.

- **OracleBI Spreadsheet Add-In** combines Oracle Database dimensional analytics with the capabilities of Microsoft Excel.

Spreadsheet Add-In enables analysts to work with live dimensional data in the familiar spreadsheet environment of Microsoft Excel. The add-in fetches data using an active connection to an OLAP data store, and displays the data in a spreadsheet. Users can use the add-in to perform OLAP operations such as drilling, rotation, and data selection.

In addition, OracleBI Beans and the OLAP API are available for developing custom applications, as described in [Chapter 5](#).

Formulating Queries

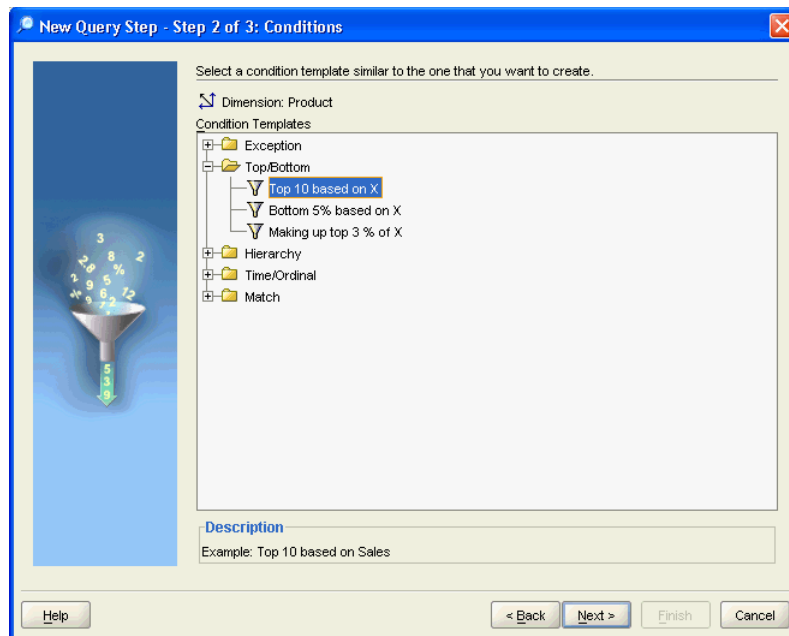
Both Discoverer Plus OLAP and Spreadsheet Add-In use a dimensional data model so that analysts can formulate their queries in the language of business. Dimensions provide the context for the data. Consider the following request for information:

For **fiscal years** 2003 and 2004, show the percent change in **sales** for the top 10 **products** for each of the top 10 **customers** based on sales.

The sales measure is dimensioned by time periods, products, and customers. This request is articulated in business terms, but easily translates into a query in the language of dimensional analysis: dimensions, levels, hierarchies, and attributes.

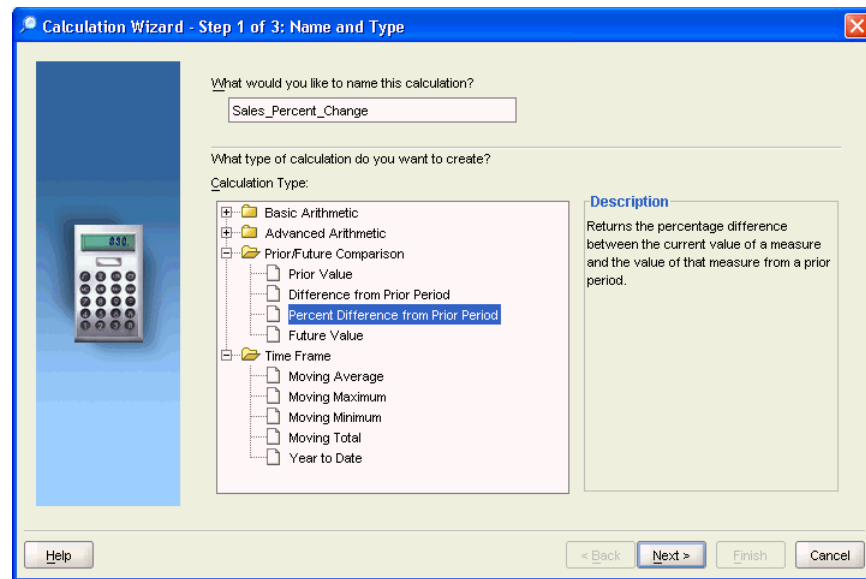
Figure 1-1 shows a step in the Query Wizard in Discoverer Plus OLAP for selecting the top 10 products. The Query Wizard assists users in selecting by criteria, by value, and by saved selections. All OLAP tools provide a Query Wizard to assist users in formulating these queries.

Figure 1-1 *Selecting Dimension Values By Criteria*



Creating Custom Measures

Multidimensional data types facilitate the creation of custom measures. From the measures stored in your data warehouse, you can use numerous operators and functions to generate a wealth of information. Figure 1-2 shows a step in the Calculation Wizard of Discoverer Plus OLAP for calculating percent change in sales. Spreadsheet Add-In has the same Calculation Wizard. Both tools use the OracleBI Beans CalcBuilder.

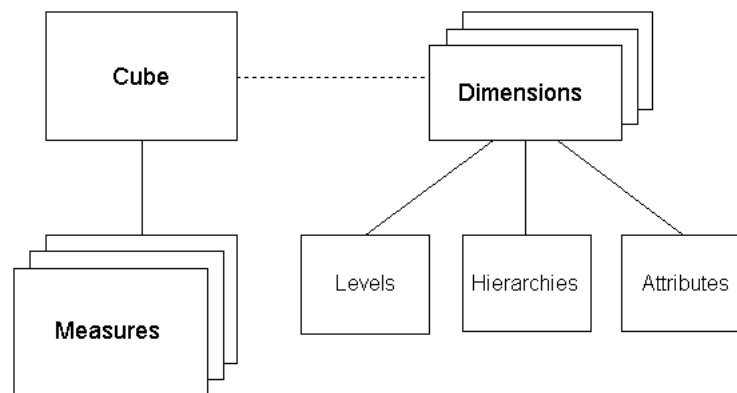
Figure 1–2 Choosing a Calculation Method for a Custom Measure

The Logical Dimensional Data Model

The dimensional data model is an integral part of On-Line Analytical Processing, or OLAP. Because OLAP is on-line, it must provide answers quickly; analysts pose iterative queries during interactive sessions, not in batch jobs that run overnight. And because OLAP is also analytic, the queries are complex.

The dimensional data model is composed of logical cubes, measures, dimensions, hierarchies, levels, and attributes. The simplicity of the model is inherent because it defines objects that represent real-world business entities. Analysts know which business measures they are interested in examining, which dimensions and attributes make the data meaningful, and how the dimensions of their business are organized into levels and hierarchies.

Figure 1–3 shows the general relationships among logical objects.

Figure 1–3 Diagram of the OLAP Logical Dimensional Model

Logical Cubes

Logical cubes provide a means of organizing measures that have the same shape, that is, they have the exact same dimensions. Measures in the same cube have the same relationships to other logical objects and can easily be analyzed and displayed together.

Logical Measures

Measures populate the cells of a logical cube with the facts collected about business operations. Measures are organized by dimensions, which typically include a Time dimension.

An analytic database contains snapshots of historical data, derived from data in a transactional database, legacy system, syndicated sources, or other data sources. Three years of historical data is generally considered to be appropriate for analytic applications.

Measures are static and consistent while analysts are using them to inform their decisions. They are updated in a batch window at regular intervals: weekly, daily, or periodically throughout the day. Some administrators refresh their data by adding periods to the time dimension of a measure, and may also roll off an equal number of the oldest time periods. Each update provides a fixed historical record of a particular business activity for that interval. Other administrators do a full rebuild of their data rather than performing incremental updates.

A critical decision in defining a measure is the lowest level of detail. Users may never view this **base level data**, but it determines the types of analysis that can be performed. For example, market analysts (unlike order entry personnel) do not need to know that Beth Miller in Ann Arbor, Michigan, placed an order for a size 10 blue polka-dot dress on July 6, 2002, at 2:34 p.m. But they might want to find out which color of dress was most popular in the summer of 2002 in the Midwestern United States.

The base level determines whether analysts can get an answer to this question. For this particular question, Time could be rolled up into months, Customer could be rolled up into regions, and Product could be rolled up into items (such as dresses) with an attribute of color. However, this level of aggregate data could not answer the question: At what time of day are women most likely to place an order? An important decision is the extent to which the data has been aggregated before being loaded into a data warehouse.

Logical Dimensions

Dimensions contain a set of unique values that identify and categorize data. They form the edges of a logical cube, and thus of the measures within the cube. Because measures are typically multidimensional, a single value in a measure must be qualified by a member of each dimension to be meaningful. For example, the Sales measure has four dimensions: Time, Customer, Product, and Channel. A particular Sales value (43,613.50) only has meaning when it is qualified by a specific time period (Feb-01), a customer (Warren Systems), a product (Portable PCs), and a channel (Catalog).

Logical Hierarchies and Levels

A **hierarchy** is a way to organize data at different levels of aggregation. In viewing data, analysts use dimension hierarchies to recognize trends at one level, drill down to lower levels to identify reasons for these trends, and roll up to higher levels to see what affect these trends have on a larger sector of the business.

Each **level** represents a position in the hierarchy. Each level above the base (or most detailed) level contains aggregate values for the levels below it. The members at different levels have a one-to-many **parent-child relation**. For example, Q1 - 02 and Q2 - 02 are the children of 2002, thus 2002 is the parent of Q1 - 02 and Q2 - 02.

Suppose a data warehouse contains snapshots of data taken three times a day, that is, every 8 hours. Analysts might normally prefer to view the data that has been aggregated into days, weeks, quarters, or years. Thus, the Time dimension needs a hierarchy with at least five levels.

Similarly, a sales manager with a particular target for the upcoming year might want to allocate that target amount among the sales representatives in his territory; the allocation requires a dimension hierarchy in which individual sales representatives are the child values of a particular territory.

Although hierarchies are typically composed of levels, they do not have to be. The parent-child relations among dimension members may not define meaningful levels. For example, in an employee dimension, each manager has one or more reports, which forms a parent-child relation. Creating levels based on these relations (such as individual contributors, first-level managers, second-level managers, and so forth) may not be meaningful for analysis.

Hierarchies and levels have a many-to-many relationship. A hierarchy typically contains several levels, and a single level can be included in more than one hierarchy.

Logical Attributes

An **attribute** provides additional information about the data. Some attributes are used for display. For example, you might have a product dimension that uses Stock Keeping Units (SKUs) for dimension members. The SKUs are an excellent way of uniquely identifying thousands of products, but are meaningless to most people if they are used to label the data in a report or graph. You would define attributes for the descriptive labels.

You might also have attributes like colors, flavors, or sizes. This type of attribute can be used for data selection and answering questions such as: Which colors were the most popular in women's dresses in the summer of 2002? How does this compare with the previous summer?

Time attributes can provide information about the Time dimension that may be useful in some types of analysis, such as identifying the last day or the number of days in each time period.

About Multidimensional Data Stores

Multidimensional data is stored in **analytic workspaces**, where it can be manipulated by the OLAP engine in Oracle Database. Individual analytic workspaces are stored in tables in a relational schema, and they can be managed like other relational tables. An analytic workspace is owned by a particular user ID, and other users can be granted access to it. Within a single database, many analytic workspaces can be created and shared among users.

Analytic workspaces have been designed explicitly to handle multidimensionality in their physical data storage and manipulation of data. The multidimensional technology that underlies analytic workspaces is based on an indexed multidimensional array model, which provides direct cell access. This intrinsic multidimensionality affords analytic workspaces much of their speed and power in performing multidimensional analysis.

Creating Analytic Workspaces

Creating an analytic workspace involves a physical transformation of the data. The first step in that transformation is defining dimensional objects such as measures, dimensions, levels, hierarchies, and attributes. Afterward, you can map the dimensional objects to the data sources. The analytic workspace instantiates the logical objects as physical objects, and the data loading process transforms the data from a relational format into a dimensional format.

Analytic workspaces have several different types of data containers, such as dimensions, variables, and relations. Each type of container can be used in a variety of ways to store different types of information, including business measures and metadata.

The analytic workspaces that are created by Oracle Warehouse Manager and Analytic Workspace Manager are in **database standard form** (typically called simply "standard form"). Standard form specifies the types of physical objects that are used to instantiate logical objects (such as dimensions and measures), and the type, form, and storage location of the metadata that describes these logical objects. This metadata is exposed to SQL in the **Active Catalog**. The Active Catalog is composed of views of standard form metadata that is stored in analytic workspaces. These views are maintained automatically, so that a change to a standard form analytic workspace is reflected immediately by a change to the Active Catalog. Discoverer Plus OLAP and Spreadsheet Add-In use the Active Catalog to query data in analytic workspaces.

Summary Data

An analytic workspace initially contains only base-level data loaded from its data sources. Summary data is calculated in the analytic workspace, and the aggregates are stored with the base data in the same object. Aggregates can be stored permanently in the analytic workspace, or for the duration of an individual session, or only for a single query. Aggregation rules identify which aggregates are stored, and which aggregates are calculated **on the fly**.

When an application queries the analytic workspace, either the aggregate values have already been calculated and can simply be retrieved, or they can be calculated on the fly from a small number of stored aggregates. The data is always presented to the application as fully solved; that is, both detail and summary values are provided, without requiring that calculations be specified in the query. Analytic workspaces are optimized for multidimensional calculations, making run-time summarizations extremely fast.

Analytic workspaces provide an extensive list of **aggregation** methods, including weighted, hierarchical, and weighted hierarchical methods.

Deciding When to Use Analytic Workspaces

To implement an OLAP solution, you must create a data store using one of these designs:

- **Dimensional schema.** An analytic workspace in database standard form
- **Relational schema.** A star schema with metadata defined in the OLAP Catalog

For most DBAs, relational tables and SQL provide a familiar environment. On the other hand, analytic workspaces require transformation of the data and learning new concepts. So why use analytic workspaces? The answer is simple: The powerful analytics and run-time performance of analytic workspaces often provide the best support for many types of decision-making. Nonetheless, relational schemas are the best choice for some other situations.

The following sections identify data characteristics that are best handled by analytic workspaces, and those that may be handled better by relational schemas. Your situation may not fit perfectly into one category, so you will need to weigh the relative importance of each characteristic as well as the long-range plans for your enterprise. For example, if your data store primarily supports the generation of routine reports, with some exploratory reporting, and this usage appears to be stable, then a relational schema might be the best choice. However, if usage is shifting toward fewer routine reports and more extensive exploratory ad-hoc querying, then you might choose an analytic workspace instead.

When to Use Analytic Workspaces

Analytic workspaces are the best choice for these requirements:

- Exploratory, ad-hoc querying of all areas of the data
- Advanced calculations such as models, forecasts, growth ratios, and trends
- What-if scenarios
- Time series calculations such as lead, lag, moving average, and year-to-date
- Complex run-time calculations
- High performance for calculating summary data and inter-row functions such as share calculations

When to Use Relational Schemas

Star schemas may be preferable to analytic workspaces for these requirements:

- Predictable querying patterns and prepared reports
- No advanced calculations (such as forecasts)
- Infrequent complex run-time calculations
- Measures with a large number of dimensions
- Dimensions with few aggregate levels

The following topics explore the technical differences between dimensional and relational data stores that support these guidelines.

Structured and Unstructured Data Stores

The dimensional data model is highly structured. Structure implies rules that govern the relationships among the data and control how the data can be queried. Analytic workspaces are the physical implementation of the dimensional model, and thus are highly optimized for dimensional queries. The OLAP engine leverages the model in performing highly efficient cross-cube joins (for inter-row calculations), outer joins (for time series analysis), and indexing. Dimensions are pre-joined to the measures.

Relational schemas can have much less structure, and the relationships among tables and views can be established on a query-by-query basis. This flexibility may be very important for some users, although it may result in reduced performance. OLAP Catalog metadata can be used to superimpose a dimensional data model on relational data stores. Nonetheless, a superimposed model does not provide the same performance as an integrated data model. The data storage design limits the opportunities the relational engine has for optimizing queries.

Processing Analytic Queries

For data stored in analytic workspaces, the OLAP calculation engine performs analytic operations and supports sophisticated analysis, such as modeling and what-if analysis. If you require these types of analysis, then you need analytic workspaces. The OLAP engine also provides the fastest run-time response to analytic queries, which is important if you anticipate user sessions that are heavily analytical.

For data stored in a relational schema, analytical operations are performed by SQL. The `SELECT MODEL` clause and the analytical functions (such as `RANK`, `LEAD`, and `LAG`) support calculations such as year-to-date totals and moving averages.

Creating Summary Data

A basic characteristic of business analysis is hierarchically structured data; detail data is summarized at various levels, which allows trends and patterns to emerge. After the analyst has detected a pattern, he or she can drill down to lower levels to identify the factors that contributed to this pattern.

The creation and maintenance of summary data is a serious issue for DBAs. If no summary data is stored, then all summarizations must be performed in response to individual queries. This can easily result in unacceptably slow response time. At the other extreme, if all summary data is stored, then the database can quickly multiply in size.

Analytic workspaces store the data much more efficiently than relational tables and return answer sets to ad-hoc queries as quickly as routine reports. Thus, analytic workspaces provide better support as ad-hoc querying and custom measures become more prevalent.

How Analytic Workspaces Store Summary Data

Analytic workspaces store aggregate data in the same objects as the base level data. Aggregates can be stored permanently in the analytic workspace, or only for the duration of an individual session, or only for a single query. Aggregation rules identify which aggregates are stored for each measure. When an application queries the analytic workspace, either the aggregate values have already been calculated and can simply be retrieved, or they can be calculated **on the fly** from a small number of stored aggregates. Analytic workspaces are optimized for multidimensional calculations, making these run-time summarizations extremely fast.

How Relational Schemas Store Aggregate Data

Relational schemas store aggregate data in materialized views. Queries can be issued directly against the source tables or the materialized views. When the queries are issued against the source tables, Oracle Database obtains stored aggregates for the answer set from the materialized views. Query rewrite is possible when the materialized views have been created using SQL similar to that used for the query. If the materialized views do not exist or the query cannot be rewritten, then the aggregates are calculated on the fly from the source tables.

A relational schema with a relatively small number of materialized views can support the reports that access a known slice of the data. However, extensive use of ad-hoc queries and user-defined custom measures create a random situation in which any part of the data store may be queried and summarized. A relational schema for OLAP may require hundreds of materialized views, which can result in degraded performance and an a significant increase in the size of the database.

Components of Oracle OLAP

The OLAP option is installed with Oracle Database 10g Enterprise Edition. The following components are installed from the database (db) disk:

- [OLAP Analytic Engine](#)
- [Analytic Workspaces](#)
- [SQL Interface to OLAP](#)
- [OLAP DML](#)
- [Analytic Workspace Java APIs](#)
- [OLAP API](#)
- [OLAP Catalog](#)

These components are installed from the client disk:

- [Analytic Workspace Manager](#)
- [OLAP Worksheet](#)

These OLAP components are described in the following paragraphs. The relationships among them are described throughout this guide.

OLAP Analytic Engine

The OLAP analytic engine supports the selection and rapid calculation of multidimensional data. The status of an individual session persists to support a series of queries, which is typical of analytical applications; the output from one query is easily used as input to the next query. A comprehensive set of data manipulation tools supports modeling, aggregation, allocation, forecasting, and what-if analysis. The OLAP engine runs within the Oracle kernel.

Analytic Workspaces

Analytic workspaces store data in a multidimensional format, as described previously in "[About Multidimensional Data Stores](#)" on page 1-7. An analytic workspace is stored as a table in a relational schema. Individual workspace objects are stored in one or more rows as LOBs. This storage structure permits the analytic workspace to be partitioned and for multiple users to write to the analytic workspace simultaneously.

Analytic Workspace Manager

Analytic Workspace Manager provides an easy-to-use interface for creating and managing analytic workspaces in **database standard form** so they can be queried by OLAP tools. It enables you to develop a logical dimensional model of your data quickly and easily, map logical objects to relational data sources, and load and aggregate the data. Using Analytic Workspace Manager, you can manage the life cycle of your analytic workspaces. You can save the logical model as an XML file.

Analytic Workspace Manager also contains tools for upgrading Oracle9i analytic workspaces and Express databases.

OLAP Worksheet

OLAP Worksheet is an interactive environment for working with analytic workspaces, similar to SQL*Plus Worksheet. It provides easy access to the OLAP DML, which is the native language of analytic workspaces. You can switch between two different modes, one for working with analytic workspaces in the OLAP DML, and the other for working with relational tables and views in SQL. It is available through Analytic Workspace Manager or as a separate executable.

SQL Interface to OLAP

The SQL interface to OLAP provides access to analytic workspaces from SQL. The SQL interface is implemented in PL/SQL packages.

For more information, refer to the *Oracle OLAP Reference*.

OLAP DML

OLAP DML is the native language of analytic workspaces. It is a data definition and manipulation language for creating analytic workspaces, defining data containers, and manipulating the data stored in these containers. All other levels of operation (GUIs, Java, and SQL) resolve to the OLAP DML. It offers the maximum power and flexibility in acquiring, manipulating, and analyzing data.

If you are upgrading from Oracle Express, or if your data is stored in formats not supported by the higher level tools, then you may work directly in the OLAP DML at an early stage. Otherwise, you may use the OLAP DML directly only to enhance the functionality of your analytic workspaces.

Analytic Workspace Java APIs

The Analytic Workspace Java APIs support the creation and maintenance of analytic workspaces in Java. They provide a programmatic method for defining a logical dimensional data model and instantiating that model in an analytic workspace. These APIs are used in Analytic Workspace Manager to create and modify analytic workspaces.

See Also: *Oracle OLAP Analytic Workspace Java API Reference*

OLAP API

The OLAP API is a Java-based programming interface for OLAP applications, and it supports OracleBI Beans.

OracleBI Beans contains building blocks for developing analytic applications in Java, and it is available for use with JDeveloper. If you are an applications developer, then

you will use OracleBI Beans in your OLAP applications. OracleBI Beans is not included with the OLAP option, but it requires a Oracle Database with the OLAP option.

See Also: *Oracle OLAP Java API Reference*

OLAP Catalog

OLAP Catalog provides OLAP applications with a query interface to data stored in star and snowflake schemas. It defines fact tables and dimension tables as logical dimensional objects such as measures, dimensions, hierarchies, levels, and attributes. OLAP Catalog consists of write APIs, which are a set of PL/SQL procedures, and read APIs, which are relational views within Oracle Database.

Implementing an Analytic Workspace

Analytic workspaces can be created in a variety of ways, depending on the characteristics of the data source and your own personal preference. However, the basic process is the same for all of them.

These are the basic stages:

1. [Identifying Business Goals](#)
2. [Identifying Data Sources](#)
3. [Defining a Logical Model](#)
4. [Mapping, Loading, and Aggregating the Data](#)
5. [Generating Information-Rich Data](#)

Identifying Business Goals

The first stage of implementing an analytic workspace is defining the analysis requirements of end users. By interviewing them, you can identify the business analysis questions they want to answer with an OLAP application. With this information, you can determine the business measures that must be available, the base level at which the measures must be stored, and the types of data calculations that must be available.

See Also: [Chapter 2](#) for a sample approach to identifying business goals.

Identifying Data Sources

To load data into an analytic workspace using OLAP tools, the source data must be in relational tables or views. The tables can be in a star, snowflake, or network schema, as described in [Chapter 3](#). Analytic Workspace Manager supports direct mapping of logical objects to relational columns. If your relational data requires transformation, then you must define views that perform the transformations.

If your source data is not stored in relational tables or requires extensive transformation, then you can choose from one of these options:

- Use Oracle Warehouse Builder to create a star schema from disparate data sources, then use Analytic Workspace Manager to create an analytic workspace from the relational data. Your Information Technology (IT) department may do this task for you. Choose this option when you are developing a new analytic workspace.

- Use Oracle Warehouse Builder to create an analytic workspace, then use Analytic Workspace Manager to manage it. Choose this option when the design phase is complete and the analytic workspace is in a production environment. The IT department can manage this task along with its other maintenance tasks.

See Also: *Oracle Warehouse Builder User's Guide*

Defining a Logical Model

A logical dimensional model defines the dimensions, levels, hierarchies, attributes, cubes, and measures of your data. The Model View in Analytic Workspace Manager enables you to define the logical model by defining the individual objects and the relationships among them. When you save the definition of a logical object, Analytic Workspace Manager creates the physical objects in an analytic workspace that are needed to instantiate the logical object in **database standard form**.

See Also: [Chapter 3](#) for an introduction to the Model View of Analytic Workspace Manager

Mapping, Loading, and Aggregating the Data

Analytic Workspace Manager provides a graphical tool for mapping the logical objects to physical data stores. You can drag-and-drop tables and views from schemas to which you have access onto a mapping canvas. You can then draw lines from the appropriate columns to the logical objects that you have defined in the analytic workspace. Using a wizard, you can load data into the analytic workspace and aggregate the data using the rules that you provided.

Generating Information-Rich Data

As part of setting up an analytic workspace, you can define numerous derived measures using the Calculation Wizard, which is described in "[Creating Custom Measures](#)" on page 1-4.

Implementing a Relational Data Warehouse for OLAP

OracleBI Discoverer Plus OLAP, OracleBI Spreadsheet Add-In, and custom OracleBI Beans applications can run directly against relational tables, either instead of or in addition to analytic workspaces.

These are the basic stages:

1. [Identifying Business Goals](#)
2. [Identifying Data Sources](#)
3. [Defining a Logical Model](#)
4. [Generating Summary Data](#)

Identifying Business Goals

The first stage is defining the analysis requirements of your end users. By interviewing end users, you can identify the business analysis questions they want to answer with an OLAP application. With this information, you can determine the business measures that must be available, the base level at which the measures must be stored, and the types of data calculations that must be available to analysts.

See Also: [Chapter 2](#) for a sample approach to identifying business goals

Identifying Data Sources

For OLAP tools to query data in a relational data warehouse, the source data must be in a star or snowflake schema that conforms to specific requirements.

If your source data does not conform to those requirements, then use Oracle Warehouse Builder to create a star schema.

See Also: *Oracle Warehouse Builder User's Guide*

Defining a Logical Model

OLAP tools query the OLAP Catalog; they do not issue queries directly against the data source. There are several methods of defining a logical dimensional model of a relational schema:

- OLAP Management tool in Oracle Enterprise Manager Database Control
- CWM2 PL/SQL procedures
- Oracle Warehouse Builder

The storage format of your data determines which of these methods you can use.

See Also: [Chapter 7](#) for a discussion of the requirements for using each method of defining OLAP Catalog metadata.

Generating Summary Data

Use the DBMS_ODM PL/SQL package to create materialized views for OLAP. Do not use any other method of generating materialized views, because they will not be used by query rewrite when formulating an answer set to an OLAP query.

See Also: [Chapter 8](#) for information about using DBMS_ODM.

Upgrading Oracle Database 10g Release 1 Analytic Workspaces

If you created an analytic workspace in Oracle 10g Release 1, you can upgrade it to Release 2 using the following procedure. Upgrading is optional. However, upgrading enables you to use the new features of Analytic Workspace Manager 10.2, such as additional aggregation operators for compressed composites, support for multiple languages, and performance improvements.

To upgrade an analytic workspace, take these steps:

1. Open Analytic Workspace Manager in the Model View.
2. In the navigation tree, select the name of the Oracle Database instance where your analytic workspace is stored.
3. On the Basic tab of the Database property sheet, verify that the database is running in 10.2 compatibility mode.
4. Right-click the analytic workspace, and select **Upgrade Analytic Workspace to 10.2**.
5. Complete the Analytic Workspace Upgrade to Version 10.2 dialog box.
Click **Help** for additional information.

Upgrading Oracle9i Analytic Workspaces

If you have analytic workspaces that were created in Oracle9i, then you should upgrade them to take advantage of new features such as partitioning and compressed composites.

Upgrading may break custom OLAP DML programs. For this reason, you can choose to upgrade at a time that is convenient for you. You can continue to manage your older analytic workspaces by using an older version of Analytic Workspace Manager (such as Oracle9i Release 9.2.0.4.1).

Any new analytic workspaces that you create using the new Oracle Database 10g version of Analytic Workspace Manager will automatically be in 10g standard form, as long as Oracle Database is running in 10g compatibility mode.

If Oracle Database is running in 9i compatibility mode, then you will continue to work the same way as before without upgrading the analytic workspaces.

To upgrade an analytic workspace, take these steps:

1. Set the `COMPATIBLE` parameter to 10.0.0.0 or later in the database initialization file.
2. Upgrade the physical storage format.
3. Upgrade the standard form metadata.

You can upgrade the physical storage format without upgrading the standard form metadata, if you wish. This change will improve performance and support partitioning. However, the analytic workspace will not be enabled dynamically for OracleBI Beans until you upgrade the metadata.

You can perform the upgrade steps either in the Object View of Analytic Workspace Manager or in PL/SQL.

Upgrading the Physical Storage Format

Convert the physical storage format by using either of these methods:

- Recreate the analytic workspace by following these steps:
 1. Export the contents to an EIF file.
 2. Delete the old analytic workspace.
 3. Create a new, empty analytic workspace.
 4. Import the contents from the EIF file.

You can export and import in Analytic Workspace Manager. For more information, see these topics in Help: “Exporting Workspace Objects” and “Importing Workspace Objects”

- Use the PL/SQL conversion program by following these steps:
 1. Rename the old analytic workspace so the upgraded analytic workspace has the original name, using the following syntax:

```
EXECUTE DBMS_AW.AW_RENAME('orig_name', 'temp_name');
```

2. Run the upgrade procedure, using the syntax:

```
EXECUTE DBMS_AW.CONVERT('temp_name', 'orig_name', 'tablespace');
```

Tip: Use a program such as SQL*Plus to execute these procedures. For their full syntax, refer to the *Oracle OLAP Reference*.

3. Delete the old analytic workspace (*temp_name*) using the PL/SQL DBMS_AW.AW_DELETE procedure.

```
EXECUTE DBMS_AW.AW_DELETE('temp_name');
```

Upgrading the Standard Form Metadata

To upgrade the standard form metadata, follow these steps:

1. In Analytic Workspace Manager, open the Object View.
2. Expand the navigation tree until you see the name of the analytic workspace.
3. Right-click the analytic workspace and choose **Upgrade Analytic Workspace From 9i to 10g Standard Form** from the popup menu.
4. Upgrade to Release 2 by following the instructions in ["Upgrading Oracle Database 10g Release 1 Analytic Workspaces"](#) on page 1-15.

Alternatively, you can use DBMS_AWM PL/SQL procedures CREATE_DYNAMIC_AW_ACCESS and DELETE_ALL_AW_ACCESS to perform the upgrade. Refer to the *Oracle OLAP Reference* for the syntax and usage notes.

The Sample Schema

This guide uses the Global schema for its examples. This chapter describes this schema and explains how it will be mapped to dimensional objects. It consists of the following topics:

- [Case Study Scenario](#)
- [Identifying Required Business Facts](#)
- [Designing a Logical Data Model for Global Computing](#)
- [The Global Schema](#)

Case Study Scenario

The fictional Global Computing Company was established in 1990. Global Computing distributes computer hardware and software components to customers on a worldwide basis. The Sales and Marketing department has not been meeting its budgeted numbers. As a result, this department has been challenged to develop a successful sales and marketing strategy.

Global Computing operates in an extremely competitive market. Competitors are numerous, customers are especially price-sensitive, and profit margins tend to be narrow. In order to grow profitably, Global Computing must increase sales of its most profitable products.

Various factors in Global Computing's current business point to a decline in sales and profits:

- Traditionally, Global Computing experiences low third-quarter sales (July through September). However, recent sales in other quarters have also been lower than expected. The company has experienced bursts of growth but, for no apparent reason, has had lower first-quarter sales during the last two years as compared with prior years.
- Global has been successful with its newest sales channel, the Internet. Although sales within this channel are growing, overall profits are declining.
- Perhaps the most significant factor is that margins on personal computers - previously the source of most of Global Computing's profits - are declining rapidly.

Global Computing needs to understand how each of these factors is affecting its business.

Current reporting is done by the IT department, which produces certain standard reports on a monthly basis. Any ad hoc reports are handled on an as-needed basis and are subject to the time constraints of the limited IT staff. Complaints have been

widespread within the Sales and Marketing department, with regard to the delay in response to report requests. Complaints have also been numerous in the IT department, with regard to analysts who change their minds frequently or ask for further information.

The Sales and Marketing department has been struggling with a lack of timely information about what it is selling, who is buying, and how they are buying. In a meeting with the CIO, the VP of Sales and Marketing states, "By the time I get the information, it's no longer useful. I'm only able to get information at the end of each month, and it doesn't have the details I need to do my job."

Reporting Requirements

When asked to be more specific about what she needs, the Vice President of Sales and Marketing identifies the following requirements:

- Trended sales data for specific customers, regions, and segments.
- The ability to provide information and some analysis capabilities to the field sales force. A Web interface would be preferred, since the sales force is distributed throughout the world.
- Detail regarding mail-order, phone, and e-mail sales on a monthly and quarterly basis, as well as a comparison to past time periods. Information must identify when, how, and what is being sold by each channel.
- Margin information on products in order to understand the dollar contribution for each sale.
- Knowledge of percent change versus the prior and year-ago period for sales, units, and margin.
- The ability to perform analysis of the data by ad hoc groupings.

The CIO has discussed these requirements with his team and has come to the conclusion that a standard reporting solution against the production order entry system would not be flexible enough to provide the required analysis capabilities. The reporting requirements for business analysis are so diverse that the projected cost of development, along with the expected turnaround time for requests, would make this solution unacceptable.

The CIO's team recommends using an analytic workspace to support analysis. The team suggests that the Sales and Marketing department's IT group work with Corporate IT to build an analytic workspace that meets their needs for information analysis.

Business Goals

The development team identifies the following high-level business goals that the project must meet:

- Global Computing's strategic goal is to increase company profits by increasing sales of higher margin products and by increasing sales volume overall.
- The Sales and Marketing department objectives are to:
 - Analyze industry trends and target specific market segments
 - Analyze sales channels and increase profits
 - Identify product trends and create a strategy for developing the appropriate channels

Information Requirements

Once you have established business goals, you can determine the type of information that will help achieve these goals. To understand how end users will examine the data in the analytic workspace, it is important to conduct extensive interviews. From interviews with key end users, you can determine how they look at the business, and what types of business analysis questions they want to answer

Business Analysis Questions

Interviews with the VP of Sales and Marketing, salespeople, and market analysts at Global Computing reveal the following business analysis questions:

- What products are profitable?
- Who are our customers, and what and how are they buying?
- What accounts are most profitable?
- What is the performance of each distribution channel?
- Is there still a seasonal variance to the business?

We can examine each of these business analysis questions in detail.

What products are profitable?

This business analysis question consists of the following questions:

- What is the percent of total sales for any item, product family, or product class in any month, quarter or year, and in any distribution channel? How does this percent of sales differ from a year ago?
- What is the unit price, unit cost, and margin for each unit for any item in any particular month? What are the price, cost, and margin trends for any item in any month?
- What items were most profitable in any month, quarter, or year, in any distribution channel, and in any geographic area or market segment? How did profitability change from the prior period? What was the percent change in profitability from the prior period?
- What items experienced the greatest change in profitability from the prior period?
- What items contributed the most to total profitability in any month, quarter, or year, in any distribution channel, and in any geographic area or market segment?
- What items have the highest per unit margin for any particular month?
- In summary, *what are the trends?*

Who are our customers, and what and how are they buying?

This business analysis question consists of the following questions:

- What were sales for any item, product family, or product class in any month, quarter, or year?
- What were sales for any item, product family, or product class in any distribution channel, geographic area, or market segment?
- How did sales change from the prior period? What was the percent change in sales from the prior period?

- How did sales change from a year ago? What was the percent change in sales from a year ago?
- In summary, *what are the trends?*

What accounts are most profitable?

This business analysis question consists of the following questions:

- What accounts are most profitable in any month, quarter, or year, in any distribution channel, by any item, product family, or product class?
- What were sales and extended margin (gross profit) by account for any month, quarter, or year, for any distribution channel, and for any product?
- How does account profitability compare to the prior time period?
- Which accounts experienced the greatest increase in sales as compared to the prior period?
- What is the percent change in sales from the prior period? Did the percent change in profitability increase at the same rate as the percent change in sales?
- In summary, *what are the trends?*

What is the performance of each distribution channel?

This business analysis question consists of the following questions:

- What is the percent of sales to total sales for each distribution channel for any item, product family, or product class, or for any geographic area or market segment?
- What is the profitability of each distribution channel: direct sales, catalog sales, and the Internet?
- Is the newest distribution channel, the Internet, "cannibalizing" catalog sales? Are customers simply switching ordering methods, or is the Internet distribution channel reaching additional customers?
- In summary, *what are the trends?*

Is there still a seasonal variance to the business?

This business analysis question consists of the following questions:

- Are there identifiable seasonal sales patterns for particular items or product families?
- How do seasonal sales patterns vary by geographic location?
- How do seasonal sales patterns vary by market segment?
- Are there differences in seasonal sales patterns as compared to last year?

Summary of Information Requirements

By examining the types of analyses that users wish to perform, we can identify the following key requirements for analysis:

- Global Computing has a strong need for profitability analysis. The company must understand profitability by product, account, market segment, and distribution channel. It also needs to understand profitability trends.

- Global Computing needs to understand how sales vary by time of year. The company must understand these seasonal trends by product, geographic area, market segment, and distribution channel.
- Global Computing has a need for ad hoc sales analysis. Analysis must identify what products are sold to whom, when these products are sold, and how customers buy these products.
- The ability to perform trend analysis is important to Global Computing.

Identifying Required Business Facts

The key analysis requirements reveal the business facts that are required to support analysis requirements at Global Computing.

These facts are ordered by time, product, customer shipment or market segment, and distribution channel:

Sales
 Units
 Change in sales from prior period
 Percent change in sales from prior period
 Change in sales from prior year
 Percent change in sales from prior year
 Product share
 Channel share
 Market share
 Extended cost
 Extended margin
 Extended margin change from prior period
 Extended margin percent change from prior period
 Units sold, change from prior period
 Units sold, percent change from prior period
 Units sold, change from prior year
 Units sold, percent change from prior year

These facts are ordered by item and month:

Unit price
 Unit cost
 Margin per unit

Designing a Logical Data Model for Global Computing

"[Business Goals](#)" on page 2-2 identifies the business facts that will support analysis requirements at Global Computing. Next, we will identify the dimensions, levels, and attributes in a logical data model. We will also identify the relationships within each dimension. The resulting data model will be used to design the Global schema, the logical dimensional model, and the analytic workspace.

Identifying Dimensions

Four dimensions will be used to organize the facts in the database.

- Product shows how data varies by product.
- Customer shows how data varies by customer or geographic area.

- Channel shows how data varies according to each distribution channel.
- Time shows how data varies over time.

Identifying Levels

Now that we have identified dimensions, we can identify the levels of summarization within each dimension. Analysis requirements at Global Computing reveal that:

- There are three distribution channels: Sales, Catalog, and Internet. These three values are the lowest level of detail in the data warehouse and will be grouped in the Channel level. From the order of highest level of summarization to the lowest level of detail, levels will be Total Channel and Channel.
- Global performs customer and geographic analysis along the line of shipments to customers and by market segmentation. Shipments and Market Segment will be two hierarchies in the Customer dimension. In each case, the lowest level of detail in the data model is the Ship To location.
 - When analyzing along the line of customer shipments, the levels of summarization will be (highest to lowest): Total Customer, Region, Warehouse, and Ship To.
 - When analyzing by market segmentation, the levels of summarization will be (highest to lowest): Total Market, Market Segment, Account, and Ship To.
- In the examples in this guide, Product is mapped to a parent-child table and is defined as a value-based hierarchy rather than a level-based hierarchy. Thus, no levels are defined for Product.
- The Time dimension will have three levels (highest to lowest): Year, Quarter, and Month.

Within the Channel, Customer, and Product dimensions, we added a Total or All level as the highest level of summarization. Adding this highest level provides additional flexibility as application users analyze data.

Identifying Hierarchies

We will identify the hierarchies that organize the levels within each dimension. To identify hierarchies, we will group the levels in the correct order of summarization and in a way that supports the identified types of analysis.

For the Channel, Product, and Time dimensions, Global Computing requires only one hierarchy for each dimension. For the Customer dimension, however, Global Computing requires two hierarchies. Analysis within the Customer dimension tends to be either by geographic area or market segment. Therefore, we will organize levels into two hierarchies, Shipments and Market Segment.

Identifying Stored Measures

"[Identifying Required Business Facts](#)" on page 2-5 lists 21 business facts that are required to support the analysis requirements of Global Computing. Of this number, only four facts need to be acquired from the transactional database:

- Units
- Sales
- Unit Price
- Unit Cost

All of the other facts can be derived from these basic facts. The derived facts can be calculated in the analytic workspace on demand. If experience shows that some of these derived facts are being used heavily and the calculations are putting a noticeable load on the system, then some of these facts can be calculated and stored in the analytic workspace as a data maintenance procedure.

The Global Schema

You can download the Global schema from

<http://www.oracle.com/technology/products/OracleBI/olap/olap.html>

and use it to try the examples shown throughout this guide. Instructions for installing the schema are provided in the readme file.

The Global schema contains alternative data sources for the logical Global model, so that you can explore these variations:

- Star schema
- Snowflake schema
- Parent-child table

The examples in this guide use a parent-child table for the Product dimension, a star schema for the Customer and Channel dimensions, and a snowflake schema for the Time dimension. See [Chapter 3](#) for schema diagrams.

Part II

Creating and Managing Analytic Workspaces

Part II contains information about creating analytic workspaces. It contains the following chapters:

- [Chapter 3, "Creating an Analytic Workspace"](#)
- [Chapter 4, "Predicting Future Performance"](#)
- [Chapter 5, "Developing Java Applications for OLAP"](#)
- [Chapter 6, "Administering Oracle OLAP"](#)

Creating an Analytic Workspace

This chapter explains how to design a logical data model and create a standard form analytic workspace using Analytic Workspace Manager.

This chapter contains the following topics:

- [Introduction to Analytic Workspace Manager](#)
- [Getting Started with Analytic Workspace Manager](#)
- [Identifying the Source Data](#)
- [Creating a Standard Form Workspace Using Analytic Workspace Manager](#)
- [Creating Logical Dimensions](#)
- [Creating Logical Cubes](#)
- [Making Data Storage Decisions](#)
- [Defining Rules for Summarizing Data](#)
- [Mapping Logical Objects to Data Sources](#)
- [Maintaining the Data](#)
- [Case Study: Creating the Global Analytic Workspace](#)
- [Case Study: Creating the Sales History Analytic Workspace](#)

Introduction to Analytic Workspace Manager

Analytic Workspace Manager is the primary tool for creating, developing, and managing analytic workspaces. The main window provides two views: the Model View and the Object View. You can switch between views using the View menu. In addition, there are menus, a toolbar, a navigation tree, and property sheets. When you select an object in the navigation tree, the property sheet to the right provides detailed information about that object. When you right-click an object, you get a choice of menu items with appropriate actions for that object.

You can also conduct an interactive session by opening OLAP Worksheet and using the OLAP DML. You can switch between the console and OLAP Worksheet, and have an up-to-date view of your workspace in each one, because they share the same session.

Analytic Workspace Manager has a full online Help system, which includes context-sensitive Help.

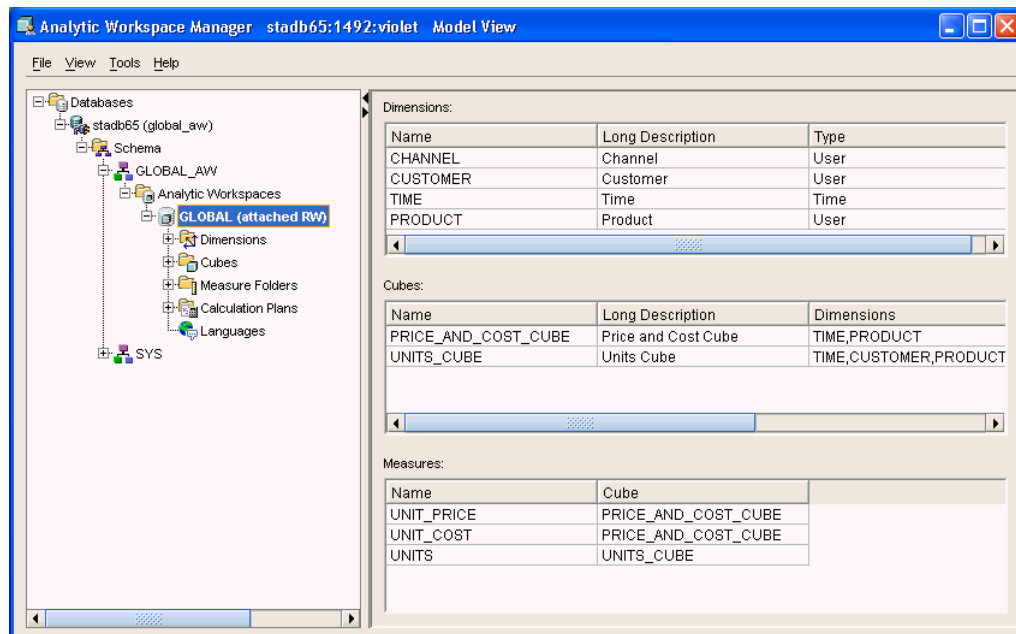
Model View

The Model View enables you to define a logical dimensional model composed of dimensions, levels, hierarchies, attributes, measures, calculated measures, and measure folders. The model is stored in the analytic workspace as **database standard form** metadata.

A drag-and-drop user interface facilitates mapping of the logical objects to columns in relational tables and views in Oracle Database. The source columns can be star, snowflake, or any other schema design that supports the logical model.

Figure 3–1 shows the logical objects created in the GLOBAL analytic workspace.

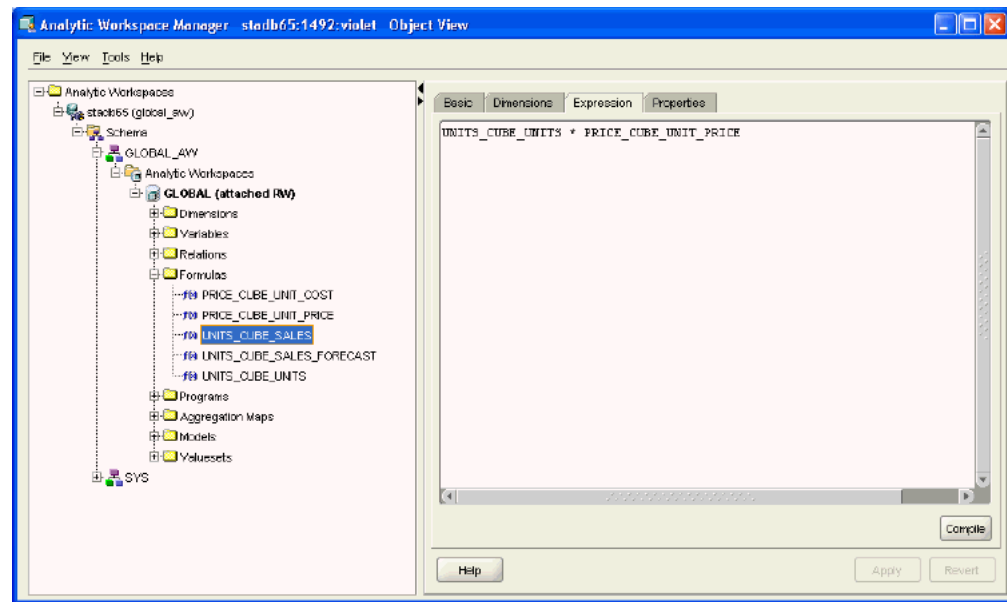
Figure 3–1 Model View in Analytic Workspace Manager



Object View

The Object View provides a graphical user interface to the OLAP DML. You can create, modify, and delete individual workspace objects. This view is provided for users who are familiar with the OLAP DML and want to upgrade Express databases, modify custom applications, or customize a new analytic workspace. Be very careful when working with a standard form analytic workspace, so that you do not create inconsistencies in the metadata.

Figure 3–2 shows the Object View. A formula named UNITS_CUBE_SALES is currently selected in the navigation tree, and the right pane shows the Expression tab of the property sheet. This tab shows the OLAP DML expression used to calculate the formula.

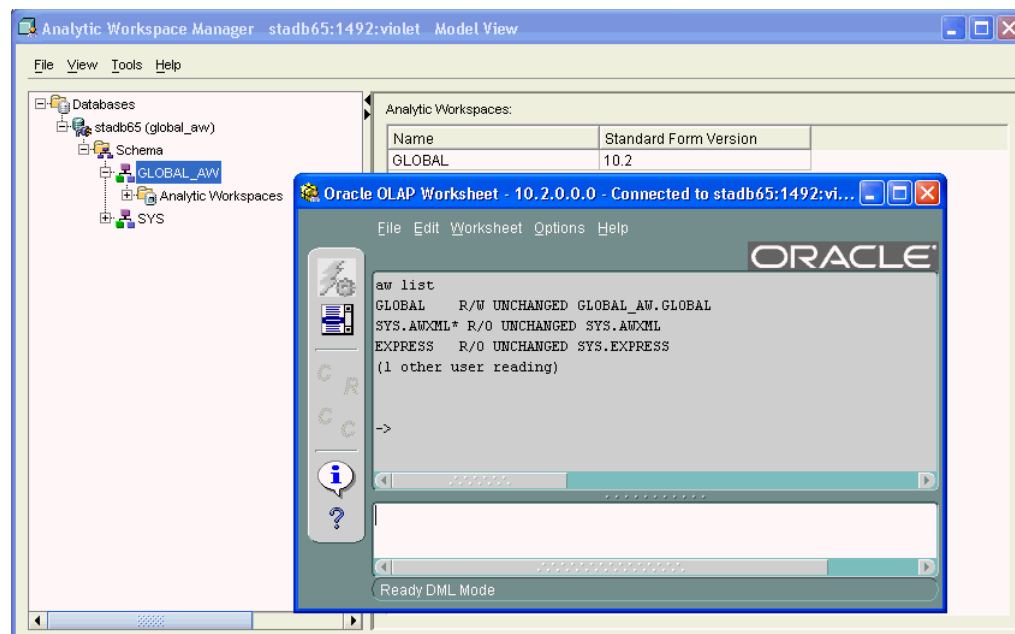
Figure 3–2 Object View in Analytic Workspace Manager

OLAP Worksheet

OLAP Worksheet provides full use of the OLAP DML for users who need to manage the contents of an object or execute a program. It opens in a separate window from the Analytic Workspace Manager console. This window provides menus, a toolbar, an input pane for OLAP DML commands on the bottom, and an output pane on the top.

Figure 3–3 shows OLAP Worksheet opened from Analytic Workspace Manager. Notice that the GLOBAL workspace is attached with read/write access in both OLAP Worksheet (as shown by the `AW LIST` command) and Analytic Workspace Manager (as shown by the Model View navigation tree). The two applications share the same session.

The OLAP DML Reference is available through the Help menu.

Figure 3–3 OLAP Worksheet Opened From Analytic Workspace Manager

Getting Started with Analytic Workspace Manager

In this section, you will learn how to obtain the Analytic Workspace Manager software, install it on your computer, and make a connection to Oracle Database.

Installing Analytic Workspace Manager

Analytic Workspace Manager is distributed with Oracle Database. Three disks compose the Oracle Database 10g Release 2 installation set: Database (db), Companion, and Client. Analytic Workspace Manager is on the Client disk.

If you are installing on the same system as the database, then choose a Custom installation and install into the same Oracle home directory as the database. Select **OLAP Analytic Workspace Manager and Worksheet** from the list of components.

If you are installing on a remote system, then choose either an Administrator or a Custom installation.

See Also: An installation guide for your platform, such as:

- *Oracle Database Client Quick Installation Guide for 32-Bit Windows*
- *Oracle Database Client Installation Guide for 32-Bit Windows*

Opening Analytic Workspace Manager

On Windows, use the Start menu to open Analytic Workspace Manager:

Start > All Programs > Oracle - Oracle_home > Integrated Management Tools > OLAP Analytic Workspace Manager and Worksheet

On Linux, open Analytic Workspace Manager from the shell command line:

```
$ORACLE_HOME/olap/awm/awm.sh
```

Defining a Database Connection

You can define a connection to each database that you use for OLAP. After you have defined a connection, the database instance is listed in the navigation tree for you to access at any time.

To define a database connection:

1. Right-click the top Databases folder in the navigation tree, then choose **Add Database to Tree** from the pop-up menu.
2. Complete the Add Database to Tree dialog box.

Opening a Database Connection

To connect to a database:

1. Click the plus icon (+) next to a database in the navigation tree.
2. Complete the Connect to Database dialog box.

Identifying the Source Data

Using Analytic Workspace Manager, you can:

- Design the logical dimensional model for the analytic workspace
- Map logical objects to relational data sources
- Load and aggregate the data

These steps are very closely related. The data that supports your logical model must exist in your database, and you must have `SELECT` privileges on the tables containing the data so you can load it into your analytic workspace.

Schema Requirements

Your goal in using Analytic Workspace Manager is to create a multidimensional data store that supports business analysis. The analytic workspace that you create must contain the logical objects described in "[The Logical Dimensional Data Model](#)" on page 1-5. For the source data to support a logical dimensional data model, these relationships must exist:

- **Dimensions.** You can map dimensions, levels, and attributes to any collection of tables or views that identify the child-parent relationships and the member-attribute relationships. The tables and views can be in one schema or owned by multiple schemas. When mapping dimensions, you can choose from these categories of schemas:
 - [Star Schema](#)
 - [Snowflake Schema](#)
 - [Other](#)

You can identify different dimensions as having different schema characteristics, for example, Customer could be a star schema (all levels and their attributes are in one table) and Time could be a snowflake schema (levels are in two or more tables with their attributes).

- **Measures.** You can map measures to any table or view that contains the appropriate data.

Hierarchies and cubes are strictly metadata objects and are not mapped to data sources.

Tables may contain columns of no importance to your analytic workspace. You can simply omit them from the mappings, and Analytic Workspace Manager will ignore them.

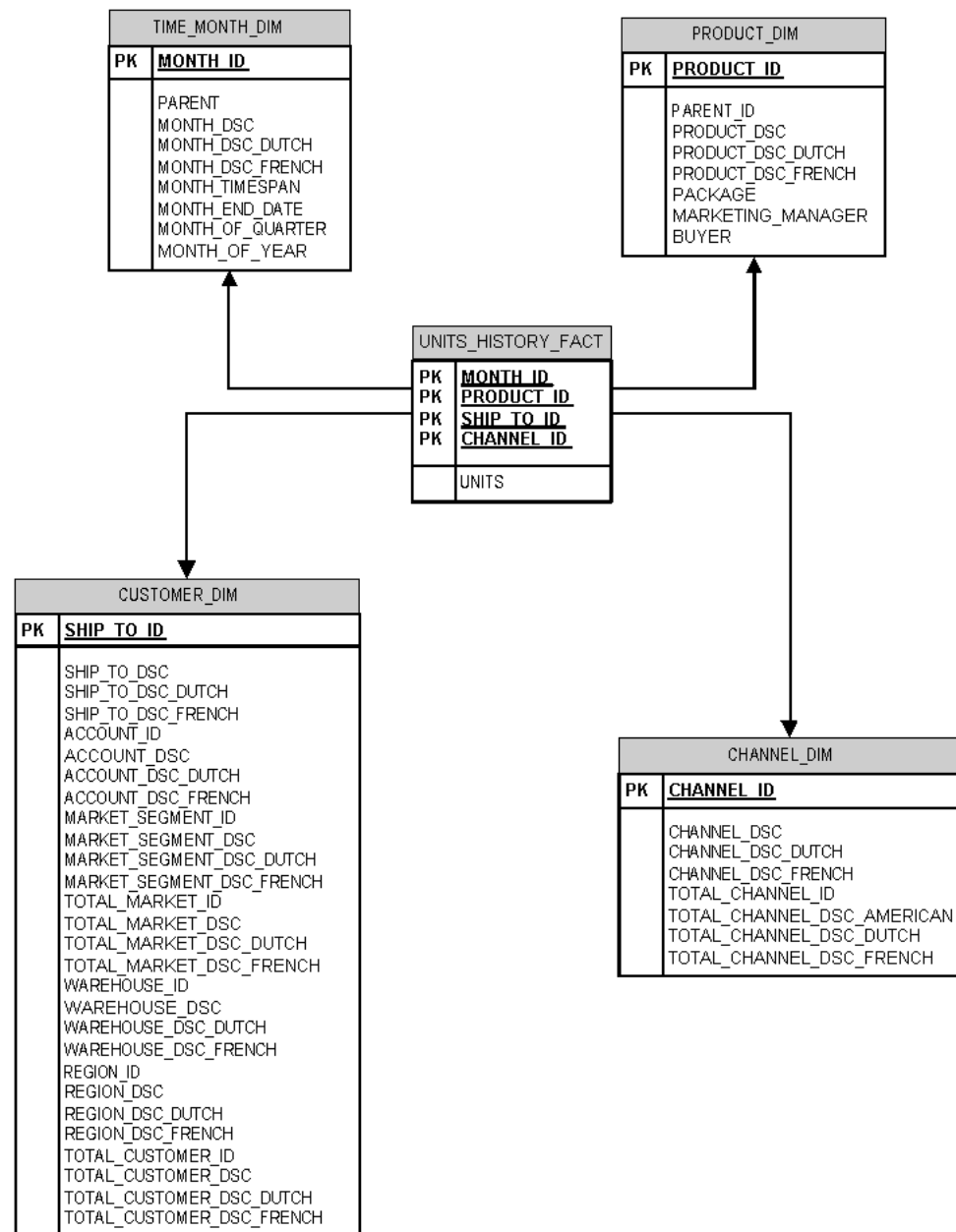
Star Schema

A star schema is the simplest of the three types. It is called a star schema because a diagram of this schema resembles a star, with points radiating from a central table. The center of the star is a fact table and the points of the star are the dimension tables.

- Dimension tables define the dimensions. In a star schema, all of the information for a dimension is stored in one table.
- Fact tables contain foreign keys from each dimension table and a column for each measure.

[Figure 3–4](#) shows the relationships in a star schema using the GLOBAL relational tables. These tables provide the data for the Units Cube. These source tables illustrate different types of schema designs:

- PRODUCT_DIM and CHANNEL_DIM are level-based dimensions in a star schema.
- PRODUCT_CHILD_PARENT is a parent-child table that supports a value-based hierarchy. There are no level columns.
- TIME_MONTH_DIM is the base-level table of a snowflake schema. The Time tables are described in "[Snowflake Schema](#)" on page 3-7.

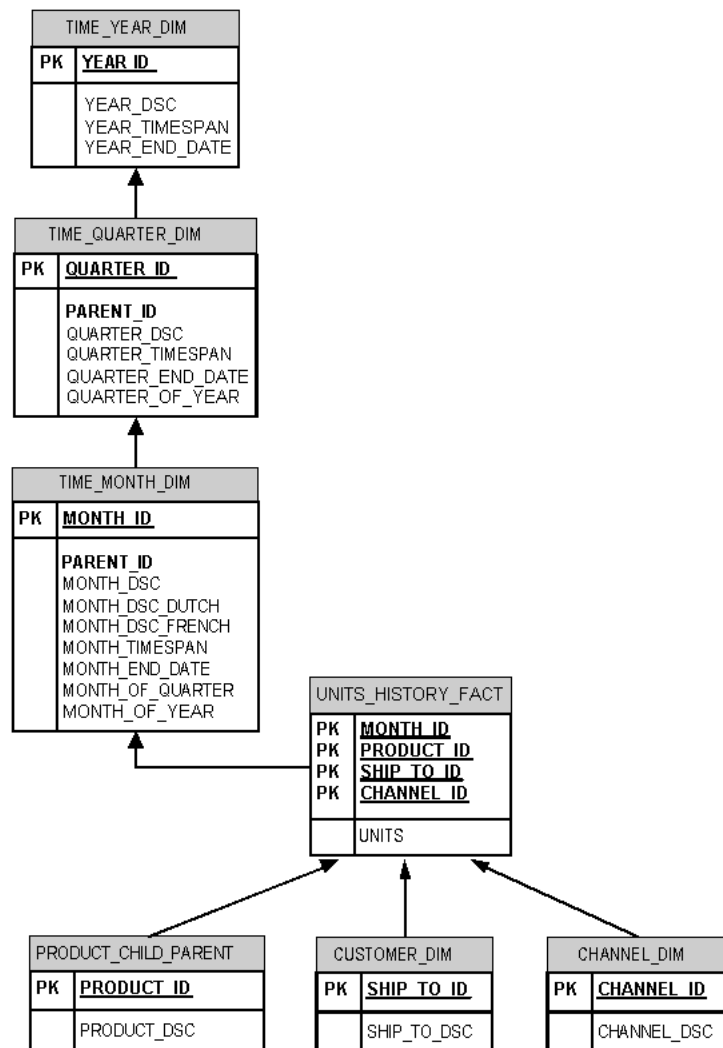
Figure 3–4 Star Schema

Snowflake Schema

A snowflake schema is a type of star schema. It is called a snowflake schema because a diagram of the schema resembles a snowflake. Snowflake schemas normalize dimensions to eliminate redundancy. That is, the dimension data has been divided into multiple tables instead of one large table. Each level may be in a separate table with its attributes.

Figure 3–5 shows the Time dimension in a snowflake schema, with separate tables for months, quarters, and years.

Note that the other dimensions are shown only partially in this snowflake diagram.

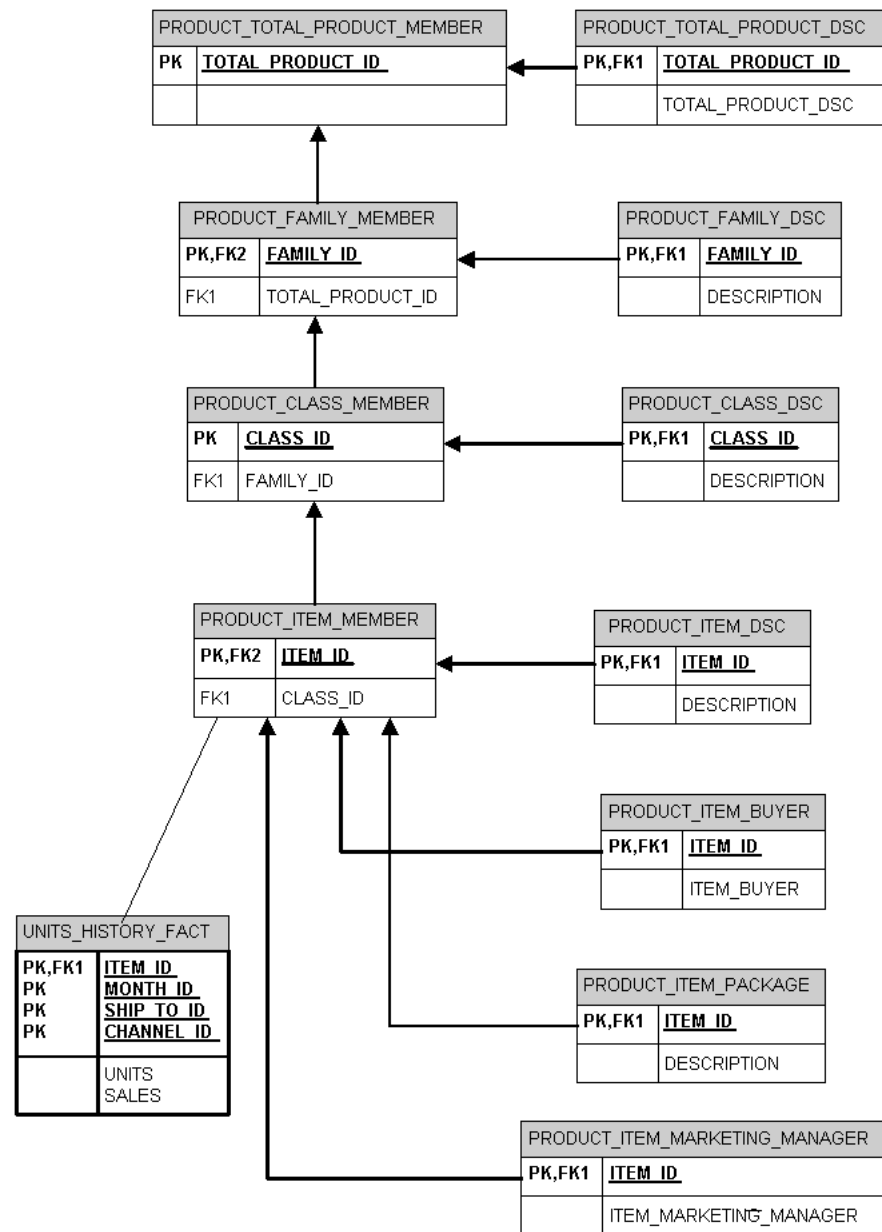
Figure 3–5 Normalized Time Dimension in a Snowflake Schema

Other

Any schema can be used that contains the parent-child relationships and the member-attribute relationships needed to implement dimensions in a dimensional data model. In the most extreme case, each parent-child and member-attribute value pairs for each hierarchy may be in a different table.

Figure 3–6 shows the Product dimension in schema that contains the appropriate relationships. Contrast these 11 tables with the single table shown in Figure 3–5 for the Product dimension in a star schema. Whereas in the star schema, the Product dimension has one source table, this schema has been normalized to store each level and each attribute in a separate table.

Note that the other dimensions are not shown in this diagram.

Figure 3–6 Product Dimension in an "Other" Schema Design

Making Transformations in Your Source Data

Analytic Workspace Manager provides direct mapping of one logical object to one column of a relational table or view. If you need to transform your data, then you can choose between these alternatives:

- Create views that perform the necessary transformations.
- Use an ETL tool such as Oracle Warehouse Builder to generate a star schema. You can then create the analytic workspace using Analytic Workspace Manager.
- Use Oracle Warehouse Builder to generate an analytic workspace in Oracle9i standard form. You can then use Analytic Workspace Manager to upgrade the analytic workspace to Oracle Database 10g standard form. (The second step is not needed when using Oracle Warehouse Builder 10g Release 2.)

Following are some of the basic types of transformations that can be handled by creating views:

- **Load a selection of data.** The Maintenance Wizard loads all rows from a mapped column into the analytic workspace. If you only want a selection of the available data, create a view with a `WHERE` clause.
- **Load data from multiple tables into one cube.** Analytic Workspace Manager currently permits you to map only one table to a cube. You can create a union view or join the tables in a view.
- **Load multiple levels of data.** Analytic Workspace Manager currently permits you to map only one level. Create a view with a `WHERE` clause that selects the base level for the analytic workspace.

Choosing a Build Tool

Both Analytic Workspace Manager and Warehouse Builder can be used to generate analytic workspaces.

Warehouse Builder is designed for Information Technology (IT) professionals who manage production systems. It is a powerful tool that can generate analytic workspaces as one element in a larger ETL process.

Analytic Workspace Manager is an easy-to-use tool designed for application developers, departmental DBAs, and other nonprofessional DBAs. It enables them to design and develop a data model quickly and interactively based on their reporting needs. After the data model has been developed and its design is stable, the IT department may assume responsibility for generating the analytic workspace using Warehouse Builder. Analytic Workspace Manager can be used to enhance the analytic workspaces created by the IT department, such as by adding custom measures.

See Also: *Oracle Warehouse Builder User's Guide*

Creating a Standard Form Workspace Using Analytic Workspace Manager

In the Model View, you can define and build an analytic workspace from relational tables and views. The tables and views can be stored in one or more schemas in which the appropriate data relationships exist, as described in "[Identifying the Source Data](#)" on page 3-5.

How Analytic Workspace Manager Saves Changes

Analytic Workspace Manager saves changes automatically that you make to the analytic workspace. You do not explicitly save your changes.

Saves occur when you take an action such as these:

- Click **OK** or the equivalent button in a dialog box.
For example, when you click **Import** in the Import From EIF File dialog box, the contents are imported, and the revised analytic workspace is committed to the database. Likewise, when you click **Create** in the Create Dimension dialog box, the new dimension is committed to the database.
- Click **Apply** in a property sheet.
For example, when you change the labels on the General property page for an object, the change takes effect when you click **Apply**.

Basic Steps for Creating a Standard Form Workspace

To create an analytic workspace in database standard form:

1. Configure your database instance for OLAP use. Define permanent, temporary, and undo tablespaces, and set the database parameters to values appropriate for data loads. Refer to [Chapter 6](#) for details.

2. Define a database user who will own the analytic workspace. Grant the user the OLAP_USER role and SELECT privileges on the source data tables.

While you can create the workspace in the same schema as the relational tables, doing so can cause problems in defining unique names within a single namespace.

3. Examine the sparsity characteristics of your data, so that you can implement a logical model for the best performance.

See ["What is Sparsity?"](#) on page 3-17 and ["Examining Sparsity Characteristics for GLOBAL"](#) on page 3-28.

4. Open Analytic Workspace Manager and connect to your database instance as the user you defined earlier for this purpose.

5. Create a new analytic workspace container in your database:

- a. In the Model View navigation tree, expand the folders until you see the schema where you want to create the analytic workspace.
- b. Right-click the schema name, then choose **Create Analytic Workspace** from the pop-up menu.
- c. Complete the Create Analytic Workspace dialog box, then choose **Create**.

The new analytic workspace appears in the Analytic Workspaces folder for the schema.

6. Define the logical dimensions for the data.

See ["Creating Logical Dimensions"](#) on page 3-12.

7. Define the logical cubes for the data.

See ["Creating Logical Cubes"](#) on page 3-15.

8. Map the logical items to their data sources.

See ["Mapping Logical Objects to Data Sources"](#) on page 3-22.

9. Load the data.

See ["Mapping Logical Objects to Data Sources"](#) on page 3-22.

10. Define measure folders to simplify access for end users.

See ["Defining Measure Folders"](#) on page 3-26.

When you have finished, you will have an analytic workspace populated with the detail data fetched from relational tables or views. You may also have summarized data and calculated measures.

Adding Functionality to a Standard Form Analytic Workspace

In addition to the basic steps, you can add functionality to an analytic workspace in these ways:

- Support multiple languages by adding translations of metadata and attribute values.

See ["Supporting Multiple Languages"](#) on page 3-26.

- Develop one or more calculation plans for the analytic workspace, so that you can establish the order of calculations. Calculation plans enable you to include dependent calculations in the model.

See ["Creating Calculation Plans"](#) on page 3-27.

Creating Logical Dimensions

Dimensions are the parents of levels, hierarchies, and attributes in the logical model. You define these supporting objects, in addition to the dimension itself, in order to have a fully functional dimension.

However, you can also define dimensions with value-based hierarchies that do not have levels defined as metadata, or "flat" dimensions that do not have hierarchies or levels. These types of dimensions must be defined with natural keys, as described in the next topic.

Creating Dimensions

Dimensions are lists of unique values that identify and categorize data. They form the edges of a logical cube, and thus of the measures within the cube. Analytic Workspace Manager supports these common dimension styles:

- List or flat dimensions have no levels or hierarchies.
- Level-based dimensions use parent-child relationships to group members into levels. Most dimensions are level-based.
- Value-based dimensions have parent-child relationships among their members, but these relationships do not form meaningful levels.

Defining a Time Dimension

You can define dimensions as either User or Time dimensions. Business analysis is performed on historical data, so fully defined time periods are vital. A time dimension table must have columns for period end dates and time span. These required attributes support time-series analysis, such as comparisons with earlier time periods. If this information is not available, then you can define Time as a User dimension, but it will not support time-based analysis.

Creating Unique Dimension Members

Every dimension member must be a unique value. Depending on your data, you can create a dimension that uses either natural keys or surrogate keys from the relational sources for its members.

- **Natural keys** are read from the relational sources without modification. To use natural keys, the values must be unique across levels. Because each level may be mapped to a different relational column, this uniqueness may not be enforced in the source data.

For example, a Geography source table might have a value of `NEW_YORK` in the `CITIES` column and a value of `NEW_YORK` in the `STATES` column. Unless you take steps to assure uniqueness, the second value for `NEW_YORK` will overwrite the first.

If a dimension is flat or value-based, then it must use natural keys. You must take whatever steps you need to assure that the dimension members are unique.

- **Surrogate keys** ensure uniqueness by adding a level prefix to the members while loading them into the analytic workspace. For the previous example, surrogate keys create two dimension members named `CITIES_NEW_YORK` and `STATES_NEW_YORK`, instead of a single member named `NEW_YORK`. A dimension that has surrogate keys must be defined with at least one level-based hierarchy.

Opening the Create Dimension Dialog Box

To create a standard form dimension:

1. Expand the folder for the analytic workspace.
An analytic workspace folder contains subfolders named Dimensions, Cubes, Measure Folders, and Calculation Plans.
2. Right-click **Dimensions**, then choose **Create Dimension** from the pop-up menu.
The Create Dimension dialog box is displayed.
3. Complete all tabs.
Click **Help** for specific information about your choices.
4. Click **Create**.
The new dimension appears as a subfolder under Dimensions.

Creating Levels

For business analysis, data is typically summarized by level. For example, your database may contain daily snapshots of a transactional database. Days are thus the base level. You might summarize this data at the weekly, quarterly, and yearly levels.

Levels have parent-child or one-to-many relationships, which form a **level-based hierarchy**. For example, each week summarizes seven days, each quarter summarizes 13 weeks, and each year summarizes four quarters. This hierarchical structure enables analysts to detect trends at the higher levels, then drill down to the lower levels to identify factors that contributed to a trend.

For each level that you define, you must identify a data source for dimension members at that level. Members at all levels are stored in the same dimension. In the previous example, the Time dimension contains members for weeks, quarters, and years.

To create a level:

1. Expand the folder for the dimension.
A dimension folder contains subfolders named Levels, Hierarchies, and Attributes.
2. Right-click **Levels**, then choose **Create Level** from the pop-up menu.
The Create Level dialog box is displayed.
3. Complete all tabs of the Create Level dialog box.
Click **Help** for specific information about these choices.
4. Click **Create**.
The new level appears as an item in the Levels folder.

Creating Hierarchies

Dimensions can have one or more hierarchies. Most hierarchies are level-based. Analytic Workspace Manager supports these common types of level-based hierarchies:

- **Normal hierarchies** consist of one or more levels of aggregation. Members roll up into the next higher level in a many-to-one relationship, and these members roll up into the next higher level, and so forth to the top level.
- **Ragged hierarchies** contain at least one member with a different base, creating a "ragged" base level for the hierarchy.
- **Skip-level hierarchies** contain at least one member whose parents are more than one level above it, creating a hole in the hierarchy. An example of a skip-level hierarchy is City-State-Country, where at least one city has a country as its parent (for example, Washington D.C. in the United States).

In relational source tables, a skip-level hierarchy may contain nulls in the level columns.

You may also have dimensions with parent-child relations that do not support levels. For example, an employee dimension might have a parent-child relation that identifies each employee's supervisor. However, levels that group together first-, second-, and third-level supervisors and so forth may not be meaningful for analysis. Similarly, you might have a line-item dimension with members that cannot be grouped into meaningful levels. In this situation, you can create a **value-based hierarchy** defined by the parent-child relations instead of levels. You can create value-based hierarchies only for dimensions that use natural keys, because surrogate keys are formed with the names of the levels.

To create a hierarchy:

1. Expand the folder for the dimension.
A dimension folder contains subfolders named Levels, Hierarchies, and Attributes.
2. Right-click **Hierarchies**, then choose **Create Hierarchy** from the pop-up menu.
The Create Hierarchy dialog box is displayed.
3. Complete all tabs of the Create Hierarchy dialog box.
If you define multiple hierarchies, be sure to define one of them as the default hierarchy.
Click **Help** for specific information about these choices.
4. Click **Create**.
The new hierarchy appears as an item in the Hierarchies folder.

Creating Attributes

Attributes provide information about the individual members of a dimension. They are used for labeling crosstabular and graphical data displays, selecting data, organizing dimension members, and so forth.

Automatically Defined Attributes

Analytic Workspace Manager creates some attributes automatically when creating a dimension. These attributes have a unique type, such as "Member Long Description," which client applications expect to find. You can create additional "User" attributes that provide supplementary information about the dimension members.

All dimensions are created with long and short description attributes. If your source tables include long and short descriptions, then you can map the attributes to the appropriate columns. However, if your source tables include only one set of labels, then you should always map the long description attributes. You can decide whether or not to map the short description attributes to the same column. If you do, the data will be loaded twice.

Discoverer Plus OLAP, Spreadsheet Add-In, and OracleBI Beans use long description attributes in selection lists and for labelling crosstabs and graphs. The Add-In initially makes limited use of short description attributes, but users can switch to long descriptions. If the appropriate descriptions are not available, then these tools use dimension members. For example, if the Product dimension has short descriptions but no long descriptions, then the tools display Product dimension members.

Time dimensions are created with time-span and end-date attributes. This information must be provided for all Time dimension members.

Be sure to examine all of these attribute definitions, because you may wish to change the default settings. In particular, expand the hierarchy tree on the Basic tab to verify that the correct levels are selected. These choices affect the number of columns that you can map to the dimension.

User Attributes

To create a new user attribute:

1. Expand the folder for the dimension.
A dimension folder contains subfolders named Levels, Hierarchies, and Attributes.
2. Right-click **Attributes**, then choose **Create Attribute** from the pop-up menu.
The Create Attribute dialog box is displayed.
3. Complete all tabs of the Create Attribute dialog box.
Click **Help** for specific information about these choices.
4. Click **Create**.
The new attribute appears as an item in the Attributes folder.

Creating Logical Cubes

Cubes are the parents of measures. They are informational objects that identify measures with the exact same dimensions and thus are candidates for being processed together at all stages: data loading, aggregation, storage, and querying.

Creating Cubes

Cubes define the shape of your business measures. They are defined by a set of ordered dimensions. The dimensions form the edges of a cube, and the measures are the cells in the body of the cube.

To create a cube:

1. Expand the folder for the analytic workspace.
An analytic workspace folder contains subfolders named Dimensions, Cubes, Measure Folders, and Calculation Plans.
2. Right-click **Cubes**, then choose **Create Cube** from the pop-up menu.

The Create Cube dialog box is displayed.

3. Complete all tabs of the Create Cube dialog box.

Important: Your decisions have a major impact on the performance of the analytic workspace. Click **Help** for specific information, and refer to "[Making Data Storage Decisions](#)" on page 3-17.

4. Click **Create**. The new cube appears as a subfolder under **Cubes**.

Creating Measures

Measures store the facts collected about your business. Each measure belongs to a particular cube, and thus shares particular characteristics with other measures in the cube, such as the same dimensions.

To create a measure:

1. Expand the folder for the cube that has the dimensions of the new measure.
A cube folder contains subfolders named Measures and Calculated Measures.
2. Right-click **Measures**, then choose **Create Measure** from the pop-up menu.
The Create Measure dialog box is displayed.
3. Complete the General, Translations, and Implementation Details tabs of the Create Measure dialog box. Complete all tabs if you wish to override the cube settings.
Click **Help** for specific information about these choices.
4. Click **Create**.

The new measure appears as an item in the Measures folder.

Creating Calculated Measures

Calculated measures add valuable information to an analytic workspace. They are created by performing calculations on the measures stored in an analytic workspace. Oracle OLAP offers an extensive range of functions and operators that can be used to define custom measures. Analytic Workspace Manager provides a Calculation Wizard, as shown in [Figure 1-2](#), which provides these calculations:

- Basic Arithmetic. Addition, subtractions, multiplication, division, ratio
- Advanced Arithmetic. Cumulative total, index, percent markup, percent variance, rank, share, variance
- Prior/Future Comparison. Prior value, difference from prior period, percent difference from prior period, future value
- Time Frame. Moving average, moving maximum, moving minimum, moving total, year to date

Calculated measures are not stored, and so they do not occupy any significant disk space. The data values are calculated in response to individual queries on the calculated measures. In this respect, calculated measures are similar to relational views.

To create a calculated measure:

1. Expand the folder for the cube that contains the base measures that will be used in the calculation.

2. Right-click **Calculated Measures**, then choose **Create Calculated Measure** from the pop-up menu.

The Calculation Wizard Welcome page is displayed.

3. Follow the steps of the wizard.

Click **Help** for specific information about these choices. When you are done, the name of the new calculated measure appears as an item in the Calculated Measures folder.

Making Data Storage Decisions

The creation of a cube requires several decisions about data storage that affect the performance of the analytic workspace. These choices are on the Implementation Details tab for the cube.

The DBMS_AW PL/SQL package contains a Sparsity Advisor and an Aggregate Advisor. These advisors analyze the tables in a relational schema and provide recommendations for data storage in an analytic workspace. These recommendations may help you to make the best choices for storing your data. Remember to always evaluate the advice generated by these tools against your own knowledge of the data.

See Also: *Oracle OLAP Reference* for information about the Sparsity Advisor and the Aggregate Advisor

What is Sparsity?

Sparsity refers to the extent to which cells contain null (NA) values instead of data. For example, if a cube is 25 percent sparse, then 25 percent of that cube's cells contain NA values and 75 percent contain data. You can also describe this cube as 75% dense.

In general, if a cube's detail-level data is more than 80 percent sparse, then you must manage sparsity by identifying the sparse dimensions in order to promote good performance.

Sparsity Patterns

There are two types of sparsity patterns:

- Controlled sparsity means that a range of values of one or more dimensions has no data. This is often a result of the way you design your analytic workspace. For example, a Time dimension might contain future time periods that will be populated by a forecast after the data is loaded into the analytic workspace. This type of sparsity is temporary and should be disregarded when evaluating the sparsity of the cube. Other types of controlled sparsity may remain sparse and should be considered as a factor in evaluating sparsity.
- Random sparsity means that NA values are scattered throughout a measure, usually because some combinations of dimension values never have any data. This is often a result of the nature of your business. Random sparsity tends to be more common than controlled sparsity.

Your data may demonstrate one or both types of sparsity.

Measures have the same sparsity pattern when all of the following are true:

- They have exactly the same dimensions
- They have many of the same empty cells
- They have roughly the same number of empty cells

Measures that share these characteristics should be created in the same cube. However, if their sparsity patterns are very different, then they should be created in different cubes even if they share the same dimensions. Typically, measures that are mapped to the same source tables have the same sparsity pattern.

Physical Storage of Sparse Data

A cube can be dense, sparse, or extremely sparse.

- Dense cubes have up to 20% empty cells. For a dense cube, do not identify any dimensions as sparse.
- For a sparse cube, identify the sparse dimensions.
- For an extremely sparse cube, identify all of the dimensions as sparse and use compressed storage.

Compressed storage is for measures that are extremely sparse. Extreme sparsity often results from these factors:

- A cube has a large number of dimensions (seven or more).
- One dimension has more than 300,000 members.
- Two dimensions have more than 100,000 members each.
- Dimension hierarchies have numerous levels, with little change to the number of dimension members from one level to the next, so that many parents have only one descendant for several contiguous levels.

Compressed storage for this type of sparsity uses less space and results in faster aggregation than normal sparse storage.

Manually Calculating Sparsity in a Cube

Sparsity is calculated as the relationship between the number of actual data values in the measures and the number of cells defined by the dimensions of the cube.

To calculate the percent sparsity in a cube, use this equation:

$$(data\ values\ in\ measure) / (cells\ in\ cube) * 100$$

To obtain the number of data values in a measure, count the number of rows in the data source. The following SQL statement returns the number of values in the UNITS measure in the GLOBAL schema:

```
SELECT COUNT(*) FROM units_history_fact WHERE units IS NOT NULL;
```

To calculate the number of cells in a cube, first count the number of base-level values for each dimension. The following SQL statement returns the number of base-level Customers in the GLOBAL schema:

```
SELECT COUNT(ship_to_id) FROM customer_dim;
```

Then multiply the number of values for each dimension in the cube:

$$time\ periods * customers * products * channels$$

Note: You must know the number of dimension values so that you can order the dimensions correctly.

All of the dimensions may be sparse, or all of them may be dense, or the cube may have a mixture of sparse and dense dimensions. You can determine these

characteristics by comparing the total number of dimension values with the number of dimension values actually used in a cube. For even more insight, you can compare these numbers at all levels.

For example, the following SQL statement returns the total number of dimension values at each level in the Global TIME_DIM dimension table:

```
SELECT COUNT(month_id), COUNT(DISTINCT quarter_id), COUNT(DISTINCT year_id)
FROM time_dim;
```

The next SQL statement returns the number of dimension values at each level of Time that are used in the Global UNITS_HISTORY_FACT fact table.

```
SELECT COUNT(DISTINCT a.month_id), COUNT(DISTINCT a.quarter_id),
COUNT(DISTINCT a.year_id)
FROM time_dim a, units_history_fact b
WHERE a.month_id=b.month_id;
```

Ordering the Dimensions in a Cube

The order in which the dimensions are listed for a cube affects performance because it determines the way the data is stored on disk. The first dimension in a cube is the fastest-varying dimension, and the last dimension is the slowest-varying dimension. The data for each measure in a cube is stored as a linear stream, in which the values of the fastest-varying dimension are clustered together and values of the slowest-varying dimension are spread far apart. Performance is optimized when values that are accessed together are stored together, because fewer pages must be swapped in and out of memory.

Data storage may be optimized for querying or loading. To optimize for loading, list the dense dimension (such as Time) before the sparse dimensions. If there is more than one dense dimension, then list the largest one first. To optimize for querying, you need to understand how your users are querying the data.

Partitioning Large Measures

Partitioning is an method of physically storing the measures in a cube. It improves the performance of large measures in the following ways:

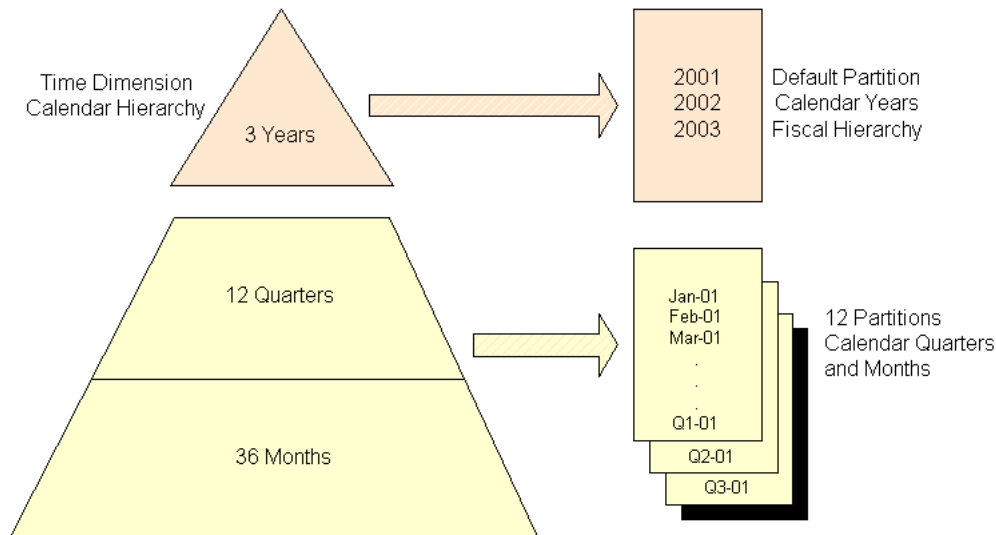
- Improves scalability by keeping data structures small. Each partition functions like a smaller measure.
- Keeps the working set of data smaller both for queries and maintenance, since the relevant data is stored together.
- Enables parallel aggregation during data maintenance. Each partition can be aggregated by a separate process.
- Allows different client sessions to have write access to different partitions of the same object at the same time.
- Simplifies removal of old data from storage. Old partitions can be dropped as a unit, and new partitions can be added.
- Stores each partition of a compressed cube in a separate analytic workspace object. If a compressed cube is not partitioned, then all measures of the cube are stored in one object.

For partitioning a cube in Analytic Workspace Manager, you must choose a dimension and one of its levels as the basis for creating the partitions. For example, you might choose the Quarter level of the Time dimension. Each Quarter and its descendants are

stored in a separate partition. If there are three years of data in the analytic workspace, then partitioning on Quarter produces 12 partitions, in addition to the default partition. The default partition contains all remaining levels, that is, those above Quarter (such as Year) and those in other hierarchies (such as Fiscal Year or Year-to-Date). The aggregate levels in the new partitions are calculated and stored in the analytic workspace as a data maintenance step, while the levels in the default partition are calculated on the fly.

Figure 3–7 illustrates the Global Time dimension partitioned by Quarter.

Figure 3–7 Partitioning Time by Quarter



The number of partitions also affects the database resources that can be allocated to loading and aggregating the data in an analytic workspace. Partitions can be aggregated simultaneously when sufficient resources have been allocated, as described in ["Maintaining the Data"](#) on page 3-25.

As this discussion explains, partitioning affects the extent to which an analytic workspace is optimized for querying or for maintenance. The fewer partitions, the more levels are precalculated (optimized for querying); the more partitions, the more levels are calculated on the fly, and the more resources can be allocated to loading and aggregating the data (optimized for maintenance).

Time is typically a good candidate for partitioning, because this choice supports life-cycle maintenance. Old time periods can be dropped as a unit in a partition, and new time periods can be added in a new partition. Moreover, the partitions will be approximately the same size because of the inherent regularity of the calendar. For example, in a calendar hierarchy, months have 28-31 children, quarters have 3 children, and years have 4 children.

However, some enterprises prefer to redeploy their analytic workspaces with new data instead of maintaining them. This is also the model used by Oracle Warehouse Builder. When life-cycle maintenance is not a factor, you should choose the most dense dimension for partitioning. The most dense dimension is frequently the one with the fewest members.

Defining Rules for Summarizing Data

Analytic Workspace Manager enables you to define summarization rules at three different levels. You can use whatever combination of levels best suits your needs:

- **Cube.** You can define default summarization rules for all measures in a cube. You define these rules when creating or modifying a cube.
- **Measure.** You can define unique summarization rules for a particular measure. These rules override the default cube summarization rules. You define these rules when creating or modifying a measure.
- **Analytic Workspace.** You can define rules for one or more measures and determine the order in which the measures are summarized. In this way, you can support dependencies among the measures. You define these rules when creating or modifying a calculation plan.

Regardless of the level at which you define the summarization rules, the decisions and the user interface are the same.

Basic Strategy for Summarizing Analytic Workspace Data

A data load typically fetches data only at the lowest, or base, level. The data cells at the higher levels are empty until the values are calculated from the base values. In analytic workspaces, aggregate data can be generated at two distinct times:

- **On the fly** in response to a query. Calculated values may be cached for use throughout the session, but they are not shared among sessions.
- **As part of the build procedure.** Calculated values are stored in the analytic workspace and shared by all sessions.

If your dimensions have multiple hierarchies or if the hierarchies have many levels, then fully aggregating the measures can increase the size of your analytic workspace (and thus your database) geometrically. At the same time, much of the intermediate level data may be accessed infrequently or not at all.

The most effective method of summarizing data in an analytic workspace is by storing some aggregates and calculating others on the fly. A typical strategy for doing this is called skip-level aggregation, because some levels are stored and others are skipped until runtime.

Selecting Levels to Aggregate in the Builds

Skip-level aggregation is a strategy for identifying levels for data storage by determining the ratio of dimension members at each level, and keeping the ratio of members to be rolled up on the fly at approximately 10:1. This ratio assures that all answer sets can be returned quickly. Either a data value is stored in the analytic workspace so it can simply be retrieved, or it can be calculated quickly from 10 stored values.

This 10:1 rule is best applied with some judgment. You might want to permit a higher ratio for levels that you know are seldom accessed. Or you might want to store levels at a lower ratio if you know they have heavy use.

Slower varying dimensions take longer to aggregate because the data is scattered throughout its storage space. If you are optimizing for data maintenance, then fully aggregate the faster varying dimensions and use skip-level aggregation on the slower varying dimensions.

Aggregation rules identify how and when the aggregate values are calculated. You define the aggregation rules for each cube, and you can override these rules by defining new ones for a particular measure.

Choosing Aggregation Methods

Analytic workspaces offer a large selection of aggregation methods, including scaled, weighted, hierarchical, and hierarchical weighted methods. Descriptions of these methods are provided in Analytic Workspace Manager Help.

You can specify different operators for different dimensions. If you do, then order the dimensions in the aggregation rules to achieve the results you want; for example, the sum of averages may yield different values than the average of sums.

Mapping Logical Objects to Data Sources

After creating logical objects, you can map them to data sources in Oracle Database. Afterward, you can load data into your analytic workspace using the Maintenance Wizard.

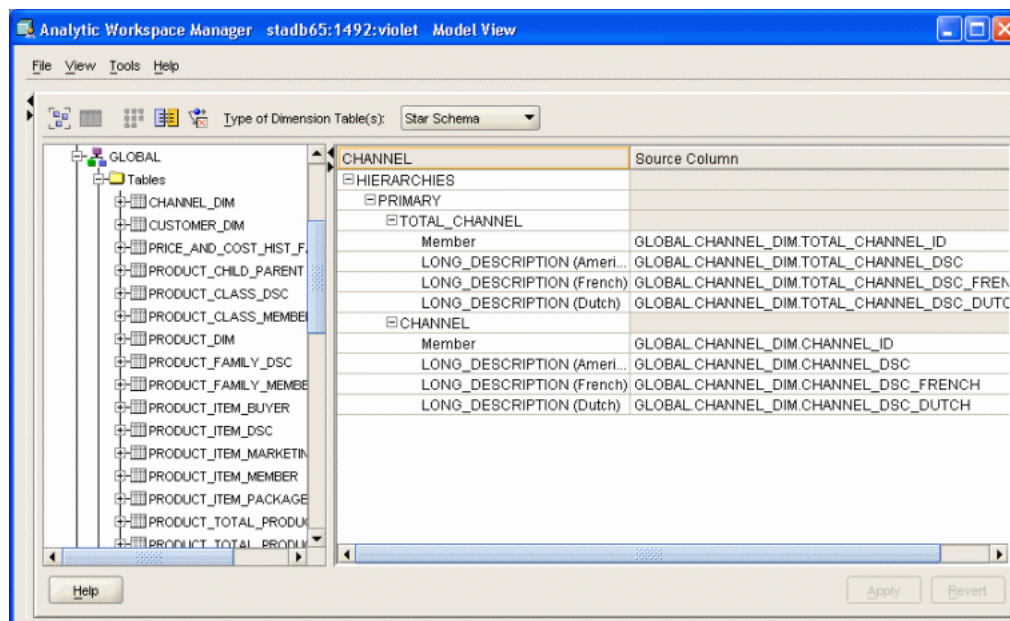
The mapping window has a tabular view and a graphical view.

- Tabular view. Drag-and-drop the names of individual columns from the schema navigation tree to the rows for the logical objects.
- Graphical view. Drag-and-drop icons, which represent tables and views, from the schema navigation tree onto the mapping canvas. Then you draw lines from the columns to the logical objects.

If you want to see the values in a particular source table or view, right-click it in either the schema tree or the mapping canvas. Choose **View Data** from the menu to fetch up to 1000 rows.

Figure 3–8 shows the CHANNEL dimension mapped in the tabular view. The toolbar appears across the top and the schema navigation tree is on the left.

Figure 3–8 Dimension Mapped in Tabular View



The following procedure explains how to map a dimension in the graphical view.

Mapping Dimensions

To map a dimension in the graphical view, take these steps:

1. Define the dimension and its levels, hierarchies, and attributes.
2. In the Model View navigation tree, expand the dimension folder and click **Mappings**.

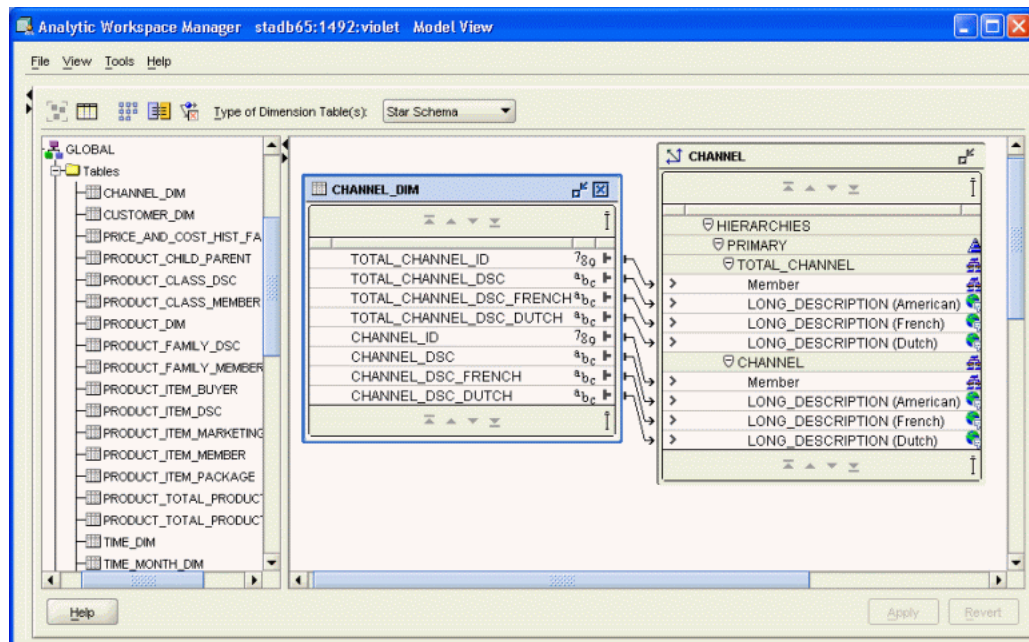
The Mapping Window will be displayed in the right pane.

3. Enlarge the mapping window by dragging the divider to the left.
4. In the toolbar, identify the source schema as [Star Schema](#), [Snowflake Schema](#), or [Other](#).
5. In the schema navigation tree, locate the tables with the dimension members and attributes for all levels. Drag-and-drop them onto the mapping canvas.
6. Draw lines from the source columns to the target objects. To draw a line, click the output connector of the source column and drag it to the input connector of the target object. Be careful to map every logical object to a source column.

Tip: For a star schema with logical names that match the column names, click Auto Map Star Schema in the toolbar. Verify that all logical objects are mapped correctly.

7. To uncross the lines, click the Auto Arrange Mappings tool.
8. Click **Apply**.
9. When you have mapped all objects for the dimension, drag the divider to the right to restore access to the navigation tree.

[Figure 3–9](#) shows the mapping canvas with the Channel dimension and its attributes mapped to columns in the CHANNEL_DIM table. The mapping toolbar is at the top, and the schema navigation tree is on the left.

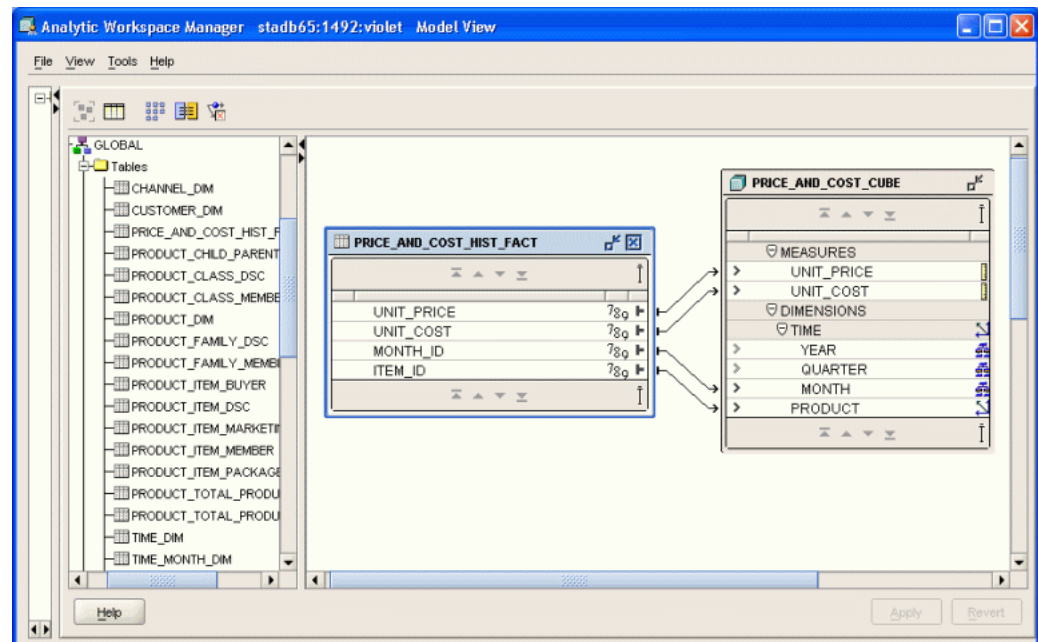
Figure 3–9 GLOBAL CHANNEL Dimension Mapped in Graphical View

Mapping Cubes

To map a cube in the graphical view, take these steps:

1. Define the cube and its measures.
You can define derived measures at any time, because they are calculated, not loaded.
2. In the Model View navigation tree, expand the **Cubes** folder and click **Mappings**.
The Mapping Window will be displayed in the right pane. You will see a schema navigation tree and a table with rows for the measures, dimensions, and levels.
3. Enlarge the mapping window by dragging the divider to the left.
4. In the schema navigation tree, locate the tables with the measures. Drag-and-drop them onto the mapping canvas.
5. Draw lines from the source columns to the target objects.
To draw a line, click the output connector of the source column and drag it to the input connector of the target object. You must map both the measures and the related dimension keys.
6. To uncross the lines, click the Auto Arrange Mappings tool.
7. When you have mapped all objects for the dimension, drag the divider to the right to restore access to the navigation tree.

Figure 3–10 shows the mapping canvas with the Price and Cost cube mapped to columns in the PRICE_AND_COST_HIST_FACT table. The mapping toolbar is at the top, and the schema navigation tree is on the left.

Figure 3–10 GLOBAL PRICE_AND_COST_CUBE Cube Mapped in Graphical View

Maintaining the Data

The Maintenance Wizard loads and aggregates the data as a single job. You can load all mapped objects in the analytic workspace, or individual dimensions and measures. You can also choose to run the job immediately, enter it in the Oracle job queue, or save it as a SQL script.

To maintain the data:

1. Right-click the name of the analytic workspace, a measure, or a dimension, then choose **Maintenance Wizard** from the pop-up menu.

Choose a folder that includes all the items that you want to maintain. For example, if you open the Maintenance Wizard from a particular cube, you will load that cube and summarize its measures. You will not load data or summarize other cubes.

2. Follow the steps of the wizard.

Click **Help** for additional information about each step.

3. Verify the results in the Data Viewer. Right-click a cube, and choose **View Data** from the pop-up menu.

Submitting Maintenance Tasks to the Oracle Job Queue

If you submit a maintenance task to the Oracle job queue, you can specify the maximum number of simultaneous processes the job can use. This number is limited by two factors:

- The number of objects in the analytic workspace that can be summarized in parallel. Each cube and each partition (including the default partition) can use a separate process.
- The number of simultaneous database processes the user is authorized to run.

This number is controlled by the `JOB_QUEUE_PROCESSES` parameter. The setting for this parameter is based on the number of processors, as described in ["Initialization Parameters for Oracle OLAP"](#) on page 6-6. You can obtain the current parameter setting with the following SQL command:

```
SHOW PARAMETER JOB_QUEUE_PROCESSES
```

Specify the smaller of these two numbers when submitting a job.

Oracle Database allocates the specified number of processes (if you have sufficient authorization) regardless of whether all of them can be used simultaneously at any point in the job. For example, if your job can use up to three processes, but you specify five, then two of the processes allocated to your job cannot be used by it or any other job.

Managing Maintenance Jobs

When submitting a maintenance task to the job queue, be sure to note the job number so that you can verify that the job completed successfully. Runtime messages are stored in a table named `OLAPSYS.XML_LOAD_LOG`. Messages in this file are identified just by the digits in the job number. The following SQL statement returns the messages for job `AWXML$_54`:

```
SELECT XML_MESSAGE FROM OLAPSYS.XML_LOAD_LOG WHERE XML_LOADID='54';
```

You can manage these jobs using tools such as Oracle Enterprise Manager Scheduler or the `DBMS_SCHEDULER` PL/SQL package.

Defining Measure Folders

You can define a measure folder for use by OLAP tools, so that the measures can be located and identified quickly by users. They may have access to several analytic workspaces or relational schemas with measures named Sales or Costs, and they will have no means of differentiating them outside of a measure folder.

To create a measure folder:

1. Expand the folder for the analytic workspace.
2. Right-click Measure Folders, then choose **Create Measure Folders** from the pop-up menu.
3. Complete the General tab of the Create Measure Folder dialog box.

Click **Help** for specific information about these choices.

Supporting Multiple Languages

A single analytic workspace can support multiple languages. This support enables users of OLAP applications and tools to view the metadata in their native languages. For example, you can provide translations for the display names of measures, cubes, and dimensions. You can also map attributes to multiple columns, one for each language.

The number and choice of languages is restricted only by the database character set and your ability to provide translated text. Languages can be added or removed at any time.

To add support for multiple languages:

1. In the Model View navigation tree, expand the folder for the analytic workspace.
2. Click the Languages folder, and select the languages for the analytic workspace on the Basic tab.
3. For each dimension, level, hierarchy, attribute, cube, measure, calculated measure, and measure folder, open the Translations tab of the property sheet. Enter the object labels and descriptions in each language.
4. For each dimension, open the Mappings window. Map the attributes to columns for each language.

Creating Calculation Plans

Calculation plans enable you to create the aggregation rules for one or more measures, as an alternative to the default aggregation plan for the cube or defining individual plans for each measure. You can also identify the order in which you want the measures aggregated when there are interdependencies.

To create a calculation plan:

1. Expand the folder for the analytic workspace.
2. Right-click **Calculation Plans**, then choose **Create Calculation Plan** from the pop-up menu.

The Create Calculation Plan dialog box is displayed.

3. Complete the General tab.

Click **Help** for specific information about these choices.

4. To create a new aggregation step, click **Add**.

The Create New Aggregation Step dialog box is displayed.

5. Complete all tabs, then click **Create**.

The new aggregation step is listed on the Calculation Plan General tab.

6. Click **Create**.

The new calculation plan appears as an item in the Calculation Plans folder.

Case Study: Creating the Global Analytic Workspace

The following case study explains the choices made in creating an analytic workspace from the GLOBAL schema. [Chapter 2](#) describes the tables.

Defining the GLOBAL_AW User

This example follows best practices by creating the GLOBAL analytic workspace in a different schema from the source tables. [Example 3-1](#) lists the SQL commands to define the GLOBAL_AW user with sufficient access rights to use Analytic Workspace Manager and to access the GLOBAL star schema. Alternatively, you can define users through Oracle Enterprise Manager.

Example 3–1 SQL Script for Defining the GLOBAL_AW User

```
CREATE USER "GLOBAL_AW" PROFILE "DEFAULT"
  IDENTIFIED BY "global_aw" DEFAULT TABLESPACE "GLOBAL"
  TEMPORARY TABLESPACE "OLAPTEMP"
  QUOTA UNLIMITED ON "GLOBAL"
  ACCOUNT UNLOCK;

GRANT OLAP_USER TO GLOBAL_AW;

GRANT SELECT ON global.channel_dim TO global_aw;
GRANT SELECT ON global.product_child_parent TO global_aw;
GRANT SELECT ON global.customer_dim TO global_aw;
GRANT SELECT ON global.time_month_dim TO global_aw;
GRANT SELECT ON global.time_quarter_dim TO global_aw;
GRANT SELECT ON global.time_year_dim TO global_aw;
GRANT SELECT ON global.units_history_fact TO global_aw;
GRANT SELECT ON global.price_and_cost_history_fact TO global_aw;
```

Examining Sparsity Characteristics for GLOBAL

By using SQL SELECT commands with the COUNT and COUNT (DISTINCT) functions, you can estimate how dense the resulting dimensional cubes will be in the analytic workspace.

The Time dimension has 96 members. However, the last 17 months have no data. These time periods and their aggregates will be used initially for forecasting and later to store actual data. These additional time periods are excluded from the following calculations because they might skew the results in such a small data set.

The PRICE_AND_COST_HISTORY_FACT table has 2023 rows out of a possible 2844 dimension value combinations (79 historic months * 36 products). The Price cube is mapped to the PRICE_AND_COST_HISTORY_FACT table and is over 70% dense.

The UNITS_HISTORY_FACT table has 222,589 rows out of a possible 520,452 dimension value combinations (79 historic months * 36 products * 61 customers * 3 channels). The Units cube, which is mapped to UNITS_HISTORY_FACT, is over 40% dense.

Because the Global data set is dense, even a regular composite should not be used.

Identifying Levels for Precalculation

To identify the levels to be precalculated, you must know the number of dimension members at each level. You can easily acquire this information using either SQL statements or OLAP DML commands.

For example, this SQL statement:

```
SELECT COUNT(DISTINCT year_id) FROM global.time_year_dim;
```

and this OLAP DML command in the GLOBAL analytic workspace (after loading the dimension):

```
SHOW NUMLINES(LIMIT(time TO time_levelrel EQ 'YEAR'))
```

both return the number of TIME dimension members at the Year level.

Global is a very small data set, so few adjacent levels have a 10:1 ratio of dimension members. [Table 3–1](#) identifies the levels to be calculated and stored in the analytic workspace.

Table 3–1 Precalculated Levels in the Global Workspace

Dimension	Level	Members	Precalculate
TIME	Month	96	Yes
TIME	Quarter	32	No
TIME	Year	8	Yes
CUSTOMER	Ship_To	61	Yes
CUSTOMER	Account	24	No
CUSTOMER	Market_Segment	5	Yes
CUSTOMER	Total_Market	1	No
CUSTOMER	Warehouse	11	No
CUSTOMER	Region	3	Yes
CUSTOMER	Total_Customer	1	No
PRODUCT	Product (no levels)	48	No
CHANNEL	Channel	3	Yes
CHANNEL	Total_Channel	1	No

Creating the GLOBAL Analytic Workspace

Take these steps to create the GLOBAL analytic workspace:

1. Open Analytic Workspace Manager and connect to Oracle Database as the GLOBAL_AW user.
2. In the Model View navigation tree, expand the GLOBAL_AW folder, and right-click **Analytic Workspaces**.
3. Choose **Create Analytic Workspace** from the pop-up menu.
4. Complete the Create Analytic Workspace dialog box, then choose **Create**.

This step creates the analytic workspace container and populates it with standard form catalogs and similar objects. You must now define the logical model.

Creating GLOBAL Dimensions and Attributes

GLOBAL has four dimensions: TIME, PRODUCT, CUSTOMER, and CHANNEL. Implement the logical model described in [Chapter 2](#) by following the basic instructions in "[Creating Logical Dimensions](#)" on page 3-12.

See Also: [Chapter 2](#) for information about installing the Global schema

Note these choices:

- **Time Dimension:** On the General tab, select **Time Dimension** as the dimension type. You can map Time to a star schema (TIME_DIM table) or to a snowflake schema (TIME_MONTH_DIM, TIME_QUARTER_DIM, and TIME_YEAR_DIM tables).
- **Product Dimension:** You can map Product to a star, level-based table (PRODUCT_DIM) or to a parent-child table (PRODUCT_CHILD_PARENT).
- **All Dimensions:** On the Implementation Details tab, select **Use Natural Keys From Data Source**.

The source tables have numeric surrogate keys that assure unique dimension members across all levels.

- **All Attributes:** On the General tab, verify that the attributes apply to all levels.
- **Languages:** Add French and Dutch.

Creating GLOBAL Cubes and Measures

GLOBAL has two cubes: UNITS_CUBE and PRICE_AND_COST_CUBE.

- UNITS_CUBE is dimensioned by TIME, PRODUCT, CUSTOMER, and CHANNEL. It contains two measures, UNITS and SALES.
- PRICE_AND_COST_CUBE is dimensioned by TIME and PRODUCT. It contains two measures, UNIT_PRICE and UNIT_COST.

Implement the logical model described in [Chapter 2](#) by following the basic instructions in "[Creating Logical Cubes](#)" on page 3-15.

UNITS_CUBE

On the Implementation Details page, list the dimensions in this order:

1. TIME
2. CUSTOMER
3. PRODUCT
4. CHANNEL

Deselect the sparsity check boxes for all dimensions. They are dense.

On the Aggregation page, select the SUM operator for all dimensions. Use [Table 3-1](#) to select levels for presummarization.

PRICE_AND_COST_CUBE

On the Implementation Details page, list the dimensions in this order:

1. TIME
2. PRODUCT

Measures in the Price Cube and the Units Cube will be used together frequently in calculated measures. For performance, the dimensions that the cubes share must be listed in the same order.

Deselect the sparsity check boxes for all dimensions. They are dense.

On the Aggregation page, select **Last Non-NA Data Value** for Time and **Average** for Product.

Mapping the GLOBAL Logical Model to Data Sources

The data for the GLOBAL analytic workspace is stored in the GLOBAL schema.

To map the PRODUCT dimension, take these steps:

1. Expand the Dimensions folder, then click the Mappings node for PRODUCT.
2. Drag the divider to the left to expand the size of the mapping canvas.
3. In the schema navigation tree, expand the GLOBAL folder, then drag-and-drop the PRODUCT_CHILD_PARENT table onto the canvas.

4. Drag a line from the output connectors in the `PRODUCT_CHILD_PARENT` table to the appropriate input connector in the `PRODUCT` table.
5. Click **Apply**.

Repeat these steps to map `CUSTOMER` to the `CUSTOMER_DIM` table and `CHANNEL` to the `CHANNEL_DIM` table. For `TIME`, select **Snowflake Schema** and map to `TIME_MONTH_DIM`, `TIME_QUARTER_DIM`, and `TIME_YEAR_DIM`.

To map `UNITS_CUBE`, take these steps:

1. Expand the Cubes folder, then click the Mappings node for `UNITS_CUBE`.
2. Drag the divider to the left to expand the size of the mapping canvas.
3. In the schema navigation tree, expand the `GLOBAL` folder, then drag-and-drop the `UNITS_DETAIL_FACT` table onto the canvas.
4. Drag lines from the output connectors in the `UNITS_DETAIL_FACT` table to the appropriate input connectors in the `UNITS_CUBE` table.
5. Click **Apply**.

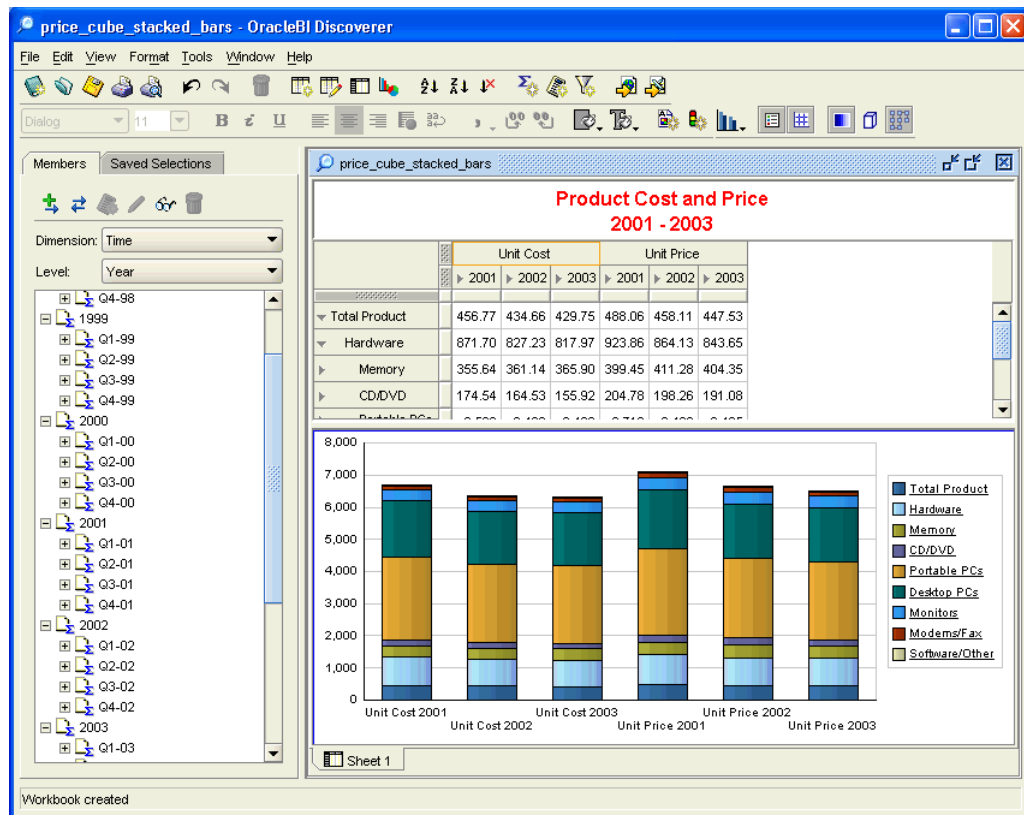
Repeat these steps to map `PRICE_AND_COST_CUBE` to the `PRICE_AND_COST_HIST_FACT` table.

Loading and Aggregating the Data

To load all of the data for `GLOBAL`, run the Maintenance Wizard as described in ["Maintaining the Data"](#) on page 3-25. Note these choices:

- Run the Maintenance Wizard from the `GLOBAL` folder in the Model navigation tree.
- Select Objects page: Select the **Add the Dimensions of the Cube** box, then move **Cubes** to the Selected Source Objects column. Click **Finish** to run the job immediately.

[Figure 3-11](#) shows the results of a query in OracleBI Discoverer Plus OLAP.

Figure 3–11 Discoverer Plus OLAP Displays Data from PRICE_AND_COST_CUBE

Creating Calculated Measures

"Identifying Required Business Facts" on page 2-5 identifies the business measures required by the Global Corporation. Only three measures were acquired from the source fact tables: Units, Unit Price, and Unit Cost. The remaining business measures can be calculated from those three. Table 3–2 shows the calculated measures for the Units Cube.

Table 3–2 Custom Measures for the GLOBAL Analytic Workspace

Required Business Measures	Calculation Type	Based On Measures
Sales	Basic Arithmetic > Multiplication	UNITS*UNIT_PRICE
Extended Cost	Basic Arithmetic > Multiplication	UNITS*UNIT_COST
Extended Margin	Basic Arithmetic > Subtraction	SALES-EXTENDED_COST
Change in sales from prior period (month, quarter, or year)	Prior/Future Comparison > Difference from Prior Period	SALES
Change in sales from prior year		
Percent change in sales from prior period	Prior/Future Comparison > Percent Difference from Prior Period	SALES
Percent change in sales from prior year		
Product share	Advanced Arithmetic > Share	SALES (PRODUCT)
Channel share	Advanced Arithmetic > Share	SALES (CHANNEL)
Market share	Advanced Arithmetic > Share	SALES (CUSTOMER)

Table 3–2 (Cont.) Custom Measures for the GLOBAL Analytic Workspace

Required Business Measures	Calculation Type	Based On Measures
Extended margin change from prior period	Prior/Future Comparison > Difference from Prior Period	EXTENDED_MARGIN
Extended margin change from prior year		
Extended margin percent change from prior period	Prior/Future Comparison > Percent Difference from Prior Period	EXTENDED_MARGIN
Extended margin percent change from prior year		
Units sold, change from prior period	Prior/Future Comparison > Difference from Prior Period	UNITS
Extended margin per unit	Basic Arithmetic > Division	EXTENDED_MARGIN/ UNITS

Creating a Measure Folder

Define a measure folder with a name such as Global Enterprises, and add all measures and calculated measures to the folder.

Case Study: Creating the Sales History Analytic Workspace

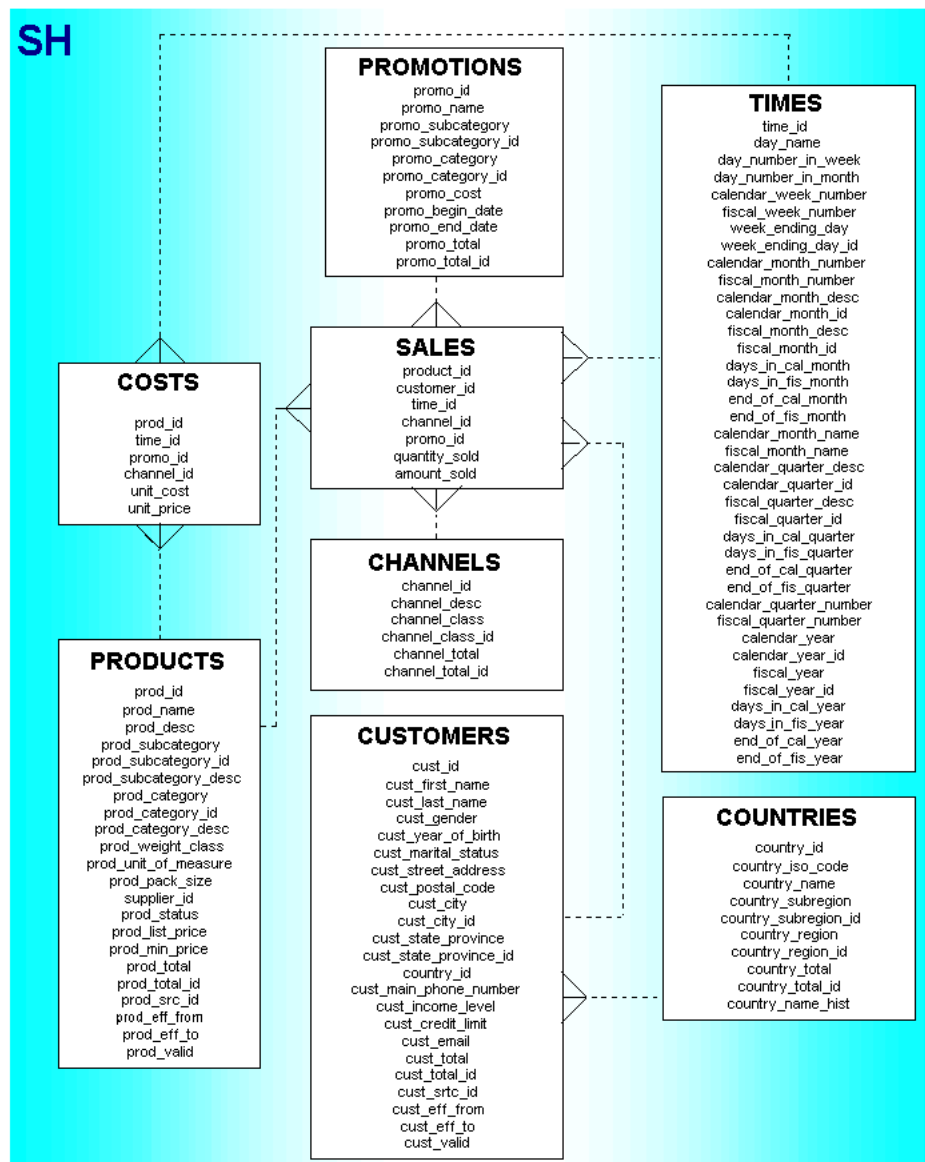
Sales History (SH) is a sample star schema that is delivered with Oracle Database, along with OLAP Catalog metadata for access directly to the relational tables by OLAP query tools. Although Global is used for most of the examples in this manual, Sales History has a very different set of data characteristics and demonstrates a correspondingly different set of build choices.

You can download a template for a Sales History analytic workspace from

<http://www.oracle.com/technology/products/OracleBI/olap/olap.html>

Figure 3–12 shows a schema diagram of Sales History.

Figure 3–12 Sales History Schema Diagram



See Also: *Oracle Database Sample Schemas* for a full description of Sales History

Creating the SH Analytic Workspace

Take these steps to create the SH analytic workspace:

1. Define database parameters for OLAP.
2. Create permanent and temporary tablespaces specifically for use by the SH analytic workspace.
3. Define the SH_AW user.
4. Open Analytic Workspace Manager and connect to Oracle Database as the SH_AW user.
5. Create the SH analytic workspace, and define the logical dimensions.

6. Examine the sparsity characteristics of the data and define the logical cube.
7. Map, load, and summarize the data.
8. Query the analytic workspace.

Defining Database Parameters

When building a large analytic workspace, the parameters for Oracle Database may affect how quickly the build proceeds. Before changing any database parameters, you should monitor performance using the default settings.

[Example 3-2](#) shows a few of the settings in the `init.ora` file for a computer with 32G of physical memory and four processors. Note that you must define an undo tablespace before you can specify it in a startup parameter. For more information about these settings, refer to [Chapter 6](#).

Example 3-2 Startup Parameters for Building Sales History

```
UNDO_MANAGEMENT=AUTO
UNDO_TABLESPACE=OLAPUNDO
SGA_TARGET=16G
PGA_AGGREGATE_TARGET=8G
JOB_QUEUE_PROCESSES=5
```

Defining Tablespaces for Sales History

While the GLOBAL analytic workspace has about a half million cells for base-level data in its largest cube, the Sales History SALES cube has over 18 trillion. This makes the Sales History analytic workspace small to average for a real application, although quite large for a sample data set. It is sufficiently large for a build to fail on a small desktop computer unless resources have been allocated for its use.

You should define temporary and permanent tablespaces for use by Sales History.

- Define a tablespace that is large enough to hold the base-level data, stored aggregates, forecast data, and so forth. If multiple physical disks are available, define an extension file for each one. For the best performance, do not use the same tablespace that the star schema uses.
- Define a temporary tablespace that is large enough to hold the data for the SALES cube. Stripe this tablespace across multiple disks the same as for the permanent tablespace. Use a small `EXTENT MANAGEMENT SIZE` value, such as 256K.

[Example 3-3](#) shows how the tablespaces might be defined for Sales History when four disk drives are available.

Example 3-3 SQL Script for Defining Tablespaces for the Sales History Analytic Workspace

```
/* Create permanent tablespaces on four disks */
CREATE TABLESPACE sh_aw DATAFILE '/disk1/oradata/sh_aw1.dbf' SIZE 64M
AUTOEXTEND ON NEXT 64M MAXSIZE 1024M
EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO;

ALTER TABLESPACE sh_aw ADD DATAFILE
'/disk2/oradata/sh_aw2.dbf' SIZE 64M REUSE AUTOEXTEND ON NEXT 64M MAXSIZE 1024M,
'/disk3/oradata/sh_aw3.dbf' SIZE 64M REUSE AUTOEXTEND ON NEXT 64M MAXSIZE 1024M,
'/disk4/oradata/sh_aw4.dbf' SIZE 64M REUSE AUTOEXTEND ON NEXT 64M MAXSIZE UNLIMITED;

/* Create temporary tablespaces on four disks */
CREATE TEMPORARY TABLESPACE sh_temp TEMPFILE '/disk1/oradata/sh_aw1.tmp' SIZE 64M REUSE
```

```
AUTOEXTEND ON NEXT 64M MAXSIZE 1024M
EXTENT MANAGEMENT LOCAL UNIFORM SIZE 256K;

ALTER TABLESPACE sh_temp ADD TEMPFILE
'/disk2/oradata/sh_aw2.tmp' SIZE 64M REUSE AUTOEXTEND ON NEXT 64M MAXSIZE 1024M,
'/disk3/oradata/sh_aw3.tmp' SIZE 64M REUSE AUTOEXTEND ON NEXT 64M MAXSIZE 1024M,
'/disk4/oradata/sh_aw4.tmp' SIZE 64M REUSE AUTOEXTEND ON NEXT 64M MAXSIZE UNLIMITED;
```

Defining the SH_AW User

[Example 3–4](#) shows a script that is similar to the one used to create the GLOBAL_AW user in [Example 3–1](#). It defines a user named SH_AW and authorizes it to access the SH star schema. The script sets the new permanent and temporary tablespaces as the defaults for the SH_AW user.

Example 3–4 Script for Creating the SH_AW User

```
/* Create the user and grant privileges */
CREATE USER sh_aw PROFILE "DEFAULT"
  IDENTIFIED BY "sh_aw"
  DEFAULT TABLESPACE sh_perm
  TEMPORARY TABLESPACE sh_temp
  QUOTA UNLIMITED ON sh_perm
  ACCOUNT UNLOCK;
GRANT OLAP_USER TO sh_aw;

/* Create a directory object*/
CREATE OR REPLACE DIRECTORY sh_scripts AS '/users/oracle/sh_scripts';
GRANT ALL ON DIRECTORY sh_scripts TO PUBLIC;

/* Grant access to SH star schema */
GRANT SELECT ON SH.CHANNELS to SH_AW;
GRANT SELECT ON SH.PRODUCTS to SH_AW;
GRANT SELECT ON SH.TIMES to SH_AW;
GRANT SELECT ON SH.CUSTOMERS to SH_AW;
GRANT SELECT ON SH.COUNTRIES to SH_AW;
GRANT SELECT ON SH.PROMOTIONS to SH_AW;
GRANT SELECT ON SH.SALES to SH_AW;
```

Defining the Logical Dimensions for Sales History

Because Sales History is a star schema, the logical model for the analytic workspace is primarily indicated by the schema design, as shown in [Figure 3–12](#).

The two fact tables, SALES and COSTS, are the data sources for two logical cubes. This case study only uses SALES.

The SALES table has a primary key composed of foreign keys from five dimension tables, which are named TIMES, PRODUCTS, CHANNELS, PROMOTIONS, and CUSTOMERS. CUSTOMERS is related to a sixth dimension table, COUNTRIES, by a foreign key. In addition, SALES has two columns that contain business measures named QUANTITY_SOLD and AMOUNT_SOLD. Thus, the star schema defines a logical SALES cube with five dimensions and two measures for the analytic workspace.

Defining TIMES_DIM

The Times table has a numeric surrogate key for each level, so you can specify natural keys as an implementation detail for TIMES_DIM.

Each level in a Time dimension must have time-span and end-date attributes. However, the Times table does not have this data for Day or Fiscal Week. One way to correct this problem is to add the columns to the Times table, using SQL statements like the following:

```
ALTER TABLE times ADD
  ( days_in_day NUMBER(1) DEFAULT 1,
    days_in_week NUMBER(1) DEFAULT 7 );
```

When you have finished mapping the dimension, run the Maintenance Wizard to load the members and attributes. Because they load quickly, you can run the job immediately (instead of in the job queue) to verify that the mappings are correct.

Defining CUSTOMERS_DIM

The Customers and Countries tables are related on the Countries key column, and together they support two hierarchies, CUST_ROLLUP and GEOG_ROLLUP. Because the two hierarchies share two aggregate levels (CITY and STATE), you must generate surrogate keys in the analytic workspace so that each hierarchy has unique dimension members. Otherwise, a single set of aggregates might not be correct for both hierarchies.

Only 7,059 customers have sales data of the 55,500 listed in the Customers table, as shown in [Example 3-7](#). You can choose the way you implement CUSTOMERS_DIM:

- Load all of the customers into the analytic workspace, regardless of their purchasing history. This case study implements this choice.
- Create a view of the Customers table with a WHERE clause in the SELECT statement that filters the customers so that only those who have made purchases are included in the analytic workspace. Map CUSTOMERS_DIM to the new view.
- Define City as the base level; do not map the Customer level or its attributes. Create a view of the SALES table with a GROUP BY clause in the SELECT statement that aggregates the data to the CITY level. This choice is appropriate only if data at the Customer level is not needed for analysis.

When you have finished mapping the dimension, run the Maintenance Wizard to load the members and attributes. Because they load quickly, you can run the job immediately (instead of in the job queue) to verify that the mappings are correct.

Defining PRODUCTS_DIM, CHANNELS_DIM, and PROMOTIONS_DIM

The three remaining dimensions do not present any new challenges. Their source tables can be identified as star schema in the Mappings canvas, because all levels and attributes are in a single source table.

The measures in the Sales cube use only 4 of the 503 promotions listed in the PROMOTIONS_DIM dimension table. You have the same choices for handling this dimension as you did for the CUSTOMERS_DIM dimension, which also has a large percentage of unused key values.

Defining the Logical Sales Cube for Sales History

The definition of a cube involves decisions that affect performance. Unlike Global, the Sales History data set is fairly large and sparse like many real data sets. It is a good candidate for using the Sparsity Advisor. The Sparsity Advisor analyzes the sparsity characteristics of the data as it is stored in the relational source tables.

About the Sparsity Advisor

The Sparsity Advisor consists of several subprograms in the DBMS_AW PL/SQL package.

- SPARSITY_ADVICE_TABLE procedure creates a table for storing the advice generated by the ADVISE_SPARSITY procedure.
- 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.
- 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.
- 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.
- 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.

It returns its results in two forms: as a table with its recommendations and as OLAP DML commands that implement these recommendations. When creating an analytic workspace using Analytic Workspace Manager, you cannot use the OLAP DML commands directly. However, you can use the table or the commands (or both) to guide your choices on the property sheets.

See Also: *Oracle OLAP Reference* for complete information about the Sparsity Advisor

Sample Program for Evaluating Sales History Tables

[Example 3-5](#) shows a sample PL/SQL program for evaluating the sparsity characteristics of the tables in the SH schema. The ADD_DIMENSION_SOURCE procedure provides the Sparsity Advisor with information about the dimensions, levels, and hierarchies.

Example 3-5 PL/SQL Program for Using the Sparsity Advisor on Sales History

```
--Connect and set display parameters
CONNECT sh/sh
SET ECHO ON
SET LINESIZE 300
SET PAGESIZE 300
SET LONG      8000
SET SERVEROUT ON FORMAT WRAPPED

--Create a table for results
BEGIN
    dbms_aw.sparsity_advice_table('sh_sparsity_advice');
EXCEPTION
    WHEN OTHERS THEN NULL;
END;
/

TRUNCATE TABLE sh_sparsity_advice;
```

```

--Define program variables
DECLARE
    dimsources dbms_aw$_dimension_sources_t;
    dimlist VARCHAR2(500);
    sparsedim VARCHAR2(500);
    counter NUMBER(2) := 1;
    maxpart NUMBER(2);
    defs CLOB;
BEGIN
--Describe the dimension hierarchies
    dbms_aw.add_dimension_source('channel', 'channel_id', dimsources,
        'channels', dbms_aw.hier_levels,
        dbms_aw$_columnlist_t('channel_id', 'channel_class_id',
        'channel_total_id'));
    dbms_aw.add_dimension_source('product', 'prod_id', dimsources, 'products',
        dbms_aw.hier_levels, dbms_aw$_columnlist_t('prod_id',
        'prod_subcategory_id', 'prod_category_id', 'prod_total_id'));
    dbms_aw.add_dimension_source('customer', 'cust_id', dimsources,
        'customers', dbms_aw.hier_levels,
        dbms_aw$_columnlist_t('cust_id', 'cust_city_id',
        'cust_state_province_id', 'cust_total_id'));
    dbms_aw.add_dimension_source('time', 'time_id', dimsources, 'times',
        dbms_aw.hier_levels, dbms_aw$_columnlist_t('time_id',
        'calendar_month_id', 'calendar_quarter_id', 'calendar_year_id'));
    dbms_aw.add_dimension_source('promotion', 'promo_id', dimsources,
        'promotions', dbms_aw.hier_levels,
        dbms_aw$_columnlist_t('promo_id', 'promo_subcategory_id',
        'promo_category_id', 'promo_total_id'));

--Analyze tables using default settings
    dbms_aw.advise_sparsity('sales', 'sales_cube', dimsources,
        dbms_aw.advice_default, dbms_aw.partby_default, 'sh_sparsity_advice');

commit;

-- Get recommendations as OLAP DML commands
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;
/

-- Get recommendations directly from the output table
COLUMN cubename          FORMAT a8
COLUMN fact              FORMAT a20
COLUMN dimension         FORMAT a12
COLUMN dimcolumn         FORMAT a12
COLUMN dimsources        FORMAT a12
COLUMN nmem              FORMAT 99999
COLUMN nleaf             FORMAT 99999
COLUMN advice            FORMAT a12
COLUMN pos               FORMAT 99

```

```

COLUMN density          FORMAT 9.9999
COLUMN partnum          FORMAT 99
COLUMN partby           FORMAT a20
COLUMN parttops         FORMAT a20

--Get basic information about the dimensions
SELECT DISTINCT(dimension), dimcolumn, position pos, membercount nmem,
       leafcount nleaf
FROM sh_sparsity_advice
WHERE cubename='sales_cube'
ORDER BY position;

--Get partitioning advice about the dimensions
SELECT dimcolumn, advice, partnum, parttops
FROM sh_sparsity_advice
WHERE cubename='sales_cube'
ORDER BY partnum;

--Identify the partition tops
SELECT DISTINCT(calendar_year_id), calendar_year FROM times
ORDER BY calendar_year;

```

Interpreting the Results from the Sparsity Advisor

The Sparsity Advisor makes very detailed recommendations, but you can derive general recommendations that can be implemented in Analytic Workspace Manager:

- List the dimensions in this order: TIME, CHANNEL, PRODUCT, PROMOTION, CUSTOMER.
- Include all dimensions in a compressed composite.
- Create five partitions for the five years of data.

On the Implementation Details page for SALES_CUBE, do the following to implement these recommendations:

- In the Dimension Order and Sparsity table, list the dimensions in the recommended order and mark all of them as sparse.
- Select **Use Compression**.
- Select **Partition Cube**.
- For partitioning, select the TIMES_DIM dimension, CAL_ROLLUP hierarchy, and YEAR level.

[Example 3–6](#) shows the recommendations as OLAP DML commands. The dimensions are listed in order within angle brackets (<>), and definition of the sparse dimension shows all dimensions in a compressed composite. The partition template creates five partitions on TIME.

Example 3–6 Recommendations for Sales History in OLAP DML

```

Dimension list: <sales_cube_composite<time channel product promotion customer>>
Sparse dimension: DEFINE sales_cube_composite COMPOSITE COMPRESSED <time 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<>>)
.
.
.

```

The Sparsity Advisor bases its OLAP DML commands on the information stored in the output table, which in this example is named `SH_SPARSITY_ADVICE`. You can also view this information directly in the table.

Example 3-7 queries `SH_SPARSITY_ADVICE`. The first `SELECT` statement retrieves basic information about the dimensions. The `POSITION` column identifies their recommended order. `MEMBERCOUNT` shows the total number of dimension members defined in the dimension tables, and `LEAFCOUNT` shows the number with data in the `SALES` fact table.

The second `SELECT` statement retrieves partitioning advice. It shows five partitions based on Time, and compression is the recommendation for four of the five. A `SELECT` statement on the `TIMES` table shows that the partition tops are the five years.

Example 3-7 Recommendations for Sales History in the Advice Table

```

SELECT distinct(dimension), dimcolumn, position, membercount, leafcount
FROM sh_sparsity_advice
WHERE cubename='sales_cube'
ORDER BY position;

```

DIMENSION	DIMCOLUMN	POSITION	MEMBERCOUNT	LEAFCOUNT
time	time_id	1	1826	1460
channel	channel_id	2	5	4
product	prod_id	3	72	72
promotion	promo_id	4	503	4
customer	cust_id	5	55500	7059

```

SELECT dimcolumn, advice, partnum, parttops
FROM sh_sparsity_advice
WHERE cubename='sales_cube'
ORDER BY partnum;

```

DIMCOLUMN	ADVICE	PARTNUM	PARTTOPS
time_id	COMPRESSED	1	1803
channel_id	COMPRESSED	1	
prod_id	COMPRESSED	1	
promo_id	COMPRESSED	1	
cust_id	COMPRESSED	1	
time_id	COMPRESSED	2	1805
channel_id	COMPRESSED	2	
prod_id	COMPRESSED	2	
promo_id	COMPRESSED	2	
cust_id	COMPRESSED	2	
time_id	COMPRESSED	3	1804
channel_id	COMPRESSED	3	
prod_id	COMPRESSED	3	
promo_id	COMPRESSED	3	
cust_id	COMPRESSED	3	
time_id	COMPRESSED	4	1813
channel_id	COMPRESSED	4	
prod_id	COMPRESSED	4	
promo_id	COMPRESSED	4	

cust_id	COMPRESSED	4
time_id	SPARSE	5 1802
channel_id	SPARSE	5
prod_id	SPARSE	5
promo_id	SPARSE	5
cust_id	SPARSE	5

25 rows selected.

```
SELECT DISTINCT(calendar_year_id), calendar_year FROM times
       ORDER BY calendar_year;
```

CALENDAR_YEAR_ID	CALENDAR_YEAR
1802	1998
1803	1999
1804	2000
1805	2001
1813	2002

Maintaining Sales History

When building the cube, submit the maintenance task to the job queue, either to run immediately or at a later time. If you are running Oracle Database on a single-processor computer, keep the number of processes at 1. Otherwise, check the value of `JOB_QUEUE_PROCESSES` to see how many jobs you can run simultaneously. As defined in this example, Sales History can use up to six processes (1 cube + 5 partitions).

Predicting Future Performance

Using the Object View in Analytic Workspace Manager and OLAP Worksheet, you can generate a forecast and store it in a standard form measure. This chapter introduces the tools you can use to generate a forecast.

This chapter contains the following topics:

- [Creating a Forecast](#)
- [Case Study: Forecasting Global Sales](#)

Creating a Forecast

The OLAP option supports various forecasting methods, including simple linear regressions, several non-linear regression methods, single exponential smoothing, double exponential smoothing, and the Holt-Winters method. You can specify which one of these methods you prefer to use, but the OLAP engine determines the best fit for your data based on past performance and will override your choice if it is inappropriate for your data. The "automatic" method, which is the default, is the best choice because it defers to the best fit.

Most forecasts are calculated at the base level, and the base-level forecast data is used to generate forecast aggregates. The examples in this chapter assume that you wish to generate forecast aggregates in this way. The alternative method, which uses actual data at all levels to generate the forecast, produces forecast aggregates that may be inconsistent with the lower-level data.

Steps for Creating a Forecast

Forecasting is not supported at this time in the Analytic Workspace Manager graphical user interface. The following sections explain how you can generate a forecast manually and store the results in a standard form measure.

These are the steps for creating a forecast. Each one is discussed in more detail in the sections that follow.

1. [Creating the Forecast Time Periods](#)
2. [Defining a Measure for the Results](#)
3. [Defining Supporting Variables \(Optional\)](#)
4. [Developing a Forecast Program](#)
5. [Generating a Forecast](#)
6. [Aggregating the Forecast Data](#)

Creating the Forecast Time Periods

The future time periods that you want to forecast must be defined as members of the time dimension in your analytic workspace. If they do not exist there already, you must:

1. Add the new time periods and attributes to the relational tables in the source schema.
2. Use the Maintenance Wizard in Analytic Workspace Manager to add the new members to the Time dimension in the analytic workspace.

You should use whatever mechanism guarantees that these Time dimension members are identical to those for loading actual data at a later date.

Defining a Measure for the Results

In the Model View, define a standard form measure for the forecast results, as described in [Chapter 3](#). You can add the measure to the cube that contains the source data for the forecast, or you can create a new cube.

When you define a standard form measure, you create several analytic workspace objects. You will use one of them, a *measure_stored* **variable**, as the target of the forecast.

Defining Supporting Variables (Optional)

If you want the forecast to use seasonal adjustment or smoothing, then define variables that the forecast can use when performing its calculations. These variables do not require any standard form metadata, because they are only used during calculation of the forecast and are not accessed by the client.

Take these steps to define the supporting variables:

1. In the Object View, expand the folder for your analytic workspace.
2. Right-click the Variables folder and choose **Create Variable** from the menu.
3. Define the variable with a DECIMAL data type.
4. On the Dimensions page, list the dimensions in the appropriate order for variables in the cube, which is typically Time first, then a composite dimension.

["Defining a Variable for Seasonal Adjustment"](#) on page 4-4 provides an example of creating supporting variables.

Developing a Forecast Program

A forecast uses several related commands that are always executed from within an OLAP DML **program**. Use the following commands in the order they are listed here.

1. FCOPEN function. Opens a **forecasting context** and returns its handle.
2. FCSET command. Specifies the characteristics of a forecast.
3. FCEXEC command. Executes a forecast and populates Oracle OLAP variables with forecasting data.
4. FCQUERY function (optional). Retrieves information about the characteristics of a forecast or a trial of a forecast.
5. FCCLOSE command. Closes a forecasting context.

See Also: *Oracle OLAP DML Reference* for descriptions of the various forecasting methods, information about querying forecast trials, and the full syntax of these commands and functions.

You can define and compile a program either in the Object View or in OLAP Worksheet. You can run a program only in OLAP Worksheet.

[Example 4-1](#) provides a template for these commands and others that are typically used in a forecast.

Example 4-1 Template for a Forecast

```
VARIABLE handle INTEGER      " Define a local variable
TRAP ON OOPS                 " Redirect processing on error to OOPS label

" Select base level time periods
LIMIT time_dim TO levelrel_time 'base_data'
" Keep historical and forecast periods
LIMIT time_dim KEEP LAST n

" Open a handle for the forecast
handle = FCOPEN('forecast_name')
" Specify the forecast method
FCSET handle METHOD 'method' descriptors
" Execute the forecast and identify source and target variables
FCEXEC handle TIME time_dim INTO target_var1 SEASONAL -
      target_var2 SMSEASONAL target_var3 source_var
FCCLOSE handle              " Close the forecast
RETURN

OOPS:
SHOW 'Error running program'
```

Generating a Forecast

To generate the forecast data, run the forecast program in OLAP Worksheet, using the OLAP DML `CALL` command:

```
CALL program [(args)
]
```

The program calculates the forecast at the base level. You can then generate summary data by running the Maintenance Wizard. Be sure not to delete dimension members as a maintenance step, because you will lose the forecast data.

Aggregating the Forecast Data

In the OLAP DML, the `AGGREGATE` command calculates summary data that is stored permanently in the analytic workspace, and the `AGGREGATE` function triggers the calculation of the remaining summary data on the fly. An object called an aggmap identifies the areas of a cube that are calculated by the `AGGREGATE` command.

When you create a forecast measure, you create a formula that uses the `AGGREGATE` function to calculate forecast aggregates on the fly. It only calculates those areas of the cube that are not identified in the aggmap as precalculated by an `AGGREGATE` command. Because the Maintenance Wizard does not load data or precalculate the variable that stores the data for the forecast measure, you must execute the appropriate `AGGREGATE` command manually.

The name of the aggmap, which has the form OBJnnnnnnnn, is identified in the formula. The AGGREGATE command has this form:

```
AGGREGATE variable USING aggmap [COUNTVAR counter]
```

Where:

variable is a MEASURE_STORED object

aggmap is an AGGREGATIONDFN object

counter is a MEASURE_COUNTVAR object

[Appendix A](#) describes these standard form objects.

Case Study: Forecasting Global Sales

This example adds a measure named Sales Forecast to the Units cube.

In the Global analytic workspace, there are 65 historical periods (Jan-98 to Jun-04) and 18 forecast periods (Jul-04 to Dec-05) already loaded from the Global relational schema.

Defining the Sales Forecast Measure for Global Sales

Take these steps to create a measure for Sales Forecast:

1. Expand the UNITS_CUBE folder so that you can see its subfolders: Dimensions, Measures, and Calculated Measures.
2. In the UNITS_CUBE folder, right-click Measures and choose **Create Measure** from the pop-up menu.
3. Create a measure named SALES_FORECAST with a decimal data type. Use the aggregation specification from the cube.

Note: Do not map this measure to a data source.

This procedure creates several objects in the analytic workspace. The UNITS_CUBE_SALES_FORECAST_STORED variable is the target for the forecast, and it will store the forecast results.

Defining a Variable for Seasonal Adjustment

The Global Sales Forecast will use seasonal adjustment.

To create a variable for the OLAP engine to use when adjusting for seasonality, take these steps:

1. In the Object View, expand the folder for the Global analytic workspace.
2. Right-click the Variables folder and choose **Create Variable** from the pop-up menu.
3. On the Basic tab, specify the name UNITS_CUBE_SALES_FORECAST_SEASONAL and the Short Decimal data type.
4. On the Dimensions tab, select TIME, CUSTOMER, PRODUCT, and CHANNEL, in this order to match the dimension order of UNITS_CUBE_SALES_FORECAST_STORED.
5. Choose **Create**.

Developing a Forecasting Program for Global Sales

[Example 4-2](#) shows a program named `FORECAST_SALES`, which forecasts sales revenue. You can use this program as the basis of forecast programs in other analytic workspaces.

Although the program contains numerous commands, only four of them are used to define and execute the forecast. The default forecast method is `AUTOMATIC`, which uses the best method based on the historical data.

Historical and Forecast Time Periods

Because the base time period in Global is a month, seasonal adjustments are based on a 12-period cycle. The program uses the `INTEGER` argument of the `LIMIT` function to obtain the numeric position of the last historical time period, and sets the status of `TIME` relative to that position.

The program arguments, along with some preset local variables, are used to select the dimension members used to generate the forecast. All dimensions are limited to the base level.

The FORECAST_SALES Program

You can define a program directly in the Global analytic workspace, or you may prefer to create a separate analytic workspace with your custom programs. Be sure to work in the Object View, not in the Model View.

To create and compile the `FORECAST_SALES` program, take these steps:

1. In the Object View, expand the Global analytic workspace folder.
2. Right-click Programs and choose **Create Program** from the pop-up menu.
3. On the Basic tab, type the name `FORECAST_SALES`.
4. On the Program tab, cut-and-paste the program code shown in [Example 4-2](#).
5. Click **Create**.

`FORECAST_SALES` appears in the Programs folder.

6. Make whatever changes you want to the program, then click **Compile**.

Make whatever corrections are needed to compile the program.

Example 4-2 Forecasting Program for Global Sales

```
ARG _method          TEXT    " Forecasting method
ARG _last_time        TEXT    " Long desc of last hist time period
ARG _histperiods      INT     " Number of historical periods
ARG _fcast_periods    INT     " Number of forecast periods
ARG _periodicity      INT     " Number of periods in a cycle
VARIABLE _time_level  TEXT    " Base level of time dimension
VARIABLE _channel_level TEXT  " Base level of channel dimension
VARIABLE _product_level TEXT  " Base level of product dimension
VARIABLE _customer_level TEXT  " Base level of customer dimension
VARIABLE _last_time_pos INT    " Numeric position of _last_time in time dim
VARIABLE _handle      INT     " Forecast handle

TRAP ON OOPS          " Divert processing on error to OOPS label

" Set default values for args
if _method eq na
  then _method = 'AUTOMATIC'
```

```
if _last_time eq na
  then _last_time = 'Jun-04'
if _histperiods eq na
  then _histperiods = 48
if _fcast_periods eq na
  then _fcast_periods = 18
if _periodicity eq na
  then _periodicity = 12

" Identify base levels of dimensions (Product is value-based)
_time_level='MONTH'
_channel_level='CHANNEL'
_customer_level='SHIP_TO'

" Set dimension status to base level
PUSH time channel product customer
LIMIT channel TO channel_levelrel EQ _channel_level
LIMIT product TO all
LIMIT customer TO customer_levelrel EQ _customer_level
LIMIT time TO time_levelrel EQ _time_level

" Check time parameters of forecast and refine status of time dimension
_last_time_pos = LIMIT(INTEGER time TO time_long_description EQ _last_time)
IF _histperiods + _fcast_periods GT STATLEN(time)
  THEN SIGNAL toosmall 'You specified more time periods than are defined.'
IF _last_time_pos - _histperiods lt 0
  THEN SIGNAL nohist 'You specified too many historical periods.'
IF _last_time_pos + _fcast_periods GT STATLAST(time)
  THEN SIGNAL nofuture 'You specified too many forecast periods.'
ELSE LIMIT time KEEP -
  (_last_time_pos - _histperiods + 1) TO (_last_time_pos + _fcast_periods)

" Run the forecast
_handle = FCOPEN('sales')
FCSET _handle METHOD _method HISTPERIODS _histperiods PERIODICITY _periodicity
FCEXEC _handle TIME time INTO units_cube_sales_forecast_stored -
  SEASONAL units_cube_sales_forecast_seasonal units_cube_sales
FCCLOSE _handle
POP time channel product customer
RETURN

OOPS:
SHOW 'Program ended in an error.'
```

Generating the Global Sales Forecast

To execute the forecast, take these steps:

1. Open OLAP Worksheet by choosing **OLAP Worksheet** from the Tools menu.
2. Run the program using a command such as this one:

```
CALL forecast_sales
```

3. Save the forecast results by typing these commands:

```
UPDATE
COMMIT
```

The FORECAST_SALES program takes five arguments:

- The forecasting method (AUTOMATIC, LINREF, NLREL1 to NLREG5, SESMOOTH, DESMOOTH, or HOLT/WINTERS). These methods are described in the *Oracle OLAP DML Reference*.
- The long description of the last time period with historical data.
- The number of historical periods to be used in the forecast.
- The number of periods to forecast.
- The number of periods in a seasonal cycle.

Default values are set for these program arguments, so that they can be omitted from the command line as shown in the previous steps. These are some additional ways that you can run this program:

```
CALL forecast_sales('holt/winters')
CALL forecast_sales(na, na, 36, 6)
```

Because arguments are passed sequentially to the program, you may need to pass an NA as a placeholder value for some arguments, as shown in the last example. Later arguments can simply be omitted.

Aggregating the Sales Forecast Measure

To aggregate the forecast measure, take these steps:

1. From the Tools menu, choose **OLAP Worksheet**.
2. Use the DESCRIBE command to view the equation for the UNITS_CUBE_SALES_FORECAST formula.

```
DESCRIBE units_cube_sales_forecast
```

```
DEFINE UNITS_CUBE_SALES_FORECAST FORMULA DECIMAL <TIME CUSTOMER PRODUCT CHANNEL>
```

```
EQ aggregate(this_aw!UNITS_CUBE_SALES_FORECAST_STORED using this_aw!-
OBJ248542689 COUNTVAR this_aw!UNITS_CUBE_SALES_FORECAST_COUNTVAR)
```

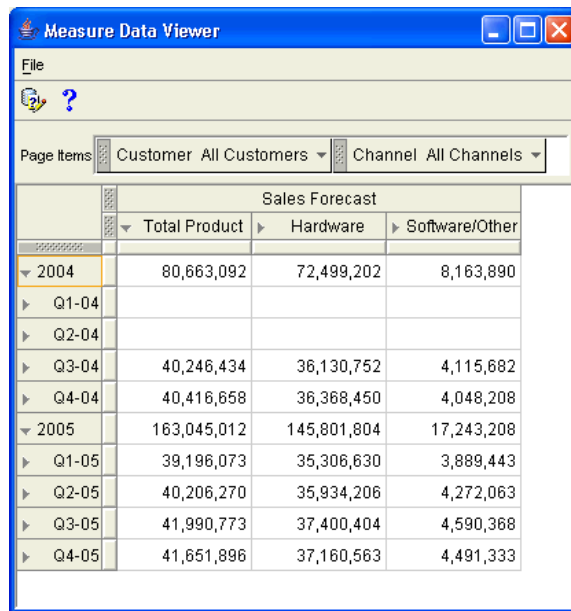
3. Issue a command like this one to aggregate the SALES_FORECAST measure:

```
AGGREGATE units_cube_sales_forecast_stored USING OBJ248542689
```

4. Save the summary data by typing these commands:

```
UPDATE
COMMIT
```

In the Model View, right-click SALES_FORECAST and choose **View Data** from the pop-up menu. Verify that the forecast results have been calculated, as shown in [Figure 4-1](#).

Figure 4–1 SALES_FORECAST displayed in Measure Data Viewer

Sales Forecast			
	Total Product	Hardware	Software/Other
▼ 2004	80,663,092	72,499,202	8,163,890
▶ Q1-04			
▶ Q2-04			
▶ Q3-04	40,246,434	36,130,752	4,115,682
▶ Q4-04	40,416,658	36,368,450	4,048,208
▼ 2005	163,045,012	145,801,804	17,243,208
▶ Q1-05	39,196,073	35,306,630	3,889,443
▶ Q2-05	40,206,270	35,934,206	4,272,063
▶ Q3-05	41,990,773	37,400,404	4,590,368
▶ Q4-05	41,651,896	37,160,563	4,491,333

Developing Java Applications for OLAP

This chapter presents the rich development environment and the powerful tools that you can use to create OLAP applications in Java. It includes the following topics:

- [Building Analytical Java Applications](#)
- [Introducing OracleBI Beans](#)
- [Understanding the OLAP API](#)
- [Managing Data Sources for OracleBI Beans and the OLAP API](#)
- [Building Java Applications that Manage Analytic Workspaces](#)

For information about SQL access to analytic workspaces, refer to the *Oracle OLAP Reference*.

Building Analytical Java Applications

Java is the language of the Internet. Using Java, application developers can write standalone Java applications (which can be launched from a browser with Java's WebStart technology) or HTML applications that access live data from Oracle Database, through servlets, JavaServer Pages (JSP), and Oracle User Interface XML (UIX).

About Java

Java is the preferred programming language for an ever-increasing number of professional software developers. For those who have been programming in C or C++, the move to Java is easy because it provides a familiar environment while avoiding many of the shortcomings of the C language. Developed by Sun Microsystems, Java is fast superseding C++ and Visual Basic as the language of choice for application developers, for the following reasons:

- Object oriented. Java enables application developers to focus on the data and methods of manipulating that data, rather than on abstract procedures; the programmer defines the desired object rather than the steps needed to create that object. Almost everything in Java is defined as an object.
- Platform independent. The Java compiler creates byte code that is interpreted at runtime by the Java Virtual Machine (JVM). As the result, the same software can run on all Windows, Linux, Unix, and Macintosh platforms where the JVM has been installed. All major browsers have the JVM built in.
- Network based. Java was designed to work over a network, which enables Java programs to handle remote resources as easily as local resources.

- Secure. Java code is either trusted or untrusted, and access to system resources is determined by this characteristic. Local code is trusted to have full access to system resources, but downloaded remote code (that is, an applet) is not trusted. The Java "sandbox" security model provides a very restricted environment for untrusted code.

The Java Solution for OLAP

To develop an OLAP application, you can use the Java programming language. Java enables you to write applications that are platform-independent and easily deployed over the Internet.

The OLAP API is a Java-based application programming interface that provides access to dimensional data for analytical business applications. Java classes in the OLAP API provide all of the functions required of an OLAP application: Connection to an OLAP instance; authentication of user credentials; access to data in the RDBMS controlled by the permissions granted to those credentials; and selection and manipulation of that data for business analysis.

OracleBI Beans simplifies application development by providing these functions as JavaBeans. Moreover, OracleBI Beans includes JavaBeans for presenting the data in graphs and crosstabs.

Note: Oracle JDeveloper and OracleBI Beans are not packaged with the Oracle RDBMS.

The OLAP API has a companion interface that can be used to build applications for OLAP DBAs. The OLAP Analytic Workspace Java API is a set of Java classes and an XML schema for designing, building, and updating analytic workspaces in the Oracle Database. For more information, see "[Building Java Applications that Manage Analytic Workspaces](#)" on page 5-7.

Oracle Java Development Environment

Oracle JDeveloper provides an integrated development environment (IDE) for developing Java applications. Although third-party Java IDEs can also be used effectively, only JDeveloper achieves full integration with the Oracle Database and OracleBI Beans wizards. The following are a few JDeveloper features:

- Remote graphical debugger with break points, watches, and an inspector.
- Multiple document interface (MDI)
- *Codecoach* feature that helps you to optimize your code
- Generation of 100% Pure Java applications, applets, servlets, Java beans, and so forth with no proprietary code or markers
- Oracle Database browser

Note: Oracle JDeveloper is an application and is not packaged with Oracle Database.

Introducing OracleBI Beans

OracleBI Beans provides reusable components that are the basic building blocks for OLAP decision support applications. Using OracleBI Beans, developers can rapidly develop and deploy new applications, because these large functional units have already been developed and tested — not only for their robustness, but also for their ease of use. And because OracleBI Beans provides a common look and feel to OLAP applications, the learning curve for end users is greatly reduced.

OracleBI Beans includes the following:

- **Presentation beans** display the data in a rich variety of formats so that trends and variations can easily be detected. Among the presentation beans currently available are Graph and Crosstab.
- **Data beans** acquire and manipulate the data. The data beans use the OLAP API to connect to a data source, define a query, manipulate the resultant data set, and return the results to the presentation beans for display. Data beans include a QueryBuilder, a CalcBuilder, and a Metadata Manager.
- **Persistence Service** is a set of packages that support the storage and retrieval of objects in the OracleBI Beans Catalog, not only so that you can save your work, but also so that you can share the work with others who have access to the Catalog.

OracleBI Beans can be incorporated in a Java client or an HTML client application. Java clients best support users who do immersed analyses, that is, use the system for extensive periods of time with a lot of interaction. For example, users who create reports benefit from a Java client. HTML clients best support remote users who use a low bandwidth connection and have basic analytical needs. Thin clients can be embedded in a portal or other Web site for these users.

Metadata

The OLAP API and OracleBI Beans use the logical model that is projected by the [Active Catalog](#) to obtain the information they need about dimensional objects defined in analytic workspaces. They use OLAP Catalog metadata to obtain information about dimensional objects defined in Oracle relational data warehouses.

OracleBI Beans generates additional metadata to support its additional functionality. This additional metadata is contained in the OracleBI Beans Catalog. The Metadata Manager presents applications with a consolidated view of metadata from the Active Catalog, OLAP Catalog, and the OracleBI Beans Catalog. For example, in the QueryBuilder, the measures obtained from the Active Catalog and the custom measures obtained from the OracleBI Beans Catalog appear together.

Navigation

The presentation beans support navigation techniques such as drilling, pivoting, and paging.

- **Drilling** displays lower-level values that contribute to a higher-level aggregate, such as the cities that contribute to a state total.
- **Pivoting** rotates the data cube so that the dimension members that labeled a graph series now label groups, or the dimension members that labeled columns in a crosstab now label rows instead. For example, if products label the rows and regions label the columns, then you can pivot the data cube so that products label the columns and regions label the rows.

- **Paging** handles additional dimensions by showing each member in a separate graph, crosstab, or table rather than nesting them in the columns or rows. For example, you might want to see each time period in a separate graph rather than all time periods on the same graph.

Formatting

The presentation beans enable you to change the appearance of a particular display. In addition, the values of the data itself can affect the format.

- **Number formatting.** Numerical displays can be modified by changing their scale, number of decimal digits and leading zeros, currency symbol, negative notation, and so forth.
- **Stoplight formatting.** The formatting of the cell background color, border, font, and so forth can be data driven so that outstanding or problematic results stand out visually from the other data values.

Graphs

The Graph bean presents data in a large selection of two- and three-dimensional business graph types, such as bar, area, line, pie, ring, scatter, bubble, pyramid, and stock market. Most graph types have several subtypes, such as clustered bar, stacked bar, and percent bar.

Bar, line, and area graphs can be combined so that individual rows in the data cube can be specified as one of these graph types. You can also assign marker shape and type, data line type, color, fill color, and width and on a row-by-row basis, depending on the type of graph.

The graph image can be exported in PNG and other image formats.

Users can zoom in and out of selected areas of a graph. They can also scroll across the axes.

Crosstabs

The Crosstab bean presents data in a two-dimensional grid similar to a spreadsheet. Multiple dimensions can be nested along the rows or columns, and additional dimensions can appear as separate pages. Among the available customizations are: Font style, size, and color; data-driven formatting, stoplight reporting, and underlining; individual cell background colors; border formats; and text alignment.

Users can navigate through the data using either a mouse or the keyboard.

Data Beans

The data beans use the OLAP API to provide the basic services needed by an application. They enable clients to identify a database, present credentials for accessing that database, and make a connection. The application can then access the metadata and identify the available data. Users can select the measures they want to see and the specific slice of data that is of interest to them. That data can then be modified and manipulated.

Wizards

OracleBI Beans offers wizards that can be used both by application developers in creating an initial environment and by end users in customizing applications to suit

their particular needs. The wizards lead you step-by-step so that you provide all of the information needed by an application. The following are some of the tasks that can be done using wizards.

- **Building a query.** Fact tables and materialized views often contain much more data than users are interested in viewing. Fetching vast quantities of data can also degrade performance unnecessarily. In addition to selecting measures, you can limit the amount of data fetched in a query by selecting dimension members from a list or using a set of conditions. Selections can be saved, and these saved selections can be used again just by picking their names from a list.

OracleBI Beans takes advantage of all of the new OLAP functions in the database, including ranking, lag, lead, and windowing. End users can create powerful queries that ask sophisticated analytical questions, without knowing SQL at all.

- **Generating custom measures.** You can define new "custom" measures whose values are calculated from data stored within the database. For example, a user might create a custom measure that shows the percent of change in sales from a year ago. The data in the custom measure would be calculated using the lag method on data in the Sales measure. Because a DBA cannot anticipate and create all of the calculations required by all users, OracleBI Beans enables users to create their own.

JSP Tag Library

OracleBI Beans includes an extensive JSP tag library that enables the development of applications without writing custom code. After you use wizards to create the presentations that are needed for an application, you can use JSP tags to insert the presentations in HTML pages and to create additional pages for the user interface.

The tags in this library are grouped in the following categories:

- **General tags.** Used to represent objects such as graphs, crosstabs, formatting tools, explorers for the OracleBI Beans Catalog, and controls for displaying messages; also includes a tag that lets you link the queries of graphs and crosstabs.
- **Dialog and wizard tags.** Used to create user interface elements that let end users manipulate presentations. For example, these tags let users change the type of a graph or export crosstab data.
- **List tags.** Used to create lists that let end users perform the following kinds of tasks: Modify queries by selecting dimensions or measures; browse for graphs or crosstabs in the Catalog; and navigate pages in an application.

OracleBI Beans also includes an extensive UNIX tag library.

Understanding the OLAP API

OLAP applications typically have object-oriented user interfaces where users manipulate objects that represent organized groupings of their data. Thus, there is a natural relationship between an object-oriented user interface and an object-oriented API such as the Oracle OLAP API. The OLAP API exploits this natural relationship by providing objects that match the end-user behavior that an application needs.

Object-oriented languages such as Java manipulate data by applying methods on objects. This approach enables the objects to maintain a current state and support incremental modifications to that state. This approach provides excellent support for common OLAP actions such as drill and rotate.

For example, a central activity for users of OLAP applications is refining queries. A user has a question in mind and devises a query to answer that question. In most cases, the initial results of the query prompt the user to want to dig deeper for a solution, perhaps by drilling to see more detailed data or by rotating the report to highlight correlations in the data. The OLAP API is able to use the result of one query as the input to the next query.

How the OLAP API Accesses Dimensional Data

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

Oracle OLAP translates all queries from the OLAP API into SQL; when a query is issued through the OLAP API, the SQL generator in Oracle OLAP issues a `SELECT` statement against a relational table or view. (When the data is stored in an analytic workspace, the relational view is dynamically generated during the query.) This has several advantages for application developers:

- The difficult task of writing the complex SQL needed to resolve dimensional queries, and even more difficult task of optimizing that complex SQL, is left for Oracle OLAP to do. Application developers can be more productive writing in the OLAP API, which is designed for OLAP.
- Updates to SQL and the OLAP DML will be incorporated into new versions of the OLAP API. Applications can make use of new analytic and performance features without recoding.

As an alternative access method, the OLAP API provides a way for a Java application to directly manipulate workspace data, without the need for any metadata and without the use of the OLAP API data manipulation classes. The Java application uses the `SPLExecutor` class in the OLAP API to send DML commands directly to Oracle OLAP for execution in the workspace.

Whichever access method is used, the application establishes a connection, opens the workspace, accesses the data (either through MDM metadata or through `SPLExecutor`), closes the workspace, and closes the connection.

See Also:

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

Calculation Capabilities

The OLAP API generates SQL commands to select and manipulate data stored in the relational tables or views. When the data is stored in an analytic workspace, the computational power of the OLAP engine can be used to manipulate the data, including:

- Modeling
- Forecasting
- What-if scenarios

When the data is stored in a star or snowflake schema, the SQL commands generated by the OLAP API can include the "N-pass" functions, such as `RANK`, `PERCENTILE`, `TOPN`, `BOTTOMN`, `LAG`, `LEAD`, `SUM`, `AVG`, `MIN`, `MAX`, `COUNT`, and `STDDEV`. Data fetches

use many database innovations, including concatenated rollup, scrollable cursors, and query rewrite.

The OLAP API provides expanded calculation capabilities beyond those that can be handled efficiently in other OLAP solutions, such as:

- Totals broken out by multiple attributes
- Suppression of NA and zero rows, columns, and pages
- Union dimensions
- Measures as dimensions
- Inter-row calculations such as the following book-to-bill ratio:

```
Balance(Account "BOOKED", Period "PRIOR") / Balance(Account "BILLED", Period "LAST")
```

- Asymmetric queries

Intelligent Caching

Analytical queries are by nature iterative. An analyst formulates a query, sees the results, and then formulates other queries based on those results. Since the likelihood is very high in business analysis of needing the same data to answer subsequent queries, the OLAP API caches the metadata so that it is available throughout the session without fetching it again. Moreover, the OLAP API defines the result set of a query geometrically. Using multidimensional cursors, the OLAP API can randomly access disparate regions of the result set. This enables an application to retrieve just the data currently of interest instead of all of the data in the result set. For example, you might scroll to the end of a page without having to fetch all of the data on the page.

Managing Data Sources for OracleBI Beans and the OLAP API

Applications built using OracleBI Beans and the OLAP API can have as a data source either an analytic workspace or a relational schema (star or snowflake). This guide is written primarily to describe the creation and management of analytic workspaces. However, the information for creating a relational data warehouse for use by OracleBI Beans and the OLAP API is also contained here.

Take these steps if you plan to use a star or snowflake schema as the data source for OLAP applications, and you do not plan to create an analytic workspace:

1. Create CWM1 or CWM2 metadata as described in ["Overview of the OLAP Catalog"](#) on page 7-3.
2. Using a SQL command processor such as SQL*Plus, issue this command to make the metadata accessible to OracleBI Beans.

```
EXECUTE CWM2_OLAP_METADATA_REFRESH.MR_REFRESH
```

3. Create materialized views as described in [Chapter 8](#).

Building Java Applications that Manage Analytic Workspaces

The Analytic Workspace application programming interface is a companion API to the OLAP API and OracleBI Beans. You can use the Analytic Workspace API to build Java applications that create and maintain analytic workspaces.

The Analytic Workspace API provides a set of Java classes that:

- Create a logical dimensional model of cubes, dimensions, measures, and attributes
- Define a set of mappings for loading data from relational columns into objects in the logical model
- Define the aggregation rules for data in the logical model
- Define advanced analytics such as allocations, forecasts, and models on objects in the logical model
- Instantiate the logical model in an analytic workspace

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 is executable by the `DBMS_AW_XML.EXECUTE` PL/SQL function. `DBMS_AW_XML.EXECUTE` can process any XML document that has been validated against the OLAP XML schema.

See Also:

- *Oracle OLAP Analytic Workspace Java API Reference*
- *Oracle OLAP Reference* for information on `DBMS_AW_XML.EXECUTE`

Administering Oracle OLAP

This chapter describes the various administrative tasks that are associated with Oracle OLAP. It contains the following topics:

- [Administration Overview](#)
- [Creating Tablespaces for Analytic Workspaces](#)
- [Setting Up User Names](#)
- [Initialization Parameters for Oracle OLAP](#)
- [Initialization Parameters for OracleBI Beans](#)
- [Permitting Access to External Files](#)
- [Understanding Data Storage](#)
- [Monitoring Performance](#)

Administration Overview

Because Oracle OLAP is contained in the database and its resources are managed using the same tools, the management tasks of Oracle OLAP and the database converge. Nonetheless, a database administrator or applications developer needs to address management tasks in the specific context of Oracle OLAP, in addition to creating and maintaining analytic workspaces. Following is a list of these tasks.

- **Tablespaces.** Create permanent and temporary tablespaces to prevent I/O bottlenecks, as described in "[Creating Tablespaces for Analytic Workspaces](#)" on page 6-2.
- **Security.** Users of OLAP applications must have database identities that have been granted the appropriate access rights. For users to have access to files, you must define directory objects and grant users access to them. Refer to "[Setting Up User Names](#)" on page 6-4.
- **Database configuration.** Set initialization parameters to optimize performance, as described in "[Initialization Parameters for Oracle OLAP](#)" on page 6-6 and "[Initialization Parameters for OracleBI Beans](#)" on page 6-7.
- **Performance.** Database monitoring tools can identify recommended changes to the database configuration based on past usage, as described in "[Monitoring Performance](#)" on page 6-11.

See Also: *Oracle Database Administrator's Guide* for detailed information about managing Oracle Database.

Creating Tablespaces for Analytic Workspaces

Before you create an analytic workspace, you should create undo, permanent, and temporary tablespaces dedicated to their use. Analytic workspaces are created in the user's default tablespace, unless the user specifies otherwise. The default tablespace for all users is set initially to `SYS`. Creating analytic workspaces in the `SYS` tablespace can degrade overall performance. Similarly, analytic workspaces should *not* share tablespaces with relational tables, especially not the source schema.

Oracle OLAP makes heavy use of temporary tablespaces, so it is particularly important that they be set up correctly to prevent I/O bottlenecks.

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

Creating an UNDO Tablespace

The following SQL commands create an undo tablespace.

```
CREATE UNDO TABLESPACE tablespace DATAFILE 'pathname'  
      SIZE size REUSE AUTOEXTEND ON NEXT size  
      MAXSIZE UNLIMITED EXTENT MANAGEMENT LOCAL;
```

Where:

tablespace is the name of the tablespace

pathname is the fully qualified file name

size is an appropriate number of bytes

For example:

```
CREATE UNDO TABLESPACE olapundo DATAFILE '$ORACLE_HOME/oradata/undo.dbf'  
      SIZE 64M REUSE AUTOEXTEND ON NEXT 8M  
      MAXSIZE UNLIMITED EXTENT MANAGEMENT LOCAL;
```

After creating the undo tablespace, change your system parameter file to include these settings, then restart the database as described in ["Initialization Parameters for Oracle OLAP"](#) on page 6-6.

```
UNDO_TABLESPACE=tablespace  
UNDO_MANAGEMENT=AUTO
```

Creating a Permanent Tablespace for Analytic Workspaces

When a user creates an analytic workspace, it is created in the user's default tablespace, which is initially set to the `SYS` tablespace. The following SQL statements create a tablespace appropriate for storing analytic workspaces.

```
CREATE TABLESPACE tablespace DATAFILE 'pathname'  
      SIZE size REUSE AUTOEXTEND ON NEXT size MAXSIZE UNLIMITED  
      EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO;
```

```
ALTER USER username DEFAULT TABLESPACE tablespace
```

Where:

tablespace is the name of the tablespace

pathname is the fully qualified file name

size is an appropriate number of bytes

username is the name of a database user

For example:

```
CREATE TABLESPACE glo DATAFILE '$ORACLE_HOME/oradata/glo.dbf'
  SIZE 64M REUSE AUTOEXTEND ON NEXT 8M MAXSIZE UNLIMITED
  EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO;
```

If your computer has multiple disks, then you can stripe the tablespace across them. The next example shows SQL statements that distribute the GLO tablespace across three physical disks:

```
CREATE TABLESPACE glo DATAFILE
  'disk1/oradata/glo1.dbf' SIZE 64M REUSE AUTOEXTEND ON NEXT 8M MAXSIZE 1024M
  EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO;

ALTER TABLESPACE glo ADD DATAFILE
  'disk2/oradata/glo2.dbf' SIZE 64M REUSE AUTOEXTEND ON NEXT 8M MAXSIZE 1024M,
  'disk3/oradata/glo3.dbf' SIZE 64M REUSE AUTOEXTEND ON NEXT 8M MAXSIZE UNLIMITED;
```

Creating a Temporary Tablespace for Analytic Workspaces

Oracle OLAP uses temporary tablespace to store all changes to the data in an analytic workspace, whether the changes are the result of a data load, what-if analysis, forecasting, aggregation, or some other analysis. An OLAP DML UPDATE command moves the changes into the permanent tablespace and clears the temporary tablespace.

Oracle OLAP also uses temporary tablespace to maintain different generations of an analytic workspace. This enables it to present a consistent view of the analytic workspace when one or more users are reading it while the contents are being updated. This usage creates numerous extensions within the tablespace, so be sure to specify a small EXTENT MANAGEMENT size.

The following commands create a temporary tablespace suitable for use by Oracle OLAP.

```
CREATE TEMPORARY TABLESPACE tablespace TEMPFILE 'pathname'
  SIZE size REUSE AUTOEXTEND ON NEXT size MAXSIZE UNLIMITED
  EXTENT MANAGEMENT LOCAL UNIFORM SIZE size;
```

Where:

pathname is a fully qualified file name
size is an appropriate number of bytes
tablespace is the name of the tablespace
username is a database user

For example:

```
CREATE TEMPORARY TABLESPACE glotmp TEMPFILE '$ORACLE_HOME/oradata/glotmp.tmp'
  SIZE 50M REUSE AUTOEXTEND ON NEXT 5M MAXSIZE UNLIMITED
  EXTENT MANAGEMENT LOCAL UNIFORM SIZE 256K;
```

You can stripe temporary tablespaces across several disks the same as permanent tablespaces. The next example shows the GLOTMP temporary tablespace striped across three physical disks.

```
CREATE TEMPORARY TABLESPACE glotmp TEMPFILE
  'disk1/oradata/glotmp1.tmp' SIZE 50M REUSE AUTOEXTEND ON NEXT 5M MAXSIZE 1024M
  EXTENT MANAGEMENT LOCAL UNIFORM SIZE 256K;

ALTER TABLESPACE glotmp ADD TEMPFILE
  'disk2/oradata/glotmp2.tmp' SIZE 50M REUSE AUTOEXTEND ON NEXT 5M MAXSIZE 1024M,
  'disk3/oradata/glotmp3.tmp' SIZE 50M REUSE AUTOEXTEND ON NEXT 5M MAXSIZE UNLIMITED;
```

Querying the Size of an Analytic Workspace

To find out the size of the tablespace extensions for a particular analytic workspace, use the following SQL statements:

```
COLUMN DBMS_LOB.GETLENGTH(AWLOB) HEADING "Bytes";
SELECT EXTNUM, SUM(DBMS_LOB.GETLENGTH(AWLOB)) FROM AW$aawname GROUP BY EXTNUM;
```

Where:

aawname is the name of the analytic workspace.

Setting Up User Names

To connect to the database, a user must present a user name and password that can be authenticated by database security. All users must have the CONNECT role. The additional privileges associated with that user name control the user's access to data. As a database administrator, you must set up user names with appropriate credentials for all users of Oracle OLAP applications.

You can define user names and grant them these rights from the Users General Page of Oracle Enterprise Manager Database Control or by using SQL commands.

Two roles are defined on installation of the database explicitly to support Oracle OLAP:

- OLAP_USER role provides users with the privileges to create, manage, or access standard form analytic workspaces. All OLAP users should have the OLAP_USER role or equivalent privileges.
- OLAP_DBA role provides a DBA or system administrator with privileges to create CWM metadata for relational tables. The OLAP_DBA role is granted with the DBA role.

See Also: *Oracle Database SQL Reference* for more information about granting privileges.

SQL Access For DBAs and Application Developers

To use Analytic Workspace Manager, users must be granted the OLAP_USER role. They also need SELECT privileges on the source schema tables, and an unlimited quota on the tablespace in which the workspace is created. [Example 6–1](#) shows the SQL statements for creating the GLOBAL_AW user.

Example 6–1 SQL Statements for Creating the GLOBAL_AW User

```
CREATE USER 'GLOBAL_AW' IDENTIFIED BY 'global_aw'
  DEFAULT TABLESPACE glo
  TEMPORARY TABLESPACE glotmp
  QUOTA UNLIMITED ON glo
  ACCOUNT UNLOCK;

GRANT SELECT ON global.channel_dim TO global_aw;
GRANT SELECT ON global.customer_dim TO global_aw;
GRANT SELECT ON global.product_dim TO global_aw;
GRANT SELECT ON global.time_dim TO global_aw;
GRANT SELECT ON global.price_and_cost_history_fact TO global_aw;
GRANT SELECT ON global.price_and_cost_update_fact TO global_aw;
GRANT SELECT ON global.units_history_fact TO global_aw;
GRANT SELECT ON global.units_update_fact TO global_aw;
```

SQL Access for Analysts

To access an existing analytic workspace, users must have these access privileges on the table in which the workspace is stored:

- To read from the analytic workspace, `SELECT` privileges.
- To write to the analytic workspace, `SELECT`, `INSERT`, and `UPDATE` privileges.

Note that the name of the table is the same as the name of the analytic workspace, with the addition of an `AW$` prefix. For example, the `GLOBAL` analytic workspace is stored in the `AW$GLOBAL` relational table.

For users to access views of workspace data, they must be granted `EXECUTE` privileges explicitly on those views.

[Example 6-2](#) shows the SQL statements that gives all users read-only privileges to the `GLOBAL` analytic workspace, and user `SCOTT` read/write privileges.

Example 6-2 Granting Access Rights to the GLOBAL Analytic Workspace

```
GRANT SELECT ON global_aw.aw$global TO public;
GRANT INSERT ON global_aw.aw$global TO scott;
GRANT UPDATE ON global_aw.aw$global TO scott;
```

Access to Database Objects Using OracleBI Beans

To connect to a database using OracleBI Beans, users must have the following access rights:

- `CONNECT` role
- `QUERY REWRITE` system privilege (for relational tables)
- `SELECT` privileges on the database objects containing the data to be analyzed, whether the data is stored in an analytic workspace or in relational tables. Refer to the previous topic, "[SQL Access for Analysts](#)", for information about granting access to analytic workspaces.

Access to the Oracle JVM

Users who want to author or execute Analytic Workspace Java API applications within the Oracle Java Virtual Machine (JVM) may need the following Java permissions, in addition to the `OLAP_DBA` or `OLAP_USER` role:

Table 6-1 Java Permissions

Permission Type	Action
<code>java.io.FilePermission</code>	read, write, execute
<code>java.util.PropertyPermission</code>	read, write
<code>java.net.SocketPermission</code>	connect, resolve
<code>java.lang.RuntimePermission</code>	null

You can grant these permissions in either Java or SQL.

See Also:

- *Oracle Database Java Developer's Guide* for information about Oracle JVM security and Java permissions
- *Oracle OLAP Analytic Workspace Java API Reference* for information about using this Java API

Initialization Parameters for Oracle OLAP

[Table 6–2](#) identifies the parameters that affect the performance of Oracle OLAP. Alter your server parameter file or `init.ora` file to these values, then restart your database instance. You can monitor the effectiveness of these settings and adjust them as necessary.

These recommendations assume that the computer is dedicated to Oracle Database, and Oracle Database is used predominately (if not exclusively) for OLAP. If you want to reserve some resources for other applications, then first calculate the percent of resources that are available to Oracle Database. For example, if your computer has 4G of physical memory and you want to reserve 25% for other applications, then you would calculate `SGA_TARGET` and `PGA_AGGREGATE_TARGET` based on 75% of 4G, which is 3G.

Table 6–2 *Initial Settings for Database Parameter Files*

Parameter	Setting
<code>JOB_QUEUE_PROCESSES</code>	Number of CPUs, plus one additional process for every three CPUs For example, <code>JOB_QUEUE_PROCESSES=5</code> for a four-processor computer
<code>PGA_AGGREGATE_TARGET</code>	25% of physical memory (increase up to 50% for builds and major query operations)
<code>SGA_TARGET</code>	50% of physical memory
<code>SESSIONS</code>	2.5 * maximum number of simultaneous OLAP users
<code>UTL_FILE_DIR</code>	Directory path where the Oracle Database can write to a file.
<code>UNDO_MANAGEMENT</code>	AUTO
<code>UNDO_TABLESPACE</code>	Name of the undo tablespace, which must be defined first as shown in " Creating an UNDO Tablespace " on page 6-2

See Also: *Oracle Database Performance Tuning Guide* for information about these parameters.

Procedure: Setting System Parameters for OLAP

Take the following steps to set system parameters:

1. Open the `init.ora` initialization file in a text editor.
2. Add or change the settings in the file.

For example, you might enter a command like this so that Oracle can write files to the `olapscripts` directory:

```
utl_file_dir=c:\olapscripts
```

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

```
SQLPLUS '/ AS SYSDBA'
SHUTDOWN IMMEDIATE
STARTUP pfile=$ORACLE_HOME/admin/rell10g/pfile/initrell10g.ora
```

About the PGA_AGGREGATE_TARGET Setting

PGA_AGGREGATE_TARGET helps the OLAP engine determine whether the OLAP page pool can grow in response to a session's demand for pages. It is also used by SQL statements, particularly when performing `SELECT` statements with `GROUP BY` and `ORDER BY` clauses. PGA_AGGREGATE_TARGET can affect the performance of OracleBI Beans when selecting data from relational tables.

Set PGA_AGGREGATE_TARGET initially to 200-400MB, and use the database performance monitoring tools to recommend adjustments.

Initialization Parameters for OracleBI Beans

OracleBI Beans performs best when the configuration parameters for the database are optimized for its use. During installation of Oracle Database, an OLAP configuration table is created and populated with `ALTER SESSION` commands that have been tested to optimize the performance of OracleBI Beans. Each time OracleBI Beans opens a session, it executes these `ALTER SESSION` commands.

If a database instance is being used only to support Java applications that use OracleBI Beans, then you can modify your server parameter file or `init.ora` file to include these settings. Alternatively, you might want to include some of the settings in the server parameter file and leave others in the table, depending upon how your database instance is going to be used. These are your choices:

- Keep all of the parameters in the configuration table, so that they are set as part of the initialization of a OracleBI Beans session. This method fully isolates these configuration settings solely for OracleBI Beans. (Default)
- Add some of the configuration parameters to the server parameter file or `init.ora` file, and delete those rows from the configuration table. This is useful if your database is being used by other applications that require the same settings.
- Add all of the configuration parameters to the server parameter file or `init.ora` file, and delete all rows from the configuration table. This is the most convenient if your database instance is being used only by OracleBI Beans.

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

See Also: *Oracle Database SQL Reference* for descriptions of initialization parameters that can be set by the `ALTER SESSION` command

Permitting Access to External Files

The OLAP DML contains three types of commands that read from and write to external files:

- File read commands that copy data from flat files to workspace objects.
- Import and export commands that copy workspace objects and their contents to files for transfer to another database instance.

- File input and output commands that read and execute DML commands from a file and redirect command output to a file.

These commands control access to files by using BFILE security. This database security mechanism creates a logical directory object to represent a physical disk directory. Permissions are assigned to the directory object, which control access to files within the associated physical directory.

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

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

Creating a Directory Object

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

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

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

Where:

directory is the name of the logical directory object
pathname is the physical directory path

Granting Access Rights to a Directory Object

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

```
GRANT permission ON DIRECTORY directory TO {user | role | PUBLIC};
```

Where:

permission is one of the following:

READ for read-only access
 WRITE for write-only access
 ALL for read and write access

directory is the name of the directory object

user is a database user

role is a database role

PUBLIC is all database users

Example: Creating and Using a Directory Object

The following SQL commands create a directory object named `OLAPFILES` to control access to a physical directory named `/users/oracle/OraHome1/olap` and grant read access to all users.

```
CREATE DIRECTORY olapfiles as '/users/oracle/OraHome1/olap';
GRANT READ ON DIRECTORY olapfiles TO PUBLIC;
```


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

```
IMPORT ALL FROM EIF FILE 'olapfiles/salesq2.eif' DATA DFNS
```

Understanding Data Storage

Oracle OLAP multidimensional data is stored in analytic workspaces, which are, in turn, stored in relational tables. An analytic workspace can contain a variety of objects, such as dimensions, variables, and OLAP DML programs. These objects typically support a particular application or set of data.

Whenever an analytic workspace is created, modified, or accessed, the information is stored in a table in the relational database.

Important: These tables are vital for the operation of Oracle OLAP. Do not delete them or attempt to modify them directly without being fully aware of the consequences.

Analytic Workspace Tables

Analytic workspaces are stored in tables in the Oracle Database. The names of these tables always begin with `AW$`.

For example, if the `GLOBAL_AW` user creates two analytic workspaces, one named `GLOBAL` and the other named `GLOBAL_PROGRAMS`, then these tables will be created in the `GLOBAL_AW` schema:

```
AW$GLOBAL
AW$GLOBAL_PROGRAMS
```

Tables are created by default with eight partitions. You can manage these partitions the same as you would for any other table in your database.

The tables store all of the object definitions and data. Each object in an analytic workspace is stored in one or more page spaces, and each page space is stored in a separate row of the table. A page space is grouping of related data pages; a page is a unit for swapping data in and out of memory.

For example, a dimension is stored in three page spaces and thus has three rows (one each for dimension members, a hash index, and a logical-to-physical map). A variable is stored in one row; a partitioned variable has a row for each partition.

[Table 6–3](#) describes the columns of a table that stores an analytic workspace.

Table 6–3 Column Descriptions for Analytic Workspace Tables

Column	Data Type	NULL	Description
EXTNUM	NUMBER (8)	-	Extension number Analytic workspaces are stored in physical LOBs (called extensions), which have a default maximum size of 500MB. The first extension is 0, the second is 1, and so forth.
PS#	NUMBER (10)	-	Page space number Each object is stored in at least one page space.

Table 6–3 (Cont.) Column Descriptions for Analytic Workspace Tables

Column	Data Type	NULL	Description
GEN#	NUMBER (10)	-	Generation number A generation (a snapshot of the page space) is maintained for each reader to assure a consistent view of the analytic workspace throughout a session.
AWLOB	BLOB	-	Analytic workspace LOB Actual storage of the analytic workspace object.
OBJNAME	VARCHAR2 (60)	-	Object name The name of the object in the analytic workspace.
PARTNAME	VARCHAR2 (60)	-	Partition name A name for the page space in which the object is stored. Each object is stored in its own page space. A partitioned variable is stored with a page space for each partition. The number of partitions and their names are specified when a partition template is created in the analytic workspace.

Table 6–4 shows a few sample rows of an analytic workspace table, which are the results of the following query.

```
SELECT ps#, gen#, objname, partname FROM aw$global WHERE
       OBJNAME = 'TIME' OR
       OBJNAME = 'UNITS_CUBE_UNITS_STORED'
ORDER BY GEN#, PS#;
```

Table 6–4 Sample Rows From AW\$GLOBAL

PS#	GEN#	OBJNAME	PARTNAME
2515	0	TIME	TIME
2516	0	TIME	TIME
2517	0	TIME	TIME
2745	0	UNITS_CUBE_UNITS_STORED	UNITS_CUBE_UNITS_STORED
2515	2	TIME	TIME
2516	2	TIME	TIME
2517	2	TIME	TIME

See Also: *Oracle OLAP DML Reference* for information about managing analytic workspaces.

System Tables and Views

The SYS user owns several tables and views associated with analytic workspaces:

```
AW$EXPRESS
AW$AWMD
AW$AWCREATE
AW$AWCREATE10G
AW$AWXML
```

AW\$
PS\$

Following are brief descriptions of these objects.

- **AW\$EXPRESS** stores the EXPRESS analytic workspace. This workspace contains objects and programs that support the OLAP DML. The EXPRESS workspace is used any time that a session is open.
- **AW\$AWCREATE** stores the AWCREATE analytic workspace, which contains programs for creating and managing standard form analytic workspaces for Analytic Workspace Manager 9.2.0.4.
- **AW\$AWCREATE10G** stores the AWCREATE10G analytic workspace, which contains programs for using OLAP Catalog metadata in Analytic Workspace Manager 10.1.0.3.
- **AW\$AWXML** stores the AXML analytic workspace, which contains programs for creating and managing standard form analytic workspaces for Analytic Workspace Manager 10g.
- **AW\$AWMD** stores the AWMD analytic workspace, which contains programs for creating standard form catalogs.
- **AW\$** maintains a record of all analytic workspaces in the database, recording its name, owner, and other information.
- **PS\$** maintains a history of all page spaces. A page space is an ordered series of bytes equivalent to a file. Oracle OLAP manages a cache of workspace pages. Pages are read from storage in a table and written into the cache in response to a query. The same page can be accessed by several sessions.

The information stored in **PS\$** enables the Oracle OLAP to discard pages that are no longer in use, and to maintain a consistent view of the data for all users, even when the workspace is being modified during their sessions. When changes to a workspace are saved, unused pages are purged and the corresponding rows are deleted from **PS\$**.

- **ALL_AW_OBJ** is a view that lists the current objects in all analytic workspaces to which the user has access rights.
- **ALL_AW_PROP** is a view that lists the current OLAP DML properties and their values in all analytic workspaces to which the user has access rights.

The **CWM1** and **CWM2** read APIs are tables owned by the **OLAPSYS** user. Public synonyms provide user access to these tables.

Monitoring Performance

Each Oracle Database instance maintains a set of virtual tables that record current database activity. These tables are called dynamic performance tables. The dynamic performance tables collect data on internal disk structures and memory structures. Among them are tables that collect data on Oracle OLAP. By monitoring these tables, you can detect usage trends and diagnose system bottlenecks. Refer to the *Oracle OLAP Reference* for information about the OLAP dynamic performance views.

Copying and Backing Up Analytic Workspaces

You can copy analytic workspaces at several levels, either as a way of replicating it on another computer or backing it up.

- **XML Template.** A template saves the XML definition of logical objects in a standard form analytic workspace. You can save the entire analytic workspace, or individual cubes, dimensions, and calculated measures. Using a saved template, you can create a new analytic workspace exactly like an existing one. The template does not save any data, nor does it save any customizations to the analytic workspace. You can copy a template to a different platform.
- **EIF File.** An EIF file saves the object definitions of any analytic workspace (not just standard form analytic workspaces), and optionally, saves the data also. You can copy an EIF file to a different platform.
- **Database Dump Files.** Analytic workspaces are copied with the other objects in a schema or database export. You can use either the `expdp/impdp` or the `exp/imp` database utilities.
- **Transportable Tablespaces.** Analytic workspaces are copied with the other objects to a transportable tablespace. However, you can only transport the tablespace to the same platform (for example, from Linux to Linux, Solaris to Solaris, or Windows to Windows). You can use either the `expdp/impdp` or the `exp/imp` database utilities. Transportable tablespaces are much faster than dump files.

The owner of an analytic workspace can create an XML template or an EIF file, or export the schema to a dump file. Only users with the `EXP_FULL_DATABASE` privilege or a privileged user (such as `SYS` or a user with the `DBA` role) can export the full database or create a transportable tablespace.

See Also:

- *Analytic Workspace Manager Help* for information about exporting to an XML template or an EIF file. Search for the topic "Saving Analytic Workspaces in Flat Files."
- *Oracle Database Utilities* for information about Oracle Data Pump and the `expdp/impdp` commands.

Part III

Creating a Relational Data Warehouse

Part 3 provides information about creating and maintaining a relational data warehouse for use by OLAP tools such as Discoverer Plus OLAP, Spreadsheet Add-In, and OracleBI Beans custom applications.

- [Chapter 7, "Using the OLAP Catalog"](#)
- [Chapter 8, "Materialized Views for the OLAP API"](#)

This information is for use only when the data source for an OLAP application is a star or snowflake schema. When analytic workspaces are used as the data source, neither OLAP Catalog metadata nor materialized views have any use.

Using the OLAP Catalog

This chapter describes methods of creating a logical dimensional model. It includes the following sections:

- [Choosing a Method for Creating OLAP Catalog Metadata](#)
- [Overview of the OLAP Catalog](#)
- [Creating Metadata Using Enterprise Manager Database Control](#)
- [Case Study: Creating Metadata for the GLOBAL Star Schema](#)
- [Creating Metadata Using PL/SQL](#)

Choosing a Method for Creating OLAP Catalog Metadata

You can use either Oracle Enterprise Manager Database Control or the CWM2 PL/SQL package to define a logical dimensional model in OLAP Catalog metadata. Both methods have restrictions on the format of the data sources, as described in the following topics. If your data sources do not conform to their requirements, then use Oracle Warehouse Builder to generate both a star schema and OLAP Catalog metadata.

For Source Data in a Basic Star or Snowflake Schema

The CWM1 write APIs, which are used by the OLAP Management tool in Database Control, create a database dimension object for each logical OLAP dimension. The database dimension object imposes the following restrictions on dimension tables and the related fact tables of a star or snowflake schema:

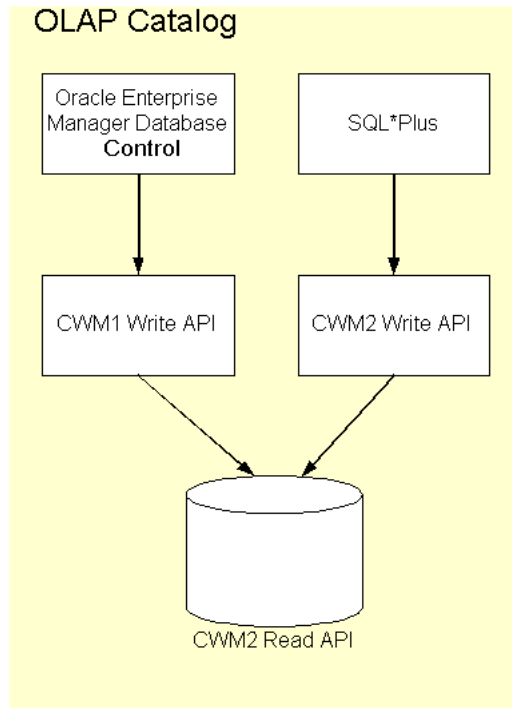
- All hierarchies must be level-based; the schema cannot use parent-child dimension tables.
- Multiple hierarchies defined for a dimension must have the same base level.
- Level columns cannot contain NULLs.
- Fact data must be unsolved, that is, it is stored only at the lowest level of the hierarchy, and all the data for a cube must be stored in a single fact table.

If your source data is a star or snowflake schema and conforms to these requirements, then you can use either Database Control or the CWM2 APIs, depending on your personal preference. The OLAP Management tool in Database Control provides a graphical user interface. The CWM2 APIs enable you to generate a SQL program that you can easily modify and port to other databases.

If your source data is a star or snowflake schema that does not conform with these requirements, then use the CWM2 APIs.

Figure 7-1 shows the tools for creating OLAP Catalog metadata.

Figure 7-1 Tools for Creating OLAP Catalog Metadata for Source Data



This chapter introduces the OLAP Management tool in Enterprise Manager Database Control and the CWM2 APIs.

See Also: *Oracle OLAP Reference* for complete syntax and descriptions of the CWM2 APIs

For Dimension Tables with Complex Hierarchies

If your source data is a star or snowflake schema, but the dimension tables include any of the following variations, then use the CWM2 APIs:

- Level columns containing NULLs, such as skip-level hierarchies
- Multiple hierarchies with different base levels (sometimes called **ragged hierarchies**)
- Multiple hierarchies with values mapped to different levels
- Embedded total dimensions
- Parent-child dimensions

If your schema contains parent-child dimension tables, then you must convert them to level-based dimension tables. The CWM2 write APIs include a package for this transforming symmetrical parent-child dimension tables.

For Other Schema Configurations

If you are using Oracle Warehouse Builder already to transform your data, then generating a metadata takes only an extra step. Warehouse Builder provides a graphical interface for designing a logical model, and deploys the model as metadata. It generates CWM2 metadata from its Design Repository.

If your data is stored in flat files or SQL tables, then you can use a manual method described in this guide. This method enables you to use the OLAP Catalog, but requires you to write data loading programs in the OLAP DML.

If you are upgrading from Oracle Express, then you may be able to automate the conversion process.

See Also:

- *Oracle Warehouse Builder User's Guide* if your data requires transformation
- [Appendix B](#) if your data is in an Express database

Overview of the OLAP Catalog

The OLAP Catalog defines logical dimensional objects and maps them to physical data sources. The logical objects are cubes, measures, dimensions, and so forth as described in "[The Logical Dimensional Data Model](#)" on page 1-5. The physical data sources are the columns of a relational star or snowflake schema.

OLAP Catalog Components

The OLAP Catalog includes the following:

- **Metadata model tables:** A set of relational tables within the database that instantiate the OLAP metadata model. These tables define all the OLAP metadata objects: dimensions, measures, cubes, measure folders, and so on. Within the metadata definitions are references to the actual data sources.
- **Write API:** A set of PL/SQL packages for creating and editing OLAP metadata. These packages contain procedures for inserting, updating, and deleting rows in the model tables.
- **Read API:** A set of relational views within the database that provide information about the metadata registered in the model tables.

Two versions of the OLAP Catalog are currently in use, CWM1 (also called CWM-Lite) and CWM2. Each version has its own metadata model tables, write API, and read API. However, applications can query a set of union views that contains all of the OLAP Catalog metadata, regardless of the write API used to generate it.

About CWM1

CWM1 is available through the OLAP Management tool of Enterprise Manager Database Control. You can use CWM1 only to describe a schema that complies with the requirements listed in "[Choosing a Method for Creating OLAP Catalog Metadata](#)" on page 7-1. You can then use the OLAP Catalog to access the relational schema directly through OracleBI Beans.

You can view CWM1 metadata in the OLAP Management tool of Database Control.

About CWM2

CWM2 is available as a set of PL/SQL packages. You can use CWM2 to describe a star or snowflake schema that does not comply with the requirements for CWM1.

You can view CWM2 metadata by querying the read API.

Steps for Creating OLAP Metadata

Whether you create OLAP metadata programmatically or by using a graphic interface, you follow the same basic steps.

To create OLAP metadata:

1. Create logical dimensions. Specify the levels, attributes, and hierarchies associated with each one. ("[Procedure: Defining a Logical Dimension in the OLAP Catalog](#)" on page 7-5)
2. Create logical cubes and specify their edges (dimensions). ("[Procedure: Defining a Logical Cube in the OLAP Catalog](#)" on page 7-6)
3. Create logical measures that represent the fact data. Associate each measure with a cube. ("[Procedure: Defining a Logical Cube in the OLAP Catalog](#)")
4. Map the logical entities to the source data. ("[Procedure: Defining a Logical Cube in the OLAP Catalog](#)" on page 7-6)

Creating Metadata Using Enterprise Manager Database Control

If your data warehouse complies with the requirements listed in "[For Source Data in a Basic Star or Snowflake Schema](#)" on page 7-1, you can create OLAP metadata using the OLAP Management tool in Enterprise Manager Database Control.

You generate the SQL statements that create the metadata primarily by following the steps presented by a wizard or by completing a property sheet. If you wish, you can display the SQL statements before executing them.

Procedure: Accessing OLAP Management

Follow these steps to access OLAP Management:

1. Open Enterprise Manager Database Control in your browser.
The login page is displayed.
2. Enter a user name and password.
3. For the Grid Control edition of Enterprise Manager, then do the following from the Grid Control home page:
 - a. Click the Targets tab.
The Hosts page is displayed.
 - b. Click the Database tab.
The Databases page is displayed.
 - c. Click the link for the database you want to manage.
The Database home page is displayed.
4. Click the Administration tab.
The Administration page is displayed.

5. Look for the Warehouse heading. Links in the left column are used for Oracle OLAP: **Cubes**, **OLAP Dimensions**, and **Measure Folders**. These links are for OLAP Management.

The other Warehouse links are used only for relational warehouses that do not use the OLAP option. Do not use those links.

Defining Metadata for Dimension Tables

When creating OLAP metadata, you must first define the metadata objects for the dimension tables. These metadata objects are logical dimensions based on database dimension objects.

Information That You Supply for Dimensions

To define a dimension, you provide all the information that will be needed to label and aggregate the measures dimensioned by it, including:

- The name of the dimension
- The name of each level, and the columns that contain the data for each one
- Join keys for levels that are stored in separate tables
- The name of each hierarchy, and the order of levels in each one
- The name of each attribute, and the columns that contain data for each one
- A display name and description for the dimension and each of its hierarchies, levels, and attributes

Time Dimension

Business analysis is performed on historical data, so fully defined time periods are vital. Your Time dimension table must have columns for period end dates and time span. This information supports time-series analysis, such as comparisons with earlier time periods. If your schema does not have these columns, then you can define Time as a normal dimension, but it will not support time-based analysis.

Procedure: Defining a Logical Dimension in the OLAP Catalog

Follow these steps to create a dimension and its associated levels, hierarchies, and attributes:

1. Start Enterprise Manager Database Control and access OLAP Management, as described in "[Procedure: Accessing OLAP Management](#)" on page 7-4.
2. Click the **OLAP Dimensions** link under Warehouse.
The Dimensions page is displayed.
3. Click **Create**.
The Create Dimension page is displayed.
4. Choose **Help** for further information.

Defining Metadata for Fact Tables

After you have defined the metadata objects for the dimension tables, you can create metadata objects for the fact tables. These metadata objects are measures and cubes. A cube is a collection of identically dimensioned measures. Cubes and measures are defined entirely in the OLAP metadata; there are no corresponding database objects.

Information That You Supply for Cubes

When you define a cube, you identify information such as the following:

- The name of the cube and the fact table associated with it. All measures in a cube must be from a single fact table.
- The names of the dimensions and the levels in the dimension hierarchies that will be used in the cube.
- The names of the measures and the columns in the fact table where the values for each measure are stored.
- Default aggregation operators for each dimension of each measure (such as sum or average).
- Any dimension dependencies.

Procedure: Defining a Logical Cube in the OLAP Catalog

Follow these steps to create a cube:

1. If you have not done so already, start Enterprise Manager Database Control and access OLAP Management, as described in "[Procedure: Accessing OLAP Management](#)" on page 7-4.
2. Click the **Cubes** link.
The Cubes page is displayed.
3. Click **Create**.
The Create Cube page is displayed.
4. Choose **Help** for further information.
5. When you are done creating metadata, open SQL*Plus and issue this command:

```
EXECUTE CWM2_OLAP_METADATA_REFRESH.MR_REFRESH;
```
6. Create materialized views as described in [Chapter 8](#).

Case Study: Creating Metadata for the GLOBAL Star Schema

The Global star schema conforms to all of the requirements of CWM1, so you can use the OLAP Management tool in Enterprise Manager Database Control.

The following procedures explain how to define just one dimension and one cube in the Global schema. However, you can follow this example by creating the metadata in a different schema.

See [Chapter 2](#) for instructions for installing the Global schema.

Defining a Logical Time Dimension for the Global Schema

The TIMES_DIM table supports a single Calendar hierarchy with three levels (Month, Quarter, and Year) as described in [Table 7-1](#).

Table 7-1 Global TIME Dimension Mapping: CALENDAR Hierarchy

TIME Objects in GLOBAL	GLOBAL.TIME_DIM Columns
CALENDAR hierarchy, MONTH level	MONTH_ID
CALENDAR hierarchy, QUARTER level	QUARTER_ID

Table 7–1 (Cont.) Global TIME Dimension Mapping: CALENDAR Hierarchy

TIME Objects in GLOBAL	GLOBAL.TIME_DIM Columns
CALENDAR hierarchy, YEAR level	YEAR_ID
MONTH Long Description attribute MONTH Short Description attribute	MONTH_DSC
QUARTER Long attribute Attribute QUARTER Short attribute Attribute	QUARTER_DSC
YEAR Long Description attribute YEAR Short Description attribute	YEAR_DSC
MONTH Time_Span attribute	MONTH_TIMESPAN
QUARTER Time_Span attribute	QUARTER_TIMESPAN
YEAR Time_Span attribute	YEAR_TIMESPAN
MONTH End_Date attribute	MONTH_END_DATE
QUARTER End_Date attribute	QUARTER_END_DATE
YEAR End_Date attribute	YEAR_END_DATE

These are the steps to define a logical Time dimension, using the general instructions in ["Procedure: Defining a Logical Dimension in the OLAP Catalog"](#) on page 7-5.

1. On the Dimensions page, choose **Create**.
The Create Dimension page is displayed.
2. On the General tab, do the following:
 - For Name, type `time`.
 - For Schema, type `global` (or click the flashlight icon to display the Search and Select dialog).
 - For Type, select **Time**.
3. On the Levels tab, click **Add** to display the Add Level page. Then do the following:
 - a. For Name, type `year`.
 - b. For Type, choose **Year** from the drop-down menu.
 - c. For Table, type `GLOBAL.TIME_DIM`.
 - d. Click **Populate Columns**.
 - e. Move `YEAR_ID` from Available Columns to Selected Columns.
 - f. Click **OK**.
 - g. Repeat these steps for the Quarter and Month levels, making the appropriate changes.
4. On the Hierarchies tab, click **Add** to display the Add Hierarchy page. Then do the following:
 - a. For Name, type `calendar`.
 - b. Choose **Move All**.
 - c. Use the up- and down-arrow keys to order the levels like this:

YEAR
QUARTER
MONTH

- d. Click **OK**.
5. On the Attributes tab, add the Time_Span and End_Date attributes. The Long_Description and Short_Description attributes are already defined. On the Add Attribute or Edit Attribute pages, select all levels and map them to the columns shown in [Table 7-1](#).
6. On the OLAP Options tab, type whatever descriptions you want to add.
7. Click **OK** to create the Time dimension.

When you have successfully created a dimension, it appears on the Dimensions page.

Defining a Logical Price and Cost Cube for the Global Schema

The PRICE_AND_COST_HISTORY_FACT table has a multi-column primary key, composed of two surrogate keys from two dimension tables, and two measures (UNIT_COST and UNIT_PRICE), as shown in [Table 7-2](#).

Table 7-2 Global PRICE_AND_COST_CUBE Mapping

PRICE_AND_COST_HISTORY_FACT Columns	Logical Objects
ITEM_ID	PRODUCT dimension
MONTH_ID	TIME dimension
UNIT_PRICE	UNIT_PRICE measure
UNIT_COST	UNIT_COST measure

These are the steps to define a logical Price cube, using the basic steps listed in "[Procedure: Defining a Logical Cube in the OLAP Catalog](#)" on page 7-6

1. On the Cubes page, choose **Create**.
The Create Cube page is displayed.
2. On the General tab, do the following:
 - a. For Cube Name, type PRICE_CUBE.
 - b. For Display Name, type Price Cube.
 - c. For Schema, type **GLOBAL**.
 - d. For Description, type your own description.
 - e. For Fact Type, choose **Table**.
 - f. For Fact Schema, type **GLOBAL**.
 - g. For Fact Table, type PRICE_AND_COST_HISTORY_FACT.
3. On the Dimension tab, add the PRODUCT and TIME dimensions. Identify the appropriate foreign key columns in the fact tables, as shown in [Table 7-2](#).
4. On the Measure tab, add the UNIT_PRICE and UNIT_COST measures.
5. On the Aggregation tab, choose **MAX** or another aggregation operator of your own choosing for both dimensions.

Creating Metadata Using PL/SQL

The CWM2 PL/SQL packages contain stored procedures that can create OLAP metadata for a variety of schema designs, as described in ["Choosing a Method for Creating OLAP Catalog Metadata"](#) on page 7-1.

Before using these packages, make sure that you have performed any required preprocessing steps.

See Also: *Oracle OLAP Reference* for the comprehensive syntax of the CWM2 packages.

CWM2 Packages for Creating OLAP Dimensions

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

- CWM2_OLAP_DIMENSION contains procedures for creating dimensions.
- CWM2_OLAP_HIERARCHY contains procedures for creating hierarchies for dimensions.
- CWM2_OLAP_LEVEL contains procedures for creating levels for dimensions and for associating levels with hierarchies.
- CWM2_OLAP_LEVEL_ATTRIBUTE contains procedures for creating level attributes and associating them with levels.
- CWM2_OLAP_DIMENSION_ATTRIBUTE contains procedures for creating dimension attributes and associating them with dimensions.

CWM2 Packages for Creating Cubes

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

- CWM2_OLAP_CUBE contains procedures for creating the dimensional structure of cubes.
- CWM2_OLAP_MEASURE contains procedures for creating measures and associating them with cubes.

CWM2 Package for Mapping Metadata

The CWM2_OLAP_TABLE_MAP package contains procedures that map logical metadata entities to their physical data source. The data may be stored in relational tables, or it may be represented by relational views.

CWM2 Package for Creating Level-Based Dimension Tables

The CWM2_OLAP_PC_TRANSFORM package contains a procedure for transforming symmetrical parent-child dimension tables to level-based dimension tables. This conversion is necessary if the dimension will be accessed by OracleBI Beans.

CWM2 Packages for Classification and Validation

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

- CWM2_OLAP_CATALOG provides procedures for creating and maintaining measure folders.

- CWM2_OLAP_VALIDATE provides procedures for validating OLAP Catalog metadata.
- CWM2_OLAP_METADATA_REFRESH provides procedures for refreshing metadata tables that support queries by OracleBI Beans against relational schemas.

Materialized Views for the OLAP API

This chapter explains how to create materialized views specific to the requirements of the OLAP API and OracleBI Beans. If you are using analytic workspaces, then you can skip this information because an analytic workspace generates and stores aggregate data so that materialized views are unnecessary. However, if you are developing a strictly relational application, then you must create materialized views using the methods described here. Otherwise, the SQL used to create the materialized views will not match the SQL generated by the OLAP API, and query rewrite will not use the materialized views to formulate the answer set to a query.

See Also:

- *Oracle OLAP Reference* for the syntax of the DBMS_ODM package
- *Oracle Data Warehousing Guide* for information on managing materialized views
- *Best Practices for Tabular Cube Aggregation and Query Operations* for optimizations to the Oracle Database environment and schema design

This chapter includes the following topics:

- [Summary Management with Oracle OLAP](#)
- [Overview and Requirements](#)
- [A Dimension Materialized View](#)
- [A Fact Materialized View](#)
- [Using the DBMS_ODM Package](#)
- [Example: Automatically Generate the Materialized Views for a Price Cube](#)
- [Example: Manually Generate the Materialized Views for a Sales Cube](#)

Summary Management with Oracle OLAP

A basic feature of online analytical processing (OLAP) is the ability to analyze and view various levels of aggregate data. With Oracle OLAP, you can choose to store aggregate data within analytic workspaces or within materialized views.

Summary management for relational warehouses is managed by Oracle's query rewrite facility. Query rewrite enables a query to fetch aggregate data from materialized views rather than recomputing the aggregates at runtime.

When the OLAP API queries a warehouse stored in relational tables, it uses query rewrite whenever possible. To prepare your relational warehouse for access by the OLAP API, you need to establish materialized views according to the guidelines described in this chapter.

Overview and Requirements

The OLAP API requires a specific set of materialized views for each OLAP Catalog cube that maps to a star schema. The cube must be mapped to a single fact table, and the fact table may contain only lowest-level data.

For each cube, there must be a separate dimension materialized view for each hierarchy of each of the cube's dimensions. For the cube's fact table, there must be a single materialized view, created with `GROUP BY GROUPING SETS` syntax.

Use the Oracle Data Management package, `DBMS_ODM`, to create materialized views.

Important: Do not use the `DBMS_OLAP` package to create materialized views for the OLAP API. Query rewrite will not map the SQL generated by the OLAP API to the materialized views generated by this package.

The `DBMS_OLAP` package is described in the *Oracle Data Warehousing Guide*.

Materialized Views Required for a Cube

The OLAP API requires a dimension materialized view for each hierarchy associated with a cube. For example, the `SALES_CUBE` cube in the Sales History (SH) schema requires seven dimension materialized views, as illustrated in [Table 8-1](#).

Table 8-1 Number of Dimension Materialized Views for `SH.SALES_CUBE`

SALES_CUBE Dimensions	Hierarchies	Number of MVs
SH.CHANNELS_DIM	CHANNEL_ROLLUP	1
SH.CUSTOMERS_DIM	CUST_ROLLUP	2
	GEOG_ROLLUP	
SH.PRODUCTS_DIM	PROD_ROLLUP	1
SH.PROMOTIONS_DIM	PROMO_ROLLUP	1
SH.TIMES_DIM	CAL_ROLLUP	2
	FIS_ROLLUP	

For the cube's fact table, the OLAP API requires a single grouping set materialized view.

Materialized Views and OLAP Metadata

Before creating materialized views, you must create OLAP metadata for the star schema. You can use Oracle Enterprise Manager, or you can write a script using the CWM2 packages. Refer to [Chapter 7](#) for information about the OLAP Catalog.

A Dimension Materialized View

The SQL script for creating dimension materialized views includes a `CREATE MATERIALIZED VIEW` statement, and statements for generating statistics and bitmap indexes.

CREATE Materialized View for a Dimension Hierarchy

The basic syntax of the `CREATE MATERIALIZED VIEW` statement for a dimension hierarchy is as follows.

```
CREATE MATERIALIZED VIEW mv_name
PARTITION BY RANGE (gid)
    (partition values less than(1) ,
      .
      .
      partition values less than(MAXVALUE))
TABLESPACE tblspace_name
BUILD IMMEDIATE
USING NO INDEX
REFRESH FORCE
ENABLE QUERY REWRITE
AS
SELECT
    COUNT(*) COUNT_STAR,
    GROUPING_ID(level_columns) gid,
    MAX(attribute_column_1)
    .
    .
    MAX(attribute_column_n)
    level_cols
FROM
    dimension_tables
GROUP BY
    hierarchy1_level1, ROLLUP(hierarchy1_level2,... hierarchy1_leveln),
    hierarchy2_level1, ROLLUP(hierarchy2_level2,... hierarchy2_leveln),
    .
    .
    hierarchyn_level1, ROLLUP(hierarchyn_level2,... hierarchyn_leveln);
```

In the `GROUP BY` clause, level columns are listed in order from most aggregate (level1) to least aggregate (leveln). The least aggregate level, or "leaf node", is also the key column. Note that level1 is excluded from the `ROLLUP` list.

Bitmap Indexes for a Dimension Hierarchy

The script includes statements like the following to generate bitmap indexes for the level columns and the `GID` column. It also calculates a bitmap index for the parent `GID` and parent `ET` key.

```
CREATE BITMAP INDEX index_name ON mv_name(level_column)
PCTFREE 0
COMPUTE STATISTICS
LOCAL
NOLOGGING;
```

Statistics for a Dimension Hierarchy

The script includes statements like the following to generate statistics.

```
execute dbms_stats.gather_table_stats(mv_owner, mv_name,
                                     degree=>dbms_stats.default_degree,
                                     estimate_percent=>dbms_stats.auto_sample_size,
                                     method_opt=>'for all hidden columns size 254') ;
method_opt=>
  'for all columns size skewonly') ;
ALTER TABLE mv_name MINIMIZE RECORDS_PER_BLOCK ;
```

A Fact Materialized View

The SQL script generated by the DBMS_ODM package for creating fact materialized views includes a CREATE MATERIALIZED VIEW statement and statements for generating statistics and bitmap indexes.

CREATE Fact Materialized View

The basic syntax of the CREATE MATERIALIZED VIEW statement with grouping sets for a fact table is as follows.

```
CREATE MATERIALIZED VIEW mv_name
PARTITION BY RANGE (gid)
  (partition values less than(1) ,
   .
   .
   partition values less than(MAXVALUE))
PCTFREE x PCTUSED y
BUILD IMMEDIATE
USING NO INDEX
REFRESH FORCE
ENABLE QUERY REWRITE
AS
SELECT
  GROUPING_ID(level_columns) gid,
  agg_method(measure_1),
  .
  .
  agg_method(measure_n),
  COUNT(*) COUNT_OF_STAR,
  level_columns
FROM
  dimension_tables, fact_table
WHERE
  (dimension_primary_key_1 = fact_foreign_key_1) AND
  .
  .
  (dimension_primary_key_n = fact_foreign_key_n)
GROUP BY GROUPING SETS (
  (level columns in grouping set_1),
  .
  .
  (level columns in grouping set_n);
```

Each grouping set contains a combination of levels specified for aggregation. For example, a grouping set could specify that the cube's data be aggregated by month for all products in each region.

The `SELECT` clause lists the levels from the dimension tables and the measures from the fact table. The selected measures will be aggregated over each combination of these levels that has been specified for aggregation. The aggregation method is typically addition (`SUM`), but it may be a method such as average or weighted average. If you want to use a different aggregation operator, you must specify it in the OLAP Catalog metadata for each of the cube's dimensions.

Bitmap Indexes for Fact Materialized Views

The script includes statements like the following to generate bitmap indexes for each level chosen for inclusion in the materialized view. It also creates a bitmap index for all higher aggregate levels within the dimension. For example, if you chose to aggregate to the quarter level of a time calendar hierarchy, a bitmap index would be created for year and quarter.

```
CREATE BITMAP INDEX index_name ON mv_name(level_col)
LOCAL
COMPUTE STATISTICS
PARALLEL PCTFREE 0
NOLOGGING;
```

Statistics for Fact Materialized Views

The script includes statements like the following to generate statistics.

```
EXECUTE dbms_stats.gather_table_stats(mv_owner, mv_name,
    degree=>dbms_stats.default_degree, estimate_percent=>
    dbms_stats.auto_sample_size, method_opt=>
    'for all columns size 1 for columns size 254 GID' , granularity=>'GLOBAL') ;
ALTER TABLE mv_name MINIMIZE RECORDS_PER_BLOCK ;
```

Using the DBMS_ODM Package

The procedures in the OLAP Data Management package, `DBMS_ODM`, generate scripts that create dimension materialized views and fact materialized views in grouping set form. You can run these scripts in their original form, modify the scripts before executing them, or use them simply as models for writing your own SQL scripts.

Important: If you choose to modify the scripts, take care to generate materialized views with the same structure as those generated by `DBMS_ODM`. Otherwise the materialized views may not be accessible to the OLAP API.

`DBMS_ODM` supports several approaches to creating the grouping set materialized view for the cube's fact table. You can choose from the following options:

- Automatically generate a materialized view that includes every level combination in the cube.

This option may potentially generate a very large materialized view, depending on the size of the fact table. In general, you should use this option only if disk space is plentiful.
- Automatically generate a materialized view that includes some of the level combinations in the cube.

This option will generate a materialized view of moderate size, depending on the size of the fact table. The summarization will be symmetric.

- Automatically generate a materialized view that includes a percentage of the level combinations in the cube.

This option may generate a materialized view of moderate size, depending on the size of the fact table and the percentage that you specify. The level combinations included in the materialized view will be random. The summarization will typically be asymmetric.

- Manually choose the level combinations to be included in the materialized view for the cube.

With this option, you can finely tune both the content and the size of the materialized view. The summarization may be symmetric or asymmetric.

Procedure: Automatically Generate the Materialized Views

Follow these steps to automatically create the materialized views for a cube:

1. Create a cube in the OLAP Catalog. You can use Enterprise Manager or you can use the CWM2 procedures. If you use the CWM2 procedures, be sure to map the cube to a star schema.
2. Configure the database to write to files. The DBMS_ODM procedures accept either a directory object to which your user ID has been granted the appropriate access, or a directory path specified by the UTL_FILE_DIR initialization parameter for the instance.
3. Log into SQL*Plus using the identity of the metadata owner.
4. Delete any materialized views that currently exist for the cube. Execute `DROP MATERIALIZED VIEW mv_name` for each materialized view you wish to delete.
5. Create scripts to generate the dimension materialized views. Execute `DBMS_ODM.CREATEDIMMV_GS` for each of the cube's dimensions.
6. Create a script to generate the fact materialized view. Execute `DBMS_ODM.CREATESTDFACTMV` and choose one of the following values for the materialization level parameter:
 - **FULL** — Fully materialize the cube's data. Include every level combination in the materialized view. See ["Example: Automatically Generate the Materialized Views for a Price Cube"](#) on page 8-7.
 - **MINIMUM** — Minimally materialize the cube's data. Include a subset of the level combinations in the materialized view. See *Oracle OLAP Reference* for more information.
 - **PERCENT** — Materialize the cube's data based on a percentage of the cube's level combinations. See *Oracle OLAP Reference* for more information.
7. Run the scripts in SQL*Plus, using commands such as the following:

```
@/users/oracle/OraHome1/olap/mvscript.sql;
```

For an example of this process, see ["Example: Automatically Generate the Materialized Views for a Price Cube"](#) on page 8-7.

Procedure: Manually Generate the Materialized Views

Follow these steps to create the materialized views with specific level combinations:

1. Follow the first five steps in "[Procedure: Automatically Generate the Materialized Views](#)" on page 8-6.
2. Use the following three step procedure to create a script to generate the fact materialized view:
 - a. Execute `DBMS_ODM.CREATEDIMLEVTUPLE` to create the table `sys.olaptablelevels`. This table lists all the dimensions of the cube and all the levels of each dimension. Edit the table to deselect any levels that you do not want to include.
 - b. Execute `DBMS_ODM.CREATECUBELEVELTUPLE` to create the table `sys.olaptableleveltuples`. This table lists all the possible combinations (grouping sets) of the levels you chose in the previous step. Edit the table to deselect any level combinations that you do not want to include.
 - c. Execute `DBMS_ODM.CREATEFACTMV_GS` to create the script.
3. Run the scripts in SQL*Plus, using commands such as the following:

```
@/users/oracle/OraHome1/olap/mvscript_fact.sql;
```

For an example of this process, see "[Example: Manually Generate the Materialized Views for a Sales Cube](#)" on page 8-8.

Example: Automatically Generate the Materialized Views for a Price Cube

This example creates materialized views for the `PRICE_CUBE` in the `GLOBAL` schema.

This cube contains unit costs and unit prices for different products over time. The dimensions are `PRODUCT`, with levels for products, families of products, classes of products, and totals, and `TIME` with levels for months, quarters, and years.

1. Generate the scripts for the dimension materialized views. The following statements create the scripts `prodmv` and `timemv` in the directory `/users/global/scripts`.
2. Run the `prodmv` and `timemv` scripts to create the dimension materialized views in the default tablespace for the `GLOBAL` schema.
3. Generate the script for the fact materialized view. The following statement creates the script `price_cost_mv` in the same script directory.

```
exec dbms_odem.CreateDimmv_gs
      ('global', 'product', 'prodmv', '/users/global/scripts');
exec dbms_odem.CreateDimmv_gs
      ('global', 'time', 'timemv', '/users/global/scripts');
```

4. Run the `price_cost_mv` script to create the fact materialized view.

The materialized view is created in the default table space for the `GLOBAL` schema. It does not use index partitioning.

The `CREATE MATERIALIZED VIEW` statement in the script specifies grouping sets for every level combination in `UNITS_CUBE`.

Example: Manually Generate the Materialized Views for a Sales Cube

This example creates materialized views for the `DRUGSTORE` cube in the `DRUG_DEPOT` schema. The cube contains sales, cost, quantity, and forecasting data. It is mapped to a fact table containing only lowest-level data and to dimension tables for `CHANNEL`, `GEOGRAPHY`, `PRODUCT`, and `TIME`. Each dimension has a single hierarchy.

1. First generate the scripts for the dimension materialized views. The following statements create the scripts `chanmv`, `prodmv`, `geogmv`, and `timemv` in `/dat1/scripts/drug_depot`.

```
EXEC DBMS_ODM.CREATEDIMMV_GS
('drug_depot', 'channel', 'chanmv', '/dat1/scripts/drug_depot');
EXEC DBMS_ODM.CREATEDIMMV_GS
('drug_depot', 'product', 'prodmv', '/dat1/scripts/drug_depot');
EXEC DBMS_ODM.CREATEDIMMV_GS
('drug_depot', 'geography', 'geogmv', '/dat1/scripts/drug_depot');
EXEC DBMS_ODM.CREATEDIMMV_GS
('drug_depot', 'time', 'timemv', '/dat1/scripts/drug_depot');
```

2. Run the scripts to create the dimension materialized views.
3. Next create the table of dimension levels for the fact materialized view.

```
EXEC DBMS_ODM.CREATEDIMLEVTUPLE('drug_depot', 'drugstore');
```

The table of levels, `SYS.OLAPTABLELEVELS`, is a temporary table specific to your session. You can view the table as follows.

```
select * from SYS.OLAPTABLELEVELS;
```

SCHEMA_NAME	DIMENSION_NAME	CUBE_NAME	LEVEL_NAME	SELECTED
DRUG_DEPOT	CHANNEL	DRUGSTORE	TOTAL	1
DRUG_DEPOT	CHANNEL	DRUGSTORE	CHANNEL_CLASS	1
DRUG_DEPOT	CHANNEL	DRUGSTORE	CHANNEL_ID	1
DRUG_DEPOT	GEOGRAPHY	DRUGSTORE	TOTAL	1
DRUG_DEPOT	GEOGRAPHY	DRUGSTORE	REGION	1
DRUG_DEPOT	GEOGRAPHY	DRUGSTORE	SUB_REGION	1
DRUG_DEPOT	GEOGRAPHY	DRUGSTORE	COUNTRY	1
DRUG_DEPOT	GEOGRAPHY	DRUGSTORE	STATE_PROVINCE	1
DRUG_DEPOT	PRODUCT	DRUGSTORE	TOTAL	1
DRUG_DEPOT	PRODUCT	DRUGSTORE	PROD_CATEGORY	1
DRUG_DEPOT	PRODUCT	DRUGSTORE	PROD_SUBCATEGORY	1
DRUG_DEPOT	PRODUCT	DRUGSTORE	ID	1
DRUG_DEPOT	TIME	DRUGSTORE	Year	1
DRUG_DEPOT	TIME	DRUGSTORE	Quarter	1
DRUG_DEPOT	TIME	DRUGSTORE	Month	1

All the levels in `SYS.OLAPTABLELEVELS` are initially selected with "1" in the `SELECTED` column.

4. Let's assume that you want to store aggregate data for each region and sub-region, across all channels and all categories of products. You do not care about data at the month level, you only want to store quarter and year data in the materialized view.

Edit `SYS.OLAPTABLELEVELS` to deselect all `CHANNEL` levels except total, the state-province level of `GEOGRAPHY`, sub-categories and individual product IDs in `PRODUCT`, and month in `TIME`.

```
update SYS.OLAPTABLELEVELS set selected = 0
  where LEVEL_NAME in ('CHANNEL_ID','CHANNEL_CLASS', 'STATE_PROVINCE',
                      'ID','PROD_SUBCATEGORY','Month');
select * from sys.olaptablelevels;
```

SCHEMA_NAME	DIMENSION_NAME	CUBE_NAME	LEVEL_NAME	SELECTED
-----	-----	-----	-----	-----
DRUG_DEPOT	CHANNEL	DRUGSTORE	TOTAL	1
DRUG_DEPOT	CHANNEL	DRUGSTORE	CHANNEL_CLASS	0
DRUG_DEPOT	CHANNEL	DRUGSTORE	CHANNEL_ID	0
DRUG_DEPOT	GEOGRAPHY	DRUGSTORE	TOTAL	1
DRUG_DEPOT	GEOGRAPHY	DRUGSTORE	REGION	1
DRUG_DEPOT	GEOGRAPHY	DRUGSTORE	SUB_REGION	1
DRUG_DEPOT	GEOGRAPHY	DRUGSTORE	COUNTRY	1
DRUG_DEPOT	GEOGRAPHY	DRUGSTORE	STATE_PROVINCE	0
DRUG_DEPOT	PRODUCT	DRUGSTORE	TOTAL	1
DRUG_DEPOT	PRODUCT	DRUGSTORE	PROD_CATEGORY	1
DRUG_DEPOT	PRODUCT	DRUGSTORE	PROD_SUBCATEGORY	0
DRUG_DEPOT	PRODUCT	DRUGSTORE	ID	0
DRUG_DEPOT	TIME	DRUGSTORE	Year	1
DRUG_DEPOT	TIME	DRUGSTORE	Quarter	1
DRUG_DEPOT	TIME	DRUGSTORE	Month	0

- Next create the table `SYS.OLAPTABLELEVELTUPLES`. This table, which is also a session-specific temporary table, contains all the possible combinations of the cube's levels. Each combination of four levels, or grouping set, has an identification number. The grouping sets that include the levels you selected in `SYS.OLAPTABLELEVELS` are marked with a 1 in the `SELECTED` column.

```
exec dbms_olap.createcubeleveltuple('drug_depot','drugstore');
select * from sys.olaptableleveltuples;
```

ID	SCHEMA_NAME	CUBE_NAME	DIMENSION_NAME	LEVEL_NAME	SELECTED
--	-----	-----	-----	-----	-----
1	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	STATE_PROVINCE	0
1	DRUG_DEPOT	DRUGSTORE	PRODUCT	ID	0
1	DRUG_DEPOT	DRUGSTORE	CHANNEL	CHANNEL_ID	0
1	DRUG_DEPOT	DRUGSTORE	TIME	Month	0
2	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	COUNTRY	0
2	DRUG_DEPOT	DRUGSTORE	PRODUCT	ID	0
2	DRUG_DEPOT	DRUGSTORE	CHANNEL	CHANNEL_ID	0
2	DRUG_DEPOT	DRUGSTORE	TIME	Month	0
.					
.					
.					
112	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	COUNTRY	1
112	DRUG_DEPOT	DRUGSTORE	PRODUCT	PROD_CATEGORY	1
112	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
112	DRUG_DEPOT	DRUGSTORE	TIME	Quarter	1
113	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	SUB_REGION	1
113	DRUG_DEPOT	DRUGSTORE	PRODUCT	PROD_CATEGORY	1
113	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
113	DRUG_DEPOT	DRUGSTORE	TIME	Quarter	1
.					
.					
.					
179	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	REGION	1

179	DRUG_DEPOT	DRUGSTORE	PRODUCT	TOTAL	1
179	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
179	DRUG_DEPOT	DRUGSTORE	TIME	Year	1
180	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	TOTAL	1
180	DRUG_DEPOT	DRUGSTORE	PRODUCT	TOTAL	1
180	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
180	DRUG_DEPOT	DRUGSTORE	TIME	Year	1

The SYS.OLAPTABLELEVELTUPLES table has 720 rows, identifying 180 unique level tuples, or grouping sets. 180 is the product of the number of levels for each of the cube's dimensions, 3*5*4*3. There are 3 levels in CHANNEL, 5 levels in GEOGRAPHY, 4 levels in PRODUCT, and 3 levels in TIME

Of the 180 grouping sets, only 16 are selected for inclusion in the materialized view. You can display the 64 selected rows (16*4) with the following statement.

```
select * from sys.olaptableleveltuples where SELECTED = 1;
```

ID	SCHEMA_NAME	CUBE_NAME	DIMENSION_NAME	LEVEL_NAME	SELECTED
---	-----	-----	-----	-----	-----
112	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	COUNTRY	1
112	DRUG_DEPOT	DRUGSTORE	PRODUCT	PROD_CATEGORY	1
112	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
112	DRUG_DEPOT	DRUGSTORE	TIME	Quarter	1
113	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	SUB_REGION	1
113	DRUG_DEPOT	DRUGSTORE	PRODUCT	PROD_CATEGORY	1
113	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
113	DRUG_DEPOT	DRUGSTORE	TIME	Quarter	1
114	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	REGION	1
114	DRUG_DEPOT	DRUGSTORE	PRODUCT	PROD_CATEGORY	1
114	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
114	DRUG_DEPOT	DRUGSTORE	TIME	Quarter	1
115	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	TOTAL	1
115	DRUG_DEPOT	DRUGSTORE	PRODUCT	PROD_CATEGORY	1
115	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
115	DRUG_DEPOT	DRUGSTORE	TIME	Quarter	1
117	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	COUNTRY	1
117	DRUG_DEPOT	DRUGSTORE	PRODUCT	TOTAL	1
117	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
117	DRUG_DEPOT	DRUGSTORE	TIME	Quarter	1
118	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	SUB_REGION	1
118	DRUG_DEPOT	DRUGSTORE	PRODUCT	TOTAL	1
118	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
118	DRUG_DEPOT	DRUGSTORE	TIME	Quarter	1
119	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	REGION	1
119	DRUG_DEPOT	DRUGSTORE	PRODUCT	TOTAL	1
119	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
119	DRUG_DEPOT	DRUGSTORE	TIME	Quarter	1
120	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	TOTAL	1
120	DRUG_DEPOT	DRUGSTORE	PRODUCT	TOTAL	1
120	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
120	DRUG_DEPOT	DRUGSTORE	TIME	Quarter	1
172	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	COUNTRY	1
172	DRUG_DEPOT	DRUGSTORE	PRODUCT	PROD_CATEGORY	1
172	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
172	DRUG_DEPOT	DRUGSTORE	TIME	Year	1
173	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	SUB_REGION	1
173	DRUG_DEPOT	DRUGSTORE	PRODUCT	PROD_CATEGORY	1
173	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
173	DRUG_DEPOT	DRUGSTORE	TIME	Year	1
174	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	REGION	1

174	DRUG_DEPOT	DRUGSTORE	PRODUCT	PROD_CATEGORY	1
174	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
174	DRUG_DEPOT	DRUGSTORE	TIME	Year	1
175	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	TOTAL	1
175	DRUG_DEPOT	DRUGSTORE	PRODUCT	PROD_CATEGORY	1
175	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
175	DRUG_DEPOT	DRUGSTORE	TIME	Year	1
177	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	COUNTRY	1
177	DRUG_DEPOT	DRUGSTORE	PRODUCT	TOTAL	1
177	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
177	DRUG_DEPOT	DRUGSTORE	TIME	Year	1
178	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	SUB_REGION	1
178	DRUG_DEPOT	DRUGSTORE	PRODUCT	TOTAL	1
178	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
178	DRUG_DEPOT	DRUGSTORE	TIME	Year	1
179	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	REGION	1
179	DRUG_DEPOT	DRUGSTORE	PRODUCT	TOTAL	1
179	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
179	DRUG_DEPOT	DRUGSTORE	TIME	Year	1
180	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	TOTAL	1
180	DRUG_DEPOT	DRUGSTORE	PRODUCT	TOTAL	1
180	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
180	DRUG_DEPOT	DRUGSTORE	TIME	Year	1

6. Suppose you want to store product totals by year for each sub-region. You do not want to store aggregates for any other grouping sets that contain the sub-region level.

Grouping sets 113, 118, 173, and 178 all use the SUB_REGION level of GEOGRAPHY.

ID	GEOGRAPHY	PRODUCT	CHANNEL	TIME
--	-----	-----	-----	-----
113	SUB_REGION	PROD_CATEGORY	TOTAL	Quarter
118	SUB_REGION	TOTAL	TOTAL	Quarter
173	SUB_REGION	PROD_CATEGORY	TOTAL	Year
178	SUB_REGION	TOTAL	TOTAL	Year

You could edit the SYS.OLAPTABLELEVELTUPLES table with a statement like the following.

```
update SYS.OLAPTABLELEVELTUPLES set selected = 0
      where ID in ('113','118', '173');
select * from sys.olaptableleveltuples where SELECTED = 1;
```

ID	SCHEMA_NAME	CUBE_NAME	DIMENSION_NAME	LEVEL_NAME	SELECTED
--	-----	-----	-----	-----	-----
112	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	COUNTRY	1
112	DRUG_DEPOT	DRUGSTORE	PRODUCT	PROD_CATEGORY	1
112	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
112	DRUG_DEPOT	DRUGSTORE	TIME	Quarter	1
114	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	REGION	1
114	DRUG_DEPOT	DRUGSTORE	PRODUCT	PROD_CATEGORY	1
114	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
114	DRUG_DEPOT	DRUGSTORE	TIME	Quarter	1
115	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	TOTAL	1
115	DRUG_DEPOT	DRUGSTORE	PRODUCT	PROD_CATEGORY	1
115	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
115	DRUG_DEPOT	DRUGSTORE	TIME	Quarter	1
117	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	COUNTRY	1
117	DRUG_DEPOT	DRUGSTORE	PRODUCT	TOTAL	1
117	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1

117	DRUG_DEPOT	DRUGSTORE	TIME	Quarter	1
119	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	REGION	1
119	DRUG_DEPOT	DRUGSTORE	PRODUCT	TOTAL	1
119	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
119	DRUG_DEPOT	DRUGSTORE	TIME	Quarter	1
120	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	TOTAL	1
120	DRUG_DEPOT	DRUGSTORE	PRODUCT	TOTAL	1
120	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
120	DRUG_DEPOT	DRUGSTORE	TIME	Quarter	1
172	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	COUNTRY	1
172	DRUG_DEPOT	DRUGSTORE	PRODUCT	PROD_CATEGORY	1
172	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
172	DRUG_DEPOT	DRUGSTORE	TIME	Year	1
174	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	REGION	1
174	DRUG_DEPOT	DRUGSTORE	PRODUCT	PROD_CATEGORY	1
174	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
174	DRUG_DEPOT	DRUGSTORE	TIME	Year	1
175	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	TOTAL	1
175	DRUG_DEPOT	DRUGSTORE	PRODUCT	PROD_CATEGORY	1
175	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
175	DRUG_DEPOT	DRUGSTORE	TIME	Year	1
177	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	COUNTRY	1
177	DRUG_DEPOT	DRUGSTORE	PRODUCT	TOTAL	1
177	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
177	DRUG_DEPOT	DRUGSTORE	TIME	Year	1
178	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	SUB_REGION	1
178	DRUG_DEPOT	DRUGSTORE	PRODUCT	TOTAL	1
178	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
178	DRUG_DEPOT	DRUGSTORE	TIME	Year	1
179	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	REGION	1
179	DRUG_DEPOT	DRUGSTORE	PRODUCT	TOTAL	1
179	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
179	DRUG_DEPOT	DRUGSTORE	TIME	Year	1
180	DRUG_DEPOT	DRUGSTORE	GEOGRAPHY	TOTAL	1
180	DRUG_DEPOT	DRUGSTORE	PRODUCT	TOTAL	1
180	DRUG_DEPOT	DRUGSTORE	CHANNEL	TOTAL	1
180	DRUG_DEPOT	DRUGSTORE	TIME	Year	1

- To create the script that will generate the fact materialized view, run the CREATEFACTMV_GS procedure.

```
exec dbms_odm.createfactmv_gs
('drug_depot','drugstore',
'drugstore_mv','/dat1/scripts/drug_depot',TRUE);
```

The CREATE MATERIALIZED VIEW statement in the script contains the following grouping sets in the GROUP BY GROUPING SETS clause.

```
(TIMES.CALENDAR_YEAR, TIMES.CALENDAR_QUARTER, CHANNELS.TOTAL,
PRODUCTS.TOTAL, PRODUCTS.PROD_CATEGORY, GEOGRAPHIES.TOTAL,
GEOGRAPHIES.REGION, GEOGRAPHIES.SUB_REGION, GEOGRAPHIES.COUNTRY ),
(TIMES.CALENDAR_YEAR, TIMES.CALENDAR_QUARTER, CHANNELS.TOTAL,
PRODUCTS.TOTAL, PRODUCTS.PROD_CATEGORY, GEOGRAPHIES.TOTAL,
GEOGRAPHIES.REGION),
(TIMES.CALENDAR_YEAR, TIMES.CALENDAR_QUARTER, CHANNELS.TOTAL,
PRODUCTS.TOTAL, PRODUCTS.PROD_CATEGORY, GEOGRAPHIES.TOTAL),
(TIMES.CALENDAR_YEAR, TIMES.CALENDAR_QUARTER, CHANNELS.TOTAL,
PRODUCTS.TOTAL, GEOGRAPHIES.TOTAL, GEOGRAPHIES.REGION,
GEOGRAPHIES.SUB_REGION, GEOGRAPHIES.COUNTRY),
(TIMES.CALENDAR_YEAR, TIMES.CALENDAR_QUARTER, CHANNELS.TOTAL,
PRODUCTS.TOTAL, GEOGRAPHIES.TOTAL, GEOGRAPHIES.REGION),
```

```

(TIMES.CALENDAR_YEAR, TIMES.CALENDAR_QUARTER, CHANNELS.TOTAL,
 PRODUCTS.TOTAL, GEOGRAPHIES.TOTAL),
(TIMES.CALENDAR_YEAR, CHANNELS.TOTAL, PRODUCTS.TOTAL,
 PRODUCTS.PROD_CATEGORY, GEOGRAPHIES.TOTAL, GEOGRAPHIES.REGION,
 GEOGRAPHIES.SUB_REGION, GEOGRAPHIES.COUNTRY),
(TIMES.CALENDAR_YEAR, CHANNELS.TOTAL, PRODUCTS.TOTAL,
 PRODUCTS.PROD_CATEGORY, GEOGRAPHIES.TOTAL, GEOGRAPHIES.REGION),
(TIMES.CALENDAR_YEAR, CHANNELS.TOTAL, PRODUCTS.TOTAL,
 PRODUCTS.PROD_CATEGORY, GEOGRAPHIES.TOTAL),
(TIMES.CALENDAR_YEAR, CHANNELS.TOTAL, PRODUCTS.TOTAL, GEOGRAPHIES.TOTAL,
 GEOGRAPHIES.REGION, GEOGRAPHIES.SUB_REGION, GEOGRAPHIES.COUNTRY),
(TIMES.CALENDAR_YEAR, CHANNELS.TOTAL, PRODUCTS.TOTAL, GEOGRAPHIES.TOTAL,
 GEOGRAPHIES.REGION, GEOGRAPHIES.SUB_REGION),
(TIMES.CALENDAR_YEAR, CHANNELS.TOTAL, PRODUCTS.TOTAL, GEOGRAPHIES.TOTAL,
 GEOGRAPHIES.REGION),
(TIMES.CALENDAR_YEAR, CHANNELS.TOTAL, PRODUCTS.TOTAL, GEOGRAPHIES.TOTAL)

```

The following statement at the end of the script sets the `MV_SUMMARY_CODE` associated with the cube in the OLAP Catalog. This setting indicates that the materialized view associated with this cube is in grouping set form.

```

execute cwm2_olap_cube.set_mv_summary_code
('DRUG_DEPOT', 'DRUGSTORE', 'GROUPINGSET') ;

```

8. Run the `drugstore_mv` script to create the fact materialized view.

Database Standard Form for Analytic Workspaces

An analytic workspace that conforms to database standard form has objects that implement a logical model for cubes, dimensions, and measures. Database standard form is a set of conventions describing the objects in an analytic workspace that can be managed by various Oracle OLAP utilities. This appendix describes database standard form conventions for users who want to add objects manually to a standard form analytic workspace. It has the following sections:

- [Overview of Database Standard Form](#)
- [Querying a Standard Form Analytic Workspace](#)
- [Object Naming Conventions](#)
- [Workspace Object Properties](#)
- [Implementation Class Objects](#)
- [Catalogs Class Objects](#)
- [Features Class Objects](#)
- [Extensions Class Objects](#)

Note: Database Standard Form 10g has a large Extensions class, which is not documented nor supported for public use. Nonetheless, these Extensions class objects are required to support OLAP tools. Customizing is thus more difficult than in Oracle9i.

Overview of Database Standard Form

Just as a relational schema can be set up in countless ways, the design of an analytic workspace can be structured in as many ways as there are application developers. However, when an application is created to run against analytic workspaces, it requires one particular design so that it can locate particular objects and identify their role within the workspace. The design for the tools available through Analytic Workspace Manager is called **database standard form**.

Analytic Workspace Manager and the current generation of tools can only be used with database standard form analytic workspaces. Database standard form (or simply, standard form) stipulates:

- Certain objects must exist in the analytic workspace. These objects and properties are used by tools in Analytic Workspace Manager that perform tasks such as aggregation, data refresh, and applications enablement. The Active Catalogs and

the DBMS_AWM PL/SQL package, described in the *Oracle OLAP Reference*, require database standard.

- Specific OLAP DML properties must be defined on these analytic workspace objects. The property values are metadata for the object, and provide information about its role in the logical model, its logical name, its relationships with other objects, and so forth. Standard form properties begin with AW\$.
- Objects must be registered in workspace catalogs. OLAP tools query these metadata catalogs to get information about how the logical cubes, measures, and dimensions are instantiated in the analytic workspace. When you define objects using the tools in Analytic Workspace Manager, the tools also maintain the catalogs. However, when you define objects manually, as described in some chapters of this guide, you must also maintain the properties and the catalogs for the tools to be aware of the new objects.

These rules impose the logical dimensional model of cubes, measures, dimensions, levels, hierarchies, and attributes on an analytic workspace.

Standard form analytic workspaces are created by all of the methods described in [Chapter 3](#). By using the Object View to browse the workspace objects, you can gain familiarity with standard form.

Terminology: Using Role Names to Identify Objects

Standard form conventions do not govern the names of workspace objects, so documentation cannot refer to the objects by name. Instead, the objects are discussed using the values of their AW\$ROLE properties as descriptors.

For example, this guide refers to the *cubedef* dimension, the *aw_names* variable, and the *default_hier* relation. These references are to the workspace objects whose AW\$ROLE property is set to CUBEDEF, AW_NAMES, and DEFAULT_HIER, respectively. The actual names of the workspace objects for most classes are typically similar, but not identical, to their roles. Roles and the AW\$ROLE property are discussed under each logical object type.

Querying a Standard Form Analytic Workspace

Standard form enables you to discover the names of logical objects and the names of the physical workspace objects that implement the logical model.

Querying the Standard Form Catalogs

You can acquire information about an analytic workspace by querying its standard form catalogs. These catalogs are implemented as dimensions, variables, relations, and valuesets in the analytic workspace. Some of these objects are in the Catalogs class, and others are in the Extensions class.

The ALL_OBJECTS dimension is a catalog that contains the names of all logical objects. ALL_OBJECTS is a concat dimension, that is, it is a concatenated list of the members of other simple dimensions. Separate dimensions for each logical object type contain the names of logical objects, for example, the ALL_HIERARCHIES dimension contains the names of all hierarchies, and the ALL_LEVELS dimension contains the names of all levels. You can query these dimensions to discover the logical model implemented by an analytic workspace.

For example, the following command displays the names of all measures in the analytic workspace.


```
REPORT W 40 all_measures
```

```
ALL_MEASURES
```

```
-----
PRICE_AND_COST_CUBE.UNIT_PRICE.MEASURE
PRICE_AND_COST_CUBE.UNIT_COST.MEASURE
UNITS_CUBE.UNITS.MEASURE
UNITS_CUBE.SALES.MEASURE
```

ALL_OBJECTS and its simple dimensions (such as ALL_LEVELS) are used in dimensional catalogs that are implemented as variables, relations, and valuesets.

Refer to ["Catalogs Class Objects"](#) on page A-19 for more information about standard form catalogs.

Querying Properties

By querying the standard form properties attached to workspace objects, you can discover the relationship between the logical model and the physical objects that implement the model.

You can query the properties on a particular object, or limit the NAME dimension to objects with particular properties or property values. The NAME dimension contains the names of all objects in an analytic workspace. By limiting the status of the NAME dimension, you can limit the scope of commands that otherwise act on all objects.

All objects have the following properties, which are described in [Table A-1](#) on page A-6.

```
AW$CLASS
AW$CREATEDBY
AW$LASTMODIFIED
AW$ROLE
```

The following commands show how you can use the AW\$ROLE property to discover the names of *measuredef* objects:

```
LIMIT name TO OBJ (PROPERTY 'AW$ROLE') EQ 'MEASUREDEF'
REPORT W 40 name
```

```
NAME
```

```
-----
PRICE_AND_COST_CUBE_UNIT_PRICE
PRICE_AND_COST_CUBE_UNIT_COST
UNITS_CUBE_UNITS
UNITS_CUBE_SALES
```

The FULLDSC command lists all the properties and their values.

```
FULLDSC units_cube_units
```

```
DEFINE UNITS_CUBE_UNITS FORMULA DECIMAL <TIME CUSTOMER PRODUCT CHANNEL>
EQ aggregate(this_aw!UNITS_CUBE_UNITS_STORED using this_aw!OBJ1176965843)
PROPERTY 'AW$CLASS' 'IMPLEMENTATION'
PROPERTY 'AW$CREATEDBY' 'AW$XML'
PROPERTY 'AW$LASTMODIFIED' '10MAY05_11:05:51'
PROPERTY 'AW$LOGICAL_NAME' 'UNITS'
PROPERTY 'AW$MEASUREDEF' NA
PROPERTY 'AW$PARENT_NAME' 'UNITS_CUBE'
PROPERTY 'AW$ROLE' 'MEASUREDEF'
PROPERTY 'AW$STATE' 'VALID_MEMBER'
```

```
PROPERTY 'COLUMN_NAME' 'MEASURE_51'  
PROPERTY 'DATA_TYPE' 'DECIMAL'  
PROPERTY 'DESCRIPTION' -  
  'LANG=AMERICAN:Units Sold-  
  LANG=FRENCH:Unités Vendues -  
  LANG=DUTCH:Verkochte Eenheden '  
PROPERTY 'DISPLAYNAME' -  
  'LANG=AMERICAN:Units Sold-  
  LANG=FRENCH:Unités Vendues -  
  LANG=DUTCH:Verkochte Eenheden '  
PROPERTY 'IS_SOLVETARGET' yes
```

Or you can use the OBJ function to get the value of a specific property:

```
SHOW OBJ(PROPERTY 'AW$PARENT_NAME', 'UNITS_CUBE_UNITS')
```

```
UNITS_CUBE
```

Standard Form Implementation of the Logical Model

The standard form logical model includes cubes, measures, and dimensions, as well as the hierarchies, levels, and attributes that are associated with dimensions. A cube is considered to be the parent of the measures that it contains, and a dimension is considered to be the parent of its hierarchies, levels, and attributes. A cube has dimensionality; that is, it is associated with its list of dimensions.

It is important to remember that this appendix describes a logical metadata model that is imposed on an analytic workspace. It does not describe the inherent relationships among workspace objects, such as the relationship between variables and formulas and their dimensions, or among dimensions in a workspace relation.

Relationships Among Logical Objects

Within the logical model of standard form are parent-child relationships among objects. Only cubes (*cubedef* objects) and dimensions (*dimdef* objects) have no parents other than the analytic workspace itself. All other objects in the logical model are descendants of these objects.

Classes of Workspace Objects

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

- **Implementation class.** Objects in this class implement the logical model. They include all the workspace objects described in the section ["Role Property Values for Implementation Class Objects"](#) on page A-7, for example the *cubedef*, *measuredef*, *dimdef*, and *hierlist* objects.
- **Catalogs class.** Objects in this class hold information about the logical model. They include a list of all the cubes in the workspace, a list of all the measures in the workspace, a list of all the dimensions in the workspace, and other lists that can facilitate the work of various utilities.
- **Features class.** Objects in this class hold information about specific objects in the logical model. For example, one object stores the descriptions of all the logical objects, while another indicates whether the object is intended to be visible to the user.

- **Extensions class.** Objects in this class are defined and maintained by the Oracle OLAP utilities. They are proprietary extensions to the standard form, and there is no commitment on the part of Oracle to maintain them from release to release.

Object Naming Conventions

There are no restrictions on the names of the workspace objects that implement a standard form logical model, other than the rules imposed by the OLAP DML. For logical objects, however, standard form imposes strict naming rules. This is because the utilities that depend on standard form reference objects by their logical names.

Standard form naming conventions for logical names are consistent with those of the Oracle Database. They establish name spaces within which logical names must be unique, and they provide rules for constructing full names to reflect the name space organization. Logical names are sometimes referred to as "simple logical names" in order to distinguish them from full names.

Logical Names

In general, the simple logical name for an object, such as a cube or dimension, conforms to the rules for a SQL simple expression, with minor differences. The rules for standard form logical names require that a name:

- Have 1 to 30 bytes.
- Cannot be an Oracle reserved word.
- Is not case-sensitive.
- Cannot contain quotation marks.
- Must begin with an alphabetic character from your database character set.
- Must contain only alphanumeric characters from your database character set and the underscore (`_`), dollar sign (`$`), and pound sign (`#`). However, Oracle strongly discourages you from using the dollar or pound sign. If your database character set contains multi byte characters, Oracle recommends that you include at least one single-byte character in each logical name.

The `AW$LOGICAL_NAME` property of a workspace object contains the simple logical name of the object that it implements. An example of a simple logical name is `PRODUCT`.

Simple Logical Names and Full Names

Because simple logical names are not unique outside their name space, standard form conventions specify a full name for each logical object. This full name includes the simple logical name, but also indicates the parent object and the role. The following is an example of a full name for an attribute whose simple name is `TIME_SPAN` and whose parent object is a logical dimension named `TIME`.

```
TIME.TIME_SPAN.ATTRIBUTE
```

The final component of a full name is the object type. In this example, it is `ATTRIBUTE`.

Full names are used in the catalog class objects that list various object types. For example, the values of the *all_dimensions*, *all_cubes*, and *all_attributes* dimensions are the full names of logical objects.

Name Space Organization

Standard form naming conventions impose an organization of logical objects that defines the following name spaces:

- **Schema.** The logical names of cubes and dimensions must be unique within a schema.
- **Cube.** The logical names of measures must be unique within a cube.
- **Dimension.** The logical names of hierarchies, levels, and attributes must be unique within a dimension. Within a given dimension, a hierarchy can have the same name as a level or attribute.

The name space organization reflects an ownership, or parent, relationship among the logical objects. For example, a measure has a cube as its parent object, and an attribute has a dimension as its parent object. The `AW$PARENT_NAME` property on workspace objects records these relationships.

Workspace Object Properties

Properties are the primary method by which logical objects are implemented by workspace objects. The properties are created on the workspace objects using the OLAP DML `PROPERTY` command.

Workspace objects in the standard form have well-defined properties that fall into three groups:

- System properties on all workspace objects.
These properties are created and given values by Oracle OLAP utilities, either `DBMS_AWM` or the utilities offered by Analytic Workspace Manager. You must never modify or delete these properties.

- Properties specific to implementation class objects.

- Role property on all workspace objects.

All objects that are in the standard form have a property called `AW$ROLE`. It indicates the role (or function) that is played by the object in the standard form.

System Properties on All Workspace Objects

All workspace objects that are part of the standard form have four system properties.

[Table A-1](#) lists the system properties and describes each one.

Table A-1 System Properties

Property	Description
<code>AW\$CLASS</code>	The class of the workspace object. Possible values are <code>IMPLEMENTATION</code> , <code>CATALOGS</code> , <code>FEATURES</code> , and <code>EXTENSIONS</code> . For a description of these classes, see " Classes of Workspace Objects " on page A-4.
<code>AW\$CREATEDBY</code>	The entity that created the workspace object. For example, if it was created by <code>DBMS_AWM</code> , then the value is <code>AW\$XML</code> .
<code>AW\$LASTMODIFIED</code>	The date and time when the workspace object was last registered.

Table A–1 (Cont.) System Properties

Property	Description
AW\$ROLE	The role (that is, function) that is performed by this object. The possible values are different for each object class. For information on property values, see "Role Property Values for Implementation Class Objects" on page A-7, "Role Property Values for Catalogs Class Objects" on page A-8, "Role Property Values for Features Class Objects" on page A-10, and "Role Property Values for Extensions Class Objects" on page A-10.
AW\$STATE	The state of the workspace object with respect to the standard form, for example, VALID_MEMBER.

Properties Specific to Implementation Class Objects

Properties for the logical name and parent name are on all implementation class objects. Three additional properties might or might not be present depending on the role of the object.

[Table A–2](#) lists the implementation class properties and describes each one.

Table A–2 Implementation Class Properties

Property	Description
AW\$LOGICAL_NAME	The simple logical name of the logical object that is implemented by this workspace object. The value is set only for objects whose role is CUBEDEF, MEASUREDEF, DIMDEF, and ATTRDEF. The property exists, but the value is NA, for all other roles in the implementation class.
AW\$PARENT_NAME	The simple logical name of the parent of the logical object that is implemented by this workspace object. The value is set for all implementation class objects except for those whose roles are CUBEDEF and DIMDEF. The value is NA for these two, because they have no parent.
AW\$TYPE	For objects with role DIMDEF and ATTRDEF, the type of the dimension or attribute. For all other roles, this property is missing. If the role is DIMDEF, this property indicates whether the dimension is a time dimension. Values are TIME or NA. If the role is ATTRDEF, this property indicates a special use for the attribute by Oracle OLAP. Values that indicate special use are END_DATE, TIME_SPAN, MEMBER_LONG_DESCRIPTION, MEMBER_SHORT_DESCRIPTION. If the value is USER or NA, then the attribute has no special meaning for Oracle OLAP.

Role Property Values for Implementation Class Objects

The AW\$ROLE property indicates the function (that is, role) that is performed by the workspace object. For implementation class objects, roles indicate fundamental building blocks of the logical model, such as cubes, measures, and dimensions.

There can be several implementation class objects that have the same role in a standard form workspace. For example, there are several objects with the role of DIMDEF because there is one such object for each dimension in the logical model.

[Table A–3](#) lists the possible values and describes each role.

Table A–3 Role Property Values: Implementation Class

Role Property Value	Role Description
CUBEDEF	Implements a cube whose logical name is in the <code>AW\$LOGICAL_NAME</code> property. For information about objects with this role, see "Cubedef Dimension" on page A-12.
MEASUREDEF	Implements a measure whose logical name is in the <code>AW\$LOGICAL_NAME</code> property. For information about objects with this role, see "Measuredef Object" on page A-13.
DIMDEF	Implements a dimension whose logical name is in the <code>AW\$LOGICAL_NAME</code> property. For information about objects with this role, see "Dimdef Dimension" on page A-15.
HIERLIST	Lists the names of the hierarchies of the dimension whose name is in the <code>AW\$PARENT_NAME</code> property. For information about objects with this role, see "Hierlist Dimension" on page A-16.
LEVELLIST	Lists the names of the levels of the dimension whose name is in the <code>AW\$PARENT_NAME</code> property. For information about objects with this role, see "Levellist Dimension" on page A-16.
MEMBER_LEVELREL	Records the level for each member of the dimension whose name is in the <code>AW\$PARENT_NAME</code> property. For information about objects with this role, see "Member_Levelrel Relation" on page A-17.
MEMBER_PARENTREL	Records the parent for each member of the dimension whose name is in the <code>AW\$PARENT_NAME</code> property. For information about objects with this role, see "Member_Parentrel Relation" on page A-17.
HIER_LEVELS	Lists the levels that are included in each hierarchy of the dimension whose name is in the <code>AW\$PARENT_NAME</code> property. For information about objects with this role, see "Hier_Levels Valueset" on page A-18.
ATTRDEF	Implements an attribute whose logical name is in the <code>AW\$LOGICAL_NAME</code> property. For information about objects with this role, see "Attrdef Object" on page A-18.

Role Property Values for Catalogs Class Objects

The `AW$ROLE` property indicates the function (or role) that is performed by the workspace object. For catalogs class objects, the objects with various roles provide information about the logical model such as a list of cubes, a list of object types, or a list of measures.

There is only one catalogs class object with a given role in a standard form workspace. For example, there is only one object that lists all the dimensions in the workspace.

[Table A–4](#) lists the possible values and describes each role.

Table A–4 Role Property Values: Catalogs Class

Role Property Value	Role Description
ALL_OBJECTS	Lists the full names of all the objects that have been registered with the standard form in this workspace. For information about the object with this role, see "ALL_OBJECTS Dimension" on page A-22.
ALL_CUBES	Lists the full names of all the cubes that have been registered with the standard form in this workspace. For information about the object with this role, see "ALL_CUBES Dimension" on page A-20.

Table A–4 (Cont.) Role Property Values: Catalogs Class

Role Property Value	Role Description
ALL_MEASURES	Lists the full names of all the measures that have been registered with the standard form in this workspace. For information about the object with this role, see " ALL_MEASURES Dimension " on page A-20.
ALL_DIMENSIONS	Lists the full names of all the dimensions that have been registered with the standard form in this workspace. For information about the object with this role, see " ALL_DIMENSIONS Dimension " on page A-20.
ALL_HIERARCHIES	Lists the full names of all the hierarchies that have been registered with the standard form in this workspace. For information about the object with this role, see " ALL_HIERARCHIES Dimension " on page A-21.
ALL_LEVELS	Lists the full names of all the levels that have been registered with the standard form in this workspace. For information about the object with this role, see " ALL_LEVELS Dimension " on page A-21.
ALL_ATTRIBUTES	Lists the full names of all the attributes that have been registered with the standard form in this workspace. For information about the object with this role, see " ALL_ATTRIBUTES Dimension " on page A-22.
ALL_DESCTYPES	Lists the types of descriptions currently supported by the standard form: SHORT, LONG, and PLURAL. For information about the object with this role, see " ALL_DESCTYPES Dimension " on page A-23.
ALL_ATTRTYPES	Lists all the attribute types that are currently supported by the standard form. These are valid values for the AW\$TYPE property of an object with the ATTRDEF role. For information about the object with the ALL_ATTRTYPES role, see " ALL_ATTRTYPES Dimension " on page A-23.
ALL_LANGUAGES	Lists the <i>language_territory</i> for the analytic workspace. For information about the object with this role, see " ALL_LANGUAGES Dimension " on page A-23.
CUBE_MEASURES	Lists the full names of the measures that belong to each cube in the workspace. For information about the object with this role, see " CUBE_MEASURES Relation " on page A-24.
DIM_HIERARCHIES	Lists the full names of the hierarchies that belong to each dimension in the workspace. For information about the object with this role, see " DIM_HIERARCHIES Relation " on page A-24.
DIM_LEVELS	Lists the full names of the levels that belong to each dimension in the workspace. For information about the object with this role, see " DIM_LEVELS Relation " on page A-25.
DIM_ATTRIBUTES	Lists the full names of the attributes that belong to each dimension in the workspace. For information about the object with this role, see " DIM_ATTRIBUTES Relation " on page A-25.
AW_NAMES	Records the name of the workspace object that implements each logical cube, measure, dimension, and attribute. For other logical objects, there is no single corresponding workspace object, so the value is NA. For information about the object with this role, see " AW_NAMES Variable " on page A-26.

Role Property Values for Features Class Objects

The `AW$ROLE` property indicates the function (or role) that is performed by the workspace object. For features class objects, roles provide various types of supplementary data for logical objects such as descriptions.

For many roles, there is a single features class object in a standard form workspace. However, for the roles that have `MEMBER` in their names, there is one object for each dimension.

[Table A-5](#) lists the possible values and describes each role that applies to features class objects.

Table A-5 Role Property Values: Features Class

Role Property Value	Role Description
<code>ALL_DESCRIPTIONS</code>	Records short, long, and plural descriptions for all objects. For information about the object with this role, see "ALL_DESCRIPTIONS Variable" on page A-27.
<code>DEFAULT_HIER</code>	Records the full name of the default hierarchy for each dimension. For information about the object with this role, see "DEFAULT_HIER Relation" on page A-27.
<code>MEMBER_CREATEDBY</code>	Records the entity that created each member of a given dimension. For information about the object with this role, see "Member_Createdby Variable" on page A-29.
<code>MEMBER_FAMILYREL</code>	Records the family relation for each hierarchy of a given dimension. For information about the object with this role, see "Member_Familyrel Relation" on page A-29.
<code>MEMBER_GID</code>	Records the grouping id for each hierarchy of a given dimension. For information about the object with this role, see "Member_Gid Variable" on page A-29.
<code>MEMBER_INHIER</code>	Indicates whether a given member of a dimension is in a given hierarchy. For information about the object with this role, see "Member_Inhier Valueset" on page A-28.
<code>OBJ_CREATEDBY</code>	Records the entity that created each object. For information about the object with this role, see "OBJ_CREATEDBY Variable" on page A-30.
<code>VERSION</code>	Records the number of the standard form version under which the workspace is being managed. For information about the object with this role, see "VERSION Variable" on page A-30.
<code>VISIBLE</code>	Indicates whether a given object should be made visible to the user by Oracle OLAP enabling utilities. For information about the object with this role, see "VISIBLE Variable" on page A-28.

Role Property Values for Extensions Class Objects

The `AW$ROLE` property indicates the function (or role) that is performed by the workspace object. For Extensions class objects, roles are for internal use of Oracle OLAP utilities such as `DBMS_AWM` and the enablers.

DBAs and users must not create, modify, or depend on objects that are in the Extensions class. The `AW$ROLE` property, and all properties, for objects in this class are for proprietary use only. Oracle makes no commitment to maintain the roles and relationships of these objects.

Implementation Class Objects

The objects in the implementation class provide the implementation for the logical objects in a given workspace. In general, they hold the data that users see as dimensions and measures. Implementation class objects differ from workspace to workspace. For example, one workspace might have measures called SALES and COST, while another workspace might have measures called BUDGET and ACTUAL.

The *cubedef*, *measuredef*, and *dimdef* objects implement cubes, measures, and dimensions respectively. In addition, each of these objects have implementation class helper objects. An overview of the objects is provided in the section "[Standard Form Implementation of the Logical Model](#)" on page A-4.

The rest of this section describes each of the implementation class objects. Note that the examples in this section show the properties required by the standard form. If you examine a workspace that was created by Analytic Workspace Manager or the DBMS_AWM package, you might find some additional properties on various objects. These are not required for compliance with the standard form.

For information about the values that should be assigned to the properties, see [Table A-1](#) and [Table A-2](#).

To list all the objects that have a given role, limit the NAME dimension to all the objects that have that role and then report the values of the NAME dimension. For example, execute the following OLAP DML commands to list all the *cubedef* objects.

```
LIMIT name TO OBJ (PROPERTY 'AW$ROLE') EQ 'CUBEDEF'
REPORT W 20 name
```

```
NAME
-----
PRICE_AND_COST_CUBE
UNITS_CUBE
```

Be sure to reset NAME afterward:

```
LIMIT name TO ALL
```

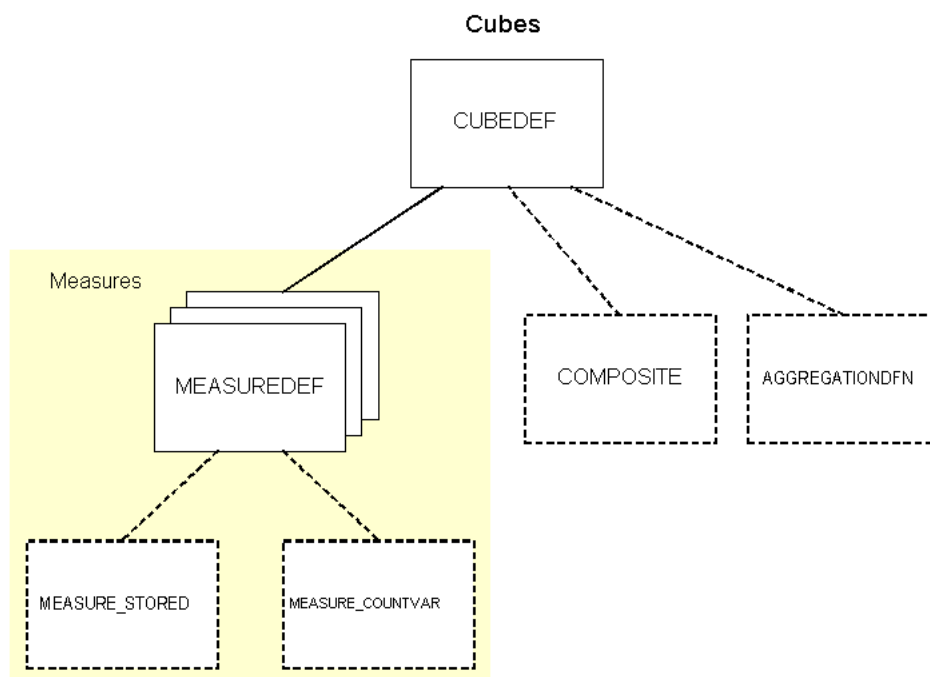
Cube Objects

A cube is implemented by a *cubedef* dimension. It is owned by the analytic workspace; it has no parent objects.

A *cubedef* dimension is the parent of one or more *measuredef* objects, and is typically the parent of two Extensions class objects:

- *composite*. A composite that dimensions the *measure_stored* variable.
- *aggregationdfn*. An aggmap that stores the aggregation rules for the cube.

[Figure A-1](#) shows the relationships among the primary objects that compose a cube.

Figure A–1 Parent-Child Relationships in a Cube

Cubedef Dimension

A logical cube is implemented by a workspace dimension that has the value `CUBEDEF` in its `AW$ROLE` property. The values of a given *cubedef* dimension are the names of the logical dimensions of the cube.

A *cubedef* dimension has no parent, so its `AW$PARENT_NAME` property is set to `NA`. A logical cube is the parent of the measures that belong to it.

The following is a full description of a *cubedef* dimension called `UNITS_CUBE`.

```
FULLDSC units_cube
```

```

DEFINE UNITS_CUBE DIMENSION TEXT
PROPERTY 'ALLDIMENSIONS' -
  'TIME-
  CUSTOMER-
  PRODUCT-
  CHANNEL'
PROPERTY 'AW$CLASS' 'IMPLEMENTATION'
PROPERTY 'AW$CREATEDBY' 'AW$XML'
PROPERTY 'AW$CUBEDEF' NA
PROPERTY 'AW$LASTMODIFIED' '10MAY05_11:05:51'
PROPERTY 'AW$LOGICAL_NAME' 'UNITS_CUBE'
PROPERTY 'AW$ROLE' 'CUBEDEF'
PROPERTY 'AW$STATE' 'VALID_MEMBER'
PROPERTY 'DENSEDIMENSIONS' -
  'TIME-
  CUSTOMER-
  PRODUCT-
  CHANNEL'
PROPERTY 'DESCRIPTION' -
  'LANG=AMERICAN:Units Cube-
  LANG=FRENCH:Cube en Unités -
  LANG=DUTCH:Kubus van Eenheden '
  
```

```
PROPERTY 'DISPLAYNAME' -
  'LANG=AMERICAN:Units Cube-
  LANG=FRENCH:Cube en Unités -
  LANG=DUTCH:Kubus van Eenheden '
```

The following report shows the values of the UNITS_CUBE dimension. The values are the names of the *dimdef* dimensions that implement the cube's logical dimensions.

```
REPORT units_cube

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

Measure Objects

A measure is implemented by a *measuredef* object. Every measure has one *cubedef* as its parent.

A *measuredef* object is typically the parent of two Extensions class objects:

- *measure_countvar*. A variable used by some aggregation operators.
- *measure_stored*. A variable used to store the data for the measure.

Measuredef Object

A logical measure is implemented by a workspace object that has the value MEASUREDEF in its AW\$ROLE property. The *measuredef* object can be a variable, formula, or relation.

The values of the *measuredef* object are the values of the logical measure, and its parent is the logical cube.

The following is a full description of a *measuredef* object for the logical measure called UNITS. The object is a formula that is dimensioned by the dimensions of the parent cube, which is called UNITS_CUBE. The formula calculates fully solved data that is stored in a variable named UNITS_CUBE_UNITS_STORED.

```
FULLDSC units_cube_units

DEFINE UNITS_CUBE_UNITS FORMULA DECIMAL <TIME CUSTOMER PRODUCT CHANNEL>
EQ aggregate(this_aw!UNITS_CUBE_UNITS_STORED using this_aw!OBJ1176965843)
PROPERTY 'AW$CLASS' 'IMPLEMENTATION'
PROPERTY 'AW$CREATEDBY' 'AW$XML'
PROPERTY 'AW$LASTMODIFIED' '10MAY05_11:05:51'
PROPERTY 'AW$LOGICAL_NAME' 'UNITS'
PROPERTY 'AW$MEASUREDEF' NA
PROPERTY 'AW$PARENT_NAME' 'UNITS_CUBE'
PROPERTY 'AW$ROLE' 'MEASUREDEF'
PROPERTY 'AW$STATE' 'VALID_MEMBER'
PROPERTY 'COLUMN_NAME' 'MEASURE_51'
PROPERTY 'DATA_TYPE' 'DECIMAL'
PROPERTY 'DESCRIPTION' -
  'LANG=AMERICAN:Units Sold-
  LANG=FRENCH:Unités Vendues -
  LANG=DUTCH:Verkochte Eenheden '
PROPERTY 'DISPLAYNAME' -
  'LANG=AMERICAN:Units Sold-
```

```

LANG=FRENCH:Unités Vendues -
LANG=DUTCH:Verkochte Eenheden '
PROPERTY 'IS_SOLVETARGET' yes

```

Dimension Objects

A dimension is implemented by a *dimdef* object. The *dimdef* object is the parent of one each of the following supporting objects:

- *hierlist* dimension
- *levellist* dimension
- *member_levelrel* relation
- *member_parentrel* relation
- *hier_levels* valueset

For each of these objects, its *AW\$ROLE* property records the object's function. For example, the *AW\$ROLE* property of a *hierlist* dimension is set to *HIERLIST*. In addition, the *AW\$PARENT* property for each of these objects contains the name of the logical dimension to which the object belongs. If a dimension does not have a hierarchy, or it does not have levels, or it has neither, then these supporting objects exist but they are not populated.

Optionally, a *dimdef* object can have one or more *attrdef* objects as its children.

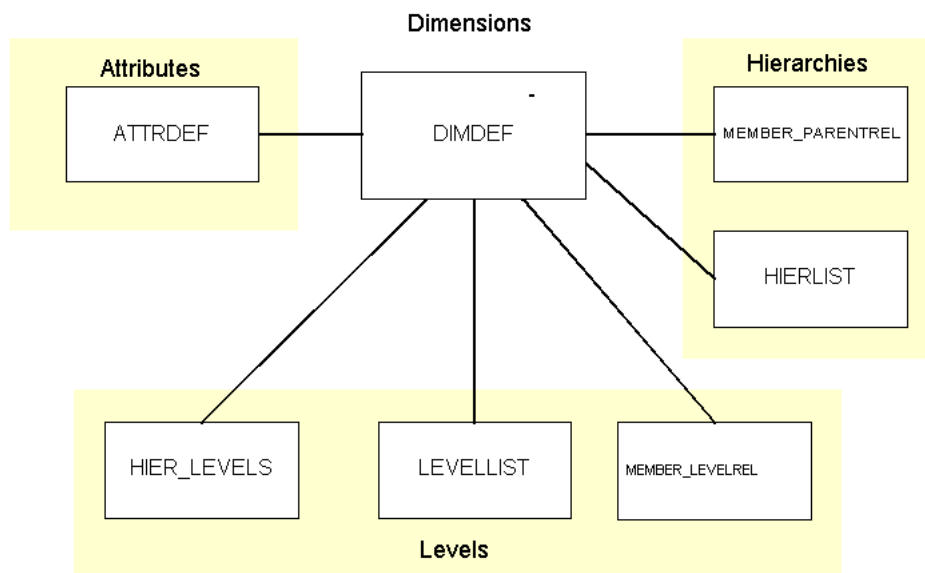
For enablement for OracleBI Beans, a *dimdef* object requires one each of these Features class objects as its children:

- *member_inhier*
- *member_familyrel*
- *member_gid*

A data refresh uses the Features class *member_createdby* object.

Figure A-2 shows the relationships among the primary objects that compose dimensions.

Figure A-2 Parent-Child Relationships in a Dimension



Dimdef Dimension

A logical dimension is implemented by a workspace dimension that has the value DIMDEF in its AW\$ROLE property. The values of a given *dimdef* dimension are the values of the logical dimension.

A *dimdef* dimension has no parent, so its AW\$PARENT_NAME property is set to NA. The AW\$TYPE property is set to TIME for time dimensions, and it is set to NA for all other dimensions.

The following is a full description of a *dimdef* dimension for the logical dimension called TIME.

FULLDSC time

```

DEFINE TIME DIMENSION TEXT
PROPERTY 'AW$CLASS' 'IMPLEMENTATION'
PROPERTY 'AW$CREATEDBY' 'AW$XML'
PROPERTY 'AW$DIMDEF' NA
PROPERTY 'AW$LASTMODIFIED' '10MAY05_11:05:29'
PROPERTY 'AW$LOGICAL_NAME' 'TIME'
PROPERTY 'AW$ROLE' 'DIMDEF'
PROPERTY 'AW$STATE' 'VALID_MEMBER'
PROPERTY 'AW$TYPE' 'TIME'
PROPERTY 'COLUMN_NAME_ET' 'ET_COL_25'
PROPERTY 'COLUMN_NAME_GID' 'GID_COL_27'
PROPERTY 'COLUMN_NAME_PRNTET' 'PET_COL_26'
PROPERTY 'COLUMN_NAME_PRNTGID' 'PGID_COL_28'
PROPERTY 'DATA_TYPE' 'TEXT'
PROPERTY 'DEFAULT_HIERARCHY' 'CALENDAR_YEAR'
PROPERTY 'DESCRIPTION' -
  'LANG=AMERICAN:Time-
  LANG=FRENCH:Temps -
  LANG=DUTCH:Tijd '
PROPERTY 'DISPLAYNAME' -
  'LANG=AMERICAN:Time-
  LANG=FRENCH:Temps -
  LANG=DUTCH:Tijd '
PROPERTY 'PLURAL_DESCRIPTION' -
  'LANG=AMERICAN:Time-
  LANG=FRENCH:Temps -
  LANG=DUTCH:Tijd '
PROPERTY 'SORT_ATTRIBUTE' 'END_DATE'

```

The following report shows sample values of this *dimdef* dimension from all the levels. This is an **embedded totals** dimension. In this example, the use of surrogate keys ensures uniqueness among the values from all levels. When surrogate keys are not used, another strategy must be used to insure uniqueness. For example, you can use the level as a prefix, such as QUARTER.142 and YEAR.145. The example includes an *attrdef* variable and *member_levelrel* relation to describe the selected dimension members.

```

LIMIT time TO '131'
LIMIT time ADD ANCESTORS USING time_parentrel
REPORT DOWN time W 25 <time_long_description time_levelrel>

```

-----ALL_LANGUAGES-----		
-----AMERICAN-----		
TIME	TIME_LONG_DESCRIPTION	TIME_LEVELREL
131	May-05	MONTH
142	Q2-05	QUARTER
145	2005	YEAR

Hierlist Dimension

A *hierlist* dimension lists the names of the hierarchies of its parent dimension. That is, the values of the *hierlist* dimension are the names of hierarchies, such as the CALENDAR and FISCAL hierarchies for a time dimension. The hierarchies do not have one-to-one implementations as workspace objects, so the names refer to logical hierarchies, not to workspace objects.

The following is a full description of a *hierlist* dimension called TIME_HIERLIST.

```
FULLDSC time_hierlist
```

```
DEFINE TIME_HIERLIST DIMENSION TEXT
PROPERTY 'AW$CLASS' 'IMPLEMENTATION'
PROPERTY 'AW$CREATEDBY' 'AW$XML'
PROPERTY 'AW$HIERLIST' NA
PROPERTY 'AW$LASTMODIFIED' '18MAR04_10:40:51'
PROPERTY 'AW$PARENT_NAME' 'TIME'
PROPERTY 'AW$ROLE' 'HIERLIST'
```

The following report shows the values of this *hierlist* dimension. TIME has one hierarchy, which is named CALENDAR_YEAR.

```
REPORT time_hierlist
```

```
TIME_HIERLIST
-----
CALENDAR_YEAR
```

Levellist Dimension

A *levellist* dimension lists the names of the levels of its parent dimension. That is, the values of the *levellist* dimension are the names of levels, such as the CITY, STATE, and COUNTRY levels for a geography dimension. The levels do not have one-to-one implementations as workspace objects, so the names refer to logical levels, not to workspace objects. The logical level for each dimension value is identified in the dimension's MEMBER_LEVELREL relation.

The following is a full description of a *levellist* dimension called TIME_LEVELLIST.

```
FULLDSC time_levellist
```

```
DEFINE TIME_HIERLIST DIMENSION TEXT
PROPERTY 'AW$CLASS' 'IMPLEMENTATION'
PROPERTY 'AW$CREATEDBY' 'AW$XML'
PROPERTY 'AW$LASTMODIFIED' '18MAR04_10:40:51'
PROPERTY 'AW$LEVELLIST' NA
PROPERTY 'AW$LEVEL_MONTH' 'MONTH'
PROPERTY 'AW$LEVEL_QUARTER' 'QUARTER'
PROPERTY 'AW$LEVEL_YEAR' 'YEAR'
PROPERTY 'AW$PARENT_NAME' 'TIME'
PROPERTY 'AW$ROLE' 'LEVELLIST'
```

The following report shows the values of this *levellist* dimension.

```
REPORT time_levellist
```

```
TIME_LEVELLIST
-----
YEAR
QUARTER
MONTH
```

Member_Levelrel Relation

A *member_levelrel* relation records the level for each value of the relation's parent dimension. For example, for a geography dimension, the *member_levelrel* relation might record the fact that BOSTON belongs to the CITY level and IOWA belongs to the STATE level.

The following is a full description of a *member_levelrel* relation called TIME_LEVELREL.

```
FULLDSC time_levelrel

DEFINE TIME_LEVELREL RELATION TIME_LEVELLIST <TIME>
PROPERTY 'AW$CLASS' 'IMPLEMENTATION'
PROPERTY 'AW$CREATEDBY' 'AW$XML'
PROPERTY 'AW$LASTMODIFIED' '03SEP03_15:27:47'
PROPERTY 'AW$PARENT_NAME' 'TIME'
PROPERTY 'AW$ROLE' 'MEMBER_LEVELREL'
```

The following report shows sample values of a *member_levelrel* relation. The levels are MONTH, QUARTER, and YEAR.

```
LIMIT time TO '75'
LIMIT time ADD ANCESTORS USING time_parentrel
REPORT DOWN time W 15 time_levelrel
```

```
TIME          TIME_LEVELREL
-----
75            MONTH
83            QUARTER
85            YEAR
```

Member_Parentrel Relation

A *member_parentrel* relation records the parent dimension value for each value of the relation's parent dimension. For example, for a geography dimension, the *member_parentrel* relation might record the fact that the parent of BOSTON is MASSACHUSETTS, and the parent of MASSACHUSETTS is USA.

The following is a full description of a *member_parentrel* relation called TIME_PARENTREL.

```
FULLDSC time_parentrel

DEFINE TIME_PARENTREL RELATION TIME <TIME TIME_HIERLIST>
PROPERTY 'AW$CLASS' 'IMPLEMENTATION'
PROPERTY 'AW$CREATEDBY' 'AW$XML'
PROPERTY 'AW$LASTMODIFIED' '03SEP03_15:27:47'
PROPERTY 'AW$MEMBER_PARENTREL' NA
PROPERTY 'AW$PARENT_NAME' 'TIME'
PROPERTY 'AW$ROLE' 'MEMBER_PARENTREL'
```

The following report shows the values of a *member_parentrel* relation. The parent of a given value can be different, depending on which hierarchy is being considered.

```
REPORT DOWN time W 20 time_parentrel

          ---TIME_PARENTREL---
          ---TIME_HIERLIST---
TIME      CALENDAR
-----
75        83
83        85
85        NA
```

Hier_Levels Valueset

A *hier_levels* valueset lists the levels that are included in each hierarchy of the parent dimension.

The following is a full description of a *hier_levels* valueset called TIME_HIER_LEVELS.

```
FULLDSC time_hier_levels
```

```
DEFINE TIME_HIER_LEVELS VALUESET TIME_LEVELLIST <TIME_HIERLIST>
LD IMPLEMENTATION Ordered from Bottom to Top list of levels in a hierarchy for
TIME
PROPERTY 'AW$CLASS' 'IMPLEMENTATION'
PROPERTY 'AW$CREATEDBY' 'AW$XML'
PROPERTY 'AW$LASTMODIFIED' '03SEP03_15:27:47'
PROPERTY 'AW$PARENT_NAME' 'TIME'
PROPERTY 'AW$ROLE' 'HIER_LEVELS'
```

The following report shows the list of levels for each hierarchy in the TIME dimension.

```
REPORT W 25 VALUES(time_hier_levels)

TIME_HIERLIST  VALUES(TIME_HIER_LEVELS)
-----
CALENDAR_YEAR  MONTH
                QUARTER
                YEAR
```

Attrdef Object

A logical attribute is implemented by a workspace object that has the value *attrdef* in its AW\$ROLE property. The *attrdef* object can be a variable, formula, or relation. The values of the *attrdef* object are the values of the logical attribute, and its parent is the logical dimension to which it belongs.

The AW\$TYPE property indicates whether Oracle OLAP has a special use for the attribute. Property values that indicate such a special use are DEFAULT_ORDER, END_DATE, TIME_SPAN, MEMBER_LONG_DESCRIPTION, MEMBER_SHORT_DESCRIPTION, and MEMBER_VISIBLE. If the value is USER or NA, then the attribute has no special meaning for Oracle OLAP.

An *attrdef* object must be dimensioned by its parent *dimdef* dimension. In addition, it can be dimensioned by the *hierlist* dimension or the ALL_LANGUAGES dimension, or both.

The following is a full description of an *attrdef* object called TIME_LONG_DESCRIPTION. This long description attribute is implemented as a variable.

```
DEFINE TIME_LONG_DESCRIPTION VARIABLE TEXT <TIME ALL_LANGUAGES>
PROPERTY '$NATRIGGER' 'if this_aw!ALL_LANGUAGES eq \'AMERICAN\' then NA
    else this_aw!TIME_LONG_DESCRIPTION(this_aw!ALL_LANGUAGES \'AMERICAN\')'
PROPERTY 'AW$ATTRDEF' NA
```



```

PROPERTY 'AW$CLASS' 'IMPLEMENTATION'
PROPERTY 'AW$CREATEDBY' 'AW$XML'
PROPERTY 'AW$LASTMODIFIED' '10MAY05_11:05:30'
PROPERTY 'AW$LNG_ATTRIBUTE' 'yes'
PROPERTY 'AW$LOGICAL_NAME' 'LONG_DESCRIPTION'
PROPERTY 'AW$PARENT_NAME' 'TIME'
PROPERTY 'AW$ROLE' 'ATTRDEF'
PROPERTY 'AW$STATE' 'VALID_MEMBER'
PROPERTY 'AW$TYPE' 'MEMBER_LONG_DESCRIPTION'
PROPERTY 'COLUMN_NAME' 'ATTRIBUTE_29'
PROPERTY 'DATA_TYPE' 'TEXT'
PROPERTY 'DESCRIPTION' -
    'LANG=AMERICAN:Long Description-
    LANG=FRENCH:Description Longue -
    LANG=DUTCH:Lange Beschrijving '
PROPERTY 'DISPLAYNAME' -
    'LANG=AMERICAN:Long Description-
    LANG=FRENCH:Description Longue -
    LANG=DUTCH:Lange Beschrijving '

```

The following is a report that shows selected values of this *attrdef* object at each level.

```

LIMIT time TO time_levelrel EQ 'YEAR'
LIMIT time KEEP LAST 1
LIMIT time ADD DESCENDANTS USING time_parentrel
REPORT DOWN time W 25 time_long_description

```

```

--TIME_LONG_DESCRIPTION--
-----ALL_LANGUAGES-----
TIME                AMERICAN
-----
145                2005
141                Q1-05
142                Q2-05
143                Q3-05
144                Q4-05
127                Jan-05
128                Feb-05
129                Mar-05
.
.
.

```

Catalogs Class Objects

Catalogs class objects hold information about the logical objects in the workspace. Catalog class objects include a list of all the cubes in the workspace, a list of all the measures in the workspace, a list of all the dimensions in the workspace, and other lists that can facilitate the work of various utilities. A given workspace has a single instance of each Catalog class object. DBMS_AWM creates these objects using the role as the name, so that the *all_languages* dimension is named ALL_LANGUAGES. For this reason, the names of objects in the CATALOGS class are shown here in capital letters to indicate actual names.

In this section, Catalogs class objects are discussed in the following groups:

- [Lists of Objects](#)
- [Lists of Types and Languages](#)

- [Lists of Cube and Dimension Objects](#)
- [Supporting Object Information](#)

Lists of Objects

The Catalogs class includes a set of dimensions, each of which lists all the objects of a given kind. For example, the ALL_MEASURES dimension lists all the logical measures.

ALL_CUBES Dimension

The ALL_CUBES dimension lists the full names of all the logical cubes in the workspace. The following is a full description of an ALL_CUBES dimension.

```
FULLDSC all_cubes

DEFINE ALL_CUBES DIMENSION TEXT
LD CATALOGS List of all cubes in the aw
PROPERTY 'AW$CLASS' 'CATALOGS'
PROPERTY 'AW$CREATEDBY' 'AW$XML'
PROPERTY 'AW$LASTMODIFIED' '04DEC02_13:09:14'
PROPERTY 'AW$ROLE' 'ALL_CUBES'
```

The following report shows the values of this ALL_CUBES dimension.

```
REPORT W 20 all_cubes

ALL_CUBES
-----
PRICE_CUBE.CUBE
UNITS_CUBE.CUBE
```

ALL_MEASURES Dimension

The ALL_MEASURES dimension lists the full names of all the logical measures in the workspace.

A full description for this dimension is similar to those presented for the ALL_CUBES dimension in "[ALL_CUBES Dimension](#)" on page A-20. The following report shows the values of an ALL_MEASURES dimension.

```
REPORT W 40 all_measures

ALL_MEASURES
-----
PRICE_AND_COST_CUBE.UNIT_PRICE.MEASURE
PRICE_AND_COST_CUBE.UNIT_COST.MEASURE
UNITS_CUBE.UNITS.MEASURE
UNITS_CUBE.SALES.MEASURE
```

ALL_DIMENSIONS Dimension

The ALL_DIMENSIONS dimension lists the full names of all the logical dimensions in the workspace.

A full description for this dimension is similar to those presented for the ALL_CUBES dimension in "[ALL_CUBES Dimension](#)" on page A-20. The following report shows the values of an ALL_DIMENSIONS dimension.

```
REPORT W 20 all_dimensions

ALL_DIMENSIONS
```

```

-----
PRODUCT.DIMENSION
TIME.DIMENSION
CHANNEL.DIMENSION
CUSTOMER.DIMENSION

```

ALL_HIERARCHIES Dimension

The ALL_HIERARCHIES dimension lists the full names of all the hierarchies in the workspace.

A full description for this dimension is similar to those presented for the ALL_CUBES dimension in "[ALL_CUBES Dimension](#)" on page A-20. The following report shows the values of an ALL_HIERARCHIES dimension.

```
REPORT W 35 all_hierarchies
```

```

ALL_HIERARCHIES
-----
CUSTOMER.AW$NONE.HIERARCHY
CUSTOMER.SHIPMENTS.HIERARCHY
CUSTOMER.MARKET_SEGMENT.HIERARCHY
PRODUCT.AW$NONE.HIERARCHY
PRODUCT.PRIMARY.HIERARCHY
TIME.AW$NONE.HIERARCHY
TIME.CALENDAR_YEAR.HIERARCHY
CHANNEL.AW$NONE.HIERARCHY
CHANNEL.PRIMARY.HIERARCHY

```

Hierarchies with a simple name of AW\$NONE indicate that a dimension has no hierarchy.

ALL_LEVELS Dimension

The ALL_LEVELS dimension lists the full names of all the levels in the workspace.

A full description for this dimension is similar to those presented for the ALL_CUBES dimension in "[ALL_CUBES Dimension](#)" on page A-20. The following report shows the values of an ALL_LEVELS dimension.

```
REPORT W 30 all_levels
```

```

ALL_LEVELS
-----
CUSTOMER.AW$NONE.LEVEL
CUSTOMER.TOTAL_CUSTOMER.LEVEL
CUSTOMER.REGION.LEVEL
CUSTOMER.WAREHOUSE.LEVEL
CUSTOMER.TOTAL_MARKET.LEVEL
CUSTOMER.MARKET_SEGMENT.LEVEL
CUSTOMER.ACCOUNT.LEVEL
CUSTOMER.SHIP_TO.LEVEL
PRODUCT.AW$NONE.LEVEL
TIME.AW$NONE.LEVEL
TIME.YEAR.LEVEL
TIME.QUARTER.LEVEL

```

```

.
.
.

```

ALL_ATTRIBUTES Dimension

The ALL_ATTRIBUTES dimension lists the full names of all the attributes in the workspace.

A full description for this dimension is similar to those presented for the ALL_CUBES dimension in "[ALL_CUBES Dimension](#)" on page A-20. The following report shows the values of an ALL_ATTRIBUTES dimension.

```
REPORT W 40 all_attributes

ALL_ATTRIBUTES
-----
CUSTOMER.LONG_DESCRIPTION.ATTRIBUTE
CUSTOMER.SHORT_DESCRIPTION.ATTRIBUTE
PRODUCT.SHORT_DESCRIPTION.ATTRIBUTE
PRODUCT.PACKAGE.ATTRIBUTE
PRODUCT.BUYER.ATTRIBUTE
PRODUCT.MARKETING_MANAGER.ATTRIBUTE
PRODUCT.LONG_DESCRIPTION.ATTRIBUTE
TIME.END_DATE.ATTRIBUTE
TIME.TIME_SPAN.ATTRIBUTE
TIME.LONG_DESCRIPTION.ATTRIBUTE
.
.
.
```

ALL_OBJECTS Dimension

The ALL_OBJECTS dimension lists the full names of all the logical objects in the workspace.

The following is a full description of an ALL_OBJECTS dimension.

```
FULLDSC all_objects

DEFINE ALL_OBJECTS DIMENSION CONCAT (ALL_DIMENSIONS ALL_CUBES ALL_MEASURES
  ALL_ATTRIBUTES ALL_HIERARCHIES ALL_LEVELS -
  ALL_SOLVES ALL_SOLVEDFNS ALL_SOLVEGROUPS ALL_MODELS ALL_MEASUREFOLDERS) UNIQUE
PROPERTY 'AW$CLASS' 'CATALOGS'
PROPERTY 'AW$CREATEDBY' 'AW$XML'
PROPERTY 'AW$LASTMODIFIED' '10MAY05_11:04:57'
PROPERTY 'AW$ROLE' 'ALL_OBJECTS'
PROPERTY 'LAST_COLUMN_ID' 78
```

ALL_OBJECTS is a concat dimension of the ALL_CUBES, ALL_MEASURES, ALL_HIERARCHIES, ALL_LEVELS, and ALL_ATTRIBUTES dimensions from the Catalogs class. It also includes dimensions from the Extensions class. Its dimension members are a concatenated list of the members of those dimensions, as shown by this example.

```
LIMIT all_cubes TO FIRST 2
LIMIT all_measures TO FIRST 2
LIMIT all_hierarchies TO FIRST 2
LIMIT all_levels TO FIRST 2
LIMIT all_attributes TO FIRST 2
LIMIT all_objects TO all_cubes
LIMIT all_objects ADD all_measures
LIMIT all_objects ADD all_hierarchies
LIMIT all_objects ADD all_levels
LIMIT all_objects ADD all_attributes
REPORT W 40 all_objects
```

```

ALL_OBJECTS
-----
PRICE_AND_COST_CUBE.CUBE
UNITS_CUBE.CUBE
PRICE_AND_COST_CUBE.UNIT_PRICE.MEASURE
PRICE_AND_COST_CUBE.UNIT_COST.MEASURE
CUSTOMER.AW$NONE.HIERARCHY
CUSTOMER.SHIPMENTS.HIERARCHY
CUSTOMER.AW$NONE.LEVEL
CUSTOMER.TOTAL_CUSTOMER.LEVEL
CUSTOMER.LONG_DESCRIPTION.ATTRIBUTE
CUSTOMER.SHORT_DESCRIPTION.ATTRIBUTE

```

Lists of Types and Languages

The Catalogs class includes dimensions that list types and languages that are supported by the current version of the standard form.

ALL_DESCTYPES Dimension

The ALL_DESCTYPES dimension lists all the description types that are recognized in the current version of the standard form. The following report lists the types.

```
REPORT all_desctypes
```

```

ALL_DESCTYPES
-----
SHORT
LONG
PLURAL

```

ALL_ATTRTYPES Dimension

The ALL_ATTRTYPES dimension lists all the attribute types that are recognized in the current version of standard form. The following report lists the types.

```
REPORT W 40 all_attrtypes
```

```

ALL_ATTRTYPES
-----
END_DATE
TIME_SPAN
MEMBER_LONG_DESCRIPTION
MEMBER_SHORT_DESCRIPTION
USER

```

ALL_LANGUAGES Dimension

The ALL_LANGUAGES dimension lists the language that is implemented in the current analytic workspace. ALL_LANGUAGES by default contains the value of the database language. You can add support for any number of additional languages, as allowed by the database character set.

Your ability to change the status of ALL_LANGUAGES is controlled by the LOCK_LANGUAGE_DIMS option, which must set to NO for the status to change. By default, it is set to YES.

```

LOCK_LANGUAGE_DIMS=NO
LIMIT all_languages TO ALL
REPORT all_languages

```

```
ALL_LANGUAGES
-----
AMERICAN
FRENCH
DUTCH

LIMIT all_languages TO 1
LOCK_LANGUAGE_DIMS=YES
```

Lists of Cube and Dimension Objects

The Catalogs class includes relations that indicate the parent-child relationships among various logical objects. These lists are specific to a given workspace.

CUBE_MEASURES Relation

The CUBE_MEASURES relation identifies the cube to which each measure belongs. The values of the relation must be listed in the ALL_CUBES dimension. The following is a full description of a CUBE_MEASURES relation in a sample analytic workspace.

```
FULLDSC cube_measures

DEFINE CUBE_MEASURES RELATION ALL_CUBES <ALL_MEASURES>
PROPERTY 'AW$CLASS' 'CATALOGS'
PROPERTY 'AW$CREATEDBY' 'AW$XML'
PROPERTY 'AW$LASTMODIFIED' '03SEP03_15:27:47'
PROPERTY 'AW$ROLE' 'CUBE_MEASURES'
```

The following report identifies the measures associated with each cube.

```
REPORT W 40 DOWN all_measures W 30 cube_measures
```

ALL_MEASURES	CUBE_MEASURES
-----	-----
PRICE_AND_COST_CUBE.UNIT_PRICE.MEASURE	PRICE_AND_COST_CUBE.CUBE
PRICE_AND_COST_CUBE.UNIT_COST.MEASURE	PRICE_AND_COST_CUBE.CUBE
UNITS_CUBE.UNITS.MEASURE	UNITS_CUBE.CUBE
UNITS_CUBE.SALES.MEASURE	UNITS_CUBE.CUBE

DIM_HIERARCHIES Relation

The DIM_HIERARCHIES relation identifies the dimension to which each hierarchy belongs. The values of the relation must be listed in the ALL_DIMENSIONS dimension. The following is a full description of the DIM_HIERARCHIES relation in a sample analytic workspace.

```
FULLDSC dim_hierarchies

DEFINE DIM_HIERARCHIES RELATION ALL_DIMENSIONS <ALL_HIERARCHIES>
PROPERTY 'AW$CLASS' 'CATALOGS'
PROPERTY 'AW$CREATEDBY' 'AW$XML'
PROPERTY 'AW$LASTMODIFIED' '16AUG04_09:43:27'
PROPERTY 'AW$ROLE' 'DIM_HIERARCHIES'
```

The following report identifies the dimension for each hierarchy.

```
REPORT W 35 DOWN all_hierarchies W 20 dim_hierarchies
```

ALL_HIERARCHIES	DIM_HIERARCHIES
-----	-----
CUSTOMER.AW\$NONE.HIERARCHY	CUSTOMER.DIMENSION
CUSTOMER.SHIPMENTS.HIERARCHY	CUSTOMER.DIMENSION
CUSTOMER.MARKET_SEGMENT.HIERARCHY	CUSTOMER.DIMENSION
PRODUCT.AW\$NONE.HIERARCHY	PRODUCT.DIMENSION
PRODUCT.PRIMARY.HIERARCHY	PRODUCT.DIMENSION
TIME.AW\$NONE.HIERARCHY	TIME.DIMENSION
TIME.CALENDAR_YEAR.HIERARCHY	TIME.DIMENSION
CHANNEL.AW\$NONE.HIERARCHY	CHANNEL.DIMENSION
CHANNEL.PRIMARY.HIERARCHY	CHANNEL.DIMENSION

DIM_LEVELS Relation

The DIM_LEVELS relation identifies the dimension to which each level belongs. The values of the relation must be listed in the ALL_DIMENSIONS dimension. The following is a full description of the DIM_LEVELS relation in a sample analytic workspace.

```
FULLDSC dim_levels
```

```
DEFINE DIM_LEVELS RELATION ALL_DIMENSIONS <ALL_LEVELS>
PROPERTY 'AW$CLASS' 'CATALOGS'
PROPERTY 'AW$CREATEDBY' 'AW$XML'
PROPERTY 'AW$LASTMODIFIED' '16AUG04_09:43:27'
PROPERTY 'AW$ROLE' 'DIM_LEVELS'
```

The following report identifies the dimension for each level.

```
REPORT W 35 DOWN all_levels W 20 dim_levels
```

ALL_LEVELS	DIM_LEVELS
-----	-----
CUSTOMER.AW\$NONE.LEVEL	CUSTOMER.DIMENSION
CUSTOMER.TOTAL_CUSTOMER.LEVEL	CUSTOMER.DIMENSION
CUSTOMER.REGION.LEVEL	CUSTOMER.DIMENSION
CUSTOMER.WAREHOUSE.LEVEL	CUSTOMER.DIMENSION
CUSTOMER.TOTAL_MARKET.LEVEL	CUSTOMER.DIMENSION
CUSTOMER.MARKET_SEGMENT.LEVEL	CUSTOMER.DIMENSION
CUSTOMER.ACCOUNT.LEVEL	CUSTOMER.DIMENSION
CUSTOMER.SHIP_TO.LEVEL	CUSTOMER.DIMENSION
PRODUCT.AW\$NONE.LEVEL	PRODUCT.DIMENSION
TIME.AW\$NONE.LEVEL	TIME.DIMENSION
TIME.YEAR.LEVEL	TIME.DIMENSION
TIME.QUARTER.LEVEL	TIME.DIMENSION
TIME.MONTH.LEVEL	TIME.DIMENSION
CHANNEL.AW\$NONE.LEVEL	CHANNEL.DIMENSION
CHANNEL.TOTAL_CHANNEL.LEVEL	CHANNEL.DIMENSION
CHANNEL.CHANNEL.LEVEL	CHANNEL.DIMENSION

DIM_ATTRIBUTES Relation

The DIM_ATTRIBUTES relation identifies the dimension to which each attribute belongs. The values of the relation must be listed in the ALL_DIMENSIONS dimension. The following is a full description of the DIM_ATTRIBUTES relation in a sample analytic workspace.

```
FULLDSC dim_attributes
```

```
DEFINE DIM_ATTRIBUTES RELATION ALL_DIMENSIONS <ALL_ATTRIBUTES>
PROPERTY 'AW$CLASS' 'CATALOGS'
PROPERTY 'AW$CREATEDBY' 'AW$XML'
PROPERTY 'AW$LASTMODIFIED' '16AUG04_09:43:27'
PROPERTY 'AW$ROLE' 'DIM_ATTRIBUTES'
```

The following report identifies the dimension for each attribute.

```
REPORT W 40 DOWN all_attributes W 20 dim_attributes
```

ALL_ATTRIBUTES	DIM_ATTRIBUTES
-----	-----
CUSTOMER.LONG_DESCRIPTION.ATTRIBUTE	CUSTOMER.DIMENSION
CUSTOMER.SHORT_DESCRIPTION.ATTRIBUTE	CUSTOMER.DIMENSION
PRODUCT.SHORT_DESCRIPTION.ATTRIBUTE	PRODUCT.DIMENSION
PRODUCT.PACKAGE.ATTRIBUTE	PRODUCT.DIMENSION
PRODUCT.BUYER.ATTRIBUTE	PRODUCT.DIMENSION
PRODUCT.MARKETING_MANAGER.ATTRIBUTE	PRODUCT.DIMENSION
PRODUCT.LONG_DESCRIPTION.ATTRIBUTE	PRODUCT.DIMENSION
TIME.END_DATE.ATTRIBUTE	TIME.DIMENSION
TIME.TIME_SPAN.ATTRIBUTE	TIME.DIMENSION
TIME.LONG_DESCRIPTION.ATTRIBUTE	TIME.DIMENSION
TIME.SHORT_DESCRIPTION.ATTRIBUTE	TIME.DIMENSION
TIME.TIME_DSO_1.ATTRIBUTE	TIME.DIMENSION
TIME.MONTH_OF_QUARTER.ATTRIBUTE	TIME.DIMENSION
TIME.MONTH_OF_YEAR.ATTRIBUTE	TIME.DIMENSION
TIME.QUARTER_OF_YEAR.ATTRIBUTE	TIME.DIMENSION
TIME.TIME_DSO_2.ATTRIBUTE	TIME.DIMENSION
TIME.TIME_DSO_3.ATTRIBUTE	TIME.DIMENSION
TIME.TIME_DSO_4.ATTRIBUTE	TIME.DIMENSION
CHANNEL.LONG_DESCRIPTION.ATTRIBUTE	CHANNEL.DIMENSION
CHANNEL.SHORT_DESCRIPTION.ATTRIBUTE	CHANNEL.DIMENSION

Supporting Object Information

The Catalogs class includes variables and formulas that list the objects that support various other objects.

AW_NAMES Variable

The AW_NAMES variable is dimensioned by ALL_OBJECTS. It contains the name of the workspace object that implements each logical object. If no workspace object implements a given logical object, the value is NA.

The following is a full description of an AW_NAMES variable.

```
FULLDSC aw_names

DEFINE AW_NAMES VARIABLE TEXT <ALL_OBJECTS>
PROPERTY 'AW$CLASS' 'CATALOGS'
PROPERTY 'AW$CREATEDBY' 'AW$XML'
PROPERTY 'AW$LASTMODIFIED' '04DEC02_13:09:14'
PROPERTY 'AW$ROLE' 'AW_NAMES'
```

The following report identifies the analytic workspace name of each logical dimension.

```
LIMIT all_objects TO all_dimensions
REPORT W 20 DOWN all_objects aw_names
```


ALL_OBJECTS	AW_NAMES
-----	-----
PRODUCT.DIMENSION	PRODUCT
TIME.DIMENSION	TIME
CHANNEL.DIMENSION	CHANNEL
CUSTOMER.DIMENSION	CUSTOMER

Features Class Objects

Features class objects hold information about specific logical objects and the workspace objects that implement them. For example, one object stores the descriptions of all the logical objects, while another indicates whether the object is intended to be visible to users.

ALL_DESCRIPTIONS Variable

The ALL_DESCRIPTIONS variable contains the short, long, and plural descriptions of various logical objects. For search convenience it is dimensioned by a composite.

The following is a full description of an ALL_DESCRIPTIONS variable.

```
FULLDSC all_descriptions
```

```
DEFINE ALL_DESCRIPTIONS VARIABLE TEXT <SPARSE <ALL_OBJECTS ALL_DESCTYPES ALL_LANGUAGES>>
LD FEATURES Descriptions for all objects
PROPERTY 'AW$CLASS' 'FEATURES'
PROPERTY 'AW$CREATEDBY' 'AW$XML'
PROPERTY 'AW$LASTMODIFIED' '04DEC02_13:09:14'
PROPERTY 'AW$ROLE' 'ALL_DESCRIPTIONS'
```

The following report shows the display names of the dimensions.

```
LIMIT all_objects TO all_dimensions
REPORT W 20 DOWN all_objects all_descriptions
```

```
ALL_LANGUAGES: AMERICAN
-----ALL_DESCRIPTIONS-----
-----ALL_DESCTYPES-----
ALL_OBJECTS      SHORT      LONG      PLURAL
-----
CUSTOMER.DIMENSION Customer  Customer  Customer
PRODUCT.DIMENSION Product   Product   Product
TIME.DIMENSION   Time     Time      Time
CHANNEL.DIMENSION Channel   Channel   Channel
```

DEFAULT_HIER Relation

The DEFAULT_HIER relation records the full name of the default hierarchy for each dimension. The base dimension for the relation is ALL_DIMENSIONS.

The following is a full description of a DEFAULT_HIER relation.

```
FULLDSC default_hier
```

```
DEFINE DEFAULT_HIER RELATION ALL_HIERARCHIES <ALL_DIMENSIONS>
LD FEATURES Default hierarchy for each dimension
PROPERTY 'AW$CLASS' 'FEATURES'
PROPERTY 'AW$CREATEDBY' 'AW$XML'
PROPERTY 'AW$LASTMODIFIED' '04DEC02_13:09:14'
PROPERTY 'AW$ROLE' 'DEFAULT_HIER'
```

The following report shows the default hierarchy for each dimension.

REPORT W 20 DOWN all_dimensions W 40 default_hier

ALL_DIMENSIONS	DEFAULT_HIER
CUSTOMER.DIMENSION	CUSTOMER.SHIPMENTS.HIERARCHY
PRODUCT.DIMENSION	PRODUCT.PRIMARY.HIERARCHY
TIME.DIMENSION	TIME.CALENDAR_YEAR.HIERARCHY
CHANNEL.DIMENSION	CHANNEL.PRIMARY.HIERARCHY

VISIBLE Variable

The `VISIBLE` variable is a boolean that indicates whether the Oracle OLAP enablement utilities should expose or ignore the objects that are registered. The variable is dimensioned by `ALL_OBJECTS` so that each object has its own setting.

The following is a full description of a `VISIBLE` variable.

FULLDSC visible

```

DEFINE VISIBLE VARIABLE BOOLEAN <ALL_OBJECTS>
PROPERTY 'AW$CLASS' 'FEATURES'
PROPERTY 'AW$CREATEDBY' 'AW$XML'
PROPERTY 'AW$LASTMODIFIED' '10MAY05_11:04:57'
PROPERTY 'AW$ROLE' 'VISIBLE'

```

The following report shows the visibility of objects in a sample analytic workspace.

REPORT W 40 DOWN all_objects visible

ALL_OBJECTS	VISIBLE
CUSTOMER.DIMENSION	yes
CUSTOMER.AW\$NONE.LEVEL	no
CUSTOMER.AW\$NONE.HIERARCHY	no
CUSTOMER.LONG_DESCRIPTION.ATTRIBUTE	yes
CUSTOMER.SHORT_DESCRIPTION.ATTRIBUTE	yes
CUSTOMER.TOTAL_CUSTOMER.LEVEL	yes
CUSTOMER.REGION.LEVEL	yes
CUSTOMER.WAREHOUSE.LEVEL	yes
.	.
.	.
.	.

Member_Inhier Valueset

The *member_inhier* valueset stores lists of the dimension members that are in each hierarchy. There is one of these valuesets for each dimension in the workspace, and that dimension is the valueset's parent.

The following is a full description of a *member_inhier* valueset for the `TIME` dimension.

FULLDSC time_inhier

```

DEFINE TIME_INHIER VALUESET TIME <TIME_HIERLIST>
PROPERTY 'AW$CLASS' 'FEATURES'
PROPERTY 'AW$CREATEDBY' 'AW$XML'
PROPERTY 'AW$LASTMODIFIED' '18MAR04_14:46:51'
PROPERTY 'AW$MEMBER_INHIER' NA
PROPERTY 'AW$PARENT_NAME' 'TIME'
PROPERTY 'AW$ROLE' 'MEMBER_INHIER'

```

Member_Createdby Variable

The *member_createdby* variable records the entity that created each member of a given dimension. There is one of these variables for each dimension in the workspace, and that dimension is the variable's parent.

The following is a full description of a *member_createdby* variable for a dimension called TIME.

```
FULLDSC time_createdby

DEFINE TIME_CREATEDBY VARIABLE TEXT <TIME>
LD FEATURES Creator of each dimension member for TIME
PROPERTY 'AW$CLASS' 'FEATURES'
PROPERTY 'AW$CREATEDBY' 'AW$XML'
PROPERTY 'AW$LASTMODIFIED' '03SEP03_15:27:47'
PROPERTY 'AW$PARENT_NAME' 'TIME'
PROPERTY 'AW$ROLE' 'MEMBER_CREATEDBY'
```

Member_Familyrel Relation

The *member_familyrel* relation records the ancestors of a given member of a dimension. There is one of these relations for each dimension in the workspace, and that dimension is the variable's parent. These relations are for internal use.

The following is a full description of a *member_familyrel* relation for the TIME dimension.

```
FULLDSC time_familyrel

DEFINE TIME_FAMILYREL RELATION TIME <TIME TIME_LEVELLIST TIME_HIERLIST>
LD FEATURES Family/Ancestry structure for TIME
PROPERTY 'AW$CLASS' 'FEATURES'
PROPERTY 'AW$CREATEDBY' 'AW$XML'
PROPERTY 'AW$LASTMODIFIED' '03SEP03_15:27:47'
PROPERTY 'AW$MEMBER_FAMILYREL' NA
PROPERTY 'AW$PARENT_NAME' 'TIME'
PROPERTY 'AW$ROLE' 'MEMBER_FAMILYREL'
```

Member_Gid Variable

The *member_gid* variable records the level depth of a given member of a dimension, within a given hierarchy. There is one of these relations for each dimension in the workspace, and that dimension is the variable's parent. These relations are for internal use.

The following is a full description of a *member_gid* relation for the TIME dimension.

```
FULLDSC time_gid

DEFINE TIME_GID RELATION GID_DIMENSION <TIME TIME_HIERLIST>
PROPERTY 'AW$CLASS' 'FEATURES'
PROPERTY 'AW$CREATEDBY' 'AW$XML'
PROPERTY 'AW$LASTMODIFIED' '16AUG04_09:45:07'
PROPERTY 'AW$MEMBER_GID' NA
PROPERTY 'AW$PARENT_NAME' 'TIME'
PROPERTY 'AW$ROLE' 'MEMBER_GID'
```

OBJ_CREATEDBY Variable

The OBJ_CREATEDBY variable records the entity that created each object that is registered in the standard form. The variable is dimensioned by ALL_OBJECTS.

The following is a full description of the OBJ_CREATEDBY variable.

```
FULLDSC obj_createdby
```

```
DEFINE OBJ_CREATEDBY VARIABLE TEXT <ALL_OBJECTS>
PROPERTY 'AW$CLASS' 'FEATURES'
PROPERTY 'AW$CREATEDBY' 'AW$XML'
PROPERTY 'AW$LASTMODIFIED' '04DEC02_13:09:14'
PROPERTY 'AW$ROLE' 'OBJ_CREATEDBY'
```

VERSION Variable

The VERSION variable records the version number of the standard form convention under which the analytic workspace is being managed.

The following is a full description of the VERSION variable.

```
FULLDSC ____xml_user_aw_version
DEFINE ____XML_USER_AW_VERSION VARIABLE TEXT
PROPERTY 'AW$CLASS' 'FEATURES'
PROPERTY 'AW$CREATEDBY' 'AW$XML'
PROPERTY 'AW$LASTMODIFIED' '16MAY05_12:28:55'
PROPERTY 'AW$NEWTIMECALCS' yes
PROPERTY 'AW$ROLE' 'VERSION'
PROPERTY 'AW$VERSION10.1.0.3' NA
PROPERTY 'AW$VERSION10.2' NA
```

The following command shows the current standard form version number.

```
SHOW ____xml_user_aw_version
10.2
```

Extensions Class Objects

Extensions class objects are defined and maintained by the Oracle OLAP utilities. They are proprietary extensions to the standard form, and there is no commitment on the part of Oracle to maintain them from release to release.

Upgrading From Express Server

This appendix provides upgrade instructions and identifies some of the major differences between Oracle Express Server 6 and Oracle OLAP. It is intended to provide a frame of reference to help you understand the material presented in this guide.

This appendix includes the following topics:

- [Administration](#)
- [Applications Support](#)
- [Programming Language Changes](#)
- [Transforming Oracle Express Databases to Standard Form](#)

Administration

Oracle OLAP is installed as an option in Oracle Enterprise Edition, and it is now integrated with Oracle Database. While Express Server runs in a service environment, Oracle OLAP runs within the Oracle kernel.

In Oracle, the term **database** refers only to the relational database. An Express database is now called an **analytic workspace**. In Oracle OLAP, an analytic workspace can be used either as a persistent data repository or as a transient data cache. A persistent analytic workspace is stored in a relational table, which in turn is stored in a tablespace. There are no ".db" files.

The administrative tasks for Oracle OLAP are merged with the database tool set.

Management Tools

Oracle Enterprise Manager encompasses the tools for administering Oracle databases, providing a common user interface across all platforms. Performance data for OLAP can be collected in system tables the same as any other Oracle database performance statistics. Oracle Enterprise Manager provides a graphical interface to SQL. Because OLAP now runs within the Oracle Database kernel, many of the basic administrative tasks (such as starting, stopping, and configuring the process) are subsumed into database management.

Analytic Workspace Manager is the tool for creating and managing analytic workspaces.

OLAP Instance Manager, oesmgr, and oescmd are not available.

Authentication of Users

Oracle OLAP does not use operating system identities, except for the installation user under whose identity Oracle Database is installed. You can delete other operating system identities created for use by Express Server (such as the DBA user, the Initialize user, the Default user, and individual user names) if they have no other purpose.

All authentication is performed by Oracle Database. Applications must always present credentials before opening a session, and those credentials must match a user name and password stored in the relational database. Before users can access Oracle OLAP, you must define user names and passwords in the database.

For users to access operating system files, they must have access rights to a directory object that is mapped to the physical directory path. This access is granted either to an individual user ID or to a database role.

See Also: [Chapter 6](#) for more information about OLAP administration tasks

Data Transfer

An Oracle OLAP session is always connected to the database. You do not open a connection with the database as a separate or optional step.

You can copy data between an analytic workspace objects (such as variables and dimensions) and relational tables in the following ways:

- A Java package named AXML provides procedures for creating analytic workspaces from relational tables. Analytic Workspace Manager provides a graphical interface to this package.
- A PL/SQL package named DBMS_AWM provides procedures for creating analytic workspaces from OLAP Catalog metadata mapped to a star or snowflake schema. Earlier versions of Analytic Workspace Manager provided a graphical interface to this package.
- The OLAP DML SQL command fetches data into dimensions and variables for further manipulation. A new SQL IMPORT command facilitates bulk data transfer from relational tables into the analytic workspace, and a new SQL INSERT DIRECT command facilitates data transfer from the analytic workspace into relational tables.
- Using SQL table functions, it is now possible for a SQL-based application to manipulate and extract data from an analytic workspace. Express Server did not permit a data transfer to be initiated externally. The SQL OLAP_TABLE function provides this capability.

ODBC is not available, and thus access to third-party databases is not available directly from Oracle OLAP. However, Oracle Database supports bridges to all major third-party databases.

Oracle Express Relational Access Administrator and Oracle Express Relational Access Manager are not available.

Localization

The Express Server language support has been replaced by Oracle Globalization Technology, which provides more extensive localization support and is much easier to administer than the localization features of Express Server. Oracle Database and Oracle OLAP use the same character set, which is selected during installation.

If you are upgrading Express databases that use translation tables, then you can delete those tables because they are not needed by Oracle OLAP. Likewise, you should check your Express programs for use of obsolete commands and keywords that supported translation tables.

Support for Globalization Technology has been added to the OLAP DML. These options enable an application to query the current localization settings and override the behaviors controlled by the default language and territory.

[Table B–1](#) identifies the Unicode character sets available in Oracle that are equivalent to the Express Server character sets. If you plan to import Express databases or to use Oracle OLAP to access multibyte data in external files, then you might find this information helpful in identifying an appropriate database character set. Note that the Express CHARSET option is now obsolete.

Table B–1 Multibyte Character Set Equivalents

Express Server	Unicode Character Set
EUC	JA16EUC
SHIFTJIS	JA16SJIS
HANGEUL	KO16KSC5601
SCHINESE	ZHS16GBK
TCHINESE	ZHT16BIG5

Applications Support

Oracle OLAP enables applications to access its dimensional data directly through either a Java API or SQL. Express SPL programs can be executed using either programming method. Be sure to review all SPL programs to remove commands that are no longer available and to take advantage of new functionality.

Analytic Workspace Manager provides a user interface for creating a [database standard form](#) analytic workspace, loading data from relational tables, and aggregating the data.

You cannot run Windows C++, HTML, or Java applications that were developed for use with Express Server.

See Also: [Chapter 3](#) for methods of creating standard form analytic workspaces from data in relational tables

Programming Environment

Applications for Oracle OLAP can be developed in Java using OracleBI Beans. SQL-based applications can access OLAP data through views or manipulate it directly through the OLAP_TABLE function.

OLAP Worksheet provides an interactive environment for developing stored procedures in either the OLAP DML or SQL. The PL/SQL DBMS_AW procedure executes OLAP DML commands from a SQL environment.

You cannot connect to Oracle OLAP using Express Administrator, Personal Express, or the Express Connection Utility.

See Also: [Chapter 5](#) for information about OracleBI Beans

Communications

Oracle OLAP provides communications through Oracle Call Interface (OCI) and Java Database Connectivity (JDBC).

XCA and SNAP I are no longer available. Session sharing is not supported.

Metadata

OracleBI Beans can query data that is stored either in an analytic workspace or in relational tables. Analytic workspaces require standard form metadata, which is stored in the same analytic workspace as the business measures. This metadata is stored in properties on workspace objects and in catalogs, which are implemented as special dimensions, variables, and valuesets. Relational data sources require OLAP Catalog metadata, which is stored in relational tables.

Oracle Express Administrator is not available in Oracle OLAP, and the Oracle Express Objects metadata that it generated is not used by OracleBI Beans. Instructions for transforming Express metadata to standard form metadata are provided in this appendix.

See Also:

- [Chapter 7](#) for information about the OLAP Catalog
- [Appendix A](#) for information about standard form

Programming Language Changes

Numerous changes have been made to the Express Stored Procedure Language (now called the OLAP Data Manipulation Language or OLAP DML).

New Commands

Support in the following areas has been added to the OLAP DML:

Partitioned variables
Compressed composites
Allocation
Dynamic model execution
Bulk data transfers between analytic workspaces and relational tables
Byte manipulation functions
Data conversion functions
New data types

Obsolete Commands

Support in the following areas has been dropped:

EXTCALL
ODBC
SNAP I
XCA
Operating system commands

Conjoint dimensions and the ROLLUP command are still available, but composite dimensions and aggmappings are strongly recommended instead, because they are easier to manage and perform better.

See Also: *OLAP DML Reference* for comprehensive lists of new, obsolete, and significantly revised commands

UPDATE and COMMIT

The UPDATE command moves analytic workspace changes from a temporary tablespace to a permanent tablespace. Your changes are not saved permanently until you execute a COMMIT command, either from your Oracle OLAP session or from SQL. A COMMIT makes the changes permanent.

Changes that have not been moved to the permanent tablespace are not committed. If you issue a COMMIT without first updating your analytic workspace, then no changes to the analytic workspace that you made after your last UPDATE are committed to disk.

The COMMIT command executes a SQL COMMIT command. All changes made during your session are committed, whether they were made through Oracle OLAP or through another form of access (such as SQL) to the database.

Transforming Oracle Express Databases to Standard Form

EIF files are used to transfer the contents of an analytic workspace from one database to another and to upgrade from an Express database. You can create an analytic workspace from an Express database simply by using EIF files to transfer the objects.

The more complex task is to create an analytic workspace in database standard form, so that you can use the current generation of Oracle OLAP tools. You may be able to leverage your investment in Express metadata to create standard form metadata. Otherwise, you must define a new logical metadata model.

Who Should Use the Transformation Tool

If your Express database contains Oracle Express Objects metadata (such as an Oracle Sales Analyzer, Oracle Financial Analyzer, or Oracle Express Administrator database), then you can use the transformation tool in Analytic Workspace Manager. Without Oracle Express Objects metadata, the transformation tool may not generate sufficient standard form metadata for the OLAP tools to work.

If your source data is in tables or views, then you have a choice of using the transformation tool to convert an Express database, or using other tools to create an analytic workspace directly from the source data. If your source data is in flat files, then you can define them as external tables first, then handle them the same as tables stored in the database.

The transformation tool enables you to use your Oracle Express Objects metadata instead of redefining the logical model in Analytic Workspace Manager. However, you must perform other steps manually, as described in "[What the Transformation Tool Does Not Do For You](#)" on page B-6. You can choose which method best suits your needs.

[Table B-2](#) identifies the upgrade options.

Table B-2 Choosing an Upgrade Path for Express Databases

If you have Oracle Express Objects metadata...	And your source data is located in...	THEN create a standard form analytic workspace using...
Yes	Tables or views	Transformation tool
Yes	Flat files	Transformation tool

Table B–2 (Cont.) Choosing an Upgrade Path for Express Databases

If you have Oracle Express Objects metadata...	And your source data is located in...	THEN create a standard form analytic workspace using...
No	Tables or views	Analytic Workspace Manager Model View (you may try the transformation tool first)
No	Flat files	Oracle Warehouse Builder or Analytic Workspace Manager Model View using the database external tables feature
No	Third-party databases or other sources outside of Oracle Database	Oracle Warehouse Builder

What the Transformation Tool Does For You

The transformation tool enables you to start using OracleBI Beans with your data in a matter of minutes. The transformation step from Oracle Express Objects metadata to database standard form metadata involves a single menu choice in Analytic Workspace Manager. The entire process, from importing the EIF file to querying views of the analytic workspace using a OracleBI Beans application, is very quick and fully automated.

If you load data only at the base level, then you can create an aggregation plan in Analytic Workspace Manager. Aggregation plans, which use the `AGGREGATE` subsystem, are faster and more flexible than the `ROLLUP` command.

If you are running Oracle Database in 9i compatibility mode, then your analytic workspace will be converted to version 9i standard form. You can upgrade your analytic workspace to 10g standard form at a later time. If you are running Oracle Database in 10g compatibility mode, then your analytic workspace will be converted to version 10g standard form.

What the Transformation Tool Does Not Do For You

The transformation process circumvents the usual first step in creating an analytic workspace, which is developing a logical data model and mapping the logical objects to the data source. The tools make the appropriate changes to the standard form catalogs. This maintenance process is not available to converted analytic workspaces. Thus, you must do the following tasks manually:

- If you want to perform time-based analysis on your data, you must identify all time dimensions and populate end date and time span attributes before transformation. A sample program is provided in this appendix.
- Your analytic workspace may contain programs with references to obsolete commands. You must revise them. You may also want to use some of the new features in the OLAP DML. For example, you can handle sparse data with composites (instead of conjoints) and partition large variables. You must define new variables and copy the data from the old variables (or reload it from the data source) to make these changes.
- If your source data is in relational tables, then you can map the logical objects (cubes, measures, dimensions, and so forth) to the appropriate columns. Then you can use the Build Wizard in Analytic Workspace Manager to refresh the data.

or

If your source data is not in relational tables, then you must revise your data load programs and run them manually to refresh the data.

Converting From Oracle Express Objects Metadata

The transformation tool operates on an analytic workspace. It uses the existing metadata to identify the roles of various objects, and then does the following:

- Populates existing objects with the appropriate standard form properties. For example, the Oracle Express Objects language dimension is given the `AW$ROLE` value of `ALL_LANGUAGES`.
- Creates and populates standard form metadata objects, such as the standard form catalogs, *member_gid* and *member_inhier* variables, and *member_familyrel* and *member_levelrel* relations. For descriptions of these standard form objects, refer to [Appendix A](#).

The transformation tool adds standard form objects and properties; it does not delete any previously existing objects or properties. You can delete them manually if you wish.

OracleBI Beans requires a level-sorted Time dimension with period end dates and time span attributes in order to support time-based analysis.

Procedure: Converting From Oracle Express Objects to Standard Form

Most of the steps for converting to standard form (such as creating a new analytic workspace and importing the EIF file) can be done using the Object View in Analytic Workspace Manager. However, this procedure uses the command-line interface provided by OLAP Worksheet, on the basis that users making this transformation are already familiar with OLAP DML commands.

Follow these steps to use the Oracle Express Objects metadata transformation tool to create a standard form analytic workspace.

1. Create an EIF file from your Oracle Express Objects database, and copy the file to a physical directory that is mapped to a directory object.

For information about database directories, refer to ["Permitting Access to External Files"](#) on page 6-7.

2. Open Analytic Workspace Manager and attach to Oracle Database, as described in ["Introduction to Analytic Workspace Manager"](#) on page 3-1.

3. From the Tools menu, choose OLAP Worksheet.

OLAP Worksheet opens in a separate window.

4. Create a new analytic workspace from the EIF file using commands like these:

```
AW CREATE aw
IMPORT ALL FROM EIF FILE 'directory/filename.eif' DATA DFNS
UPDATE
COMMIT
```

5. Identify the Time dimensions:

```
LIMIT name TO OBJ(PROPERTY 'DIMTYPE') EQ 1
REPORT name
```

6. Identify the hierarchy dimension for each Time dimension:

```
SHOW OBJ(PROPERTY 'HIERDIM' timedim)
```

Note: The Oracle Express Objects metadata identifies all of the objects that support hierarchies and levels for a dimension. You can use the FULLDSC command to see all of the properties of a dimension, or use the OBJ function as shown here to obtain the value of particular properties, such as HIERDIM, LEVELDIM, and LEVELREL.

7. Create date and time span attributes for each Time dimension.

```
DEFINE TIME_TIME_SPAN VARIABLE INTEGER <timedim hierdim>
PROPERTY 'USERDATA' FALSE
```

```
DEFINE TIME_END_DATE VARIABLE DATE <timedim hierdim>
PROPERTY 'USERDATA' FALSE
```

8. Populate the end date and time span attributes, as described in ["Populating Time Attributes"](#) on page B-8.
9. Set properties on the Time dimension:

```
CONSIDER timedim
PROPERTY 'END_DATE' attribute_name
PROPERTY 'TIME_SPAN' attribute_name
```

The END_DATE and TIME_SPAN (*attribute_name*) values identify the names of the variables that you just created.

10. Save these changes.

```
UPDATE; COMMIT
```

11. Return to the Analytic Workspace Manager window, and choose **View > Object View**.
12. In the navigation tree, expand the Analytic Workspaces folder for your schema.
13. Right-click the analytic workspace, then choose **Transform Analytic Workspace to Standard Form**.

A message appears to advise you to follow these directions. You can continue.
14. Review the output from the transformation tool to assure that all measures and dimensions have been identified properly.
15. To refresh the data from relational tables or views, map the logical objects and run the Build Wizard as described in [Chapter 3](#).

To refresh the analytic workspace from other sources, revise and run the data load programs, as described in ["Revising the Load Programs"](#) on page B-9.

Populating Time Attributes

A standard form Time dimension has the following characteristics:

- Dimension members are sorted chronologically within level.
- The AW\$TYPE property has a value of 'Time'.
- Period end date and time span attributes are defined and populated.

The transformation process sets the AW\$TYPE property, defines standard form attributes for period end dates and time span, and registers this information in the standard form catalogs. It does not change the order of the Time dimension members nor populate the attributes.

Sorting Time Dimension Members

If the Time members are not already sorted in chronological order within levels, then commands like the ones shown in [Example B–1](#) to sort them correctly. Refer to [Appendix A](#) for information about the standard form objects used in the example.

Example B–1 *Template for Sorting the Time Dimension*

```
LIMIT time_dim TO ALL
"Sort levels in descending order and time periods in ascending order
SORT time_dim D time_dim_LEVELREL A time_dim_END_DATE
LIMIT member_inhier TO time_dim
MAINTAIN time_dim MOVE VALUES(valueset) FIRST

"Save these changes
UPDATE
COMMIT
```

Creating and Populating End Date and Time Span Attributes

The end date and time span attributes are variables dimensioned by Time. The end date variable must be defined as a DATE data type. The time span variable is typically defined as an INTEGER data type, but any numeric data type is acceptable.

The method that you use to populate the end date and time span attributes depends on your data source and the format of your Time dimension members. If the information is available from your original data source (that is, the source from which you populated the Express database), then you can load the information using the OLAP DML. Otherwise, you must derive the information from the dimension members or their descriptions. An example of this method is shown in ["Populating the XADEMO Time Attributes"](#) on page B-13.

Setting Properties on Time Objects

You must define and set the following properties before running CREATE_DB_STDFORM:

- On the Time dimension, set the END_DATE and TIME_SPAN properties to the object names for these attributes. The DIMTYPE property should be set to 1 already.
- On the end-date and time-span attributes, set the USERDATA property to FALSE.

Revising the Load Programs

If the source data is stored in relational tables or views in Oracle Database, then you can map the logical objects of your analytic workspace to the appropriate columns, using the Model View in Analytic Workspace Manager. After mapping the objects, you can run the Build Wizard to refresh the data.

If your source data is stored in a different format, such as flat files, your analytic workspace probably contains programs generated by Express Administrator for refreshing your Express database. You can begin by modifying these programs for use in your analytic workspace; they are unusable in their current state.

Delete the following code from your load programs:

- Calls to EDDE.MSG. This program displayed Express error messages in the Administrator graphical interface, and deleting calls to it does not affect the operation of your program.

- Calls to `EDDE.HIERMNT`. This program managed the metadata associated with dimension hierarchies. It is not available for use in analytic workspaces, nor is any of the information about your data that was stored in an `XPDDDATA` database.
- Code to establish a connection with Oracle. Since the analytic workspace is part of Oracle Database, a connection to relational tables and views is always open.

The load programs only refresh the dimensions and measures. They do not refresh the dimension attributes, the hierarchy and level objects, or the standard form catalogs.

See Also: ■ [Appendix A](#) for information about the standard form catalogs

- Oracle OLAP DML Reference

Example: Converting the XADEMO Database to Standard Form

This example uses an EIF file that contains objects and Oracle Express Objects metadata from an Express database named `XADEMO`. If you are converting an Express database, you are probably already familiar with `XADEMO`.

Creating a Standard Form XADEMO Analytic Workspace

Suppose that an EIF file named `xademo.eif` is located in a system directory named `\users\oracle\xademo_files`. Take these steps to create a standard form analytic workspace from this file.

1. Log in to your Oracle database as the `SYSTEM` user and create the `XADEMO` user, permanent and temporary tablespaces, and a directory object for access to the EIF file.

```
CREATE TABLESPACE olapdata DATAFILE '$ORACLE_HOME/oradata/olapdata.dbf'
  SIZE 5M REUSE AUTOEXTEND ON NEXT 5M MAXSIZE UNLIMITED
  EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO;

CREATE TEMPORARY TABLESPACE olaptmp TEMPFILE '$ORACLE_HOME/oradata/olaptmp.tmp'
  SIZE 5M REUSE AUTOEXTEND ON NEXT 5M MAXSIZE UNLIMITED
  EXTENT MANAGEMENT LOCAL UNIFORM SIZE 256K;

CREATE USER xademo IDENTIFIED BY 'xademo'
  DEFAULT TABLESPACE olapdata
  TEMPORARY TABLESPACE olaptmp
  QUOTA UNLIMITED ON olapdata
  ACCOUNT UNLOCK;

CREATE DIRECTORY xademo_dir as '/users/oracle/OraHome1/xademo_files';
GRANT READ ON DIRECTORY xademo_dir TO xademo;
```

Refer to [Chapter 6](#) for information about performing these tasks.

2. Open Analytic Workspace Manager and connect to Oracle Database as the `XADEMO` user.
3. Open OLAP Worksheet.
4. Create an analytic workspace from the EIF file:

```
AW CREATE xademo
IMPORT ALL FROM EIF FILE 'xademo_dir/xademo.eif' DATA DFNS
UPDATE
COMMIT
```

5. Identify the Time dimensions:

```
LIMIT name TO OBJ (PROPERTY 'DIMTYPE') EQ 1
REPORT name
```

```
NAME
```

```
-----
```

```
TIME
```

```
QUARTER
```

```
YEAR
```

```
MONTH
```

This example shows how to provide support to the TIME dimension. Repeat these procedures for the other Time dimensions.

6. Identify the HIERDIM dimension for TIME.

```
SHOW OBJ (PROPERTY 'HIERDIM' 'TIME')
```

```
T0.HIERDIM
```

7. Create the TIME_END_DATE and TIME_TIME_SPAN variables.

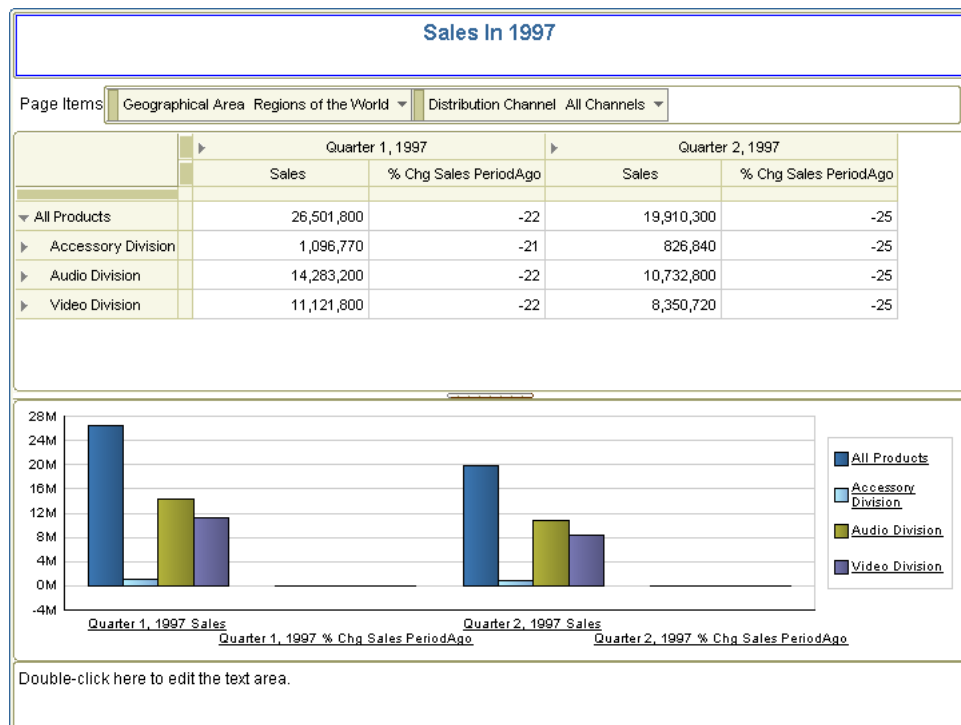
```
DEFINE TIME_END_DATE VARIABLE DATE <TIME>
PROPERTY 'USERDATA' FALSE
DEFINE TIME_TIME_SPAN VARIABLE INTEGER <TIME>
PROPERTY 'USERDATA' FALSE
```

8. Populate the TIME_END_DATE and TIME_TIME_SPAN variables, as described in the following sections.**9. Set the properties on TIME.**

```
CONSIDER time
PROPERTY 'END_DATE' 'TIME_END_DATE'
PROPERTY 'TIME_SPAN' 'TIME_TIME_SPAN'
```

10. In the Object View, right-click XADEMO, then choose Transform Analytic Workspace to Standard Form.

Figure B-1 shows a report generated by Discoverer Plus OLAP with data from the transformed XADEMO analytic workspace.

Figure B–1 Report of XADEMO Measures

About the Time Dimension in XADEMO

Oracle Express Objects metadata stores the names of supporting objects in properties on the TIME dimension, as shown in [Table B–3](#).

Table B–3 Oracle Express Objects Properties for Hierarchy and Level Support

Property	Description
HIERDIM	List of hierarchies (dimension)
LEVELDIM	List of levels (dimension)
LEVELREL	Level associated with each dimension member (relation)
LEVELLABELFRM	Description of each level (formula)

By using the OBJ function, you can discover the names of objects that support the TIME dimension:

```
SHOW OBJ (PROPERTY 'LEVELDIM' 'TIME')
TO .LEVELDIM
```

```
SHOW OBJ (PROPERTY 'LEVELLABELFRM' 'TIME')
TO .LVLLABFRM
```

The TIME dimension has two hierarchies, which are listed in the TO . LEVELDIM dimension. They are named STANDARD and YTD. The following report shows sample TIME members at each level.

```
LIMIT time TO FIRST 2
LIMIT time ADD ANCESTORS
REPORT DOWN time W 12 t0.levelrel W 20 t0.lvllabfrm
```


TIME	-----T0.HIERDIM-----			
	-----STANDARD-----		-----YTD-----	
	T0.LEVELREL	T0.LVLLABFRM	T0.LEVELREL	T0.LVLLABFRM
JAN96	L3	Month(s)	L5	YTD Month(s) Detail
FEB96	L3	Month(s)	L5	YTD Month(s) Detail
Q1.96	L2	Quarter(s)	NA	NA
LAST.YTD	NA	NA	L4	YTD Summaries
1996	L1	Year(s)	NA	NA

Populating the XADEMO Time Attributes

The POP_TIME_ATTRS program shown in [Example B-2](#) populates the TIME_END_DATE and TIME_TIME_SPAN variables.

For TIME_END_DATE, the program uses the ENDDATE function to identify the last day of each time period. The ENDDATE function only operates on dimensions with a time data type (such as MONTH and YEAR). However, the XADEMO TIME dimension has a TEXT data type. Several transformations are needed before the ENDDATE function can be used. The program takes these steps:

1. For each level, defines a dimension with the appropriate data type (MONTH, QUARTER, or YEAR). In the example, the dimensions are named M_TEMP, Q_TEMP, and Y_TEMP.
2. Stores the names of the dimension members for particular level in a valueset. In the example, the valueset is named T_LIST.
3. Uses the current status of the T_LIST valueset to add members to the new dimensions (M_TEMP, Q_TEMP, and Y_TEMP).

For TIME_TIME_SPAN, the program extracts the first two characters from TIME_END_DATE at the month level, which has values like 30APR96, to get the number of days in each month.

The program then uses the ROLLUP command to calculate the number of days in each quarter and year. T0.PARENT is a self-relation that identifies the parent-child relationships among dimension members. However, T0.PARENT is dimensioned by T0.HIERDIM, so ROLLUP cannot use T0.PARENT. Instead, the program creates a relation named TIME_PARENTREL dimensioned only by TIME, populates it from T0.PARENT, and uses the new relation in the ROLLUP command.

Note that AGGREGATE is more efficient than ROLLUP, but since this case involves just a single dimension in which all aggregate values are stored, ROLLUP is slightly more convenient and the performance differences are negligible.

Example B-2 OLAP DML Program for Populating TIME Attributes

```

DEFINE POP_TIME_ATTRS PROGRAM
PROGRAM
VARIABLE _ytd TEXT           " Stores YTD time members
TRAP ON cleanup              " Divert processing on error to CLEANUP label

" Define dimensions for each level with date data types
IF NOT EXISTS('m_temp')
  THEN DEFINE m_temp DIMENSION MONTH
  ELSE MAINTAIN m_temp DELETE ALL

IF NOT EXISTS('q_temp')
  THEN DEFINE q_temp DIMENSION QUARTER
  ELSE MAINTAIN q_temp DELETE ALL

```

```
" Format years like TIME year members (1997 instead of YR97)
IF NOT EXISTS('y_temp')
  THEN DO
    DEFINE y_temp DIMENSION YEAR
    CONSIDER y_temp
    VNF <YYYY>
    DOEND
  ELSE MAINTAIN y_temp DELETE ALL

" Define a valueset to store time members
IF NOT EXISTS('t_list')
  THEN DEFINE t_list VALUESET TIME
  ELSE LIMIT t_list TO NA

" Initialize target variables
ALLSTAT
time_time_span = NA
time_end_date = NA
" *****
"   Set values for the STANDARD hierarchy
" *****
LIMIT t0.hierdim TO 'STANDARD'
" Select all time members at the month level
LIMIT time TO t0.levelrel 'L3'
" Store months in the valueset
LIMIT t_list TO time
" Populate M_TEMP so all months have a MONTH data type
MAINTAIN m_temp MERGE values(t_list)
" Calculate the end date
FOR m_temp
  time_end_date(time, m_temp) = ENDDATE(m_temp)
" Extract the number of days in each month
time_time_span = CONVERT(EXTCHARS(time_end_date, 1, 2), DECIMAL)

" Store quarters in q_temp
LIMIT time TO t0.levelrel 'L2'
LIMIT t_list TO time
MAINTAIN q_temp MERGE VALUES(t_list)
FOR q_temp
  time_end_date(time, q_temp) = ENDDATE(q_temp)

" Store years in y_temp
LIMIT time TO t0.levelrel 'L1'
LIMIT t_list TO time
MAINTAIN y_temp MERGE VALUES(t_list)
FOR y_temp
  time_end_date(time, y_temp) = ENDDATE(y_temp)
" *****
"   Set values for the YTD hierarchy
" *****
LIMIT t0.hierdim TO 'YTD'
" Limit status of months to YTD
LIMIT time TO t0.levelrel 'L5'
LIMIT t_list TO time
LIMIT m_temp TO t_list

" Calculate end date and time span for months
FOR m_temp
  time_end_date(time, m_temp) = ENDDATE(m_temp)
time_time_span = CONVERT(EXTCHARS(time_end_date, 1, 2), DECIMAL)
```

```
" Get current and previous YTD
LIMIT time TO t0.parent EQ 'LAST.YTD'
LIMIT time KEEP LAST 1
_ytd = time
time_end_date(time, 'LAST.YTD') = time_end_date(time, _ytd)
LIMIT time TO t0.parent EQ 'CURRENT.YTD'
LIMIT time KEEP LAST 1
_ytd = time
time_end_date(time, 'CURRENT.YTD') = time_end_date(time, _ytd)

" Rollup time span for quarters and years
LIMIT t0.hierdim TO ALL
LIMIT time TO ALL
FOR t0.hierdim
  DO
    time_parentrel = t0.parent
    ROLLUP time_time_span OVER time USING time_parentrel
  DOEND

CLEANUP:
" Delete temporary objects
DELETE m_temp q_temp y_temp t_list time_parentrel
END
```

Glossary

Active Catalog

A set of relational views that expose the standard form metadata stored in analytic workspaces, where it can be accessed by SQL. Applications that use OracleBI Beans query the Active Catalog.

See also [database standard form](#).

abstract data type (ADT)

See [object type](#).

additive

Describes a fact (or measure) that can be summarized through addition. An additive fact is the most common type of fact. Examples include sales, cost, and profit.

Contrast with [nonadditive](#), [semi-additive](#).

aggregation

The process of consolidating data values into a single value. For example, sales data could be collected on a daily basis and then be aggregated to the week level, the week data could be aggregated to the month level, and so on. The data can then be referred to as aggregate data. Aggregation is synonymous with summarization, and aggregate data is synonymous with summary data.

analytic workspace

A dimensional schema stored in a LOB table in the relational database. An analytic workspace can contain a variety of objects. Some of these objects may be integrally connected to other objects, while others are totally independent. Some objects store data that is useful to applications, and other objects may only exist for the purposes of the DBA or developer. There are several basic types of objects which play a variety of roles in the dimensional model. In these respects, an analytic workspace is very similar to a relational schema.

The OLAP DML is the basic, low-level language for working in an analytic workspace. Tools are available in PL/SQL and Java that provide an interface to the OLAP DML for users already familiar with those languages.

See also [OLAP DML](#).

ancestor

A value at any level higher than a given value in a hierarchy. For example, in a Time dimension, the value 2002 might be the ancestor of the values Q1-02 and Jan-02. In a

dimension hierarchy, the data value of the ancestor is the aggregated value of the data values of its descendants.

Contrast with [descendant](#). See also [hierarchy](#), [level](#), [parent](#).

attribute

A descriptive characteristic of either a single dimension member or a group of dimension members. When applied to a single member, attributes provide supplementary information that can be used for display (such as a descriptive name) or in analysis (such as the number of days in a time period). When applied to a group, attributes represent logical groupings that enable users to select data based on like characteristics. For example, in a database representing footwear, you can use a shoe color attribute to select all boots, sneakers, and slippers that share the same color.

base level data

Data at the lowest level, often acquired from another source, such as a transactional database.

Contrast with [aggregation](#).

cell

A single data value of an expression. In a dimensioned expression, a cell is identified by one value from each of the dimensions of the expression. For example, if you have a variable with the dimensions MONTH and DISTRICT, then each combination of a month and a district identifies a separate cell of that variable.

See also [dimension](#), [variable](#).

child

A value at the level under a given value in a hierarchy. For example, in a Time dimension, the value Jan-02 might be the child of the value Q1-2002. A value can be a child for more than one parent if the child value belongs to multiple hierarchies.

Contrast with [parent](#). See also [descendant](#), [hierarchy](#), [level](#).

composite

An analytic workspace object that lists dimension-value combinations (also called a tuple) 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.

See also [dimension](#), [sparsity](#), [variable](#).

container

See [object](#).

cube

A logical organization of measures with identical dimensions. The edges of the cube contain dimension members and the body of the cube contains data values. For example, sales data can be organized into a cube, whose edges contain values from the time, product, and customer dimensions and whose body contains Volume Sales and Dollar Sales data. In a star schema, a cube is represented by a fact table.

custom measure

A measure that is calculated at run-time and presented as one or more additional columns of data added to a result set. The result set includes a value for each dimension member currently in status. For example, a custom measure might calculate the difference in costs from the prior period by using the OLAP DML LAGDIF function on the COSTS measure. Another custom measure might calculate profits by subtracting the COSTS measure from the SALES measure.

See also [dimension member](#), [OLAP DML](#), [measure](#), [status](#).

custom member

A member of a dimension created at run-time and defined as the parent of one or more existing dimension members. The values of a measure for a custom member are calculated using the aggregation rules for that dimension.

See also [aggregation](#), [dimension member](#), [parent](#).

data source

A database, application, repository, or file that provides data.

data warehouse

A relational database that is designed for query and analysis rather than transaction processing. A data warehouse usually contains historical data that is derived from transaction data, but it can include data from other sources. It separates analysis workload from transaction workload and enables a business to consolidate data from several sources.

database standard form

An analytic workspace that has been constructed with a specific set of objects, such as hierarchy dimensions, level dimensions, parent relations, and level relations. Each object must be defined with a set of properties that identify its role and its relationships with other objects in the analytic workspace. The standard form is required for an analytic workspace to be accessible to OLAP tools, however, it is not a prerequisite for multidimensional analysis.

derived measure

A measure that is calculated from one or more stored measures. The calculated data may be stored in the analytic workspace, or it may be calculated entirely in response to a query.

See also [custom measure](#).

DBA

Database administrator. The person responsible for creating, installing, configuring and maintaining Oracle Databases.

definition

The description of an analytic workspace object. An object definition includes characteristics such as the object's name, type (for example, dimension or variable), data type, dimensions, long description, permission specifications, and properties.

See also [dictionary](#), [object](#), [property](#).

denormalized

Permit redundancy in a table. Contrast with [normalize](#).

derived fact (or measure)

A fact (or measure) that is generated from existing data using a mathematical operation or a data transformation. Examples include averages, totals, percentages, and differences.

descendant

A dimension member at any level below a particular member in a hierarchy. The level immediately below is the child.

Contrast with [ancestor](#). See also [aggregation](#), [child](#), [hierarchy](#), [level](#).

dictionary

The collection of object definitions in an analytic workspace. The dictionary is also called the workspace dictionary.

See also [definition](#), [object](#).

dimension

A structure that categorizes data. Among the most common dimensions for sales-oriented data are time, geography, and product. Most dimensions have hierarchies.

In an analytic workspace, a dimension is a container for a list of values. A dimension acts as an index for identifying the values of a variable. For example, if sales data has a separate sales figure for each month, then the data has a month dimension; that is, the data is organized by month.

In SQL, a dimension is a type of object that defines hierarchical (parent/child) relationships between pairs of column sets.

See also [hierarchy](#).

dimension member

One element in the list that makes up a dimension. Also called a dimension value. A computer company might have dimension members in the product dimension called LAPPC and DESKPC. Members in the geography dimension might include Boston and Paris. Members in the time dimension might include NOV02, DEC02, JAN03, FEB03, MAR03, and so forth.

dimension table

A relational table that stores all or part of the values for a logical dimension in a star or snowflake schema. Dimension tables describe the business entities of an enterprise, represented as hierarchical, categorical information such as time, departments, locations, and products. They are sometimes called lookup or reference tables.

dimension value

See [dimension member](#).

dimension view

A relational view of data in an analytic workspace that contains the same types of data as a dimension table in a star schema, that is, columns for dimension members and attributes. A dimension view typically lists all dimension members in the key column, regardless of their level in the dimension hierarchy.

See also [dimension table](#), [star schema](#).

drill

To navigate from one item to a set of related items. Drilling typically involves navigating up and down through the levels in a hierarchy. When selecting data, you can expand or collapse a hierarchy by drilling down or up in it, respectively.

drill down

To expand the view to include child values that are associated with parent values in the hierarchy.

drill up

To collapse the list of descendant values that are associated with a parent value in the hierarchy.

edge

A set of one or more dimensions that are displayed together in a cube or document. Although there is no limit to the number of edges in a cube, data is often organized for display purposes along three edges, which are referred to as the row edge, the column edge, and the page edge.

In a cross-tab report, dimension members on the row edge appear in the first column and identify the rows, dimension members on the column edge appear in the first row and identify the columns, and dimension members on the page edge label the individual pages of the report.

See also [cube](#).

EIF file

A specially formatted file for transferring data between analytic workspaces. Using the OLAP DML, you can create an EIF file using the `EXPORT` command and read an EIF file using the `IMPORT` command.

embedded total

A predefined level of aggregation built into a dimension for which a hierarchy exists. For example, in a time dimension, each quarter represents the total for the months in the quarter. Data for embedded totals is calculated in the analytic workspace by the aggregation system.

See also [aggregation](#), [dimension](#), [hierarchy](#).

fact

See [measure](#). See also [additive](#), [derived fact \(or measure\)](#).

fact table

A table in a star schema that contains facts. A fact table typically has two types of columns: those that contain facts and those that are foreign keys to dimension tables. The primary key of a fact table is usually a composite key that is made up of all of its foreign keys.

A fact table might contain either detail level facts or facts that have been aggregated. Fact tables that contain aggregated facts are typically called summary tables or materialized views. A fact table usually contains facts with the same level of aggregation.

family relation

An analytic workspace relation object that identifies the complete parentage of each dimension member. A family relation has at least two dimensions: the data dimension

and a level dimension. The contents of the relation identify, for each dimension member, the ancestor at each level of the hierarchy.

See also [ancestor](#), [level](#), [relation](#).

formula

A type of workspace object that represents a stored calculation, expression, or procedure that produces a value. A formula provides a way to define and save complex or frequently used relationships within the data without storing the result set. Each time you query a formula, the OLAP engine performs the calculation or procedure that is required to produce the value.

hierarchy

A logical structure that uses ordered levels as a means of organizing data. A hierarchy can be used to define data aggregation; for example, in a time dimension, a hierarchy might be used to aggregate data from the month level to the quarter level to the year level. A hierarchy can be used to define a navigational drill path, regardless of whether the levels in the hierarchy represent aggregated totals.

In PL/SQL, hierarchies can be defined as part of a dimension object.

level

A position in a hierarchy. For example, a time dimension might have a hierarchy that represents data at the month, quarter, and year levels.

level relation

An analytic workspace relation object that identifies the level of each dimension member.

See also [level](#), [relation](#).

mapping

The definition of the relationship and data flow between source and target objects.

materialized view

A precomputed relational table comprising aggregated or joined data from fact and possibly dimension tables. Also known as a summary or aggregate table.

measure

Data that can be examined and analyzed, such as sales or cost data. You can select and display the data in a measure. Measures can be stored as variables or relations, or measures can be calculated by means of formulas. The terms **measure** and **fact** are synonymous; measure is more commonly used in a dimensional environment and fact is more commonly used in a relational environment.

There are both base measures and custom measures. Base measures, such as Volume Sales and Dollar Sales, are stored. Custom measures, such as Volume Share Year Ago, are calculated from base measures.

See also [formula](#), [relation](#), [variable](#).

measure view

A relational view of data in analytic workspace that contains the same types of data as a fact table in a star schema. However, in addition to the base-level facts, a measure view also contains derived data, such as aggregates and inter-row calculations.

See also [fact table](#), [star schema](#).

metadata

Data that describes data and other structures, such as objects, business rules, and processes.

See also [OLAP Catalog](#).

model

A type of analytic workspace object that contains a set of interrelated equations, which are used to calculate data and assign it to a variable or dimension value. Models are used frequently when working with financial data.

See also [dimension member](#), [object](#), [variable](#).

NA value

A special data value that indicates that data is "not available" (NA). It is the value of any cell to which a specific data value has not been assigned or for which data cannot be calculated.

See also [cell](#), [sparsity](#).

nonadditive

Describes a fact (or measure) that cannot be summarized through addition, such as average. Contrast with [additive](#), [semi-additive](#).

normalize

In a relational database, the process of removing redundancy in data by separating the data into multiple tables. Contrast with [denormalized](#).

object

In an analytic workspace, a distinct item in the workspace dictionary. Analytic workspaces consist of one or more objects, such as variables, formulas, dimensions, relations, and programs, which are used to organize, store, and retrieve data. Each object is created with a particular object type and stores a particular type of information. Objects that are the same type (for example, three variables) can have different roles within the analytic workspace.

See also [role](#).

object type

In Oracle object technology, a form of user-defined data type that is an abstraction of a real-world entity. An object type is a schema object with the following components:

- A name, which identifies the object type uniquely within a schema
- Attributes, which model the structure and state of the real-world entity
- Methods, which implement the behavior of the real-world entity, in either PL/SQL or Java

OLAP Catalog

A metadata package consisting of a set of read and write APIs that describe data in dimensional terms, such as cubes, measures, dimensions, and attributes.

See also [metadata](#).

OLAP DML

The low-level data definition and manipulation language for analytic workspaces.

on the fly

Calculated at run-time in response to a specific query. In an analytic workspace, custom measures and custom members are typically calculated on the fly. Aggregate data can be precalculated, calculated on the fly, or a combination of the two methods.

Contrast with [precalculate](#).

online analytical processing (OLAP)

Functionality characterized by dynamic, dimensional analysis of historical data, which supports activities such as the following:

- Calculating across dimensions and through hierarchies
- Analyzing trends
- Drilling up and down through hierarchies
- Rotating to change the dimensional orientation

online transaction processing (OLTP)

Systems optimized for fast and reliable transaction handling. Compared to data analysis systems, most OLTP interactions involve a relatively small number of rows, but a larger group of tables.

parent

A dimension member at the level immediately above a particular member in a hierarchy. In a dimension hierarchy, the data value of the parent is the aggregated total of the data values of its children.

Contrast with [child](#). See also [hierarchy](#), [level](#).

parent-child relation

A one-to-many relationship between one parent and one or more children in a hierarchical dimension. For example, New York (at the state level) might be the parent of Albany, Buffalo, Poughkeepsie, and Rochester (at the city level).

See also [child](#), [parent](#).

parent relation

An analytic workspace relation object that defines a dimension's hierarchies by storing the parent of each dimension member.

See also [parent](#), [relation](#).

precalculate

Calculated and stored as a data maintenance procedure. In an analytic workspace, aggregate data can be precalculated, calculated on the fly, or a combination of the two methods.

Contrast with [on the fly](#).

program

A type of database object that contains a series of OLAP DML commands. A program executes a set of related commands. Programs can be nested, with one calling another. A program can return a value; in this case, it is called a user-defined function.

See also [object](#).

property

A characteristic of an object or component. Properties provide identifiers and descriptions, define object features (such as the number of decimal places or the color), or define object behaviors (such as whether an object is enabled). Properties are used extensively in standard form analytic workspaces.

See also [object](#).

QDR

See [qualified data reference](#).

qualified data reference

A qualifier that limits one or more dimensions to a single value for the duration of an OLAP DML command. A QDR is useful when you want to temporarily reference a value without affecting the current status. In the following example of an OLAP DML command, the QDR limits the MONTH dimension to JUN02.

```
SHOW sales(month 'JUN02')
```

See also [dimension](#), [dimension member](#), [status](#).

relation

A type of workspace object that is similar to a variable, except that it restricts its data values to the members of a particular dimension (such as PRODUCT) instead of to a particular data type (such as NUMBER). A relation establishes a correspondence between the values of a given dimension and the values of that dimension or other dimensions in the database.

For example, you might have a relation between cities and sales regions, such that each city belongs to a particular region. In a relation between cities and sales regions, the relation is dimensioned by CITY. Each cell holds the corresponding value of the REGION dimension.

See also [cell](#), [dimension](#), [dimension member](#), [variable](#).

role

The function of a workspace object within its broader categorization of object type. For example, a variable that stores numeric business measures has a role of measure. A variable that stores descriptive product names has a role of attribute. Both are variables, but they contain different types of information and play different roles in the dimensional model.

See also [object](#).

rollup form

A table that displays the full ancestry of each dimension member within a row. The table provides a column for each level of the hierarchy.

For example, a row for base-level dimension member Florence has FLORENCE in the City column, ITALY in the Country column, and EUROPE in the Region column. A row for Italy has null in the City column, ITALY in the Country column, and EUROPE in the Region column.

Contrast with [embedded total](#). See also [ancestor](#), [dimension member](#), [hierarchy](#).

schema

A collection of related database objects. Relational schemas are grouped by database user ID and include tables, views, and other objects. Multidimensional schemas are

called analytic workspaces and include dimensions, relations, variables, and other objects.

See also [analytic workspace](#), [snowflake schema](#), [star schema](#).

semi-additive

Describes a fact (or measure) that can be summarized through addition along some, but not all, dimensions. Examples include head count and on-hand stock.

Contrast with [additive](#), [nonadditive](#).

snowflake schema

A type of star schema in which the dimension tables are partly or fully normalized.

See also [normalize](#), [schema](#), [star schema](#).

solved data

A result set in which all derived data has been calculated. Data fetched from an analytic workspace is always fully solved, because all of the data in the result set is calculated before it is returned to the SQL-based application. The result set from the analytic workspace is the same whether the data was precalculated or calculated on the fly.

See also [on the fly](#), [precalculate](#).

source

A database, application, file, or other storage facility from which the data in a data warehouse is derived.

sparsity

A concept that refers to multidimensional data in which a relatively high percentage of the combinations of dimension values do not contain actual data. Such "empty," or NA, values can take up storage space in an analytic workspace. To handle sparse data efficiently, you can create a composite.

There are two types of sparsity.

- Controlled sparsity occurs when a range of values of one or more dimensions has no data; for example, a new variable dimensioned by month for which you do not have data for past months. The cells exist because you have past months in the month dimension, but the cells contain NA values.
- Random sparsity occurs when NA values are scattered throughout the variable, usually because some combinations of dimension values never have any data. For example, a district might only sell certain products and never have data for other products. Other districts might sell some of those products and other ones, too.

See also [composite](#), [NA value](#).

standard form

See [database standard form](#).

star query

A join between a fact table and a number of dimension tables. Each dimension table is joined to the fact table using a primary key to foreign key join, but the dimension tables are not joined to each other.

star schema

A relational schema whose design represents a dimensional data model. The star schema consists of one or more fact tables and one or more dimension tables that are related through foreign keys.

See also [schema](#), [snowflake schema](#)

status

The list of currently accessible values for a given dimension. If the status of a given dimension is limited to a subset of its stored values, then all expressions that are based on that dimension will be limited to the corresponding subset of data. The status of a dimension persists within a particular session, and does not change until it is changed deliberately. When an analytic workspace is first attached to a session, all members are in status.

See also [dimension](#), [dimension member](#).

summary

See [aggregation](#), [materialized view](#).

update window

The length of time available for loading new data into your database.

valueset

A type of workspace object. A valueset contains a list of dimension members for a particular dimension. After defining a valueset, you use the LIMIT command to assign members from the dimension to the valueset. The values in a valueset can be saved across Oracle OLAP sessions.

When you begin a new Oracle OLAP session or start up a workspace, each dimension has all values in status. You can then limit a dimension to the values stored in the valueset for that dimension.

See also [dimension](#).

variable

A type of workspace object that stores data. The data type of a variable indicates the kind of data that it contains, such as numeric or text data.

If a variable has dimensions, then those dimensions organize its data, and there is one cell for each combination of dimension members. A dimensioned variable is an array whose cells are individual data values. If a variable has no dimensions, then it is a single-cell variable, which contains one data value.

See also [cell](#), [dimension](#), [dimension member](#), [object](#).

Index

A

- access rights, 6-5
- active catalogs, 5-3
- ALL_ATTRIBUTES dimension, A-22
- ALL_ATTRTYPES dimension, A-23
- ALL_CUBES dimension, A-20
- ALL_DESCRIPTIONS variable, A-27
- ALL_DESCTYPES dimension, A-23
- ALL_DIMENSIONS dimension, A-20
- ALL_HIERARCHIES dimension, A-21
- ALL_LANGUAGES dimension, A-23
- ALL_LEVELS dimension, A-21
- ALL_MEASURES dimension, A-20
- ALL_OBJECTS dimension, A-22
- allocation, B-4
- ALTER SESSION commands, 6-7
- analysis tools
 - described, 1-3
- analytic capabilities
 - compared, 1-10
- Analytic Workspace Java API, 5-7, 7-3
- Analytic Workspace Manager, 3-1 to 3-11
- analytic workspaces
 - basic process overview, 1-13
 - basic steps for creating, 3-11
 - best practices, 1-9
 - database storage, 6-9
 - defined, 1-7, 1-11
 - standard form, 3-10
- applications
 - differences from Express, B-3
- ATTRDEF object, A-18
- attributes
 - See Also* dimension attributes
 - See Also* level attributes
 - creating logical, 7-5, 7-9
 - logical, 1-7
- authentication, 6-4
- AW\$ tables, 6-11
- AW\$CLASS property, A-6
- AW\$CREATEDBY property, A-6
- AW\$LASTMODIFIED property, A-6
- AW\$STATE property, A-7
- AW_NAMES variable, A-26
- AWXML package, 5-7, 7-3

B

- BFILE security, 6-8
- BI Beans
 - described, 5-2, 5-3
 - relational data source, 5-7
 - relational data sources, 5-7
 - See also* OLAP API

C

- caches
 - use in iterative queries, 5-7
- calculation engine
 - defined, 1-11
- catalogs class
 - database standard form, A-19
- character sets, B-2
- CHARSET option, B-3
- classes
 - database standard form, A-4
- COMMIT command, B-5
- configuration procedures, 6-1
- conjoins, B-4
- CONNECT role, 6-5
- connect string
 - for Analytic Workspace Manager, 3-5
- CREATE_DB_STDFORM program, B-5
- crosstab bean, 5-4
- CUBE_MEASURES valueset, A-24
- CUBEDEF dimension, A-12
- cubes, 1-8
 - defined, 7-5
- cursors, 5-7
- custom measures
 - BI Beans support, 5-5
- CWM2
 - write APIs, 7-9
- CWM2_OLAP_CATALOG package, 7-9
- CWM2_OLAP_CUBE package, 7-9
- CWM2_OLAP_DIMENSION package, 7-9
- CWM2_OLAP_HIERARCHY package, 7-9
- CWM2_OLAP_LEVEL package, 7-9
- CWM2_OLAP_LEVEL_ATTRIBUTE package, 7-9
- CWM2_OLAP_MEASURE package, 7-9

CWM2_OLAP_PC_TRANSFORM package, 7-9
CWM2_OLAP_VALIDATE package, 7-10

D

data formatting, 5-4
data models, 1-5
data stores
 described, 1-9
data striping, 6-2
database configuration, 6-1
database security, 6-4
database standard form
 described, A-1
 specification, A-1 to A-30
DEFAULT_HIER relation, A-27
demand planning systems, 1-3
DIM_ATTRIBUTES relation, A-25
DIM_HIERARCHIES relation, A-24
DIM_LEVELS relation, A-25
DIM_LEVELS valueset, A-25
DIMDEF dimension, A-15
dimension hierarchies
 See hierarchies
dimension order
 basic rules, 3-19
dimension tables
 defining metadata, 7-5
dimensions
 creating logical, 7-5
 logical, 1-6, 7-5
 time, 7-5
directories
 database, 6-7
directory object, 8-6
drilling, 5-3
dynamic performance tables, 6-11

E

EDDE.HIERMNT program (obsolete), B-10
EDDE.MSG program (obsolete), B-9
EIF files, B-5
Excel add-in, 1-3
Express Connection Utility (obsolete), B-3
Express databases
 converting to standard form, B-5
Express Relational Access Administrator
 (obsolete), B-2
Express Relational Access Manager (obsolete), B-2
EXTCALL (obsolete), B-4
extensions class
 database standard form, A-30

F

fact tables
 defining metadata, 7-5
features class
 database standard form, A-27

files
 allowing access, 6-7
financial applications, 1-3
forecasting commands
 OLAP DML, 4-1
formatting
 data, 5-4

G

Global Computing Company
 data requirements, 2-2 to 2-7
Global star schema, 2-7
GLOBAL_AW user
 defining, 3-27
globalization, B-2
Globalization Technology *See* NLS
graph bean, 5-4

H

HIER_LEVELS valueset, A-18
hierarchies
 creating logical, 7-5
 logical, 1-7
HIERLIST dimension, A-16

I

IDE
 defined, 5-2
implementation class
 database standard form, A-11
initialization parameters, 6-7
init.ora file, 6-6

J

Java
 described, 5-1
 sandbox security, 5-2
Java APIs for OLAP, 5-1 to 5-8
JDeveloper, 5-2
JOB_QUEUE_PROCESSES parameter, 6-6

L

language support, B-2
LEVELLIST dimension, A-16
LEVELREL relation, A-17
levels
 creating logical, 7-5
 logical, 1-7
localization, B-2
login names, 6-4

M

materialized views
 CWM2, 8-5, 8-6, 8-7
 grouping sets, 8-5 to 8-13

- MDI
 - defined, 5-2
- MEASUREDEF object, A-13
- measures
 - custom, 5-5
 - logical, 1-6
- MEMBER_CREATEDBY variable, A-29
- MEMBER_FAMILYREL relation, A-29
- MEMBER_INHIER variable, A-28
- modeling support, B-4
- MOLAP
 - analytic operations, 1-10
 - defined, 1-9
- multibyte character sets
 - Express equivalents, B-3
- multidimensional data, 1-8

N

- naming conventions
 - database standard form, A-5
- NLS_LANG configuration parameter, B-2
- n-pass functions, 5-6
- number formatting, 5-4

O

- object-oriented programming, 5-5
- ODBC (obsolete), B-4
- ODBC support (obsolete), B-2
- oescmd program (obsolete), B-1
- oesmgr program (obsolete), B-1
- OLAP
 - defined, 1-2
- OLAP API
 - described, 1-12, 5-2, 5-5
 - relational data source, 5-7
 - See also* BI Beans
- OLAP Beans, 5-3, 5-4
- OLAP Catalog
 - described, 1-11
 - metadata model tables, 7-3
 - read APIs, 7-3
 - write APIs, 7-3
- OLAP DML
 - described, 1-12
- OLAP Instance Manager (obsolete), B-1
- OLAP Management tool, 7-4
- OLAP metadata
 - creating in Enterprise Manager, 7-4
 - creating with CWM2 APIs, 7-9
 - materialized views, 8-2
 - steps for creating, 7-4
- OLAP Worksheet
 - described, 3-3
 - session sharing, 3-1
- OLTP
 - defined, 1-2
- operating system commands (obsolete in OLAP

- DML), B-4
- optimization techniques, 6-2

P

- paging, 5-4
- parameter file, 6-6
- parent-child relations
 - described, 1-7
- PARENTREL relation, A-17
- partitioning, 6-9
- performance counters, 6-11
- Personal Express (obsolete), B-3
- pfile settings, 6-6
- PGA_AGGREGATE_TARGET parameter, 6-6
- PGA_TARGET parameter, 6-6
- pivoting, 5-3
- predictive analysis applications, 1-3
- Presentation Beans, 5-3
- PS\$ tables, 6-11

Q

- query builder, 5-5
- QUERY REWRITE system privilege, 6-5
- query tools
 - described, 1-3

R

- random sparsity, definition, 3-17
- regressions
 - OLAP DML, 4-1
- Relational Access Administrator (obsolete), B-2
- Relational Access Manager (obsolete), B-2
- relational data sources, 5-7, 8-1
- relational schema
 - best practices, 1-9
- result sets, 5-7
- ROLAP
 - analytic operations, 1-10
 - defined, 1-9
- roles, 6-5
- ROLLUP command, B-4, B-6

S

- schemas
 - star, snowflake, 7-4, 8-2
- SELECT privilege, 6-5
- server parameter file, 6-6
- session sharing, B-4
- SESSIONS parameter, 6-6
- SNAPI (obsolete), B-4
- SNAPI communications (obsolete), B-4
- sparsity characteristics, 3-28
- sparsity pattern, definition, 3-17
- Spreadsheet Add-In
 - described, 1-3
- SQL analytic operations, 1-10
- SQL command (OLAP DML), B-2

- SQL interface, 1-12
- standard form
 - see* database standard form
- startup parameters
 - database, 3-35
- stoplight formatting, 5-4
- striping, 6-2
- summary data
 - creating, 1-8
- summary data methods
 - compared, 1-10
- system properties
 - database standard form, A-6

T

- tablespaces
 - defining, 3-35
 - for analytic workspaces, 6-2
- Time attributes
 - creating for converted Express databases, B-8
- time dimensions, 7-5
- translation tables, B-3

U

- UNDO_MANAGEMENT parameter, 6-6
- UNDO_TABLESPACE parameter, 6-6
- Unicode, B-3
- UPDATE command, B-5
- user access rights, 6-5
- user names, 6-4
- UTL_FILE_DIR parameter, 6-6, 8-6

V

- VISIBLE variable, A-28

W

- wizards
 - BI Beans, 5-4

X

- XCA (obsolete), B-4
- XML metadata, 5-7, 7-3
- XPDDDATA database (obsolete), B-10