Oracle® OLAP

Application Developer's Guide,

10*g* Release 2 (10.2.0.3)

B14349-03

September 2006

Oracle OLAP Application Developer’s Guide, 10*g* Release 2 (10.2.0.3)

B14349-03

Contents

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. It also provides information for Oracle DBAs about managing resources for OLAP.

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 10*g* documentation set:

- *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.

For documentation about Oracle Business Intelligence, view the Web page at `http://www.oracle.com/technology/documentation/bi.html`.

Conventions

The following text conventions are used in this document:

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

What's New in Oracle OLAP Applications Development?

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

Oracle Database 10g Release 10.2.0.3 Oracle OLAP

Analytic Workspace Manager in Release 10.2.0.3 provides a Sparsity Advisor, which examines the source data and makes recommendations for defining OLAP cubes that will provide the best performance. The functionality of calculation plans has been enhanced to provide post-load forecasting, allocation, and aggregation. Analytic Workspace Manager also supports Java add-ins, so that any Java developer can extend and customize the tools in Analytic Workspace Manager.

See Also:

- Chapter 5 for information about the Sparsity Advisor and plug-ins
- Chapter 7 for information about aggregation
- Chapter 8 for information about allocation
- Chapter 9 for information about forecasting

Oracle Database 10*g* Release 10.2 Oracle OLAP

Oracle OLAP in Oracle Database 10*g* 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 5 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 10*g* 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 10*g*, 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 10*g* 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 5 for instructions on using the Model View

Database Standard Form 10*g*

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 10*g*.

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:

1 Overview

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

OLAP Technology Within Oracle Database

Multidimensional technology has been available in the Oracle database since Oracle9*i*. Each release since then has provided enhanced integration, functionality, and performance. 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 is available for developing custom applications, as described in Chapter 4.

See Also: *Oracle Business Intelligence Concepts Guide*, which is available at `http://www.oracle.com/technology/documentation/bi.html`

Formulating Queries

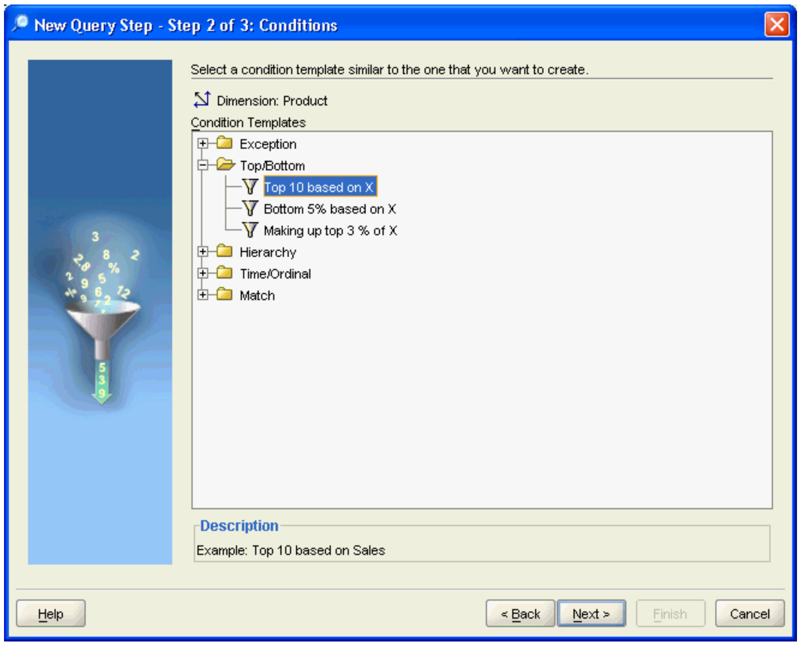
Both Discoverer Plus OLAP and Spreadsheet Add-In use a dimensional data model so that analysts can formulate their queries in the language of business. Dimensions provide the context for the data. Consider the following request for information:

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


New Query Step - Step 2 of 3: Conditions

Select a condition template similar to the one that you want to create.

Dimension: Product

Condition Templates

- Exception
- Top/Bottom
  - Top 10 based on X
  - Bottom 5% based on X
  - Making up top 3 % of X
- Hierarchy
- Time/Ordinal
- Match

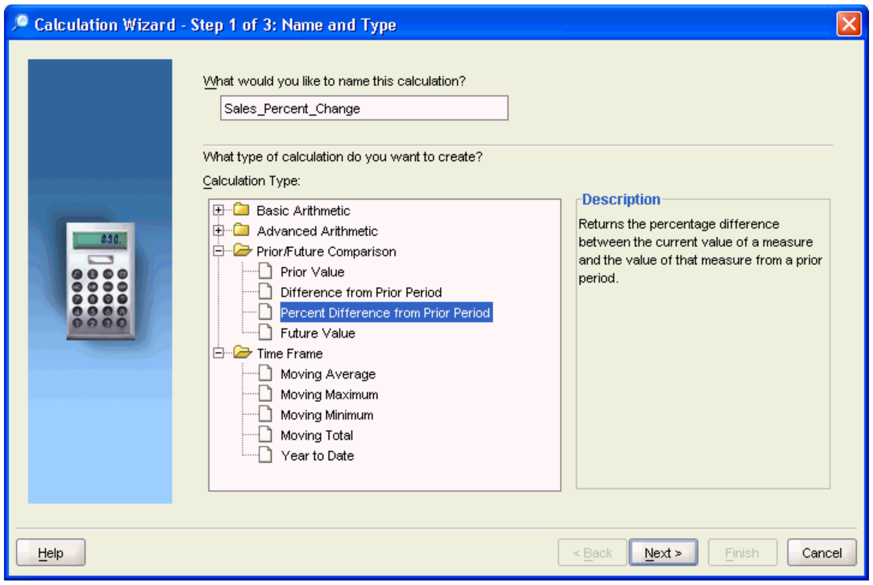
Description

Example: Top 10 based on Sales

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



Calculation Wizard - Step 1 of 3: Name and Type

What would you like to name this calculation?

Sales_Percent_Change

What type of calculation do you want to create?

Calculation Type:

- Basic Arithmetic
- Advanced Arithmetic
- 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

Description

Returns the percentage difference between the current value of a measure and the value of that measure from a prior 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.

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.

Structured 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.

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.

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. An analyst who has detected a pattern 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 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. 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.

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
OLAP DML
SQL Interface to OLAP

Analytic Workspace Java API
OLAP API

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 1-5. 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 from Oracle9*i* and Oracle Express Server.

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.

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.

If you are upgrading from Oracle Express or you plan to develop all the tools for working with analytic workspaces, then you may work directly in the OLAP DML. If you plan to use Oracle OLAP tools and applications, then do not work directly in the OLAP DML; your manual changes may invalidate the metadata.

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*.

Analytic Workspace Java API

The Analytic Workspace Java API supports the creation and maintenance of analytic workspaces in Java. It provides a programmatic method for defining a logical dimensional data model and instantiating that model in an analytic workspace. This API is 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 Oracle Database with the OLAP option.

See Also: *Oracle OLAP Java API Reference*

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 3 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 other schema, as described in Chapter 5. 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 5 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 calculated measures using the Calculation Wizard, which is described in "Creating Custom Measures" on page 1-4. In addition, you can create forecasts, allocations, and post-load aggregations.

Upgrading Oracle Database 10*g* Release 1 Analytic Workspaces

If you created an analytic workspace in Oracle 10*g* 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 Oracle9*i* Analytic Workspaces

If you have analytic workspaces that were created in Oracle9*i*, 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 Oracle9*i* Release 9.2.0.4.1).

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

If Oracle Database is running in 9*i* 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:

  ```
  EXECUTE DBMS_AW.CONVERT('aw_name');
  ```

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

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 1-10.

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.

2

The Logical Dimensional Data Model

This chapter describes the logical dimensional data model, which is used by Oracle OLAP. It consists of the following topics:

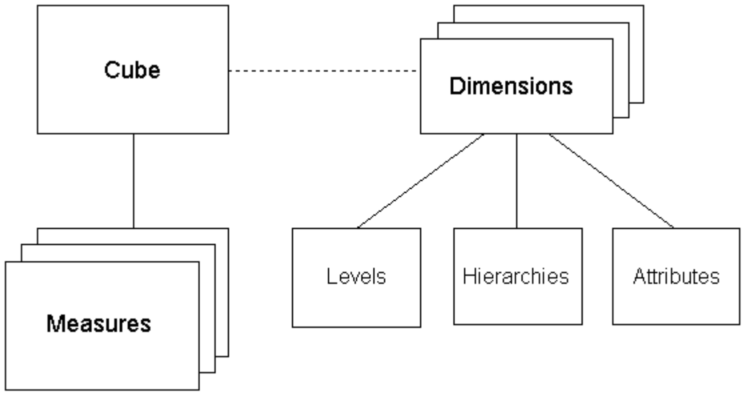
Overview of the Data Model

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 2–1 shows the general relationships among logical objects.

Figure 2–1 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, 2005, at 2:34 p.m. But they might want to find out which color of dress was most popular in the summer of 2005 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.

Level-Based Hierarchies

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, `Q1-05` and `Q2-05` are the children of `2005`, thus `2005` is the parent of `Q1-05` and `Q2-05`.

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.

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.

Value-Based Hierarchies

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. Likewise, the line item dimension of financial data does not have levels. This type of hierarchy is called a **value-based 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 a 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 2005? 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.

3 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

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 3-2 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 is 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 3-5 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 and install the Global schema from the Oracle Web site and use it to try the examples shown throughout this guide:

`http://www.oracle.com/technology/products/bi/olap/doc_sample_schemas/globalschemawithxml.html`

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 5 for schema diagrams.

4 Developing Java Applications for OLAP

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

- Building Analytical Java Applications
- Introducing OracleBI Beans
- Building Java Applications That Manage Analytic Workspaces

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 4-5.

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 UIX tag library.

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

Part II

Creating and Managing Analytic Workspaces

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

5 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

Your goal in using Analytic Workspace Manager is to create a multidimensional data store that supports business analysis. 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.

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

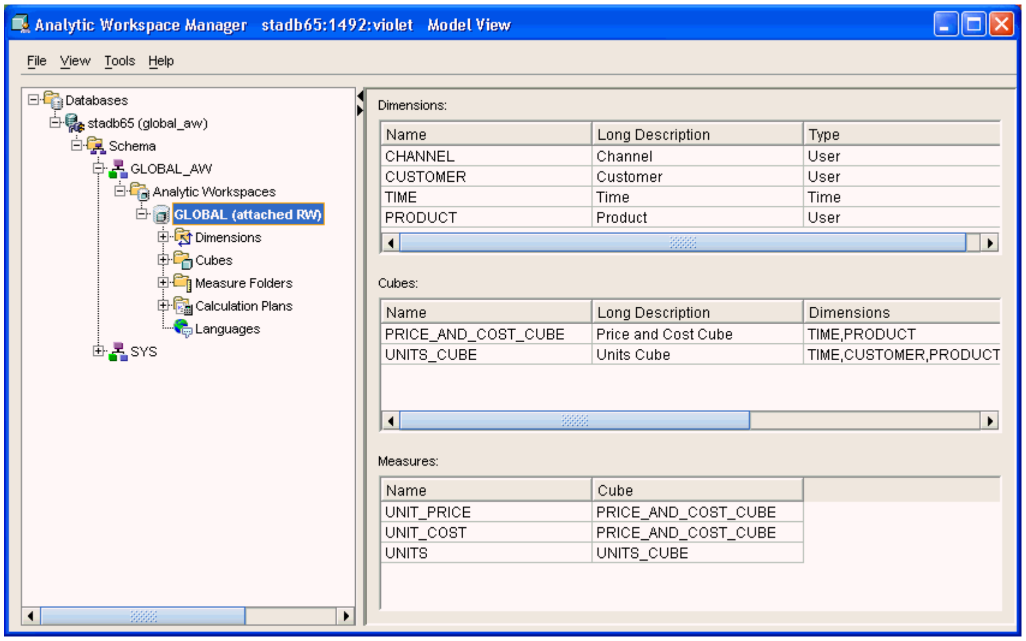
Model View

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, views, and synonyms in Oracle Database. The source columns can be star, snowflake, or any other schema design that supports the logical model.

Figure 5–1 shows the logical objects created in the `GLOBAL` analytic workspace.

Figure 5–1 Model View in Analytic Workspace Manager



Dimensions:

Name	Long Description	Type
CHANNEL	Channel	User
CUSTOMER	Customer	User
TIME	Time	Time
PRODUCT	Product	User

Cubes:

Name	Long Description	Dimensions
PRICE_AND_COST_CUBE	Price and Cost Cube	TIME,PRODUCT
UNITS_CUBE	Units Cube	TIME,CUSTOMER,PRODUCT

Measures:

Name	Cube
UNIT_PRICE	PRICE_AND_COST_CUBE
UNIT_COST	PRICE_AND_COST_CUBE
UNITS	UNITS_CUBE

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 from Express databases or modify custom applications. Do not use this view to manually change a standard form analytic workspace, because you may create inconsistencies in the metadata.

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

The most recent version of Analytic Workspace Manager is available for download from the Oracle Technology Network:

`http://www.oracle.com/technology/products/bi/olap/index.html`

Follow the installation instructions provided in the README file.

Opening Analytic Workspace Manager

On Windows, open Analytic Workspace Manager from the Start menu. Choose **Oracle - *Oracle_home***, then **Integrated Management Tools**, and then **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

The analytic workspace that you create must contain the logical objects described in Chapter 2. 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

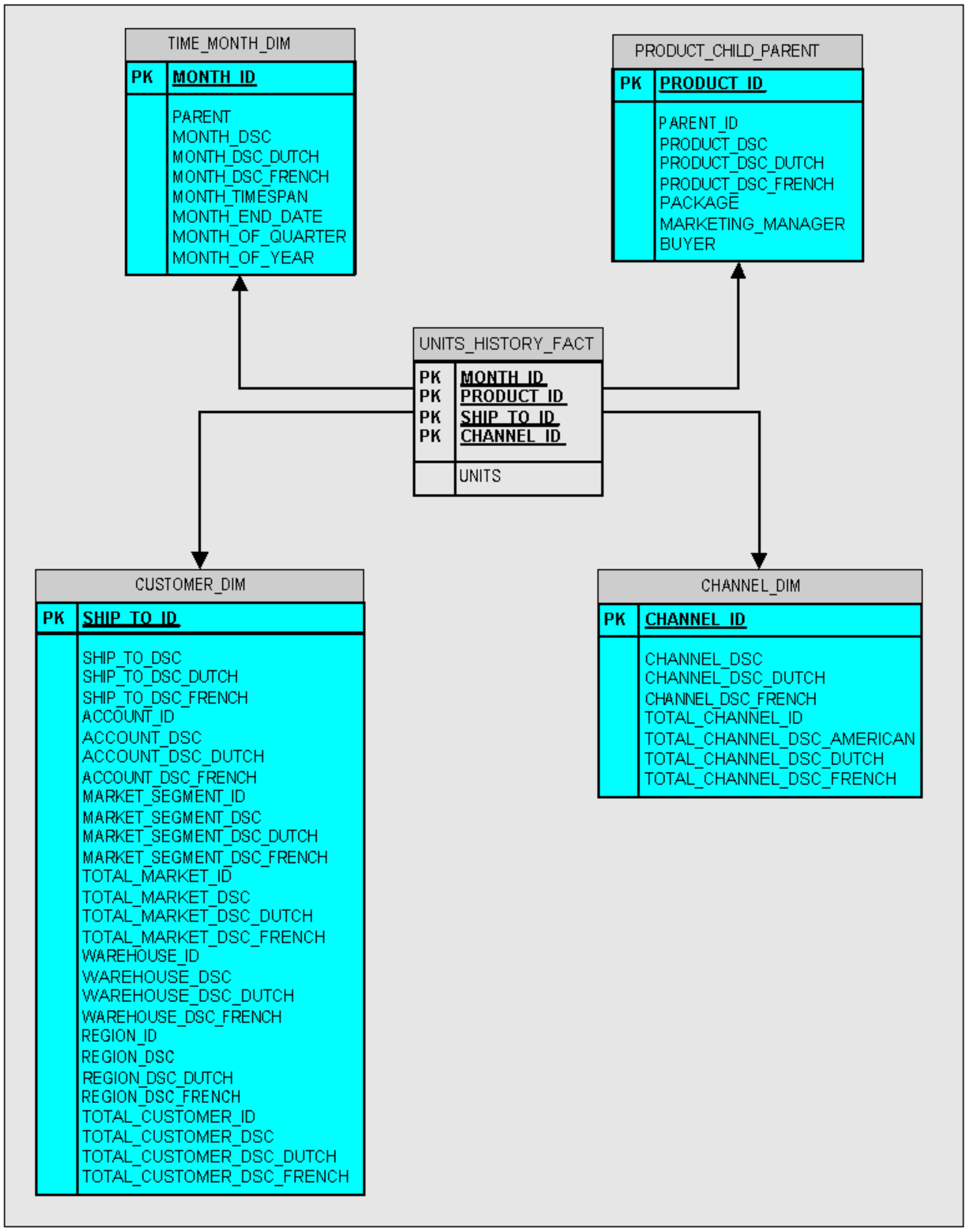
A star schema is the simplest of the three types. It is called a star schema because a diagram of this schema resembles a star, with points radiating from a central table. The center of the star is a fact table and the points of the star are the dimension tables.

- 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 5–2 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:

- `CUSTOMER_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 5-5.

Figure 5–2 Star Schema



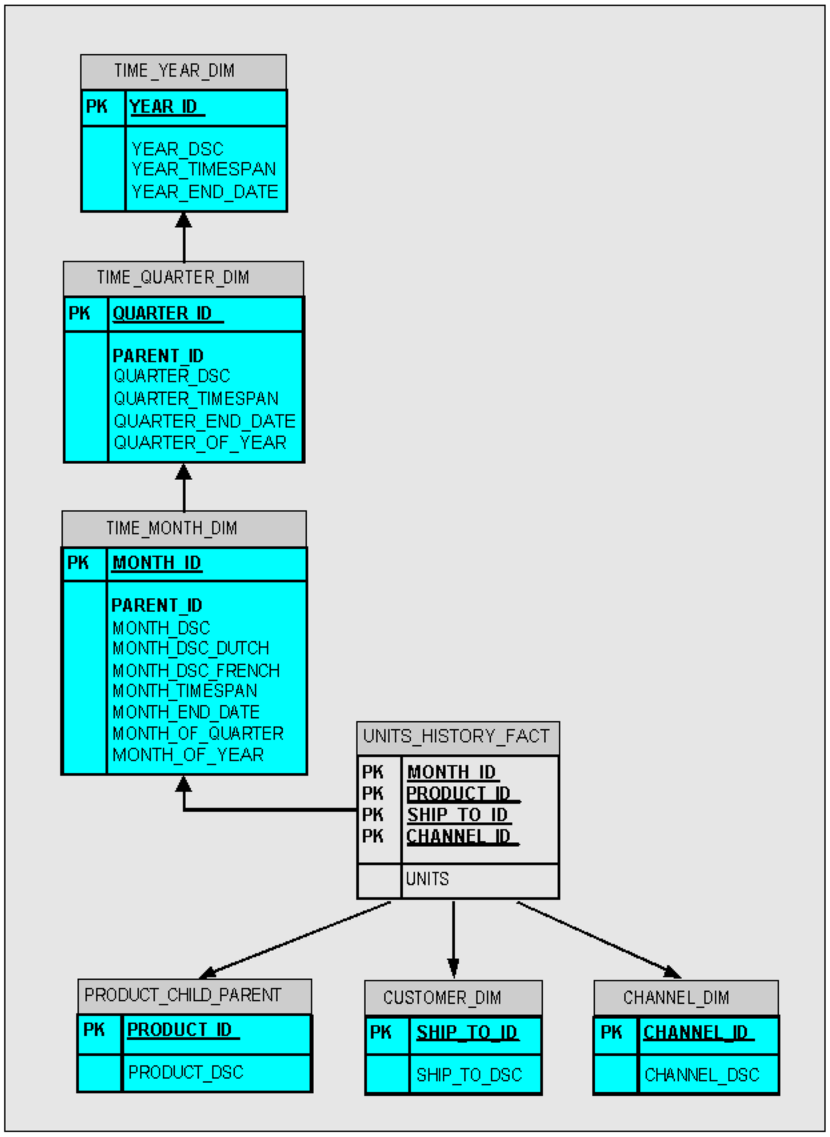
Snowflake Schema

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 5–3 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 5–3 Normalized Time Dimension in a Snowflake Schema



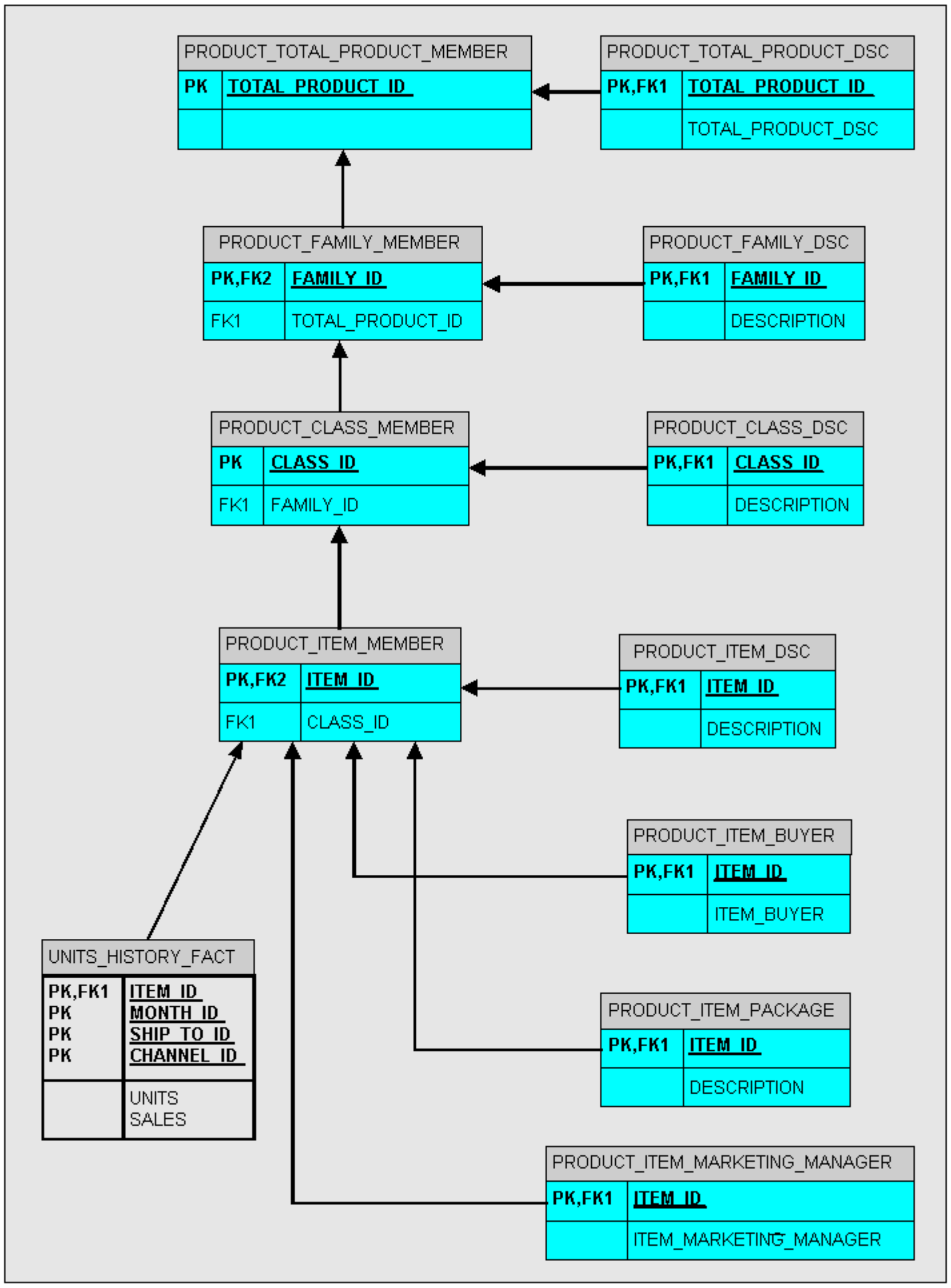
Other

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 5–4 shows the Product dimension in a schema that contains the appropriate relationships. Contrast these 11 tables with the single table shown in Figure 5–2 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 5–4 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 the analytic workspace.

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 multiple levels of data.** Analytic Workspace Manager 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 5-3.

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 an Analytic Workspace

To create an analytic workspace in database standard form:

1. Configure your database instance for OLAP use. Define permanent, temporary, and undo tablespaces as needed, 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.
3. Open Analytic Workspace Manager and connect to your database instance as the user you defined earlier for this purpose.
4. 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.
5. Define the logical dimensions for the data.

   See "Creating Logical Dimensions" on page 5-10.
6. Define the logical cubes for the data.

   See "Creating Logical Cubes" on page 5-13.
7. Map the logical items to their data sources.

   See "Mapping Logical Objects to Data Sources" on page 5-15.
8. Run the Sparsity Advisor to get recommendations for the physical implementation of the cubes.

   See "Using the Sparsity Advisor" on page 5-18.
9. Load the data.

   See "Maintaining the Data" on page 5-21.
10. Define measure folders to simplify access for end users.

    See "Defining Measure Folders" on page 5-22.

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 5-23.
- Develop one or more calculation plans for the analytic workspace. Calculation plans enable you to generate forecasts, allocate data down the dimension

hierarchies, aggregate data up the dimension hierarchies, and specify the order of these calculations.

See "Creating and Executing Calculation Plans" on page 5-23.

Creating Logical 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.

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.

You can define dimensions that have any of these common forms:

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

Dimension Members Must Be Unique

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 `CITY` column and a value of `NEW_YORK` in the `STATE` 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 because no levels are defined as metadata. 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 `CITY_NEW_YORK` and `STATE_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.

Time Dimensions Have Special Requirements

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.

You must define a Time dimension with at least one level to support time-based analysis, such as a custom measure that calculates the difference from the prior period.

To create a 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, which does not have named 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 OLAP client applications expect to find.

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-Defined Attributes

You can create additional "User" attributes that provide supplementary information about the dimension members.

To create a new 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.

The physical storage of the cube can have a great impact on performance. After you map the cube, you can run the Sparsity Advisor to get recommendations for the appropriate settings.

The measures inherit the characteristics of the cube. For cubes that use compressed composites to handle sparsity, you cannot override the default characteristics for individual measures. For cubes that use regular composites, you can use the inherited characteristics or override them with different choices.

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 except Implementation Details of the Create Cube dialog box.

   Important: After mapping the cube, run the Sparsity Advisor to see the recommended settings for the Implementation Details tab. For more information about the Summary To tab, refer to Chapter 7.

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.

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.

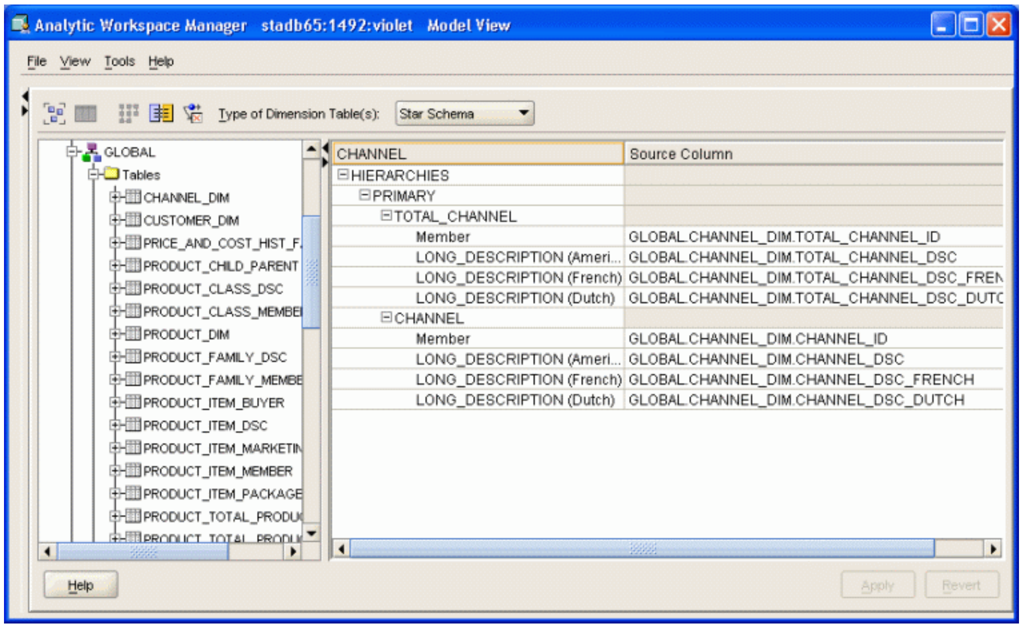
The mapping window has a tabular view and a graphical view.

- **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 5–5 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 5–5 Dimension Mapped in Tabular View



Type of Dimension Table(s): Star Schema

CHANNEL	Source Column
⊟HIERARCHIES	
⊟PRIMARY	
⊟TOTAL_CHANNEL	
Member	GLOBAL.CHANNEL_DIM.TOTAL_CHANNEL_ID
LONG_DESCRIPTION (Ameri...	GLOBAL.CHANNEL_DIM.TOTAL_CHANNEL_DSC
LONG_DESCRIPTION (French)	GLOBAL.CHANNEL_DIM.TOTAL_CHANNEL_DSC_FREN
LONG_DESCRIPTION (Dutch)	GLOBAL.CHANNEL_DIM.TOTAL_CHANNEL_DSC_DUTC
⊟CHANNEL	
Member	GLOBAL.CHANNEL_DIM.CHANNEL_ID
LONG_DESCRIPTION (Ameri...	GLOBAL.CHANNEL_DIM.CHANNEL_DSC
LONG_DESCRIPTION (French)	GLOBAL.CHANNEL_DIM.CHANNEL_DSC_FRENCH
LONG_DESCRIPTION (Dutch)	GLOBAL.CHANNEL_DIM.CHANNEL_DSC_DUTCH

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:

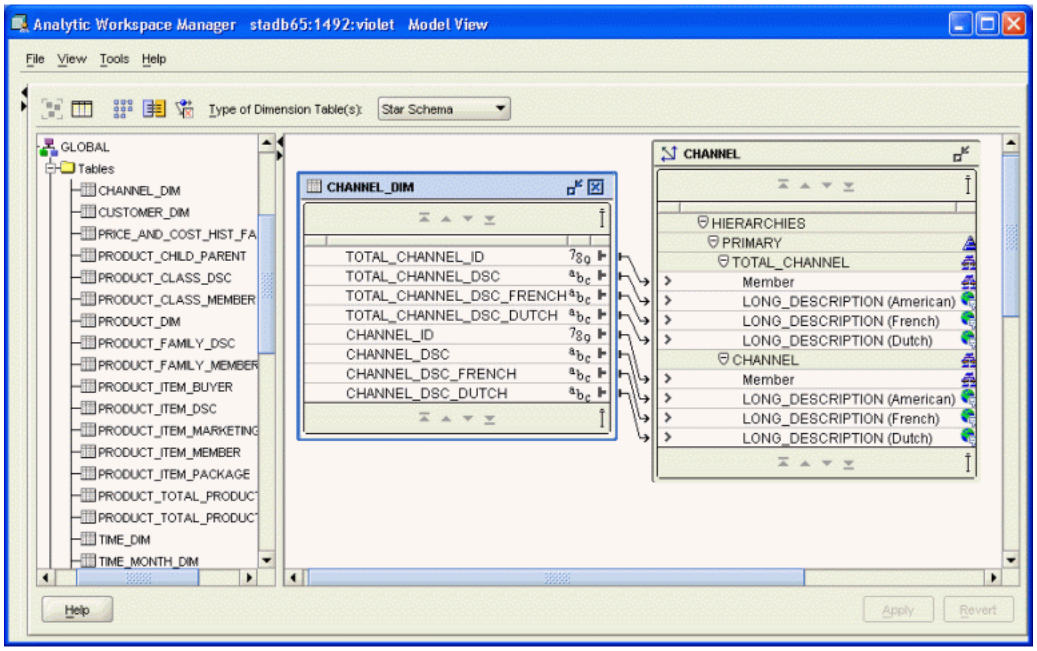
1. Define the dimension and its levels, hierarchies, and attributes.
2. In the Model View navigation tree, expand the dimension folder and click **Mappings**.

   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 5–6 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 5–6 GLOBAL CHANNEL Dimension Mapped in Graphical View



Type of Dimension Table(s): Star Schema

Mapping Cubes

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

1. Define the cube and its measures.

   You can define calculated 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.

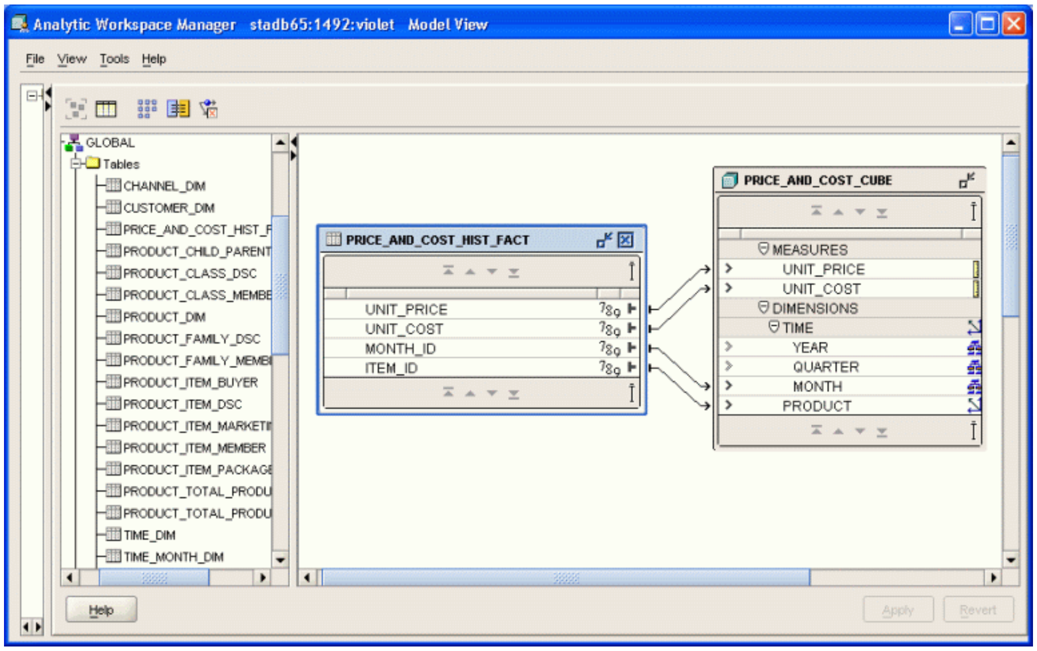
   To draw a line, click the output connector of the source column and drag it to the input connector of the target object. You must map both the measures and the related dimension keys.

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 5–7 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 5–7 GLOBAL PRICE_AND_COST_CUBE Cube Mapped in Graphical View



Using the Sparsity Advisor

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 Sparsity Advisor in Analytic Workspace Manager evaluates the data in the relational tables and recommends appropriate settings. You can accept all the recommendations or modify them before implementing them in the cube.

To run the Sparsity Advisor:

1. Create a cube and map it to a relational data source.
2. In the navigation tree, right-click the cube and choose Sparsity Advisor.

   Wait while the Sparsity Advisor analyzes the cube. When it is done, the Sparsity Advisor for Cube dialog box displays the recommendations.
3. For compressed cubes, be sure to select a data type for the cube. In most cases, `DECIMAL` is the best choice.

   For a discussion of `NUMBER` and `DECIMAL` data types, refer to "Keep Within Allocated Resources" on page 7-8.
4. Look over the recommendations and make any changes.
5. Click **Recreate Cube** to implement the recommendations.

   These changes cannot be made just by editing the definitions of objects in the analytic workspace. The objects are deleted and re-created, and any data stored in them is lost in the process.

The following topics provide information that will help you evaluate the recommendations of the Sparsity 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.

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

- Dense cubes have up to 20% empty cells. Dense data should be stored directly in the dimensional format.

  Dense cubes are extremely rare. If you know that a cube is dense, then do not use the Sparsity Advisor, because it is optimized for use on sparse data.

- Sparse cubes have more than 20% empty cells. Sparse data should be stored in a special format called a composite.
- Extremely sparse cubes often display these characteristics:
  - The 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.

  Extremely sparse data should be stored in another special format called a compressed composite. Compressed storage for this type of sparsity uses less space and results in faster aggregation than normal sparse storage.

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. You can easily project the sparsity of a dimensional cube from the relational source tables. It is the number of rows in the fact table with data values divided by the product of the unique keys in the dimension tables.

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. Performance is optimized when values that are accessed together are stored together, because fewer pages must be swapped in and out of memory.

For regular composites and dimensional storage, order the dimensions from largest to smallest.

For compressed composites, order the dimensions from smallest to largest.

Choosing a Data Type

The most commonly used data types are `NUMBER` and `DECIMAL`.

The `NUMBER` data type provides the same results on all platforms. All calculations are performed as integer arithmetic. Because of this, results based on a `NUMBER` will match those stored in relational tables. Choose `NUMBER` when you need a high level of precision, or when you need to match values. An unscaled `NUMBER` value is 22 bytes.

The `DECIMAL` data type is smaller at 8 bytes for each value and thus takes up less disk space than `NUMBER` data types. All calculations are done in the CPU Floating Point Unit, which is many times faster than integer arithmetic. However, floating point calculations produce slightly differently results on different platforms, typically at the seventh decimal place. `DECIMAL` is the best choice when the analytic workspace will be used heavily for computations.

Choosing Composite Types

A composite is an index into one or more sparse measures, and is used to store sparse data in a compact form. There are two types of composites: regular and compressed. A regular composite is used to store measures with moderate sparsity, and compressed composites are used to store measures with extreme sparsity.

For a cube using regular composites and partitions, you can choose between creating a single, global composite for use by all partitions and creating a composite for each partition. When in doubt, do not use global composites. The cube will have one composite for each partition.

Partitioning Large Measures

Partitioning is a 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.

Effects of Partitioning on Performance

The number of partitions 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 5-21.

Choosing a Dimension for Partitioning

The Sparsity Advisor can operate at a very granular level by assigning each dimension member to a particular partition. If you want to override its recommendations, you must choose a dimension with a level-based hierarchy and use one of its levels as the basis for creating the partitions.

Some enterprises redeploy their analytic workspaces with new data instead of maintaining them. When life-cycle maintenance is not a factor, you may choose the most dense dimension for partitioning. The most dense dimension is frequently the one with the fewest members.

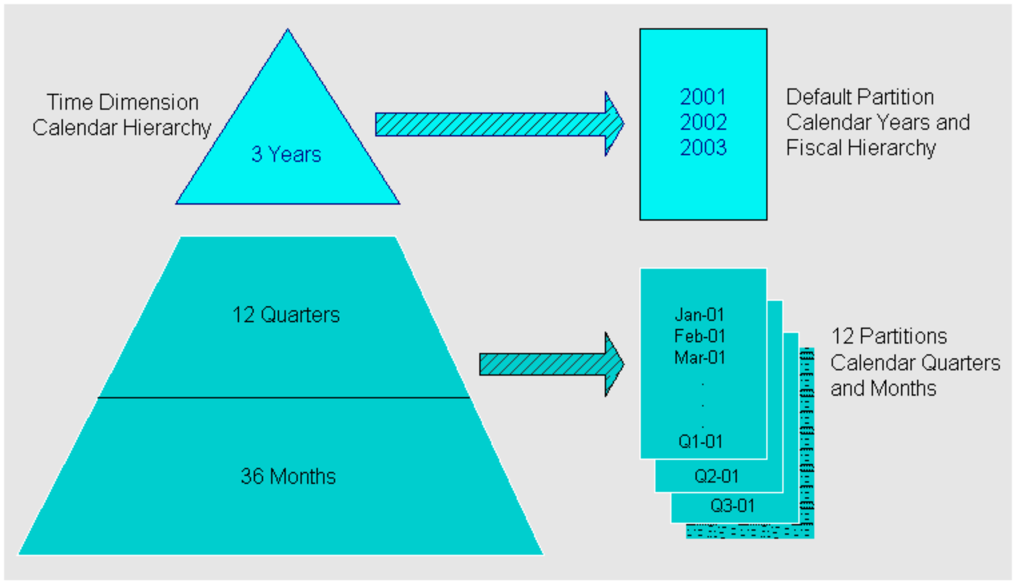
Other enterprises maintain their analytic workspaces. For them, the Time dimension may be a good candidate for partitioning, because this choice supports life-cycle maintenance. Old time periods can be dropped as a unit in a partition, and new time periods can be added in a new partition. Moreover, the partitions will be approximately the same size because of the inherent regularity of the calendar. In a calendar hierarchy, months have 28-31 children, quarters have 3 children, and years have 4 children.

Example of a Partitioned Dimension

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 5–8 illustrates a Time dimension partitioned by Quarter.

Figure 5–8 Partitioning Time by Quarter



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, measures, or cubes. 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 cube, 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 or summarize data for 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.

You can also create subfolders.

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 and Executing Calculation Plans

Calculation plans are composed of an ordered list of steps. Each step is either an aggregation, an allocation, or a forecast. By specifying the order in which these steps are performed, you can allow for interdependencies.

You execute calculation plans using the Maintenance Wizard, typically after loading new data into the analytic workspace.

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 step, click **New Step**.
5. Choose the type of step: Forecast, allocation, or aggregation.

   The New Step dialog box is displayed for that type of calculation.
6. Complete all tabs, then click **Create**.

   The new step is listed on the Calculation Plan General tab.
7. Click **Create**.

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

8. To run the calculation plan:

   a. Right-click it on the navigation tree and choose **Execute Calculation Plan**.

      The Maintenance wizard opens.

   b. Follow the steps of the wizard.

      See Also: Part III for information about creating forecasts, allocations, and aggregations.

Using Templates to Re-Create a Logical Model

Analytic Workspace Manager enables you to save all or part of the logical model as a text file. This text file contains the XML definitions of the logical objects, such as dimensions, levels, hierarchies, attributes, and measures. Only the metadata is saved, not the data or any customizations. Templates are small files, so you can easily distribute them by email or on a Web site, just as the templates for Global and Sales History are distributed on the Oracle Web site. To re-create the logical objects, you simply identify the templates in Analytic Workspace Manager.

You can save the following types of objects as XML templates:

- **Analytic workspace**: Saves all logical objects. You can save measure folders and calculation plans only by saving the complete analytic workspace.
- **Cube**: Saves the cube and its measures, calculated measures, and mappings.
- **Calculated measure**: Saves just the calculated measure.
- **Dimension**: Saves the dimension and its levels, hierarchies, attributes, and mappings.

To create a template:
In the navigation tree, right-click the object and choose **Save *object* to Template**.

To create logical objects from a template:
In the navigation tree, right-click the object type and choose **Create *object* From Template**.

Using Plug-Ins

Plug-ins extend the functionality of Analytic Workspace Manager. Any Java developer can create a plug-in. Plug-ins are distributed as JAR files. The developer should provide information about what the plug-in does and how to use it.

If you have one or more plug-ins, then you only need to identify their location to Analytic Workspace Manager.

To use plug-ins:

1. Create a local directory for storing plug-ins for Analytic Workspace Manager.
2. Copy the JAR files to that directory.
3. Open Analytic Workspace Manager.
4. Choose **Configuration** from the Tools menu.

   The Configuration dialog box opens.

5. Complete the Plug-in tab and click **OK**.
6. Close and reopen Analytic Workspace Manager.

   The new functionality provided by the plug-ins is available as right-click menu choices in the navigator.
7. Right-click an object and select one of the new choices from the pop-up menu.

> **See Also:** *Developing Analytic Workspace Manager Plug-ins*, which you can download from the Oracle Technology Network at `http://www.oracle.com/technology/products/bi/olap`.

Case Study: Creating the Global Analytic Workspace

You can download and install the Global relational tables from this Oracle Web site, following the instructions provided with the download:

`http://www.oracle.com/technology/products/bi/olap/`

You can then either create an analytic workspace from a template or create it manually by following the instructions given here.

> **See Also:** Chapter 3 for additional information about installing the Global schema

Defining the GLOBAL_AW User

This example creates the `GLOBAL` analytic workspace in a different schema from the source tables. Example 5–1 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.

The installation scripts for Global create the `GLOBAL_AW` user, so you do not need to run the script shown here. However, you may wish to use it as a model for creating other users.

Example 5–1 SQL Script for Defining the GLOBAL_AW User

```
CREATE USER "GLOBAL_AW" PROFILE "DEFAULT"
    IDENTIFIED BY "global_aw" DEFAULT TABLESPACE "GLOBAL"
    TEMPORARY TABLESPACE "GLOBAL_TEMP"
    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;
```

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, using GLOBAL_AW as the password.
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 3 by following the basic instructions in "Creating Logical Dimensions" on page 5-10.

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) as described in this example.
- **Product Dimension**: You can map Product to a star, level-based table (PRODUCT_DIM) or to a parent-child table (PRODUCT_CHILD_PARENT) as described in this example.
- **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 is a very small and dense sample data set, so that some decisions that are crucial when handling huge data sets are simply non-consequential in this case. Nonetheless, this example shows the best practices for handling dense data. "Case Study: Creating the Sales History Analytic Workspace" on page 5-29 shows the best practices for handling sparse data; sparsity is typical of real data sets.

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 3 by following the basic instructions in "Creating Logical Cubes" on page 5-13.

UNITS_CUBE

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

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

Time is first to facilitate data maintenance. The other dimensions are listed in order from largest to smallest.

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

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.

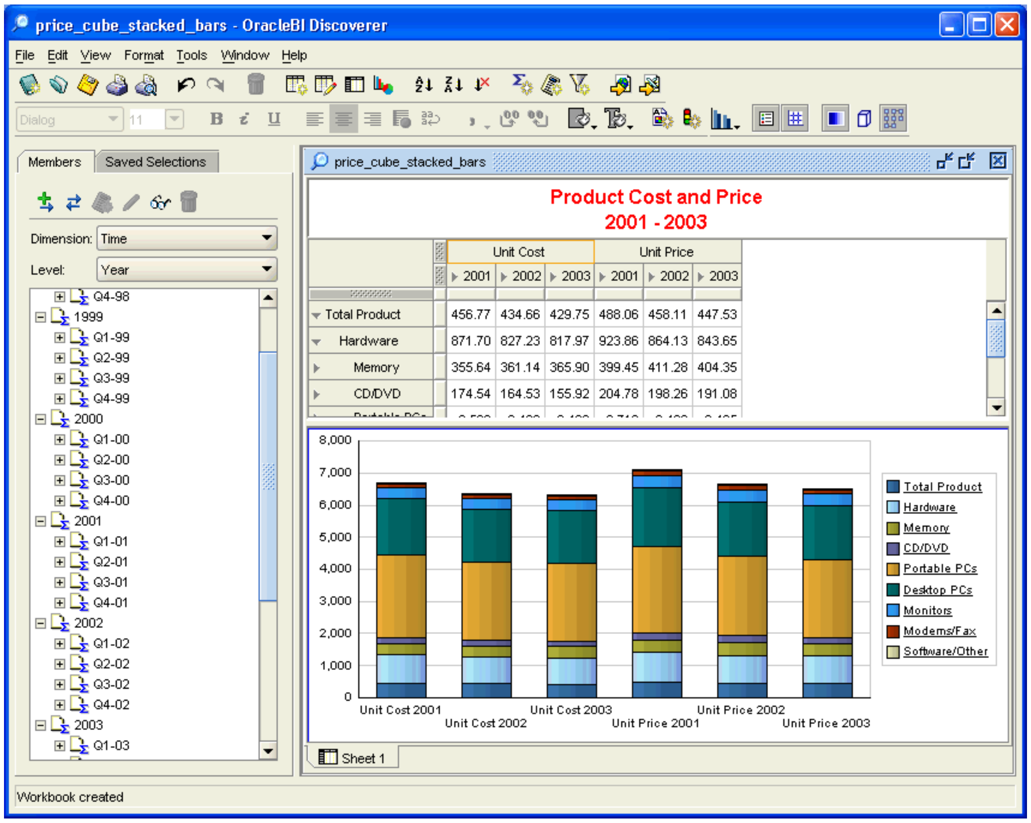
Loading and Aggregating the Data

To load all of the data for `GLOBAL`, run the Maintenance Wizard as described in "Maintaining the Data" on page 5-21. 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 Cubes** box, then move **Cubes** to the Selected Source Objects column. Click **Finish** to run the job immediately.

Figure 5–9 shows the results of a query in OracleBI Discoverer Plus OLAP.

Figure 5–9 Discoverer Plus OLAP Displays Data from PRICE_AND_COST_CUBE



Product Cost and Price
2001 - 2003

	Unit Cost			Unit Price		
	2001	2002	2003	2001	2002	2003
Total Product	456.77	434.66	429.75	488.06	458.11	447.53
Hardware	871.70	827.23	817.97	923.86	864.13	843.65
Memory	355.64	361.14	365.90	399.45	411.28	404.35
CD/DVD	174.54	164.53	155.92	204.78	198.26	191.08

Creating Calculated Measures

"Identifying Required Business Facts" on page 3-5 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 5–1 shows the calculated measures for the Units Cube.

Table 5–1 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) Change in sales from prior year	Prior/Future Comparison > Difference from Prior Period	`SALES`
Percent change in sales from prior period Percent change in sales from prior year	Prior/Future Comparison > Percent Difference from Prior Period	`SALES`
Product share	Advanced Arithmetic > Share	`SALES(PRODUCT)`
Channel share	Advanced Arithmetic > Share	`SALES(CHANNEL)`
Market share	Advanced Arithmetic > Share	`SALES(CUSTOMER)`
Extended margin change from prior period Extended margin change from prior year	Prior/Future Comparison > Difference from Prior Period	`EXTENDED_MARGIN`
Extended margin percent change from prior period Extended margin percent change from prior year	Prior/Future Comparison > Percent Difference from Prior Period	`EXTENDED_MARGIN`
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. 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.

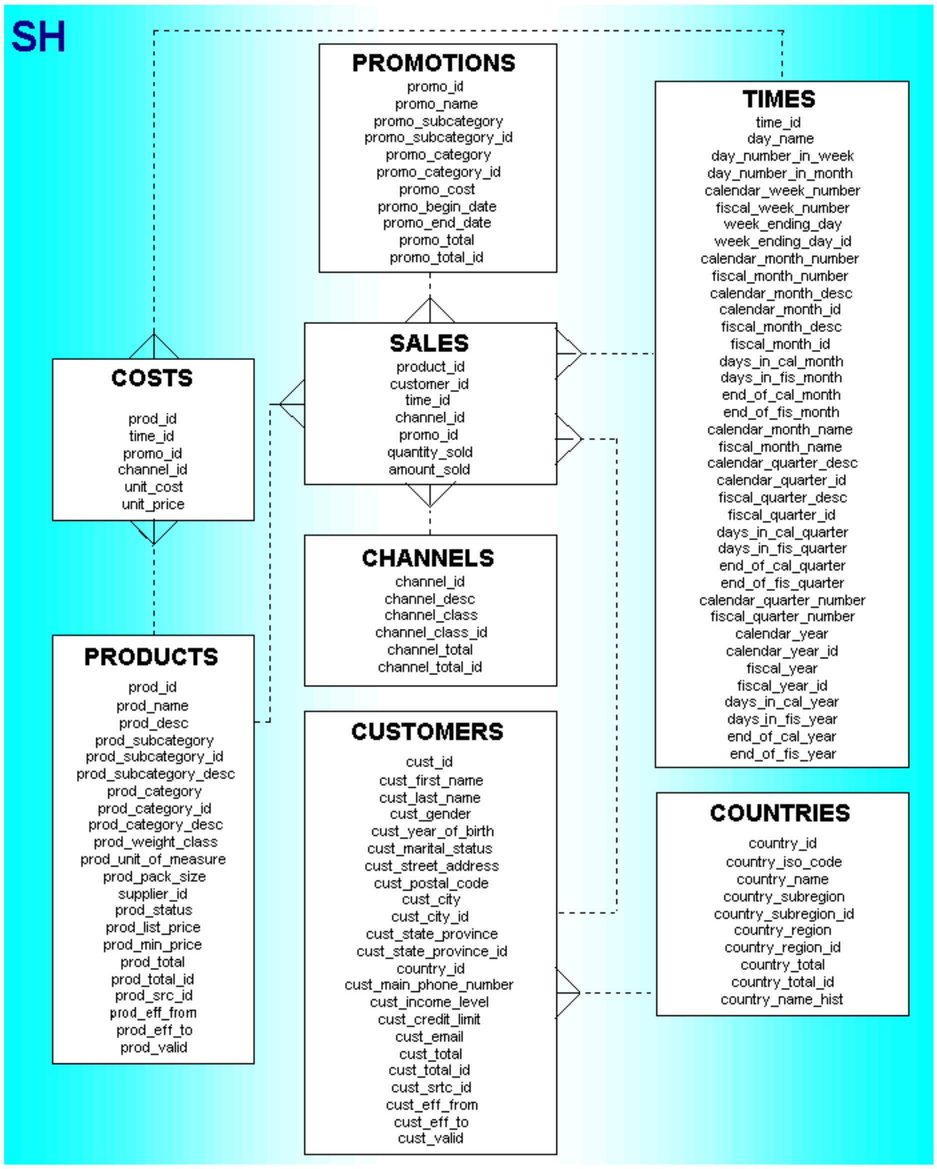
You can download a template for a Sales History analytic workspace from:

`http://www.oracle.com/technology/products/bi/olap`

Then you can simply examine the definitions of various objects instead of creating them manually. You will still need to run the Maintenance wizard to load the data.

Figure 5–10 shows a schema diagram of Sales History.

Figure 5–10 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. Define the logical cube and map it to the relational tables.
7. Run the Sparsity Advisor.
8. Load and summarize the data.
9. 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 5–2 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 5–2 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.
- 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 5–3 shows how the tablespaces might be defined for Sales History when four disk drives are available.

Example 5–3 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 5–4 shows a script that is similar to the one used to create the GLOBAL_AW user in Example 5–1. 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 5–4 SQL 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;

/* 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 5–10.

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. 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 most 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.

Run the Sparsity Advisor. Be sure that the data type is set to DECIMAL, then re-create the cube using the recommended settings.

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.

6 Administering Oracle OLAP

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

Administration Overview

Because Oracle OLAP is contained in the database and 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 6-13.

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 that are appropriate for their use. Analytic workspaces contain many objects and each one occupies at least one extent. You should create tablespaces with `EXTENT MANAGEMENT LOCAL` and allow an automatic allocation. Otherwise, with a fixed extent size, you may waste most of the allocated space. For example, if an object is 64K and the extents are set to a uniform size of 1M, then only a small portion of the extent is used.

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.

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 with the appropriate parameters for use by analytic workspaces:

```
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:

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

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;
```

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 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.
- `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.
- `QUERY REWRITE` system privilege (for relational tables).
- `OLAP_USER` role (for relational tables).

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
`java.io.FilePermission`	`read`, `write`, `execute`
`java.util.PropertyPermission`	`read`, `write`
`java.net.SocketPermission`	`connect`, `resolve`
`java.lang.RuntimePermission`	`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.

The recommendations for memory assume that the computer is dedicated to running Oracle Database. 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 `PGA_AGGREGATE_TARGET` and `SGA_TARGET` based on 75% of 4G, which is 3G.

Table 6–2 Initial Settings for Database Parameter Files

Parameter	Setting
`JOB_QUEUE_PROCESSES`	Number of CPUs, plus one additional process for every three CPUs For example, `JOB_QUEUE_PROCESSES=5` for a four-processor computer
`OPEN_CURSORS`	300 or more to support Analytic Workspace Manager
`PGA_AGGREGATE_TARGET`	50% of physical memory to start, then tune as indicated by performance statistics
`SGA_TARGET`	25% or less of physical memory to start, then tune as indicated by performance statistics
`SESSIONS`	2.5 * maximum number of simultaneous OLAP users
`UNDO_MANAGEMENT`	`AUTO`
`UNDO_TABLESPACE`	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:

   ```
   PGA_AGGREGATE_TARGET=1G
   ```

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/rel10g/pfile/initrel10g.ora
```

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
```

Basic Queries for Monitoring the OLAP Option

The following queries extract OLAP information from the data dictionary. In most of the SELECT statements, you must replace GLOBAL with the name of your analytic workspace.

More complex queries are provided in a script that you can download from the Oracle OLAP Web site. For descriptions of these scripts and download instructions, refer to "Monitoring Performance" on page 6-13.

Is the OLAP Option Installed in the Database?

The OLAP option is provided with Oracle Database Enterprise Edition. To verify that the OLAP components have been installed, issue this SQL command:

```
SELECT COMP_NAME, VERSION, STATUS FROM DBA_REGISTRY WHERE COMP_NAME LIKE '%OLAP%';

COMP_NAME                 VERSION      STATUS
------------------------- ------------ -----------
OLAP Analytic Workspace   10.2.0.3.0   VALID
Oracle OLAP API           10.2.0.3.0   VALID
OLAP Catalog              10.2.0.3.0   VALID
```

What Analytic Workspaces are in the Database?

The DBA_AWS view provides information about all analytic workspaces. Use the following SQL command to get a list of names and their owners:

```
SELECT OWNER, AW_NAME FROM DBA_AWS;

OWNER                          AW_NAME
------------------------------ ------------------------------
SYS                            EXPRESS
SYS                            AWMD
SYS                            AWCREATE
SYS                            AWCREATE10G
SYS                            AWXML
SYS                            AWREPORT
GLOBAL_AW                      GLOBAL
```

How Big is the Analytic Workspace?

To find out the size of the tablespace extensions for a particular analytic workspace, use the following SQL statements, replacing GLOBAL with the name of your analytic workspace.

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

    EXTNUM SUM(DBMS_LOB.GETLENGTH(AWLOB))
---------- ------------------------------
         0                       50450928
```

The DBMS_LOB PL/SQL package includes a program for reporting the size of a LOB table that stores an analytic workspace. Use a SQL command like the following, replacing GLOBAL with the name of your analytic workspace and GLOBAL_AW with the name of the schema.

```
SELECT ROUND(SUM(DBMS_LOB.GETLENGTH(AWLOB))/1024,0) "KB" FROM global_aw.aw$global;

        KB
----------
     53700
```

How Is the Analytic Workspace Stored?

The `DBMS_METADATA` PL/SQL package contains a subprogram that shows how any particular analytic workspace is stored in the database. Use a SQL command like the following, replacing `GLOBAL` with the name of your analytic workspace and `GLOBAL_AW` with the name of the schema.

```
SELECT DBMS_METADATA.GET_DDL('TABLE', 'AW$GLOBAL', 'GLOBAL_AW') FROM DUAL;

DBMS_METADATA.GET_DDL('TABLE','AW$GLOBAL','GLOBAL_AW')
--------------------------------------------------------------------------------

  CREATE TABLE "GLOBAL_AW"."AW$GLOBAL"
   (    "PS#" NUMBER(10,0),
        "GEN#" NUMBER(10,0),
        "EXTNUM" NUMBER(8,0),
        "AWLOB" BLOB,
        "OBJNAME" VARCHAR2(256),
        "PARTNAME" VARCHAR2(256)
   ) PCTFREE 10 PCTUSED 40 INITRANS 4 MAXTRANS 255
  STORAGE(
  BUFFER_POOL DEFAULT)
  TABLESPACE "GLOBAL"
 LOB ("AWLOB") STORE AS (
  DISABLE STORAGE IN ROW CHUNK 8192 PCTVERSION 0
  CACHE
  STORAGE(
  BUFFER_POOL DEFAULT))
  PARTITION BY RANGE ("GEN#")
  SUBPARTITION BY HASH ("PS#","EXTNUM")
  SUBPARTITIONS 8
        .
        .
        .
```

When Were the Analytic Workspaces Created?

The `DBA_OBJECTS` view provides the creation date of the objects in your database. The following SQL command generates an easily readable report for analytic workspaces.

```
SELECT OWNER ||'.'||SUBSTR(OBJECT_NAME,4) AS AW_NAME, CREATED FROM DBA_OBJECTS
WHERE OBJECT_NAME LIKE 'AW$%' AND OBJECT_NAME != 'AW$' GROUP BY OWNER, OBJECT_
NAME, CREATED ORDER BY OWNER, AW_NAME;

AW_NAME              CREATED
-------------------- ---------
GLOBAL_AW.GLOBAL     10-JUL-06
SYS.AWCREATE         08-JUL-06
SYS.AWCREATE10G      08-JUL-06
SYS.AWMD             08-JUL-06
SYS.AWREPORT         08-JUL-06
SYS.AWXML            08-JUL-06
SYS.EXPRESS          08-JUL-06
```

How Dimensional Data is Stored in the Database

Oracle OLAP multidimensional data is stored in analytic workspaces. 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.

Each analytic workspace is stored in a relational table. 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 measure is stored in one row; a partitioned measure 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.
`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.

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

Column	Data Type	NULL	Description
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

System Tables

The SYS user owns several tables and views associated with analytic workspaces. Most of them are LOB tables that contain analytic workspaces, which are attached automatically to a user's session as needed. Following are brief descriptions of these objects.

- AW$ maintains a record of all analytic workspaces in the database, recording its name, owner, and other information.
- The following tables contain analytic workspaces:
  - AW$AWCREATE10G stores the AWCREATE10G analytic workspace, which contains programs for using OLAP Catalog metadata in Oracle Database 10*g* Release 10.1.0.3.
  - AW$AWMD stores the AWMD analytic workspace, which contains programs for creating standard form catalogs.

  - AW$AWREPORT stores the AWREPORT analytic workspace, which contains a program named AWREPORT for generating a summary space report.
  - AW$AWXML stores the AWXML analytic workspace, which contains programs for creating and managing standard form analytic workspaces for Oracle Database 10*g* Release 10.2 and later.
  - 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.

- 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$.

Static Data Dictionary Views

Among the static views of the database data dictionary are several that provide information about analytic workspaces. Table 6–5 brief descriptions of them. There are corresponding DBA and USER views.

Table 6–5 Static Data Dictionary Views for OLAP

View	Description
ALL_AWS	Describes the analytic workspaces accessible to the current user.
ALL_AW_OBJ	Describes the current objects in all analytic workspaces accessible to the current user.
ALL_AW_PROP	Describes the OLAP DML properties defined in all analytic workspaces accessible to the current user.
ALL_AW_PS	Describes the page spaces currently in use by all analytic workspaces accessible to the current user.

See Also: *Oracle Database Reference* for descriptions of these and other data dictionary views.

Monitoring Performance

Each Oracle Database instance maintains fixed tables that record current database activity. These tables collect data on internal disk structures and memory structures. Among them are tables that collect data on Oracle OLAP.

These tables are available to users through a set of dynamic performance views. By monitoring these views, you can detect usage trends and diagnose system bottlenecks. Table 6–6 provides a brief description of each view. Global dynamic performance views (GV$) are also provided.

See Also: *Oracle OLAP Reference* for full descriptions of the OLAP dynamic performance views.

Table 6–6 OLAP Dynamic Performance Views

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

You can download from the Oracle OLAP Web site a file that contains several SQL scripts. These scripts extract information from two or more system views and generate a report that may be useful in tuning a database. To download the file, go to this URL:

`http://www.oracle.com/technology/products/bi/olap/DBA_scripts.zip`

Table 6–7 describes these scripts. For more information, refer to the README file provided with the scripts.

Table 6–7 OLAP DBA Scripts

SQL Script	Description
`aw_objects_in_cache`	Identifies the objects in the buffer cache that are related to analytic workspaces.
`aw_rows_rw`	Tallies the number of reads from temporary segments and the LOB tables where analytic workspaces are stored, the number of cache rights, and the number of rows processed.
`aw_size`	Displays the amount of disk space used by each analytic workspace.
`aw_tablespaces`	Provides extensive information about the tablespaces used by analytic workspaces.
`aw_users`	Identifies the users of analytic workspaces.
`cursor_parameters`	Indicates whether the database parameters that limit the number of open cursors are set too low.
`olap_pga_performance`	Determines how much PGA is in use, the size of the OLAP page pool, and the hit/miss ratio for OLAP pages for each user.
`olap_pga_use`	Determines how much PGA is consumed by the OLAP page pool to perform operations on analytic workspaces.
`session_resources`	Identifies the use of cursors, PGA, and UGA for each open session.

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 Templates.** 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 Files**. 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. Use the `expdp/impdb` 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). Use the `expdp/impdb` 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

Generating Quality Information

Part III contains information about generating information from the data loaded into an analytic workspace. It contains the following chapters:

7 Aggregating Data

An analytic workspace always returns summary data to a query as needed. While the analytic workspace may store data, for example, at the day level, it will return a result at the quarter or year level without requiring a calculation in the query. This chapter explains how to optimize the unique aggregation subsystem of Oracle OLAP to provide the best performance for both data maintenance and querying.

This chapter contains the following topics:

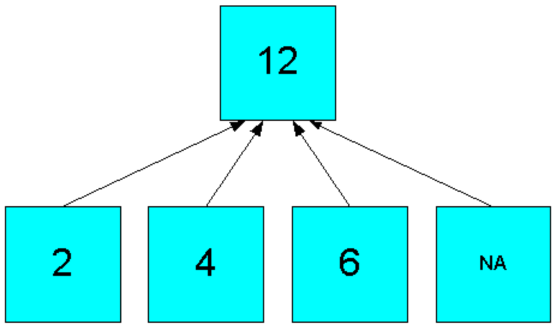
What is Aggregation?

Aggregation is the process of consolidating multiple values into a single value. For example, data can be collected on a daily basis and aggregated into a value for the week, the weekly data can be aggregated into a value for the month, and so on. Aggregation allows patterns in the data to emerge, and these patterns are the basis for analysis and decision making. When you define a data model with hierarchical dimensions, you are providing the framework in which aggregate data can be calculated.

Aggregation is frequently called summarization, and aggregate data is called summary data. While the most frequently used aggregation operator is Sum, there are many other operators, such as Average, First, Last, Minimum, and Maximum. Oracle OLAP also supports weighted and hierarchical methods. Following are some simple diagrams showing how the basic types of operators work. For descriptions of all the operators, refer to "Aggregation Operators" on page 7-12.

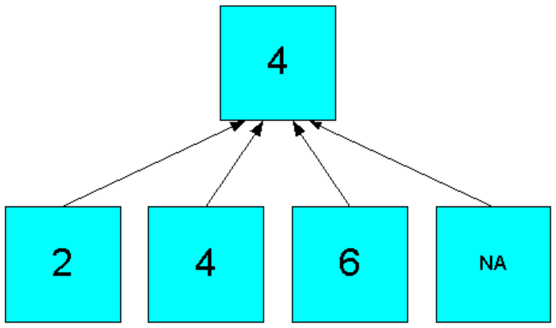
Figure 7–1 shows a simple hierarchy with four children and one parent value. Three of the children have values, while the fourth is empty. This empty cell has a null or `NA` value. The Sum operator calculates a value of 12 (2 + 4 + 6) for the parent value.

Figure 7–1 Summary Aggregation in a Simple Hierarchy



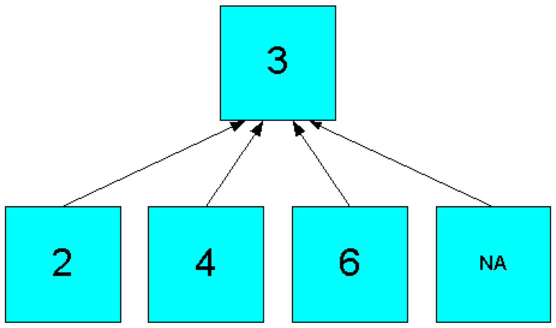
The Average operator calculates the average of all real data, producing an aggregate value of 4 ((2 + 4 + 6)/3), as shown in Figure 7–2.

Figure 7–2 Average Aggregation in a Simple Hierarchy



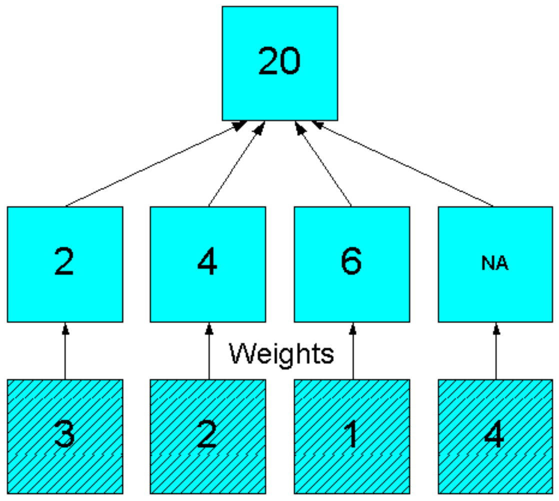
The hierarchical operators include null values in the count of cells. In Figure 7–3, the Hierarchical Average operator produces an aggregate value of 3 ((2 + 4 + 6 +NA)/4).

Figure 7–3 Hierarchical Average Aggregation in a Simple Hierarchy



The weighted operators use the values in another measure to generate weighted values before performing the aggregation. Figure 7–4 shows how the simple sum of Figure 7–1 is altered by using weights ((3*2) + (2*4) + (1*6) +(4*NA)).

Figure 7–4 Weighted Sum Aggregation in a Simple Hierarchy



Managing Aggregate Data

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 may slow the response time. At the other extreme, if all summary data is stored, then the database multiplies in size.

Managing Aggregate Data in Relational Tables

Relational schemas store aggregate data in materialized views and summary tables. The query rewrite feature of Oracle Database redirects queries for summary data from the fact tables to the materialized views. When predefined reports are run on a routine basis, the DBA knows which areas of the data are queried and what summary data is needed. This situation may be handled easily with a relatively small number of materialized views.

However, extensive use of ad-hoc queries and user-defined calculated measures create a random situation in which any part of the data store may be queried and summarized. Because a materialized view only has data for one combination of levels for the dimensions of the data, thousands of materialized views might be needed to provide coverage for most queries. A large number of materialized views requires extensive storage space and slows the query rewrite process for every request. When a particular materialized view is not available, the summary data is generated at runtime from the data in the fact table.

Managing Aggregate Data in Analytic Workspaces

Analytic workspaces use an entirely different paradigm for managing aggregate data than relational tables. This paradigm is based on the dimensional model, which is inherent in analytic workspaces.

A cube is defined by its dimensions. Each cell in the cube is identified by a unique combination of dimension members, with one member from each dimension. The dimension members function as an index into the cube.

For each measure in the cube, a cell either has a data value or an NA value. While an NA is equivalent to empty or null, it is handled as the value of the cell. Thus, a cell with a value of NA can be queried the same as a cell with an actual data value. This is an important feature of dimensional data analysis, because it enables analysts to investigate the absence of data (that is, no activity) as well as the actual data.

A dimension is a list of all its members from the base to the most aggregate. Thus, a measure stores all of its data, regardless of its level of aggregation. There are no additional objects for storing aggregate data, and thus no need to redirect queries.

Basic Strategies for Aggregating Data

In an analytic workspace, 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.

Table 7–1 shows a small portion of the Global Sales measure before aggregation.

- The Channel dimension is set to a single base-level member named Catalog.
- The Product dimension is set to a single base-level member named Sentinel Financial.
- The Time dimension is set to Jul-04 and its ancestors, Q3-04 and 2004.
- The Customer dimension is set to KOSH Enterprises Boston and its ancestors, United States, North America, and All Customers.

Out of the 12 cells shown, only one has a data value. If the three ancestors of Sentinel Financial and the one ancestor of Catalog were added, then this number would expand to one out of 96 cells.

Table 7–1 Portion of Sales Data Before Aggregation

Catalog **Sentinel Financial**	**2004**	**Q3-04**	**Jul-04**
All Customers	NA	NA	NA
North America	NA	NA	NA
United States	NA	NA	NA
KOSH Entrpr Boston	NA	NA	12,281.79

Nonetheless, 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.

In an analytic workspace, aggregate data is generated at two distinct times:

- As part of the build procedure. You can calculate and store all of the aggregates, none, or a portion of them. Calculated values are stored in the analytic workspace and shared by all sessions. This is often called **pre-aggregation**.
- In response to a query. Any "missing" aggregate data will be calculated for the query, even if the analytic workspace has no stored aggregate data. Calculated values may be cached for use throughout a session, but they are not shared among sessions. This is often called aggregating **on the fly.**

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 any analytic workspace is by storing some aggregates and calculating others on the fly. When choosing which aggregate values to store, your goal is to select those that require time- and resource-intensive calculations. Calculations that can be performed quickly can be left until runtime.

Table 7–2 shows the same portion of the Global Sales measure after it has been aggregated as part of the data maintenance process. The Warehouse level of Customer and the Year level of time have been calculated and stored. Out of the 12 cells shown, four now have data values, which is one-third the total. The entire measure has this ratio when Channel and Product are fully aggregated during maintenance.

Table 7–2 Portion of Sales Data After Data Maintenance Aggregation

Catalog **Sentinel Financial**	**2004**	**Q3-04**	**Jul-04**
All Customers	`NA`	`NA`	`NA`
North America	`2,819,969.60`	`NA`	`368,453.61`
United States	`NA`	`NA`	`NA`
KOSH Entrpr Boston	`91,208.57`	`NA`	`12,281.79`

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. In addition, you can choose to cache aggregate values for the duration of a session, so that they are calculated on the fly only once. Table 7–3 shows the Sales measure when it is fully aggregated in response to a query. Eight values are calculated on the fly to return the answer set to the client.

Table 7–3 Portion of Sales Data Fully Aggregated For a Query

Catalog **Sentinel Financial**	**2004**	**Q3-04**	**Jul-04**
All Customers	`4,415,575.54`	`600,053.02`	`600,053.02`
North America	`2,819,969.60`	`368,453.61`	`368,453.61`
United States	`2,660,444.61`	`352,662.74`	`352,662.74`
KOSH Entrpr Boston	`91,208.57`	`12,281.79`	`12,281.79`

Aggregating Non-Compressed Composites

The strategy for aggregation shown in "Basic Strategies for Aggregating Data" on page 7-4 is called skip-level aggregation, because some levels are stored and others are skipped until runtime. The success of this strategy depends on choosing the right levels to skip, which are those that can be calculated quickly in response to a query.

Note: Skip-level aggregation is used only for regular (non-compressed) composites. For the recommended strategy for compressed composites, refer to "Aggregating Compressed Composites" on page 7-6.

Selecting Dimensions for Skip-Level Aggregation

As a general rule, you should skip levels for only one or two dimensions and for no more than half of the dimensions of the cube. Choose the dimensions with the most levels in their hierarchies for skip-level aggregation.

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.

Selecting the Levels to Skip

You can identify the best levels to skip 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 or less. 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.

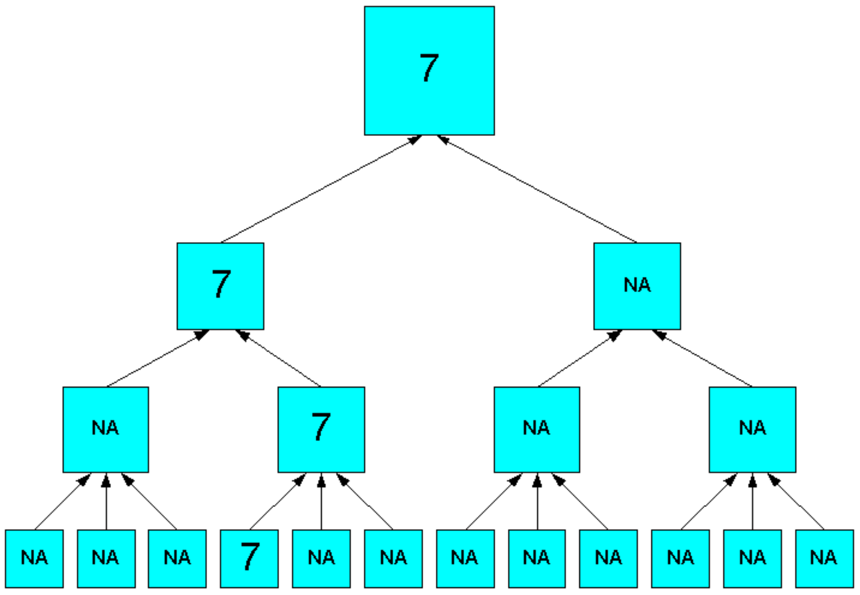
This 10:1 rule is best applied with some judgment. You might want to permit a higher ratio for levels that you know are seldom accessed. Or you might want to store levels at a lower ratio if you know they have heavy use. Generally, you should strive for a lower ratio instead of a higher one to maintain the best performance.

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.

Aggregating Compressed Composites

Compressed composites are used to store extremely sparse data. They are designed specifically to handle data structures in which several levels may store the same value. Figure 7–5 shows an aggregation in which one out of 12 cells has a data value. The skip-level strategy described in "Aggregating Non-Compressed Composites" on page 7-5 does not produce any benefit under these circumstances.

Figure 7–5 Aggregation of Very Sparse Data



Use this aggregation strategy for compressed cubes:

- Identify the dimension with the most members. If several dimensions have about the same number, then choose the dimension with the most levels. Do not pre-aggregate this dimension.
- Pre-aggregate all other dimensions up to, but not including, the top level, unless the next level down has a large number of members.

You can adjust these basic guidelines to the particular characteristics of your data. For example, you may skip levels that are seldom queried from pre-aggregation. Or you may need to pre-aggregate a level with a large number of child values, to provide acceptable run-time performance.

Improving Aggregation Performance

The previous guidelines provide an approach to aggregation that should help you meet these basic goals:

- **Finish Data Updates on Time**
- **Keep Within Allocated Resources**
- **Provide Good Response Time**

If you anticipate problems with one or more of these goals, then you should keep them in mind while devising your aggregation rules. Otherwise, you may need to make adjustments after the initial build, if you experience problems meeting all of these goals.

Often the problem can be solved by changing factors other than the aggregation rules, as described in the following topics.

Note: Be sure to run the Sparsity Advisor so that the data is structured in the most efficient way. Refer to "Using the Sparsity Advisor" on page 5-18.

Finish Data Updates on Time

Most organizations allocate a batch window in which all data maintenance must be complete. If you are unable to finish refreshing the data in the allotted time, then you can make the following adjustments.

The more levels that you aggregate for storage, the longer the maintenance process will take. Review your reasons for choosing levels for pre-aggregation. If you know that some levels are seldom queried, you may skip them during the builds. Exercise care in skipping additional levels, however, because you run the risk of degrading run-time performance.

Be sure that you have set the database initialization parameters correctly for data maintenance, as described in "Initialization Parameters for Oracle OLAP" on page 6-6. You can make significant improvements in build performance by setting `SGA_TARGET`, `PGA_AGGREGATE_TARGET`, and `JOB_QUEUE_PROCESSES`.

After the initial build, you can save time by aggregating only newly loaded values, instead of aggregating all of them again. Partial aggregation is a choice you can make in the Maintenance Wizard.

Analytic workspaces are stored in partitioned tables, and you can create partitioned cubes. You can use these partitions to stripe the data across several disks, thus avoiding bottlenecks in I/O operations, if you have purchased the partitioning option to Oracle Database.

See Also: Chapter 6, "Administering Oracle OLAP"

Keep Within Allocated Resources

Your analytic workspace must fit within the allocated resources. The more levels of aggregate data that you store, the larger the tablespaces must be to store the analytic workspace.

The data type is an important consideration when estimating the size of an analytic workspace. The most commonly used data types for measures are `NUMBER` and `DECIMAL`. The difference in size is significant: an unscaled `NUMBER` value is 22 bytes and a `DECIMAL` value is 8 bytes.

Refer to "Choosing a Data Type" on page 5-19 for a comparison between these two data types.

Provide Good Response Time

An analytic workspace must provide good performance for end users. When pre-aggregation is done correctly, the response time for queries does not noticeably slow down. Analytic workspaces are optimized for multidimensional calculations, so that run-time summarizations should be extremely fast. However, runtime performance will suffer if the wrong choices were made.

If response time is poor, then review the decisions you made in skipping levels and find those that should be pre-aggregated. Try to identify and pre-aggregate those areas of the data that are queried heavily. Check the level on which you partitioned the cube. Remember that all levels above the partition are calculated on the fly. When partitioning over Time, the Month level is a much better choice than Day.

Read the recommendations given in the previous topics. The savings in maintenance time and disk storage may be used to pre-aggregate more of the data.

Selecting Dimension Members for Aggregation

The aggregation rules defined for a cube or a measure are always performed over all dimension members. You can perform a partial aggregation only in a calculation plan and only for regular composites.

Note: Do not set status for a compressed cube. All members must be in status.

To aggregate over a portion of a measure, you select the dimension members that identify the cells containing the source data, using the Status page of the Aggregation property sheet. You do not need to select the target cells. All of the cells identified by the ancestors of the selected dimension members are aggregated, either when you execute the calculation plan or when a user queries the measure.

When you select the dimension members, they are **in status**. This means that the dimension members have been selected for use in a calculation, a query, or other data manipulation. Likewise, **out of status** means that the dimension members have been excluded from use.

Figure 7–6 shows an aggregation in which the 12 months of 2006 are in status. Neither the quarters nor the year are in status, but aggregates are generated for all levels.

Figure 7–6 Sum Aggregation With All Source Values in Status

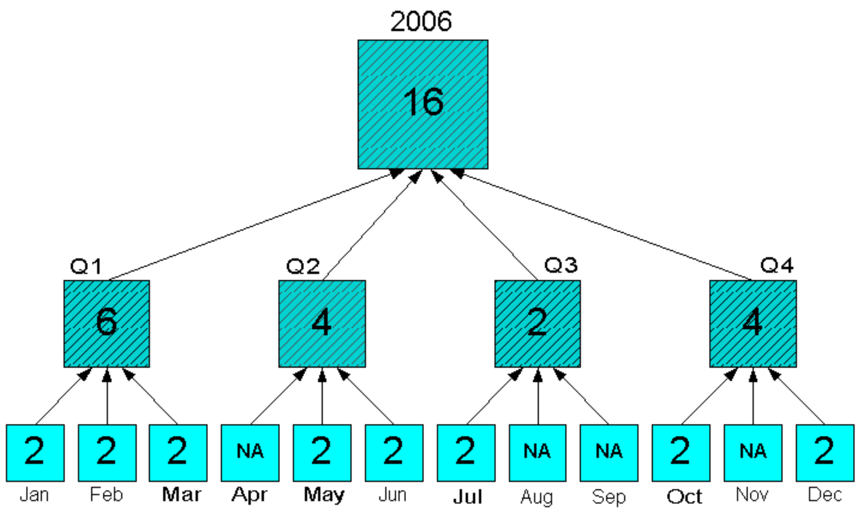
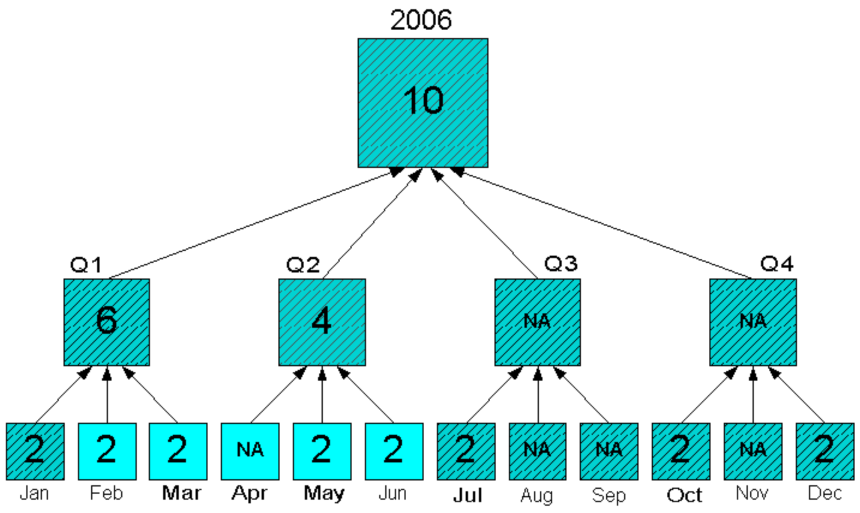


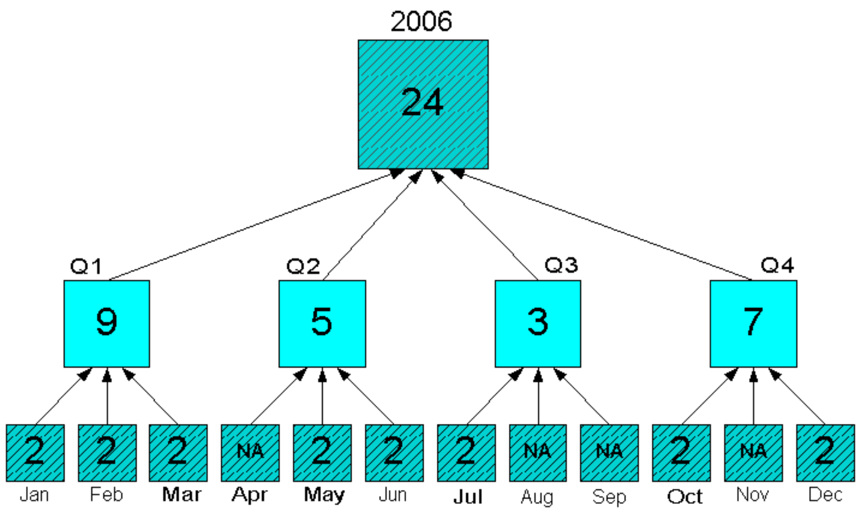
Figure 7–7 shows the same portion of data, but with only Feb to Jun in status. Aggregates are calculated only for Q1, Q2, and 2006. Note that Jan is included in the aggregation, even though it is out of status. The aggregation engine adds the ancestors, then the children to status before aggregating the data, as a means of maintaining the integrity of the data. The values for Jul to Dec are not included in the aggregation.

Figure 7–7 Sum Aggregation With Some Source Values Out of Status



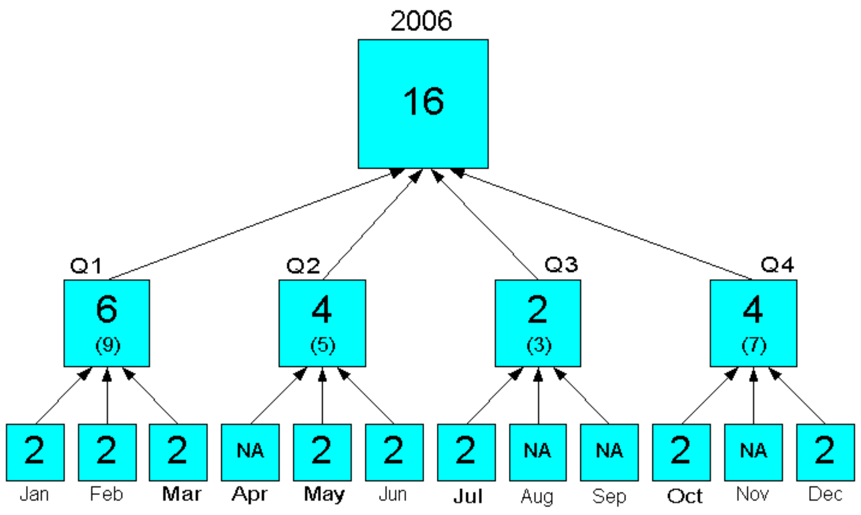
You may need to aggregate data that is stored in the middle of a hierarchy, perhaps if the data for a particular measure is not available or needed at the base level. You must be sure that the cells with the data are the lowest levels in the hierarchy in status. Figure 7–8 shows quarterly forecast data in status and aggregated to the year. The monthly values are not in status, and thus are excluded from the aggregation.

Figure 7–8 Sum Aggregation From the Quarterly Level



Aggregation begins at the lowest level in status and rolls up the hierarchy. The aggregate values will overwrite any pre-existing values higher in the hierarchy. Figure 7–9 shows that when the Month level is in status, those values will overwrite the forecast values at the Quarter level. The status of Quarter and Year has no effect on the aggregation.

Figure 7–9 Sum Aggregation From the Month Level Overwrites Quarters



Defining an Aggregation

Analytic Workspace Manager enables you to define aggregations at three different times. You can use whatever combination best suits your needs:

- **Cube**. You can define default aggregation rules for all measures in a cube by defining them just once. You define these rules when creating or modifying a cube.
- **Measure**. You can define unique aggregation rules for a particular measure that will override the default cube rules. You define these rules when creating or modifying a measure. You cannot specify aggregation rules for individual measures for compressed cubes.

- **Calculation Plan**. You can define rules for one or more measures and determine the order in which the measures are calculated. In this way, you can support dependencies among the measures and the calculations, such as aggregating data that is generated by a forecast. You define these rules when creating or modifying a calculation plan. This type of aggregation is performed after the measures are aggregated using either the Cube or the Measure rules.

> **Note:** All measures are aggregated during data maintenance using the rules defined for the cube or the rules defined for the measure. Any additional aggregation rules defined in a calculation plan are calculated separately.

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

- Select an aggregation operator and, depending on the operator, a weight measure on the Rules tab.
- Select the levels for storing values on the Summarize To tab.

To aggregate a measure using the cube rules:

1. On the General page of the Cube property sheet, select **Use Default Aggregation Plan for Cube Aggregation**.
2. On the General page of the Measure property sheet, select **Use Aggregation Specification From the Cube**.

To aggregate a measure using its individual rules:

1. On the General page of the Cube property sheet, select **Use Default Aggregation Plan for Cube Aggregation**.

   Any measure in the cube without its own individual rules will use this default plan.
2. On the General page of the Measure property sheet, select **Override the Aggregation Specification of the Cube**.

To aggregate a measure using a calculation plan:

1. Define the aggregation rules for the measure or the cube, as described previously.

   You can aggregate all the values first, then execute the calculation plan to overwrite a portion of the aggregate values. Or you can define the rules for the measure so that no aggregates are created during data maintenance.
2. Create a calculation plan that will aggregate all or part of the measure.

To aggregate all measures in a cube using a calculation plan:

1. On the General page of the Cube property sheet, clear the **Use Default Aggregation Plan for Cube Aggregation** box.
2. Create a calculation plan containing one or more aggregation steps that will aggregate all measures in the cube.

Aggregation Operators

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

Basic Operators

These are the basic aggregation operators:

- **Average**: Adds non-null data values, then divides the sum by the number of data values that were added together.
- **First Non-NA Data Value**: The first real data value.
- **Last Non-NA Data Value**: The last real data value.
- **Maximum**: The largest data value among the children of each parent.
- **Minimum**: The smallest data value among the children of each parent.
- **Nonadditive**: Do not aggregate any data for this dimension.
- **Sum**: Adds data values.

Hierarchical Operators

These are the hierarchical operators. They include all cells identified by the hierarchy in the calculations, whether or not the cells contain data. You should use these operators only when you want null values to be treated as zeroes instead of as missing data.

- **Hierarchical Average**: Adds data values, then divides the sum by the number of the children in the dimension hierarchy. Unlike Average, which counts only non-null children, hierarchical average counts all of the logical children of a parent, regardless of whether each child does or does not have a value.
- **Hierarchical First Member**: The first data value in the hierarchy, even when that value is null.
- **Hierarchical Last Member**: The last data value in the hierarchy, even when that value is null.
- **Hierarchical Weighted Average**: Multiplies non-null child data values by their corresponding weight values, then divides the result by the sum of the weight values. Unlike Weighted Average, Hierarchical Weighted Average includes weight values in the denominator sum even when the corresponding child values are null.
- **Hierarchical Weighted First**: The first data value in the hierarchy multiplied by its corresponding weight value, even when that value is null.
- **Hierarchical Weighted Last**: The last data value in the hierarchy multiplied by its corresponding weight value, even when that value is null.

Scaled and Weighted Operators

These are the scaled and weighted aggregation operators. They require a measure containing the weights in the same cube.

- **Scaled Sum**: Adds the value of a weight object to each data value, then adds the data values.

- **Weighted Average**: Multiplies each data value by a weight factor, adds the data values, and then divides that result by the sum of the weight factors.
- **Weighted First**: The first non-null data value multiplied by its corresponding weight value.
- **Weighted Last**: The last non-null data value multiplied by its corresponding weight value.
- **Weighted Sum**: Multiplies each data value by a weight factor, then adds the data values.

Case Study: Aggregating a Moderately Sparse or Dense Cube

The Global data set is a good candidate for skip-level aggregation. This example discusses aggregation of the Units Cube, which has four dimensions: Time, Customer, Product, and Channel. You could skip levels on one or two dimensions. For this example, skip levels on only one dimension; because Global is small, precalculating the additional levels does not create a problem with time or disk space.

The Customer dimension has two hierarchies with a total of seven levels. Because it has the most levels of any dimension, it is the best choice for skipping levels.

To identify the levels to be precalculated, you must know the number of dimension members at each level. You can easily acquire this information in SQL, using this SQL command:

```
SELECT COUNT(DISTINCT ship_to_id), COUNT(DISTINCT warehouse_id),
COUNT(DISTINCT region_id),COUNT(DISTINCT total_customer_id),
COUNT(DISTINCT account_id), COUNT(DISTINCT market_segment_id),
COUNT(DISTINCT total_market_id), FROM global.customer_dim;
```

Global is a very small data set, so few adjacent levels have the desired 10:1 ratio of children-to-parent dimension members. Table 7–4 and Table 7–5 identify the appropriate levels to be calculated and stored for the two hierarchies. Only eight members are stored out of a total of 45 aggregate members.

On the Summarize To page for the Units Cube, select the precalculated levels for Customer, and select all levels for Time, Product, and Channel.

Table 7–4 Precalculated Levels in the Customer Shipments Hierarchy

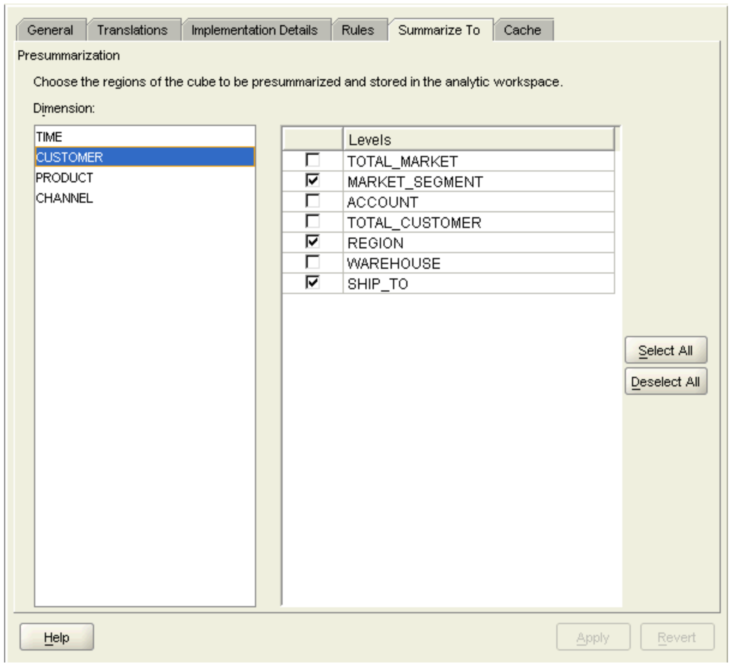
Level	Members	Precalculate?
`Total_Customer`	1	No
`Region`	3	**Yes**
`Warehouse`	11	No
`Ship_To`	61	**Yes**

Table 7–5 Precalculated Levels in the Customer Market Segment Hierarchy

Level	Members	Precalculate?
`Total_Market`	1	No
`Market_Segment`	5	**Yes**
`Account`	24	No
`Ship_To`	61	**Yes**

Figure 7–10 shows this selection on the Summarize To page of the Units Cube property sheet.

Figure 7–10 Selection of Customer Levels



Presummarization

Choose the regions of the cube to be presummarized and stored in the analytic workspace.

Dimension:

TIME
CUSTOMER
PRODUCT
CHANNEL

	Levels
☐	TOTAL_MARKET
✓	MARKET_SEGMENT
☐	ACCOUNT
☐	TOTAL_CUSTOMER
✓	REGION
☐	WAREHOUSE
✓	SHIP_TO

Case Study: Aggregating a Very Sparse Cube

Sales History is a very sparse data set, so it uses compressed composites for all cubes. You should fully aggregate all but the largest dimension. Table 7–6 shows the number of base-level members for each dimension. This information is easily acquired by counting the unique values in the dimension tables like this:

```
SELECT COUNT(DISTINCT time_id) FROM sh.times;
```

Because Customers is far larger than any of the other dimensions, you do not need to count the aggregate members or the number of levels to identify it as the largest one. On the Summarize To page for the cubes, deselect all levels for Customers and select all but the top level for Channels, Products, Promotions, and Times.

Table 7–6 Precalculated Dimensions in Sales History

Dimension	Members	Precalculate?
`Channels`	5	Yes
`Customers`	55,500	**No**
`Products`	72	Yes
`Promotions`	503	Yes
`Times`	1,826	Yes

8 Allocating Data

In Analytic Workspace Manager, you can create forecasts, set goals, and create budgets at a high level, and then allocate those numbers down a hierarchy to see how those numbers impact the contributing values.

This chapter contains the following topics:

What Is an Allocation?

Allocations distribute aggregate level data to detail level data, sometimes using an existing set of data as the basis for the allocation. This technology is often used in forecasting and budgeting systems. An example of a financial allocation is the automated distribution of a bonus pool, based on the current salaries and performance ratings of the employees.

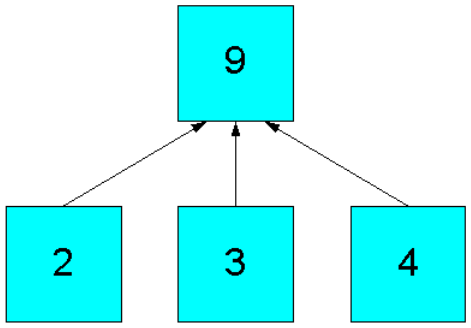
You can think of allocations as inverse aggregations.

- In aggregations, a group of child values are aggregated into a single parent value using an aggregation method, such as Sum.
- In allocations, a parent value is distributed to a group of child cells using an allocation method that is the inverse of the aggregation method, such as Average.

One important difference between aggregation and allocation is that an aggregation has one defined answer. An allocation has many possible answers for the same source value.

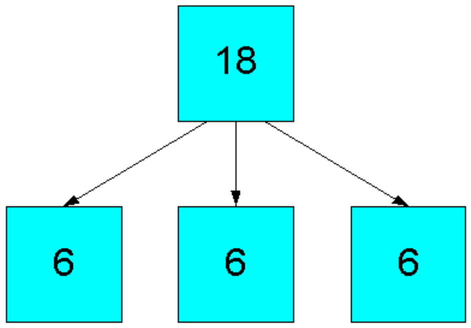
For example, consider the hierarchy in Figure 8–1. The value 9 is derived by aggregating the values 2, 3 and 4 using the Sum operator.

Figure 8–1 Aggregation in a Simple Hierarchy



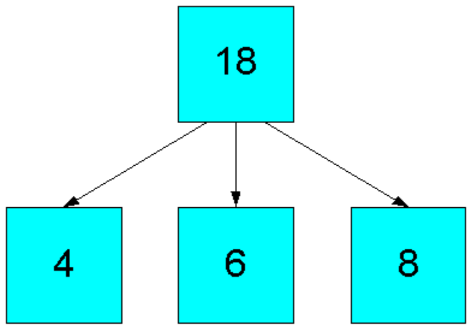
Now change the value of 9 to 18 and allocate the results to the children. The Even allocation operator divides the source value evenly by the number of children, and so assigns each child a value of 6, as shown in Figure 8–2.

Figure 8–2 Even Allocation In a Simple Hierarchy



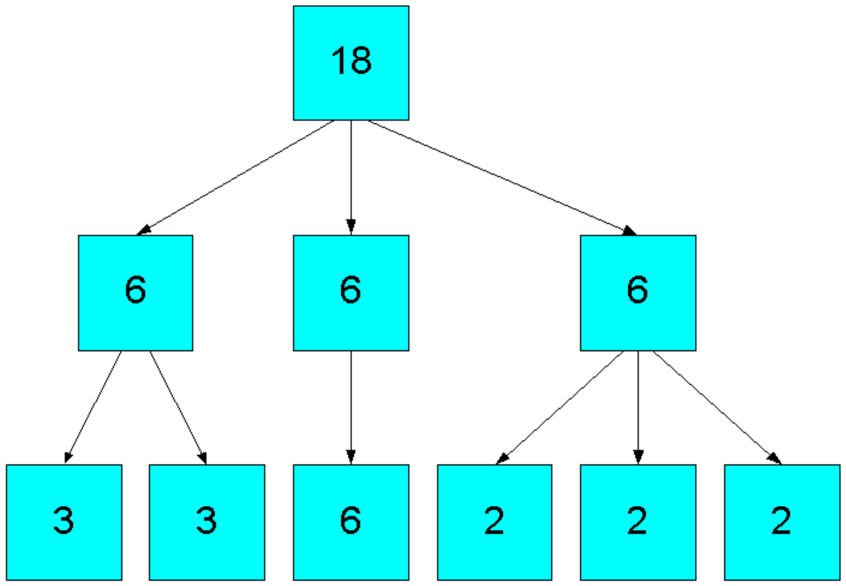
In contrast, the Proportional allocation operator divides the value into proportions based on the current value of each target cell, and so assigns values of 4, 6 and 8, as shown in Figure 8–3.

Figure 8–3 Proportional Allocation In a Simple Hierarchy



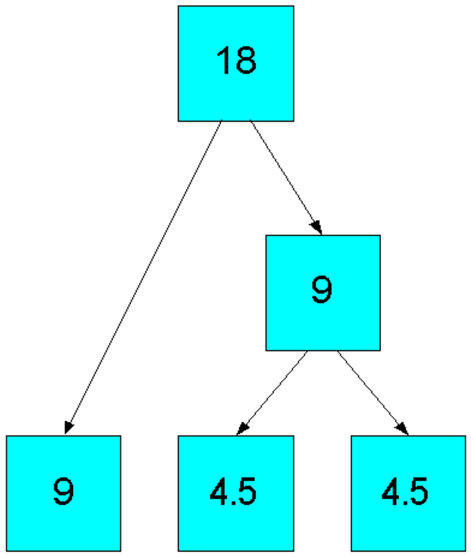
The previous examples show direct allocation, that is, where there is a parent-child relation between the source cell and the target cells. However, most hierarchies have multiple levels, and an allocation may assign values down the hierarchy, as shown in Figure 8–4.

Figure 8–4 Even Allocation in a Multilevel Hierarchy



Next, consider a skip level hierarchy. The source value is allocated down the hierarchy, as shown in Figure 8–5. The relationship of the target cell to the allocation source, not the hierarchical level of a cell, determines the allocation. Note that, as the result of an intermediate value in one branch, the base-level cells are allocated different values than in the simple hierarchy shown in Figure 8–2.

Figure 8–5 Even Allocation in a Skip Level Hierarchy



Creating Measures to Support an Allocation

Source, basis, and target are the most fundamental terms for describing allocation. You may use the same measure for all three roles or assign a different measure to each role. All allocation operators require a source and a target, but some operators do not use a basis. You can also multiply the results of an allocation by a weight measure.

Source Measures

The source measure contains the set of numbers that you want to allocate. You may use an existing measure, or you may perform some computation on existing data to construct new source values. For instance, you might want to budget 30 percent growth over the next year and perform an allocation to see the sales targets required for each product to meet that budget. You would create a calculated measure based on actual sales to use as the allocation source. Alternatively, you might generate a forecast at the middle or top of a hierarchy and then allocate the forecast results down to the lower levels.

Basis Measures

Depending on the type of allocation, the basis measure may identify which cells will be the targets of an allocation, and what proportion of the allocation each target cell will receive. Different operators use the basis measure in different ways, as illustrated by the diagrams of Even and Proportional operators in "What Is an Allocation?" on page 8-1. Note that a basis measure is not used by the hierarchical operators. Refer to "Allocation Operators" on page 8-9 for descriptions of all the operators and their use of a basis measure.

The basis measure can be the same as the target measure, or it can be a different measure. For example, suppose you want to calculate the sales of each individual product for an increase in total sales of 15 percent. You would create a calculated measure from Sales that contains the desired aggregate values, and use it as the allocation source. By using the original Sales measure as both the target and the allocation basis, and allocating with the Proportional distribution method, you can generate the individual product sales figures that are needed to produce the desired total sales figure.

If, however, you want to write the results of the allocation to a completely new measure, you would still use the Sales measure as the basis. The new target enables you to preview the allocated results before overwriting the original data. Similarly, you may want to allocate data into a Budget target measure and use an Actuals measure as the basis of the allocation.

Target Measures

The target measure stores the results of an allocation. By default, the target and the basis are the same measure. However, you may prefer to use a different target measure so that you can preview the results of an allocation before overwriting any original values.

Weight Measures

You can perform a calculation on the allocated values before they are stored in the target measure. For example, you might need to convert Sales numbers to a different currency. You might create a budget in US dollars, and then translate the allocation target into local currencies. To accomplish this, you would multiply the target values by a weight measure that contains the currency translation rates.

Selecting Dimension Members for an Allocation

You can perform an allocation over an entire measure or over selected branches of the hierarchy. You must restrict the allocation to a portion of the measure under these circumstances:

- You want to allocate some of the values at the top of the hierarchy, but not all the values.

  For example, you may need to restrict the Time dimension to a few future periods to prevent allocating over all the historical data.

- You want to allocate some values that are in the middle of the hierarchy.

  For example, you may have generated a forecast at the Month level of Time and the Brand level of Product, and you want to allocate those numbers down to the base.

- You want to allocate down to the middle of the hierarchy, not to the base.

  For example, you do not want to proliferate data to the Day level of Time and the SKU level of Product, because you are setting sales quotas, which do not need that level of detail.

Identifying the Sources and Targets

The dimension members that you select for the allocation is used to identify the source and the target cells. The selection must include:

- In the source measure, the cells at the top of the hierarchy that contain the values to be allocated.
- In the target measure, the cells down the hierarchy that will be allocated values.

Figure 8–6 shows a portion of a Time hierarchy with the source allocation values at the Quarter level. How the allocation is performed depends on which members are selected (or **in status**). Table 8–1 describes various status settings and their effect on the allocation.

Figure 8–6 Allocating at the Quarter Level

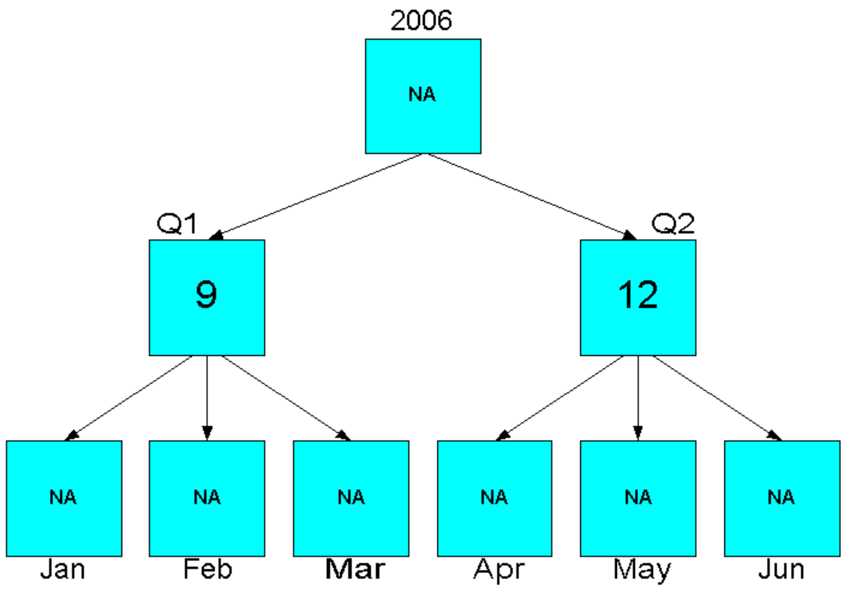
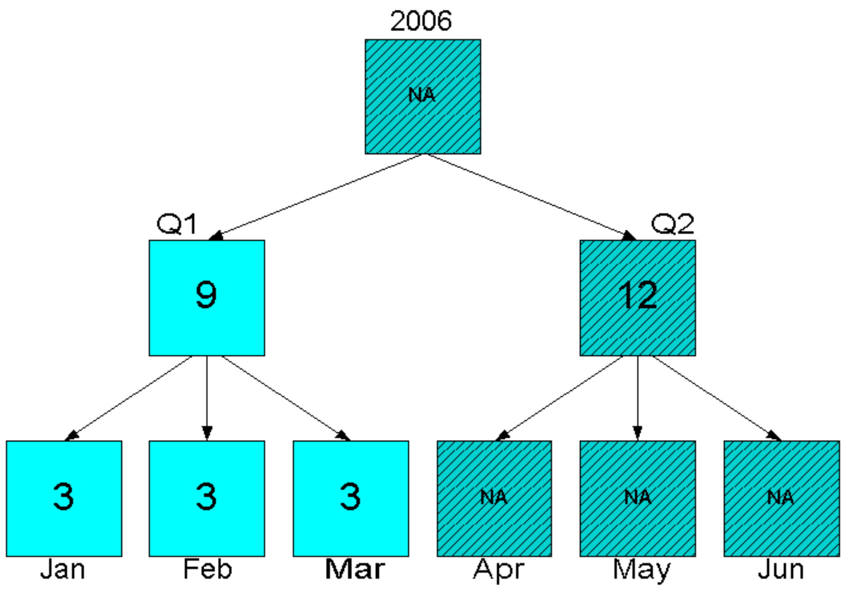


Table 8–1 Results of Status on Allocation at the Quarter Level

Status	Allocation	Explanation
All	None	The top member of the hierarchy (2006) has no value, so there is no source value to allocate.
All quarters	None	The children of Q1 and Q2 are not in status, so there is no target for allocation.
All quarters, all months	Jan to Jun	Q1 and Q2 are in status, so the value 9 is allocated to Jan, Feb, and Mar, and the value 12 is allocated to Apr, May, and Jun.
Q1, Jan to Mar	Jan to Mar	Q1 and its children are in status, so the value 9 is allocated to Jan, Feb, and Mar. Q2 is not in status and is not allocated.

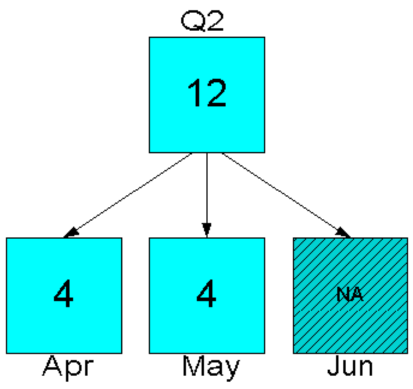
Figure 8–7 shows the correct status for allocating only Q1.

Figure 8–7 Status for Allocating One Mid-Level Branch of a Hierarchy



When calculating the allocation, the OLAP engine will, if necessary, expand the current status to include siblings. Figure 8–8 shows an even allocation when Q2, Apr and May are in status. Jun is not a target and does not get a value. Nonetheless, the engine divides the allocated value of 12 by all three children, not just the two targets, to calculate the values for Apr and May.

Figure 8–8 Even Allocation to Selected Child Members

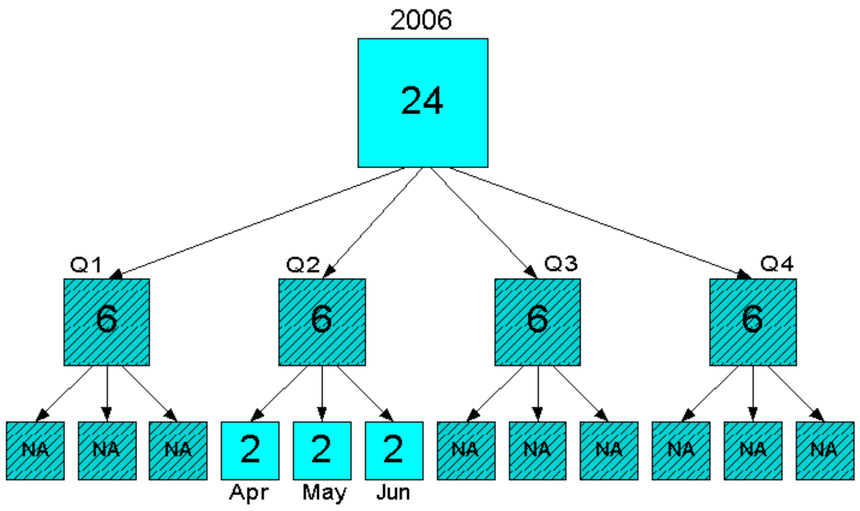


Identifying the Allocation Path

When the allocation path from the source to the target cells is not defined by the current status, the engine may populate the siblings of cells along the path. This information is important only if you want to avoid overwriting existing values or unnecessarily proliferating data.

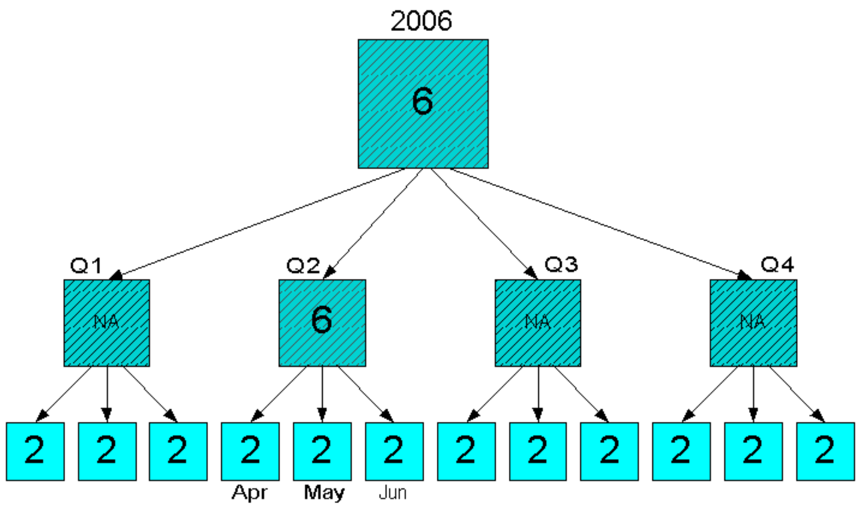
Figure 8–9 shows the results of an allocation from 2006 to the three months in Q2. Only 2006, Apr, May, and Jun are in status. This status does not define a path from the source to the target. Because the Quarter level is on the path to the target, all of the quarters are allocated a value.

Figure 8–9 Even Allocation Without a Defined Allocation Path



However, when Q2 is included in status, it is the only quarter to get an allocated value, as shown in Figure 8–10.

Figure 8–10 Even Allocation With a Defined Allocation Path



Creating an Allocation

You can create allocations in Analytic Workspace Manager by defining an allocation step in a Calculation Plan. Take these steps:

1. Create the source, basis, target, and weight measures. They must be in the same cube. The source, basis, and weight measures can be either stored measures or calculated measures. The target measure must be a stored measure.
2. Create an allocation step:
    a. In the navigation tree, create a new Calculation Plan or open an existing plan.
    b. On the General tab of the Calculation Plan property page, click **New Step**, then choose **New Allocation Step**.

       The New Allocation Step property pages are displayed.
    c. Complete the General page, being careful to select the correct source, target, and basis measures.
    d. On the Rules page, use the up- and down-arrows to list the dimensions in the order you want them calculated. If you assign different operators to different dimensions, then the allocated values may be different depending on the order.
    e. Select an operator for each dimension that you want to allocate, and a weight measure if desired.
    f. On the Status page, select the members for each dimension of the measure. To allocate values from the top down to the base, retain the default selection of All Levels. Otherwise, select the dimension members with the source data and the target members.

       Refer to "Selecting Dimension Members for an Allocation" on page 8-5 for information on selecting the dimension values.

g. Click **Create** to save the allocation step, then **Apply** to save the Calculation Plan.

3. To allocate the data, right-click the Calculation Plan in the navigation tree, then choose **Execute Calculation Plan**.
4. To view the results of the allocation, right-click the target measure and choose **View Data**.

Allocation Operators

Allocation operators determine the methodology for distributing source values to their targets. There are three basic types of allocation operators: Copy, Even Distribution, and Proportional Distribution.

Within these basic types are regular operators and hierarchical operators. The regular operators only assign values to cells identified by the basis measure as having a value. The hierarchical operators do not use a basis measure. They assign values to all target cells.

Note: The hierarchical operators may increase the size of a measure dramatically by allocating values to previously empty cells. Be careful to set the status of all dimensions.

Copy Operators

These are the copy operators:

- **Copy**: Copies the allocation source to all of the target cells that have a basis value that is not NA (null).
- **Hierarchical Copy**: Copies the allocation source to all of the target cells specified by the hierarchy, regardless of the basis value.
- **Minimum**: Copies the allocation source to the target that has the smallest basis value.
- **Maximum**: Copies the allocation source to the target that has the largest basis value.
- **First non-NA Data Value**: Copies the allocation source to the first target cell that has a non-NA basis value.
- **Hierarchical First Member**: Copies the allocation source to the first target cell specified by the hierarchy, regardless of the basis value.
- **Last non-NA Data Value**: Copies the allocation source to the last target cell that has a non-NA basis value.
- **Hierarchical Last Member**: Copies the allocation source to the last target cell specified by the hierarchy, regardless of the basis value.

Even Distribution Operators

These are the even distribution operators:

Even: Divides the allocation source by the number of target cells that have non-NA basis values and applies the quotient to each target cell.

Hierarchical Even: Divides the allocation source by the number of target cells, including the ones that have NA values, and applies the quotient to each target cell.

Proportional Distribution Operator

The proportional distribution operator is:

Proportional: Divides the allocation source by the sum of the basis values, then multiplies the quotient by the individual basis value for each target cell.

Relationships Between Allocation and Aggregation Operators

The allocation system operates as the logical inverse of the aggregation system. In other words, if you allocate down from a middle level of a hierarchy, you can aggregate up to the top of the hierarchy using an aggregation operator that corresponds to the allocation operator. Table 8–2 shows the correspondence between allocation operators and aggregation operators.

Table 8–2 Corresponding Allocation and Aggregation Operators

Allocation Operator	Aggregation Operator
Copy	Average
Hierarchical Copy	Average
Minimum	Minimum
Maximum	Maximum
First non-NA Data Value	First Non-NA Data Value
Last non-NA Data Value	Last Non-NA Data Value
Hierarchical First Member	Hierarchical First Member
Hierarchical Last Member	Hierarchical Last Member
Even	Sum or Average
Hierarchical Even	Hierarchical Average
Proportional	Sum

Case Study: Allocating a Budget

This example creates a sales budget that is 10% higher than the previous year's sales. It uses a calculated measure to generate the increase, then distributes the total increase evenly down the dimension hierarchies.

Creating the Source Measure

To create the source measure:

1. Expand the `UNITS_CUBE` folder, right-click **Calculated Measures**, and choose **Create Calculated Measure**.

   The Calculation Wizard opens.

2. Complete the Name and Type page with these values:

   - Name: `sales_py`
   - Calculation Type: **Prior Value** (under Prior/Future Comparison)

3. Complete the Prior Value page with these values:

   - Measure: **Sales**

   - Over Time in: **Calendar Year**
   - Go back by: **1 Year**
4. Create a second calculated measure with the name SALES_BUDGET.
5. For the calculation, expand the Basic Arithmetic folder and choose Multiplication.
6. On the Multiplication page, multiply SALES_PY by 1.1.

Creating the Target Measure

This example stores the allocated data in a separate measure from the source data to assure that the allocated data does not overwrite any source data.

To create the target measure:

1. In the UNITS_CUBE folder, right-click Measures and select **Create Measure**.

   The Create Measure dialog box opens.
2. On the General page, create a measure named ALLOC_SALES_BUDGET.
3. Select **Override the Aggregation Specification of the Cube.**
4. On the Summarize To page, deselect all levels for all dimensions.

The measure is not mapped to a data source, so no aggregation needs to be done during regular builds. Instead, aggregation will be defined in the Calculation Plan. The aggregation step is not shown in this example; refer to "Case Study: Forecasting Sales for Global Enterprises" on page 9-14 for an example that shows forecasting, allocation, and aggregation.

Creating the Calculation Plan

Budget Plan will have an allocation step and an aggregation step (not shown).

To create a new Calculation Plan:

1. Right-click Calculation Plans and select **Create Calculation Plan**.

   The Create Calculation Plan dialog box opens.
2. Create a new plan named BUDGET_PLAN. Click **Create**.

   BUDGET_PLAN appears as a new item in the Calculation Plans folder. It does not yet contain any steps.

Creating the Allocate Budget Step

The SALES_BUDGET calculated measures generates data at all levels. The allocation will redistribute the data from the top of the hierarchy to the lowest levels and store it in the target measure.

To create an allocation step:

1. On the General page of Sales Plan, click **New Step**, then select **New Allocation Step**.

   The Create Allocation Step dialog box opens.
2. Complete the General page with these values:
   - Name: allocate_budget_step
   - Cube: UNITS_CUBE

- Source Measure: `SALES_BUDGET`
- Target Measure: `ALLOC_SALES_BUDGET`
- Basis Measure: `SALES_BUDGET`

3. On the Rules page, assign **Hierarchical Even** for the Time operator. For the other dimensions, assign the **Proportional** operator.
4. On the Status page, keep the default status of **All Levels** for all dimensions.
5. Click **Create** to save the allocation step.
6. Click **Apply** to save the Calculation Plan.

Generating and Validating the Allocation

To generate the allocation:

1. Expand the Calculation Plans folder. Right-click `BUDGET_PLAN` and choose **Execute Calculation Plan BUDGET_PLAN**.

   The Maintenance Wizard opens, and `BUDGET_PLAN` is a selected target object.
2. Click **Finish**.

   The build log is displayed when the Calculation Plan is done executing.

To view the forecast results, take these steps:

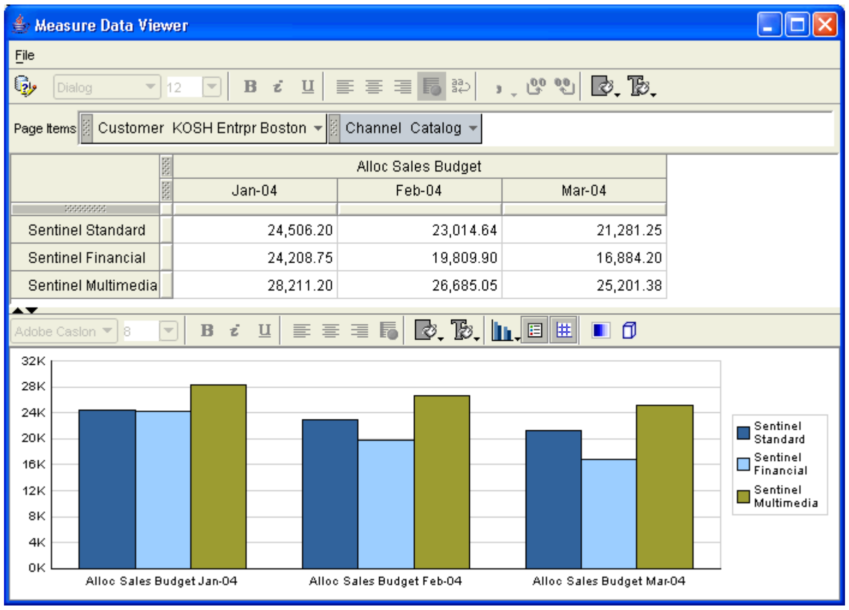
1. Fully expand the `UNITS_CUBE` folder, right-click the `ALLOC_SALES_BUDGET` measure, and choose **View Data ALLOC_SALES_BUDGET**.

   The Measure Data Viewer opens. No data is displayed, because the top dimension levels provide the source data, not the allocated data.
2. From the File menu, choose **Query Builder**.

   The Query Builder opens.
3. On the Layout tab, switch Product and Customer. Click **Help** for instructions.
4. On the Dimensions tab, set the status of all dimensions to the base level. You may wish to select just a few values from these lists. For Time, limit the months to 2004, since that it is only allocated year.
5. Click **OK** to close the Query Builder.

Figure 8–11 shows a sample of the allocated data. The allocated data should be aggregated from these base levels to the top by an aggregation step.

Figure 8–11 Allocated Sales Budget Data



Page Items: Customer KOSH Entrpr Boston | Channel Catalog

	Alloc Sales Budget		
	Jan-04	Feb-04	Mar-04
Sentinel Standard	24,506.20	23,014.64	21,281.25
Sentinel Financial	24,208.75	19,809.90	16,884.20
Sentinel Multimedia	28,211.20	26,685.05	25,201.38

9 Generating Forecasts

Forecasting is a natural extension to the types of data analysis typically performed on the historical data stored in analytic workspaces. Using Analytic Workspace Manager, you can quickly generate forecasts of your measures. This chapter provides a basic framework for generating and using quantitative forecasting methods for those who do not have a strong statistical background. It also provides specific information about the particular forecasting engine provided with Oracle OLAP.

This chapter contains the following topics:

Introduction to Forecasting Considerations

Forecasts are predictions about future events. They provide a basis for making decisions in a timely manner, which is often in advance of the facts. There are many ways of creating forecasts, and the best method for a particular forecast depends on many factors. Consider this question: Will it rain tomorrow?

The degree of difficulty in correctly predicting tomorrow's weather depends on where you live. You may live where the weather is extremely stable, with little or no variation from one day to the next. In this situation, if it is raining today, then you can be fairly certain that it will rain tomorrow.

However, if you live where the weather is in constant flux, with sudden and dramatic changes, then today's rainfall is not a reliable predictor. You may just make an informed guess, based on your analysis of the current weather pattern, or you might consult an arthritis sufferer whose joints ache with changes in the weather. Nonetheless, all of these methods (today's rainfall, informed guess, or swollen joints) should over time prove to be more accurate than just flipping a coin.

Now consider this question: Will it rain three months from today? Instead of basing your prediction on today's weather, you need to consider the frequency of rainfall for the forecast period in previous years. If you live where rainy seasons and dry seasons are clearly defined, then you can probably answer this question with relative certainty based on the season. Otherwise, your ability to predict rainfall on a particular day that far into the future may be no better than a coin toss. To make a meaningful prediction, you may need to expand the forecast period to a week or more. You may also need to expand the size of the area in which you are predicting rain from your neighborhood to a larger region.

Finally, how important is it to correctly predict the weather on a particular day and at a particular place? If accuracy is critical -- such as planning a large outdoor event -- then an accurate forecast is worth some effort, and you might try several forecasting methods to see if their predictions converge. Regardless, you might still plan to erect a tent in case you get a downpour instead of the forecasted clear skies.

This simple example demonstrates several characteristics of forecasting:

- Stable patterns in historical data are more likely to generate an accurate forecast.
- Different methods are appropriate for different forecasts, depending on how far into the future you want to make a forecast and how stable your data is.
- Some forecasting methods are experiential or qualitative (informed guess or aching joints), and others are quantitative (historical data).
- The season may be an important factor in the forecast.
- Forecasting is not 100% accurate.
- The more precise the forecast, the more prone it is to error.
- Longer-range forecasts should generate data at higher levels to offset the increasing likelihood of error.
- The degree of error may be offset by your tolerance for error.
- If you have a low tolerance for error, then you may want to make some provisions that will lessen the consequences of forecasting incorrectly.

These observations may help give you a perspective on what you want to forecast, how you want to design the forecast, and how you want to use the forecast in making decisions about your business.

Choosing a General Forecasting Approach

The first step in generating a forecast is to decide how far into the future you want to make your predictions. The approach that produces the best results for short-term forecasts is not a good predictor of long-term performance. The opposite is also true.

The critical question is, of course, how far into the future these time frames reach. Is "short" five weeks or five months? Is "long" five quarters or five years? As illustrated by the rain prediction example in "Introduction to Forecasting Considerations" on page 9-1, it all depends on a variety of factors:

- What are you trying to forecast?
- How stable is the historical data?
- How are you going to use this information?

These are just a few of the questions that you need to answer in order to define the forecasting time frames for your specific business. Table 9–1 provides some general guidelines for these time periods.

Table 9–1 Guidelines for Choosing a Forecasting Approach

Time Frame	Typical Forecasting Horizon	Best Approach
Short	Up to 18 months	Time Series
Medium	6 to 36 months	Causal Analysis
Long	19 months to 5 years	Expert Opinion

Time Series

Time series forecasting methods are based on the premise that you can predict future performance of a measure simply by analyzing its past results. These methods identify a pattern in the historical data and use that pattern to extrapolate future values.

Past results can, in fact, be a very reliable predictor for a short period into the future. You can generate this type of forecast very quickly and easily, and you do not need either forecasting expertise or an in-depth knowledge of your data. The modeling techniques used by the time-series methods are relatively simple and run very fast. Time-series forecasting is extremely useful when hundreds or thousands of items must be forecast.

You may also use time-series methods to generate forecast data further into the future. However, the results will not be as accurate, because factors other than past performance have a greater impact over time. For example, you or your competitors may change the pricing structure or run advertising campaigns, competitive products may come onto the market, or shifts in the economy or political events may affect performance. You should consider the forecast data generated by time series methods to be one component of a medium- or long-range forecast, which may be adjusted by expert opinion and other factors.

Analytic Workspace Manager provides access to a time-series forecasting engine, which is described in this chapter.

Causal Analysis

Causal analysis takes into consideration the external factors (the causes) that can affect a forecast, as described under "Time Series". Statistical regression methods are the basis for causal analysis. They use the forecasts for several independent measures to forecast a dependent measure. This type of forecast requires considerable skill and understanding of forecasting methodology and the relationships between independent and dependent variables. A good regression model will produce the best results for medium-range forecasts.

However, because of the time, expense, and expertise needed to develop a model, most businesses restrict regression analysis to a few key business measures. For the other measures, they use a combination of methods including time-series and expert opinion.

The forecasting engine used by Analytic Workspace Manager does not support causal analysis. The linear and nonlinear regression methods in the forecasting engine are time-series regression methods that use historical data from a single measure.

Note: Oracle Data Mining supports both time series and causal analysis methods for data stored in a relational format. This type of forecasting is done using the SQL `PREDICTION` function within a Data Mining model. Refer to *Oracle Data Mining Concepts*.

Expert Opinion

As the time horizon for the forecast moves further out into the future, expert opinion becomes the most reliable predictor. The experts, who are usually corporate executives, have their fingers on the pulse of myriad factors that may influence future performance, such as the general direction of the market and plans for new products. Customer surveys also provide input to long-term forecasts. An equivalent computer model to rival expert opinion for long-term forecasts would be too complex to generate within a usable time frame.

About the Forecasting Engine

Oracle OLAP incorporates a statistical forecasting engine that is used extensively for demand planning applications. This engine has a variety of time-series forecasting methods, which are described in "Forecasting Method Descriptions" on page 9-9.

The forecasting engine incorporates advanced filtering technology to identify and process outliers, which are data values that are extremely high or low in relation to the mean. Exception handing is a critical component of forecasting efficiency, and the forecasting engine reduces the time and money spent analyzing exceptions. This technology also enables the forecasting engine to produce accurate short-term forecasts using wildly fluctuating historical data.

Typical applications for OLAP forecasting include the following:

- Distribution requirements planning for seasonal monthly forecasts of retail sales for products reaching market saturation.
- Business planning with seasonal quarterly forecasts of expenses with upward linear trends.
- Sales quota management by forecasting exponential decay in company sales for aging products.
- Materials requirement planning with trends in raw material prices with cyclical behavior.
- Sales forecasts with exponential growth in industry sales.
- Inventory control planning by forecasting S-curve demand growth from increasing distribution.

Creating a Forecast

You can create forecasts in Analytic Workspace Manager by defining a forecast step in a Calculation Plan. 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. Creating a Forecast Measure
3. Selecting the Historical Data
4. Identifying the Levels for the Forecast
5. Creating a Forecast Step
6. Generating the Forecast
7. Evaluating the Forecast Results

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 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.

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

Creating a Forecast Measure

You can store the forecast data in the same measure as the actual data, or you can store it in a separate measure. If you store the forecast in the same measure, then the actual data will eventually overwrite it. This prevents you from monitoring the accuracy of the forecast. For this reason, you should create a separate forecast measure in the same cube as the source measure.

To create a forecast measure:

1. In the navigation tree, expand the cube for the actual data.
2. Right-click Measures and choose **Create Measure**.
3. Complete the Create Measure property sheet. Do not map the measure to a data source.

Selecting the Historical Data

The forecasting engine needs only a year of data to detect trends and seasonality. Business cycles may take two or three years of data to detect.

If your business has experienced a paradigm shift, then you should exclude previous data from your forecast as irrelevant. The following are examples of events that might cause a paradigm shift:

- Cellular telephones on the telecommunications industry.
- Digital cameras on the photography industry.
- The Internet on the book and music publishing industries.

You will select the historical data when creating the forecast step.

Identifying the Levels for the Forecast

To generate consistent data at all levels of a hierarchy, you must generate the forecast data at a single level and use it to populate the other forecast levels by aggregation or allocation. If you generate a forecast from multiple levels, then the aggregate forecast data may be inconsistent with the lower levels of forecast data.

The "correct" levels are determined by the time frame of your forecast and by your reasons for making the forecast. For example, you may forecast Customers at the Total level for manufacturing, but at a lower level for marketing. Table 9–2 shows the recommended dimension levels for forecasting products over various time frames.

If you set the levels too low, then large variations in the data may decrease accuracy. These inaccuracies may be magnified in the aggregated forecasts. If you set the levels too high, then the aggregated forecasts may smooth out localized trends and allocate them incorrectly.

Table 9–2 Example of Dimension Levels for Forecasts

Time Frame	Time Level	Product Level	Other Dimension Levels
Short	Week, Biweek, or Month	UPC, SKU, NDC, ISBN	Level of interest
Medium	Month or Quarter	Brand	Level of interest
Long	Quarter or higher	Brand, Company, Market	Level of interest

You will select the levels when creating the forecast step.

Creating a Forecast Step

To create a forecast step in Analytic Workspace Manager:

1. In the navigation tree, create a new Calculation Plan or open an existing plan.
2. On the General tab of the Calculation Plan, click **New Step**, then select **New Forecast Step**.

   The New Forecast Step property pages are displayed.
3. Complete the General page. For the forecast method, select **Automatic**.

   For information about using other methods, refer to "Designing Your Own Forecast" on page 9-7. For information about completing the other fields, click **Help**.
4. Keep the default values on the Advanced Settings page unless you have expertise in time-series forecasting.
5. On the Status page, select the historical time periods and other dimension values that will be used as the basis for the forecast. Select only one level for each dimension.
6. Save the forecast step, then save the Calculation Plan.

Note: Depending on how you set up the forecast, you may need to follow it with an allocation step, or an aggregation step, or both to populate all levels.

Generating the Forecast

To generate the forecast data:

If all the time periods and data are already loaded into the analytic workspace, then right-click the Calculation Plan and choose **Execute Calculation Plan**.

or

If you need to load new data, then include the Calculation Plan in the regular maintenance process using the Maintenance Wizard.

Afterward, you can view the forecast data in the Measure Viewer.

Evaluating the Forecast Results

If the forecast does not initially look plausible to you, then check that there are no errors in the design of the forecast:

- Compare the first few forecast periods to the last few historical periods to verify that a discrepancy exists.
- Use the forecast step editor to check the number of forecast periods against the status of the Time dimension. The forecast periods are the last ones in status. For example, if the Time dimension has dimension members defined through the next five months and you designed a 4-month forecast, then you must remove the last month from status. Otherwise, the forecast will be based on a month of null historical data.
- Use the Measure Viewer to verify that all of the historical data has been loaded in the source measure. If several periods immediately prior to the forecast period have not been loaded, then the forecast will be 0.
- If you used a specific forecasting method (not Automatic):
  - Compare its results with those of the Automatic option.
  - Verify that you set Forecast Approach to Manual and Data Filter to the appropriate choice.
- If you set any of the advanced parameters, then compare the results against a forecast that uses the default settings.

A standard part of forecasting is to continually monitor the accuracy of the forecast data. The easiest way to compare the forecast data with the actual data is to set up a standard report that includes a line graph. Then you can see how closely the forecast data converges with the actual data.

Short-term forecasts should be fairly precise, with only a small difference between forecast and actual data. If this is not the case, then you should consider modifying the forecast using some of the suggestions listed previously. You may even want to create several forecasts and compare their results over time.

Medium- and long-range forecasts generated by time-series forecasting methods should be qualified by other input, such as expert opinion, because external factors will affect performance in these time frames.

Designing Your Own Forecast

The OLAP forecast engine provides an Expert System that generates the best short-term forecasts over the long run, so you should use the Automatic method and the default parameters for most forecasts. However, there may be times when you should override the Expert System and design the forecast yourself.

What is the Expert System?

The Expert System supports the Automatic method by identifying the best statistical method and selecting the best parameter settings for your data. It also distinguishes outliers from factors like trend and seasonality.

The Expert System separates the data into seasonal effects and trend effects. It then uses an iterative approximation method to forecast the seasonal component of the data. After completing the trend forecast, it factors the seasonality portion into the trend forecast for all methods except Holt-Winters, which calculates its own seasonal factors.

The Expert System represents a type of artificial intelligence for statistical forecasting that has been in common use ever since computers took over the task of performing complex and lengthy numerical calculations. Instead of the analyst's having to evaluate the data and make an educated guess as to the best method, the software can quickly try all methods and select the best one based on the results.

You can override the Expert System by setting the Forecast Approach parameter to Manual. The default value of Automatic gives the Expert System the most control in overriding your choices. This is the appropriate setting when using the Automatic method, but it will invalidate your attempt to design a forecast.

What is the Verification Window?

The Expert System always tests the accuracy of a forecast method using a portion of the historical data called a verification window. For the Automatic method, the Expert System uses this window to select the best statistical method. For the other methods, it verifies that your selection of a method and the parameter settings provide a good fit to the historical data.

For this test, the Expert System divides the historical time periods into two groups. The older time periods retain their role as historical data. The newer historical time periods become the "forecast" periods and form the verification window. The Expert System generates forecast data for the newer time periods, using the older time periods as the basis for the forecast.

The Expert System calculates the precision of the method by comparing the forecast data to the actual data in the verification window. The precision is the distance between the forecast data and the actual data.

The Expert System uses several standard calculations to compare the precision of different forecasting methods: Mean Absolute Deviation (MAD), Mean Absolute Percentage Error (MAPE), and Root Mean Square Error (RMSE).

When Should You Design a Forecast?

You may want to control the forecast when you have special knowledge that future performance will deviate from past results.

For example, you may recently have entered an agreement for a major national chain of stores to carry your products, so you anticipate a dramatic increase in sales. Or your company might have been an innovator in developing a new product line, but your competitors are about to introduce rival products. In this case, you expect sales to level off. You or your competitors might also be negotiating a corporate merger, and you expect that transaction to affect performance.

Under circumstances like these, your special knowledge may enable you to design a more accurate forecast than the Expert System.

Overriding the Expert System

To override the Expert System, take these steps:

1. Create or edit a forecast step, as described in "Creating a Forecast Step" on page 9-6.
2. On the General page, select the method that best describes the future performance that you expect, based on your expert knowledge.
3. On the Advanced Settings page, set **Forecast Approach** to **Manual**.

4. Set the Data Filter parameter to an appropriate setting for your data.
5. Change the Verification Window Size parameter as desired.
6. Make whatever other changes to the parameter settings are appropriate.
7. Complete the definition of the forecast, and run it as described in "Creating a Forecast" on page 9-4.

Forecasting Method Descriptions

The forecasting methods represent several basic approaches to time-series forecasting. This topic provides descriptions of the various approaches, the methods that use each approach, and the optimization parameters that apply specifically to them.

Automatic

The Expert System identifies the best fit by quickly testing each statistical method against the portion of historical data specified by the Verification Window Size parameter. The Expert System selects the method and the parameter settings that would have generated the most accurate forecast in the past. It automatically detects and handles outliers, removing noise so that it can better detect trends and seasonality.

The forecasting engine generates a forecast for every combination of dimension members. The Expert System evaluates each forecast separately and picks the best method and parameter settings for each one.

In general, Automatic is the best choice unless you have knowledge that future performance will deviate from the past. Under these special circumstances, you can substitute your own expert judgment for the Expert System.

"What is the Verification Window?" on page 9-8 provides more information about how the Expert System selects a method.

Regressions

Time series regression methods relate a variable (measure) to functions of time describing trend and seasonal components. Regression generates the most reliable forecasts when the trend or seasonal components remain constant.

OLAP forecasting provides both linear and nonlinear regression models.

Linear Regression

Linear regression attempts to fit the historical data to a straight line (y=ax+b), and extends that line into future time periods for the forecast. All data points have equal weight. This method identifies steady, long-term trends in the data.

Nonlinear Regression

Nonlinear regression attempts to fit the historical data to a curve, and extrapolates that curve into the forecast time periods. All data points have equal weight. The curved lines are defined by mathematical equations. You can choose from the following types of curves:

- **Polynomial**: Fits data that fluctuates with a rise and a drop (x'=log(x); y'=log(y)).
- **Exponential**: Fits data points that rise or drop at an increasingly faster rate (x'=x; y'=ln(y)).

- **Logarithmic**: Fits data points that rise or drop quickly and then level off (x'=log(x); y'=y).
- **Asymptotic**: Fits data points that rise or drop until they approach a fixed value and then level off (x'=1/x; y'=1/y).
- **Exponential Asymptotic**: Fits data points that rise or drop at an increasingly faster rate until they approach a fixed value and then level off (x'=x; y'=ln(y/(K-y))).

For more information about the equations used by each method, refer to the topic "Equations for Forecasting Methods" in Analytic Workspace Manager Help.

Advanced Parameter for Regressions

The Cyclical Decay smoothing constant is used in the equations for linear and nonlinear regression. This constant determines how quickly a cycle reverts to the mean. A higher value implies slower decay while a lower value implies faster decay. The smaller the value, the less effect cyclical activity has on the forecast.

You can specify a maximum value, a minimum value, and a step, or you can specify the same value for both the maximum and the minimum. The step is an incremental value between the maximum and minimum, which the forecasting engine uses to find the optimal value of the constant. Keep the default settings unless you have a strong background in time-series forecasting.

Note: For the optimization parameters, a maximum of 3 to 5 steps is sufficient to find the best value. Increasing the number of steps increases the time it takes to generate a forecast, so that an increase in just two or three parameters may noticeably impact performance. However, it may also yield a small improvement in the forecast. The extra time may be worthwhile for small- to medium-sized measures (up to 10,000 products), but not for larger measures.

Exponential Smoothing

The exponential smoothing methods weight the historical data using exponentially decreasing weights. The prior period has the most weight and each period prior to it has comparatively less weight. The decline in weight is expressed mathematically as an exponential function. The smoothing parameters determine the weights.

Comparison Among Exponential Smoothing Methods

You can choose from the following methods of exponential smoothing:

- **Single Exponential Smoothing**: Identifies the percentage of weight given to the prior period and all other historical periods. It does not adjust for trend or for seasonal variance.
- **Double Exponential Smoothing**: Identifies the trend, and adjusts the forecast data to reflect this trend instead of generating a single parameter for all forecast periods.
- **Holt-Winters Exponential Smoothing**: Identifies both trend and seasonal variance, and adjusts the forecast data to reflect these factors. This method is particularly sensitive to both high and low outliers. A better choice for handling seasonality is Double Exponential Smoothing with the Data Filters parameter set to Seasonal Adjustment.

Advanced Parameters for Exponential Smoothing

These smoothing constants are used in the equations for exponential smoothing methods. Keep the default settings unless you have a strong background in time-series forecasting.

- **Alpha**: Determines how responsive a forecast is to sudden jumps and drops. It is the percentage weight given to the prior period, and the remainder is distributed to the other historical periods. Alpha is used in all exponential smoothing methods.

  The lower the value of alpha, the less responsive the forecast is to sudden change. A value of 0.5 is very responsive. A value of 1.0 gives 100% of the weight to the prior period, and gives the same results as a prior period calculation. A value of 0.0 eliminates the prior period from the analysis.

- **Beta**: Determines how sensitive a forecast is to the trend. The smaller the value of beta, the less weight is given to the trend. The value of beta is usually small, because trend is a long-term effect. Beta is not used in Single Exponential Smoothing.

- **Gamma**: Determines how sensitive a forecast is to seasonal factors. The smaller the value of gamma, the less weight is given to seasonal factors. Gamma is used only by the Holt-Winters method.

- **Trend Dampening**: Determines how sensitive the forecast is to large trends in recent time periods. Dampening identifies how quickly the trend reverts to the mean. A higher value implies slower dampening while a lower value implies faster dampening. The smaller the value, the less effect the trend has on the forecast.

For each constant, you can specify a maximum value, a minimum value, and a step. The step is an incremental value between the maximum and minimum, which the forecasting engine uses to find the optimal value of the constant. For more information about steps, refer to "Advanced Parameter for Regressions" on page 9-10.

Advanced Parameter Descriptions

Following are descriptions of the advanced parameters that can be used with all methods.

Parameters that are specific to a particular approach are described in "Forecasting Method Descriptions" on page 9-9.

Note: When using a specific forecasting method (not Automatic), be sure to set the following parameters:

- Forecast Approach
- Data Filter

Setup Parameters

These parameters provide the forecasting engine with basic information about how you want it to approach a forecast. Always set the Forecast Approach and Data Filter parameters when using a specific forecasting method.

- **Forecast Approach**: Specifies whether the forecasting engine gives control to the Expert System.
  - **Automatic**: Give control to the Expert System. Use this setting with the Automatic method.
  - **Manual**: Give control to the user. It enables you to choose a method and set the parameters that are appropriate for the historical data. Use this setting with all methods other than Automatic.
- **Data Filter**: Identifies a basic characteristic of the data.
  - **Non-Seasonal Data**: No seasonality.
  - **Seasonal Data**: Adjust for seasonal patterns in the data. You can use this filter with Double Exponential Smoothing to get a more accurate forecast than Holt-Winters.
  - **Intermittent Data** Adjusts for sporadic or intermittent data and, if appropriate, seasonal patterns. Intermittent data has null or zero for over 50% of the values. Do not use median smoothing with this filter, because smoothing eliminates the intermittent characteristic of the data. The purpose of the intermittent data filter is to forecast intermittent demand.

    Set the Moving Periodic Total Decay parameter when using this filter.
- **Verification Window Size**: The Expert System uses the verification window to determine the best method and parameter settings, as described in "What is the Verification Window?" on page 9-8.

  The verification window is specified as a fraction of the total number of historical periods. For example, assume that you have three years of historical data for 2004, 2005, and 2006. The default window size is .3333, so the Expert System will use 1/3 of the historical data for the verification window. Thus, the data for 2004 and 2005 will be used to generate a "forecast" for 2006. The difference between the forecast data and the actual data for 2006 indicates the precision of the method.

  You may want to adjust the window size, depending on the granularity of the data. For monthly data, use a window size of 20% (1/5) or more. For weekly data, use a window size of 12.5% (1/8) or more. For daily or hourly data, you can use a window size of 11.1% (1/9) or less.

General Parameters

These parameters apply to all of the specific forecasting methods.

- **Allocate Last Cycle**: Controls whether the last cycle is calculated by forecasting alone or with allocation. Allocation may reduce the risk of overadjustment for trend or seasonality.

  Allocation forecasts an average value for one period of the last cycle. That average value is then multiplied by factors to give the remaining points in that period. For example, a forecast at the day level would calculate an average for all days in the last week rather than forecasting individual days.

  Set Periodicity to a value greater than 1 when using this parameter.

- **Boundary Maximum and Minimum**: Boundaries constrain the forecasting engine from occasionally generating unreasonably high or low values. The upper boundary is calculated by multiplying Boundary Maximum by the largest value in the historical series. The lower boundary is calculated by multiplying Boundary Minimum by the smallest value in the historical series.

  For example, if the Boundary Maximum parameter is 100.0 and the largest historical value 5,600, then no forecast value can be greater than 560,000. If the Boundary Minimum parameter is 0.5 and the smallest historical value 300, then no forecast value can be less than 150.

- **Moving Periodic Total Decay**: The maximum value of a decay constant that is inversely related to noise, random deviation, and stability in the history of intermittent data. Set this value higher when the history is evolving rapidly from one cycle to the next or when the noise level is low. This parameter is used only with the Intermittent Data filter. The difference between the maximum and the minimum must be evenly divisible by 0.4.

- **Periodicity**: The number of periods in a single cycle or the number of periods in each set of nested cycles. The default value of 1 does not group the periods at all, so each period is logically independent.

  For example, if you are using Month as the base level for the forecast, and the time hierarchy has levels for Month, Quarter, and Year, then the cycles are 12 months in a year and 3 months in a quarter. For a single cycle, enter the number of periods. For nested cycles, list the cycles in parentheses from the most aggregate to the least aggregate, separated by commas, such as (12,3).

- **Trials**: The number of trials that are run to determine the best method and combination of parameter settings.

Historical Data Smoothing Parameters

These parameters help generate a smoother forecast from intermittent historical data. Alternatively, you can use the intermittent data filter to forecast intermittent demand. Do not combine the smoothing parameters with the intermittent demand filter, because these adjustments are contradictory.

- **Use Smoothed Historical Data**: Controls whether the historical data is smoothed. Smoothing is typically used for weekly or finer-grained data that has many missing values. Smoothing the historical data produces a smoother baseline forecast.

- **Interpolate Missing Values**: Specifies whether you want to smooth the data by inserting estimates for missing values instead of by averaging. This parameter is useful when missing values indicate incomplete data instead of a lack of activity.

- **Median Smoothing Window**: The number of time periods used in a median smoothing window to identify outliers and replace them with adjusted data values. Median smoothing eliminates extreme variations in the data by replacing each data point in a series by the median value of itself and its neighbors. This setting must be an odd number, so that the current time period is in the center of the window.

  The larger the window, the smoother the data. If the window is too large, smoothing may eliminate important data patterns. If the window is too small, then smoothing may include outliers that could not be filtered out. As a rule, you should not set this parameter below 3; setting it to 1 has the effect of turning off smoothing.

For monthly data, use a maximum value of 5 to prevent excessive flattening of the data. For weekly data, use a maximum of 13. Use a longer window (15 or more) for daily or hourly data.

Case Study: Forecasting Sales for Global Enterprises

The `GLOBAL` analytic workspace has historical data from January 1998 to July 2004. Thus, the last five months of 2004 and all of 2005 is NA. This example creates a Calculation Plan that generates a four-month Sales forecast from August 2004 to December 2005. An allocation step distributes the forecast data down to the base levels of all dimensions. An aggregation step generates and stores some of the aggregate values to improve runtime performance.

Creating the Sales Forecast Target Measure

This example stores the forecast data in a separate measure from the historical data so that the results of the forecast can be evaluated more easily.

To create the target measure:

1. In the `UNITS_CUBE` folder, right-click Measures and select **Create Measure**.

   The Create Measure dialog box opens.

2. On the General page, create a measure named `SALES_FORECAST`.
3. Select **Override the Aggregation Specification of the Cube.**
4. On the Summarize To page, deselect all levels for all dimensions.
5. Click **Create**.

The measure is not mapped to a data source, so no aggregation needs to be done during regular builds. Instead, aggregation will be defined in the Calculation Plan.

Creating the Calculation Plan

Sales Plan will have a forecast step, an allocation step, and an aggregation step.

To create a new Calculation Plan:

1. Right-click Calculation Plans and select **Create Calculation Plan**.

   The Create Calculation Plan dialog box opens.

2. Create a new plan named `SALES_PLAN`. Click **Create**.

   `SALES_PLAN` appears as a new item in the Calculation Plans folder. It does not yet contain any steps.

Creating the Sales Forecast Step

To create the forecast step:

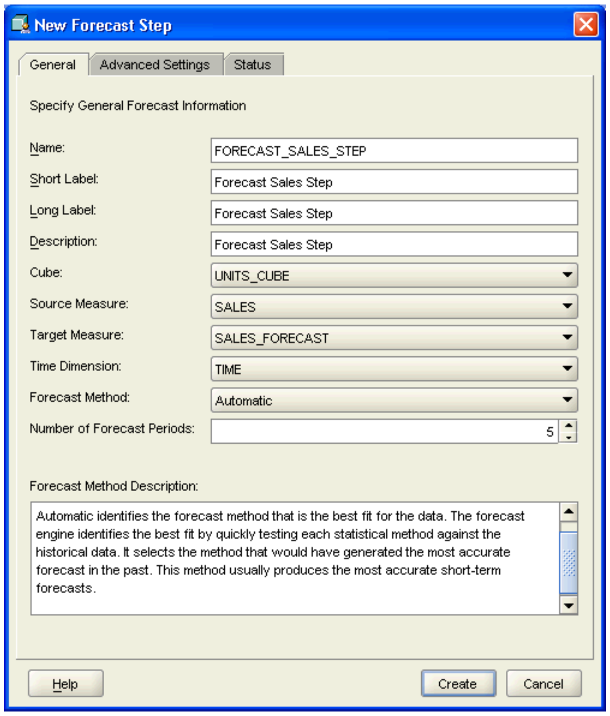
1. On the General page of `SALES_PLAN`, click **New Step**, then select **New Forecast Step**.

   The Create Forecast Step dialog box opens.

2. Complete the General page with these values, as shown in Figure 9–1
   - Name: `forecast_sales_step`
   - Cube: `UNITS_CUBE`

- Source Measure: `SALES`
- Target Measure: `SALES_FORECAST`
- Time Dimension: `TIME`
- Forecast Method: `Automatic`
- Number of Forecast Periods: `5`

Figure 9–1 Forecasting Global Sales



Specify General Forecast Information

Name:	FORECAST_SALES_STEP
Short Label:	Forecast Sales Step
Long Label:	Forecast Sales Step
Description:	Forecast Sales Step
Cube:	UNITS_CUBE
Source Measure:	SALES
Target Measure:	SALES_FORECAST
Time Dimension:	TIME
Forecast Method:	Automatic
Number of Forecast Periods:	5

Forecast Method Description:

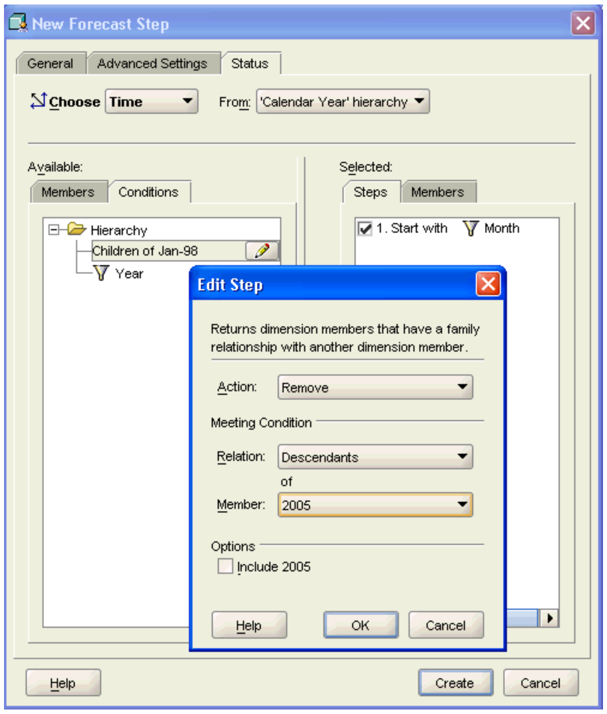
Automatic identifies the forecast method that is the best fit for the data. The forecast engine identifies the best fit by quickly testing each statistical method against the historical data. It selects the method that would have generated the most accurate forecast in the past. This method usually produces the most accurate short-term forecasts.

3. Keep the default settings on the Advanced Parameters page.
4. On the Status page, set the Time dimension:
   a. On the Selected Steps tab, click **All Levels** and select **Month** from the drop-down list.
   b. On the Available Conditions tab, expand the Hierarchy folder. Select **Children of Jan-98** and click the Edit Step icon.

      The Edit Step dialog box opens, as shown in Figure 9–2.
   c. Set Action to **Remove**, and set Relation to **Descendants**.
   d. Click **Member** and choose **More** from the list.

      The Select Members dialog box opens.
   e. Select **2005**.
   f. Click **OK** to close the Select Members dialog box, then click **OK** to close the Edit Step dialog box.
   g. Add this condition to the Selected Steps.
   h. On the Members tab, verify that only months are in the list and **Dec-04** is the last value.

Figure 9–2 Selecting Time Dimension Members


New Forecast Step

General | Advanced Settings | Status

Choose: Time — From: 'Calendar Year' hierarchy

Available: Members | Conditions

Hierarchy
- Children of Jan-98
- Year

Selected: Steps | Members

1. Start with Month

Edit Step

Returns dimension members that have a family relationship with another dimension member.

Action: Remove

Meeting Condition

Relation: Descendants

of

Member: 2005

Options

☐ Include 2005

Help | OK | Cancel

Help | Create | Cancel

5. Set the status of the Customer dimension:
   a. On the Steps tab, click **All Levels** and choose **Total Customer**.
   b. On the Members tab, verify that **All Customers** is the only value.
6. Set the status of the Product dimension:
   a. On the Steps tab, remove the initial selection.
   b. On the Conditions tab, expand the Hierarchy folder. Add the **Children of Total Product** condition to the Selected Steps.
   c. On the Members tab, verify that **Hardware** and **Software/Other** are the only values.
7. Set the status of the Channel dimension:
   a. On the Steps tab, click **All Levels** and choose **Total Channel**.
   b. On the Members tab, verify that **All Channels** is the only value.
8. Click **Create** to save the forecast step.
9. Click **Apply** to save the Calculation Plan.

Generating and Validating the Forecast

You do not need to generate the forecast now. You can wait until you have created all the steps of the Calculation Plan. However, this step-by-step approach simplifies troubleshooting.

To generate the forecast:

1. Expand the Calculation Plans folder. Right-click **SALES_PLAN** and choose **Execute Calculation Plan SALES_PLAN**.

   The Maintenance Wizard opens, and **SALES_PLAN** is a selected target object.

2. Click **Finish**.

   The build log is displayed when the Calculation Plan is done executing.

To view the forecast results, take these steps:

1. Fully expand the **UNITS_CUBE** folder, right-click the **SALES_FORECAST** measure, and choose **View Data SALES_FORECAST**.

   The Measure Data Viewer opens. No data is displayed, because the base levels for Product, Customer, and Channel are `NA`.

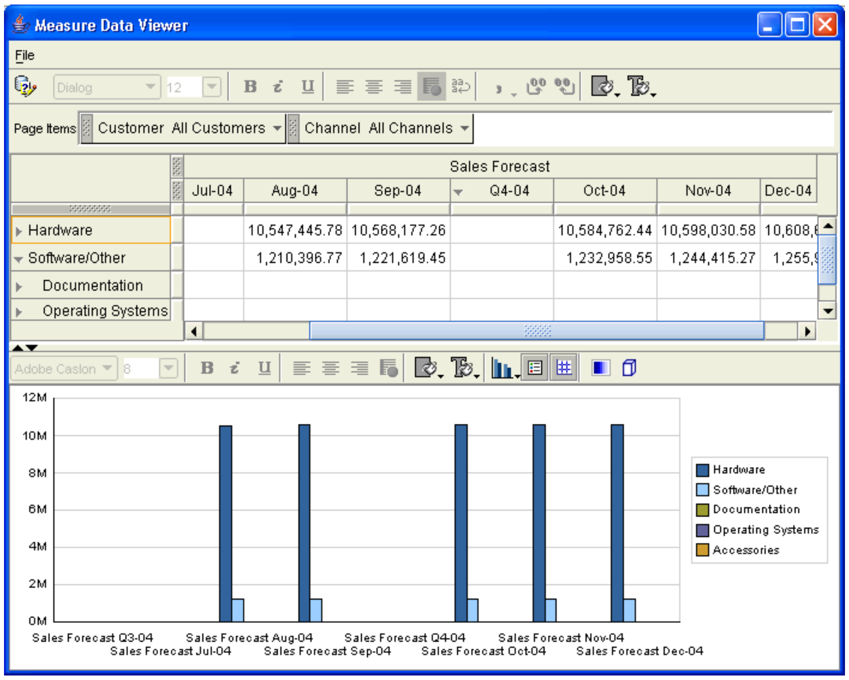
2. From the File menu, choose **Query Builder**.

   The Query Builder opens.

3. On the Layout tab, switch Product and Customer. Click **Help** for instructions.
4. On the Dimensions tab, set the status of Time:
   a. On the Steps tab, remove the initial selection.
   b. On the Conditions tab, expand the Hierarchy folder.
   c. Change **Children of 1998** to **Children of Q3-04, Q4-04**, and add this condition to the Selected Steps.
   d. On the Members tab, verify that only months are in the list from **Jul-04** to **Dec-04**.
5. Set the status of Product:
   a. On the Steps tab, remove the initial selection.
   b. On the Available Members tab, expand **Total Product**.
   c. Select **Hardware** and **Software/Other**, and add them to the Selected list.
6. Click **OK** to close the Query Builder.

Figure 9–3 shows the results of the forecast, which are displayed in the Measure Viewer. Notice the empty cells in the crosstab that need to be populated by allocation.

Figure 9–3 Forecast Data Displayed in the Measure Viewer



Measure Data Viewer

Page Items: Customer All Customers | Channel All Channels

	Sales Forecast						
	Jul-04	Aug-04	Sep-04	Q4-04	Oct-04	Nov-04	Dec-04
Hardware		10,547,445.78	10,568,177.26		10,584,762.44	10,598,030.58	10,608,
Software/Other		1,210,396.77	1,221,619.45		1,232,958.55	1,244,415.27	1,255,
Documentation							
Operating Systems							

Creating an Allocation Basis Measure

This example uses the Proportional method to distribute the values based on the sales performance for the previous year. Before creating the allocation step, you must create a calculated measure for sales values for the prior year to use as the basis measure.

To create the basis measure if you did not already create `SALES_PY` in Chapter 8:

1. Expand the `UNITS_CUBE` folder, right-click **Calculated Measures**, and choose **Create Calculated Measure**.

   The Calculation Wizard opens.

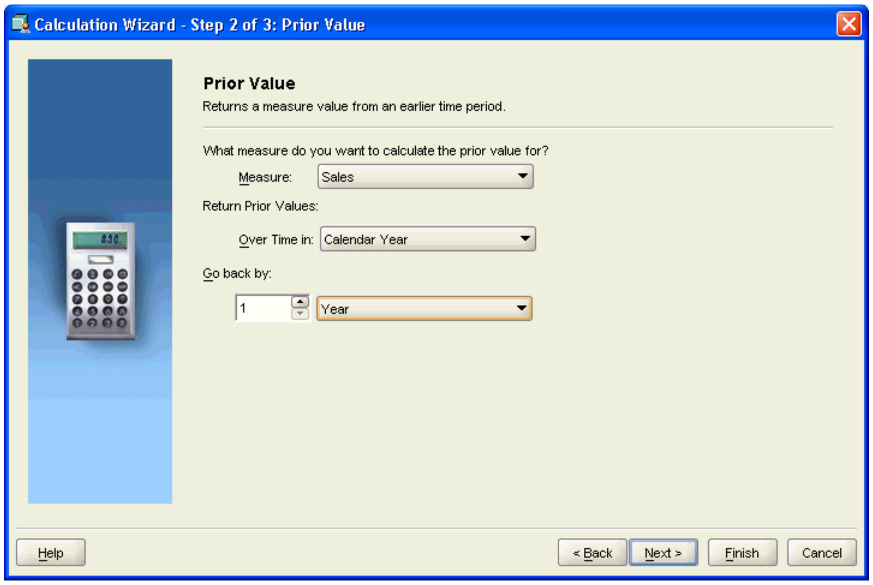
2. Complete the Name and Type page with these values:

   - Name: `sales_py`
   - Calculation Type: **Prior Value** (under Prior/Future Comparison)

3. Complete the Prior Value page with these values, as shown in Figure 9–4.

   - Measure: **Sales**
   - Over Time in: **Calendar Year**
   - Go back by: **1 Year**

Figure 9–4 Creating a Calculated Measure


Calculation Wizard - Step 2 of 3: Prior Value

Prior Value
Returns a measure value from an earlier time period.

What measure do you want to calculate the prior value for?
Measure: Sales
Return Prior Values:
Over Time in: Calendar Year
Go back by:
1 Year

Creating the Allocate Sales Forecast Step

The forecast created the data only for a single level of each dimension. Only Time is populated at the base level. The data must be allocated to the base levels of the other dimensions before it can be aggregated.

To create an allocation step:

1. On the General page of Sales Plan, click **New Step**, then select **New Allocation Step**.

   The Create Allocation Step dialog box opens.

2. Complete the General page with these values:

   - Name: `allocate_sales_forecast_step`
   - Cube: `UNITS_CUBE`
   - Source Measure: `SALES_FORECAST`
   - Target Measure: `SALES_FORECAST`
   - Basis Measure: `SALES_PY`

3. On the Rules page, select **None** for the Time operator. For the other dimensions, select **Proportional**.

4. On the Status page, set the dimension status using conditions:

   - Time: Start with **Month**

     On the Members tab, verify that only months are listed.

   - Customer: Start with **All Levels**

     On the Members tab, verify that all members are listed.

   - Product: Start with **Descendants of Total Product**

     On the Members tab, verify that all members except `Total Products` are listed.

- Channel: Start with **All Levels**

  On the Members tab, verify that all members are listed.

5. Click **Create** to save the allocation step.
6. Click **Apply** to save the Calculation Plan.

Generating and Validating the Allocation

Rerun the Calculation Plan, as described in "Generating and Validating the Forecast" on page 9-17. Both the forecast step and the allocation step will be executed.

To view the allocation results, use the Measure Viewer to see the data in the Sales Forecast measure. The allocation populated the base levels of Product, Customer, and Time.

Creating the Sales Forecast Aggregation Step

As the previous display of the allocated measure shows, the aggregated data is available without an aggregation step. All aggregated values are generated on the fly. The aggregation step simply generates and stores some of the aggregate values to improve runtime performance.

To create a new aggregation step:

1. On the General page of Sales Plan, click **New Step**, then select **New Aggregation Step**.

   The Create Aggregation Step dialog box opens.

2. Complete the General page with these values:
   - Name: `aggregate_sales_forecast_step`
   - Cube: `UNITS_CUBE`
   - Selected Measures: `SALES_FORECAST`
3. On the Summarize To page, select the levels the same as for `UNITS_CUBE`, that is, select all levels for Time, Product, and Channel, and skip levels for Customer. Refer to "Case Study: Aggregating a Moderately Sparse or Dense Cube" on page 7-13.
4. On the Status page, keep the default status of `All Levels` for all dimensions. Keep the default settings for Rules and Cache also.
5. Click **Create** to save the aggregation step.
6. Click **Apply** to save the Calculation Plan.

Generating the Aggregation

Rerun the Calculation Plan, as described in "Generating and Validating the Forecast" on page 9-17. The forecast step, the allocation step, and the aggregation step will be executed. You can view the data in the Measure Viewer.

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.

calculated measure

A measure that is calculated at run-time. The result set includes a value for each dimension member currently in status. For example, a calculated measure might calculate the difference in costs from the prior period by using the OLAP DML `LAGDIF` function on the `COSTS` measure. Another calculated measure might calculate profits by subtracting the `COSTS` measure from the `SALES` measure.

See also **dimension member**, **OLAP DML**, **measure**, **status**.

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. Standard form is required for an analytic workspace to be accessible to OLAP tools, however, it is not a prerequisite for multidimensional analysis.

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**.

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.

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.

fact

See **measure**. See also **additive**.

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.

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 calculated measures. Base measures, such as Volume Sales and Dollar Sales, are stored. calculated 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.

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 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, calculated 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**.

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