

Oracle® Database
2 Day + Data Warehousing Guide
11g Release 1 (11.1)
B28314-01

July 2007

Oracle Database 2 Day + Data Warehousing Guide, 11g Release 1 (11.1)

B28314-01

Copyright © 2007, Oracle. All rights reserved.

Primary Author: Michelle Bird, Paul Lane

Contributing Author: Mark van de Wiel

Contributor: Tammy Bednar, Hermann Baer, Kristy Browder Edwards, John Haydu, Sushil Kumar, George Lumpkin, Mughees Minhas, Mark Townsend, Murali Thiyagarajan

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

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

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

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

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

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

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

Contents

Preface	ix
Audience	ix
Documentation Accessibility	ix
Related Documents	x
Conventions	x
Part I Building Your Data Warehouse	
1 Introduction to Data Warehousing	
About This Guide	1-1
Before Using This Guide	1-1
What This Guide Is Not	1-1
What is a Data Warehouse?	1-2
The Key Characteristics of a Data Warehouse	1-2
Common Oracle Data Warehousing Tasks	1-2
Tasks Illustrated in this Guide	1-3
Tools for Administering the Data Warehouse	1-4
2 Setting Up Your Data Warehouse System	
General Steps for Setting up a Data Warehouse System	2-1
Preparing the Environment	2-1
Balanced Hardware Configuration	2-2
How Many CPUs and What Clock Speed Do I Need?	2-2
How Much Memory Do I Need?	2-2
How Many Disks Do I Need?	2-3
How Do I Determine Sufficient I/O Bandwidth?	2-3
Verifying the Data Warehouse Hardware Configuration	2-4
About the dd Utility	2-4
Example: Using the dd Utility	2-4
About the Orion Utility	2-5
Example: Using the Orion Utility	2-6
Setting Up a Database for a Data Warehouse	2-6
How Should I Set the Memory Management Parameters?	2-6
Example: Setting an Initialization Parameter	2-7
What Other Initialization Parameter Settings Are Important?	2-7

Accessing Oracle Warehouse Builder.....	2-9
Installing the Oracle Warehouse Builder Demonstration.....	2-10

3 Identifying Data Sources and Importing Metadata

General Steps for Importing Metadata from Sources.....	3-1
About Workspaces, Projects, and other Devices in Warehouse Builder.....	3-2
Example: Importing Metadata from Flat Files.....	3-2
Specifying Locations for the Flat Files.....	3-3
Creating Modules in the Project.....	3-3
Launching the Import Metadata Wizard.....	3-4
Using the Flat File Sample Wizard.....	3-4
Importing the Flat File Data.....	3-5

4 Defining Warehouses in Oracle Warehouse Builder

General Steps for Defining a Relational Target Warehouse.....	4-1
Identifying the Warehouse Target Schema.....	4-2
About Flat File Sources in Warehouse Builder.....	4-3
Exercise: Adding External Tables to the Target Module.....	4-3
About Dimensions.....	4-3
Exercise: Understanding Dimensions.....	4-4
About Levels.....	4-5
Defining Level Attributes.....	4-6
Defining Hierarchies.....	4-6
Dimension Roles.....	4-6
Level Relationships.....	4-7
Dimension Example.....	4-7
Control Rows.....	4-7
Implementing a Dimension.....	4-8
Star Schema.....	4-8
Binding.....	4-9
About Cubes.....	4-11
Defining a Cube.....	4-11
Cube Measures.....	4-11
Cube Dimensionality.....	4-11
Cube Example.....	4-12
Implementing a Cube.....	4-12
Relational Implementation of a Cube.....	4-12
Binding.....	4-13

Part II Loading Data into Your Data Warehouse

5 Defining ETL Logic

About Mappings and Operators.....	5-1
Instructions for Defining Mappings.....	5-2
Creating a Mapping.....	5-2
Types of Operators.....	5-3

Adding Operators	5-4
Adding Operators that Bind to Workspace Objects.....	5-5
Create Unbound Operator with No Attributes	5-5
Select from Existing Workspace Object and Bind	5-6
Editing Operators	5-6
Connecting Operators	5-7
Connecting Attributes	5-7
Connecting Groups	5-8
Example: Using the Mapping Editor to Create Staging Area Tables	5-8
Setting Mapping Properties	5-10
Target Load Order	5-10
Reset to Default	5-11
Setting Operator, Group, and Attribute Properties	5-11
Synchronizing Operators and Workspace Objects	5-12
Synchronizing An Operator	5-12
Synchronizing From a Workspace Object to an Operator	5-13
Synchronizing Operators based on Workspace Objects	5-14
Synchronizing from an Operator to a Workspace Object	5-15

6 Improving Loading Performance

Improved Performance via Partition Exchange Loading	6-1
About Partition Exchange Loading	6-1
Configuring a Mapping for Partition Exchange Loading	6-2
Direct and Indirect Partition Exchange Loading	6-3
Using Indirect Partition Exchange Loading.....	6-3
Example: Using Direct Partition Exchange Loading to Publish Fact Tables.....	6-4
Using Partition Exchange Loading Effectively	6-4
Configuring Targets in a Mapping.....	6-5
Step 1: Create All Partitions.....	6-5
Step 2: Create All Indexes Using the LOCAL Option	6-7
Step 3: Primary/Unique Keys Use "USING INDEX" Option.....	6-7
Restrictions for Using Partition Exchange Loading in Warehouse Builder	6-8

7 Deploying to Target Schemas and Executing ETL Logic

About Deployment	7-1
What is a Control Center?.....	7-2
Configuring the Physical Details of Deployment.....	7-2
Deployment Actions	7-3
The Deployment Process.....	7-3
About Schedules	7-3
Process for Defining and Using Schedules	7-4
Deploying Objects	7-5
Starting ETL Jobs	7-6
Viewing the Data.....	7-6

Part III Analyzing and Reporting on a Data Warehouse

8 Analyzing Metadata Dependencies

Introduction to the Metadata Dependency Manager	8-1
Usage Scenario	8-2
What are Lineage and Impact Analysis Diagrams?	8-3
Generating an LIA Diagram	8-4
Modifying the Display of an LIA Diagram	8-5
Using Groups in an LIA Diagram	8-5
Displaying an Object's Attributes	8-5
Modifying Objects in the Dependency Manager	8-6

9 SQL for Reporting and Analysis

Use of SQL Analytic Capabilities to Answer Business Queries	9-1
How to Add Totals to Reports Using the ROLLUP Function	9-2
When to Use the ROLLUP Function	9-2
Example: Using the ROLLUP Function	9-2
How to Separate Totals at Different Levels Using the CUBE Function	9-4
When to Use the CUBE Function	9-4
Example: Using the CUBE Function	9-4
How to Add Subtotals Using the GROUPING Function	9-5
When to Use the GROUPING Function	9-5
Example: Using the GROUPING Function	9-5
How to Combine Aggregates Using the GROUPING SETS Function	9-6
When to Use the GROUPING SETS Function	9-6
Example: Using the GROUPING SETS Function	9-7
How to Calculate Rankings Using the RANK Function	9-8
When to Use the RANK Function	9-8
Example: Using the RANK Function	9-8
How to Calculate Relative Contributions to a Total	9-9
Example: Calculating Relative Contributions to a Total	9-9
How to Perform Interrow Calculations with Window Functions	9-11
Example: Performing Interrow Calculations	9-11
How to Calculate a Moving Average Using a Window Function	9-12
Example: Calculating a Moving Average	9-13
Use of Partition Outer Join to Handle Sparse Data	9-13
When to Use Partition Outer Join	9-14
Example: Using Partition Outer Join	9-14
Use of the WITH Clause to Simplify Business Queries	9-14
When to Use the WITH Clause	9-15
Example: Using the WITH Clause	9-15

Part IV Managing a Data Warehouse

10 Refreshing a Data Warehouse

About Refreshing Your Data Warehouse	10-1
Example: Refreshing Your Data Warehouse	10-1
Using Rolling Windows to Offload Data	10-5

Example: Using a Rolling Window	10-5
11 Optimizing Data Warehouse Operations	
Avoiding System Overload	11-1
Monitoring System Performance	11-1
Monitoring Parallel Execution Performance.....	11-1
Monitoring I/O	11-2
Using Database Resource Manager	11-3
Optimizing the Use of Indexes and Materialized Views	11-4
Example: Optimizing Indexes and Materialized Views Using the SQL Access Advisor.....	11-4
Optimizing Storage Requirements	11-5
Using Data Compression to Improve Storage	11-5
12 Eliminating Performance Bottlenecks	
Verifying That SQL Runs Efficiently	12-1
Analyzing Optimizer Statistics	12-1
Analyzing an Execution Plan	12-2
Example: Analyzing Explain Plan Output	12-2
Using Hints to Improve Data Warehouse Performance.....	12-3
Example: Using Hints to Improve Data Warehouse Performance	12-4
Using Advisors to Verify SQL Performance	12-4
Improving Performance by Minimizing Resource Consumption	12-5
Improving Performance: Partitioning.....	12-5
Improving Performance: Partition Pruning.....	12-5
Improving Performance: Partitionwise Joins.....	12-6
Example: Evaluating Partitioning with the SQL Access Advisor.....	12-6
Improving Performance: Query Rewrite and Materialized Views.....	12-7
Improving Performance: Indexes	12-8
Using Resources Optimally	12-8
Optimizing Performance with Parallel Execution.....	12-8
How Parallel Execution Works.....	12-9
Setting the Degree of Parallelism.....	12-9
Example: Setting the Degree of Parallelism	12-9
About Wait Events	12-10
13 Backing up and Recovering a Data Warehouse	
How Should I Handle Backup and Recovery for a Data Warehouse?	13-1
Strategies and Best Practices for Backup and Recovery	13-2
Best Practice A: Use ARCHIVELOG Mode	13-2
Is Downtime Acceptable?	13-3
Best Practice B: Use RMAN	13-3
Best Practice C: Use Read-Only Tablespaces	13-4
Best Practice D: Plan for NOLOGGING Operations.....	13-4
Extraction, Transformation, and Loading	13-5
The ETL Strategy and NOLOGGING Operations	13-5
Sizing the Block Change Tracking File	13-6

Incremental Backup	13-7
The Incremental Approach.....	13-7
Best Practice E: Not All Tablespaces are Equally Important	13-7

14 Securing a Data Warehouse

Overview of Data Warehouse Security	14-1
Why is Security Necessary for a Data Warehouse?	14-1
Using Roles and Privileges for Data Warehouse Security	14-2
Using Virtual Private Database in Data Warehouses	14-2
How Virtual Private Database Works.....	14-3
Overview of Oracle Label Security	14-3
How Oracle Label Security Works	14-3
How Data Warehouses Benefit from Labels	14-4
Overview of Fine-Grained Auditing in Data Warehouses	14-4
Overview of Transparent Data Encryption in Data Warehouses	14-4

Index

Preface

This preface contains these topics:

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

Audience

Oracle Database 2 Day + Data Warehousing Guide is for anyone who wants to perform common day-to-day warehousing tasks with the Oracle Database. The main prerequisite is to have gone through *Oracle Database 2 Day DBA* and have a basic knowledge of computers.

In particular, this guide is targeted toward the following groups of users:

- Oracle DBAs wanting to acquire data warehouse administrative skills
- DBAs who have some data warehouse background but are new to Oracle

Documentation Accessibility

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

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

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

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

TTY Access to Oracle Support Services

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

Related Documents

For more information, see these Oracle resources:

- *Oracle Database 2 Day DBA*
- *Oracle Database Data Warehousing Guide*
- *Oracle Database Administrator's Guide*

Conventions

The following text conventions are used in this document:

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

Part I

Building Your Data Warehouse

This section discusses building a data warehouse, and includes:

- [Chapter 1, "Introduction to Data Warehousing"](#)
- [Chapter 2, "Setting Up Your Data Warehouse System"](#)
- [Chapter 3, "Identifying Data Sources and Importing Metadata"](#)
- [Chapter 4, "Defining Warehouses in Oracle Warehouse Builder"](#)

Introduction to Data Warehousing

As someone responsible for administering, designing, and implementing a data warehouse, you are responsible for the overall operation of the Oracle data warehouse and maintaining its efficient performance.

This section contains the following topics:

- [About This Guide](#)
- [What is a Data Warehouse?](#)
- [Common Oracle Data Warehousing Tasks](#)
- [Tools for Administering the Data Warehouse](#)

About This Guide

Oracle Database 2 Day + Data Warehousing Guide teaches you how to perform common day-to-day tasks necessary to implement and administer a data warehouse. The goal of this guide is to introduce you to the data warehousing solutions available in the Oracle Database. This guide teaches you how to perform common administration and design tasks needed to keep the data warehouse operational, including how to perform basic performance monitoring tasks.

The primary interfaces used in this guide are Oracle Enterprise Manager, Oracle Warehouse Builder, and SQL*Plus.

Before Using This Guide

Before using this guide, you must perform the following:

- Become familiar with the *Oracle Database 2 Day DBA Guide*.
- Obtain the necessary tools described in "[Tools for Administering the Data Warehouse](#)".

What This Guide Is Not

Oracle Database 2 Day + Data Warehousing Guide is task-oriented. The objective is to describe why and when tasks need to be performed. Where appropriate, it describes the concepts necessary for understanding and completing the current task. This guide is not an exhaustive discussion of implementing a data warehouse on Oracle. For complete conceptual information about these features and detailed instructions for using them, see the appropriate Oracle documentation:

- *Oracle Database Data Warehousing Guide*

- *Oracle Warehouse Builder User's Guide*
- *Oracle Database Administrator's Guide* for a discussion of administrative tasks
- *Oracle OLAP Reference* for a discussion of OLAP
- *Oracle Data Mining Concepts* for a discussion of data mining

Also, this guide discusses using some of the features available in Oracle Enterprise Manager, but this guide does not provide exhaustive information about Enterprise Manager. For information about Enterprise Manager, see:

- The Enterprise Manager online help
- *Oracle Enterprise Manager Concepts*

What is a Data Warehouse?

A **data warehouse** is a relational or multidimensional database that is designed for query and analysis. They are not optimized for transaction processing, which is the domain of OLTP systems. Data warehouses usually consolidate historical and transactional data derived from multiple sources. Data warehouses separate analysis workload from transaction workload and enable an organization to consolidate data from several sources.

Data in a data warehouse usually stores many months or years of data to support historical analysis. The data in a data warehouse is typically loaded through an extraction, transformation, and loading (ETL) process from one or more data sources such as OLTP applications, mainframe applications, or external data providers.

End users of the data warehouse perform data analyses that are often time-related. Examples include consolidation of last year's sales figures, inventory analysis, and profit by product and by customer. More sophisticated analyses include trend analyses and data mining, which use existing data to forecast trends or predict futures. The data warehouse typically provides the foundation for a business intelligence environment.

This guide covers relational implementations, including star schemas. See *Oracle Database Data Warehousing Guide* for more details regarding multidimensional data warehouses.

The Key Characteristics of a Data Warehouse

The key characteristics of a data warehouse are as follows:

- Some data is denormalized for simplification and to improve performance.
- Large amounts of historical data are used.
- Queries often retrieve large amounts of data.
- Both planned and ad hoc queries are common.
- The data load is controlled.

In general, high data throughput is the key to a successful data warehouse.

Common Oracle Data Warehousing Tasks

As an Oracle data warehousing administrator or designer, you can expect to be involved in the following tasks:

- Configuring an Oracle database for use as a data warehouse

- Designing data warehouses
- Performing upgrades of the database and software to new release levels
- Managing schema objects, such as tables, indexes, and materialized views
- Managing users and security
- Developing routines used for the Extraction, Transformation, and Loading (ETL) process
- Creating reports based on the data in the data warehouse
- Backing up the data warehouse and performing recovery when necessary
- Monitoring the data warehouse's performance and taking preventive or corrective action as required

In a small-to-midsized data warehouse environment, you might be the sole person performing these tasks. In large, enterprise environments, the job is often divided among several DBAs and designers, each with their own specialty, such as database security or database tuning.

Tasks Illustrated in this Guide

This guide, the *Oracle Database 2 Day + Data Warehousing Guide*, illustrates the following tasks:

1. Configure an Oracle database for use as a data warehouse.
Refer to [Chapter 2, "Setting Up Your Data Warehouse System"](#). This section also includes instructions on how to access a demonstration that is referenced in exercises throughout this guide.
2. Take the initial steps in consolidating data.
Follow the instructions in [Chapter 3, "Identifying Data Sources and Importing Metadata"](#).
3. Begin to define the target objects in the warehouse.
[Chapter 4, "Defining Warehouses in Oracle Warehouse Builder"](#) describes how to define external tables, dimensions, and cubes for the target warehouse.
4. Define strategies for extracting, transforming, and loading data into the target.
[Chapter 5, "Defining ETL Logic"](#) describes how to define ETL logic to extract data from the source you identified in step 2, transform the data, and then load it into the target you designed in step 3.
5. Consider performance issues and advanced ETL concepts.
[Chapter 6, "Improving Loading Performance"](#) provides conceptual information for designing complex mappings and improving ETL performance.
6. Deploy to target schemas and execute ETL logic.
[Chapter 7, "Deploying to Target Schemas and Executing ETL Logic"](#) describes how to prepare a target schema with code from mappings and also describes how to subsequently execute that code.
7. Analyze metadata dependencies.
Read and complete the tasks in [Chapter 8, "Analyzing Metadata Dependencies"](#).
8. Write efficient SQL.

Read and complete the tasks in [Chapter 9, "SQL for Reporting and Analysis"](#). This section describes how to write efficient SQL.

9. Refresh the data warehouse.

Read and complete the tasks in [Chapter 10, "Refreshing a Data Warehouse"](#).

10. Optimize operations.

Read and complete the tasks in [Chapter 11, "Optimizing Data Warehouse Operations"](#).

11. Eliminate performance bottlenecks.

Read and complete the tasks in [Chapter 12, "Eliminating Performance Bottlenecks"](#).

12. Review some basics of data warehouse backup and recovery.

[Chapter 13, "Backing up and Recovering a Data Warehouse"](#) describes some considerations for how to back up and recover a data warehouse.

13. Review some basics of data warehouse security.

[Chapter 14, "Securing a Data Warehouse"](#) describes some considerations for how to create a secure data warehouse.

Tools for Administering the Data Warehouse

The intent of this guide is to enable you to quickly and efficiently create and administer an Oracle data warehouse. The following are some of the products, tools, and utilities you can use to achieve your goals with your data warehouse:

- **Oracle Universal Installer**

Oracle Universal Installer installs your Oracle software and options. It can automatically start the Database Configuration Assistant (DBCA) to install a database.

- **Oracle Enterprise Manager**

The primary tool for managing your database is Oracle Enterprise Manager, a Web-based interface. After you have installed the Oracle software, created or upgraded a database, and configured the network, you can use Oracle Enterprise Manager for managing your database. In addition, Oracle Enterprise Manager also provides an interface for performance advisors and for Oracle utilities such as SQL*Loader and Recovery Manager.

- **Oracle Warehouse Builder**

The primary product for populating and maintaining a data warehouse, Oracle Warehouse Builder provides ETL, data quality management, and metadata management functionality in a single product.

Warehouse Builder includes a unified repository hosted on an Oracle Database. Warehouse Builder leverages Oracle Database functionality to generate code optimized for loading into and maintaining Oracle Database targets.

- **Database Tuning Pack**

Oracle Database Tuning Pack offers a set of new technologies that automate the entire database tuning process, which significantly lowers database management costs and enhances performance and reliability. The key features of Oracle

Database Tuning Pack that will be used in this guide are the SQL Access and SQL Tuning Advisors.

Setting Up Your Data Warehouse System

An Oracle data warehouse is an Oracle Database specifically configured and optimized to handle the size of data and types of queries that are intrinsic to data warehousing. This section discusses how to initially configure your data warehouse environment. It includes the following topics:

- [General Steps for Setting up a Data Warehouse System](#)
- [Preparing the Environment](#)
- [Setting Up a Database for a Data Warehouse](#)
- [Accessing Oracle Warehouse Builder](#)

General Steps for Setting up a Data Warehouse System

In these instructions, you configure an Oracle Database for use as a data warehouse. Subsequently, you install Oracle Warehouse Builder which leverages the Oracle Database and provides graphical user interfaces for designing data management strategies.

To set up a data warehouse system, complete the following steps:

1. Size and configure your hardware as described in "[Preparing the Environment](#)" on page 2-1.
2. Install the Oracle Database software.
3. Optimize the Database for use as a data warehouse as described in "[Setting Up a Database for a Data Warehouse](#)" on page 2-6.
4. Access the Oracle Warehouse Builder software.

Oracle Warehouse Builder is the data integration product that is packaged with the Standard and Enterprise editions of the Oracle Database.

Follow the instructions in "[Accessing Oracle Warehouse Builder](#)" on page 2-9. Subsequently, you can install a demonstration to assist you in learning how to complete common data warehousing tasks using Warehouse Builder.

Subsequent Steps

[Chapter 3, "Identifying Data Sources and Importing Metadata"](#)

Preparing the Environment

The basic components for a data warehouse architecture are the same as for an online transaction processing (OLTP) system. However, due to the sheer size of the data, you

have to choose different quantities to balance the individual building blocks differently. The starting point for sizing a data warehouse is the throughput that you require from the system. This can be one or both of the following:

- The amount of data that is being accessed by queries hitting the system at peak time, in conjunction with the acceptable response time. You may be able to use throughput numbers and experience from an existing application to estimate the required throughput.
- The amount of data that is loaded within a window of time.

In general, you need to estimate the highest throughput you need at any given point.

Hardware vendors can recommend balanced configurations for a data warehousing application and can help you with the sizing. Contact your preferred hardware vendor for more details.

Balanced Hardware Configuration

A properly sized and balanced hardware configuration is required to maximize data warehouse performance. The following sections discuss some important considerations in achieving this balance:

- [How Many CPUs and What Clock Speed Do I Need?](#)
- [How Much Memory Do I Need?](#)
- [How Many Disks Do I Need?](#)
- [How Do I Determine Sufficient I/O Bandwidth?](#)

How Many CPUs and What Clock Speed Do I Need?

Central processing units (CPUs) provide the calculation capabilities in a data warehouse. You must have sufficient CPU power to perform the data warehouse operations. Parallel operations are more CPU-intensive than the equivalent serial operation would be.

Use the estimated highest throughput as a guideline for the number of CPUs you need. As a rough estimate, use the following formula:

- $\text{<number of CPUs>} = \text{<maximum throughput in MB/s>} / 200$

In other words, a CPU can sustain up to about 200 MB per second. For example, if a system requires a maximum throughput of 1200 MB per second, then the system needs $\text{<number of CPUs>} = 1200 / 200 = 6$ CPUs. A configuration with 1 server with 6 CPUs can service this system. A 2-node clustered system could be configured with 3 CPUs in both nodes.

How Much Memory Do I Need?

Memory in a data warehouse is particularly important for processing memory-intensive operations such as large sorts. Access to the data cache is less important in a data warehouse because most of the queries access vast amounts of data. Data warehouses do not have memory requirements as critical as OLTP applications.

The number of CPUs provides you a good guideline for the amount of memory you need. Use the following simplified formula to derive the amount of memory you need from the CPUs you selected:

- $\text{<amount of memory in GB>} = 2 * \text{<number of CPUs>}$

For example, a system with 6 CPUs needs $2 * 6 = 12$ GB of memory. Most standard servers fulfill this requirement.

How Many Disks Do I Need?

A common mistake in data warehouse environments is to size the storage based on the maximum capacity needed. Sizing based exclusively on storage requirements will likely create a throughput bottleneck.

Use the throughput you require to find out how many disk arrays you need. Use the storage provider's specifications to find out how much throughput a disk array can sustain. Note that storage providers measure in Gb per second, and your initial throughput estimate is based on MB second. An average disk controller has a maximum throughput of 2 Gb second, which translates to a sustainable throughput of about $(70\% * 2 \text{ Gbit/s}) / 8 = 180 \text{ MB/s}$.

Use the following formula to determine the number of disk arrays you need:

- $\text{<number of disk controllers>} = \text{<throughput in MB/s>} / \text{<individual controller throughput in MB/s>}$

For example, our system with 1200 MB per second throughput requires at least $1200 / 180 = 7$ disk arrays.

Make sure you have enough physical disks to sustain the throughput you require. Ask your disk vendor for the throughput numbers of the disks.

How Do I Determine Sufficient I/O Bandwidth?

The end-to-end I/O system consists of more components than just the CPUs and disks. A well-balanced I/O system has to provide approximately the same bandwidth across all components in the I/O system. These components include the following:

- Host Bus Adapters (HBAs), the connectors between the server and the storage.
- Switches, in between the servers and a Storage Area Network (SAN) or Network Attached Storage (NAS).
- Ethernet adapters for network connectivity (GigE NIC or Infiniband). In a clustered environment, you need an additional private port for the interconnect between the nodes that you should not include when sizing the system for I/O throughput. The interconnect must be sized separately, taking into account factors such as internode parallel execution.
- Wires that connect the individual components.

Each of the components has to be able to provide sufficient I/O bandwidth to ensure a well-balanced I/O system. The initial throughput you estimated and the hardware specifications from the vendors are the basis to determine the quantities of the individual components you need. Use the conversion in the following table to translate the vendors' maximum throughput numbers in bits into sustainable throughput numbers in bytes.

Table 2-1 Throughput Performance Conversion

Component	Bits	Bytes Per Second
HBA	2 Gbit	200 MB
16 Port Switch	8 * 2 Gbit	1200 MB
Fibre Channel	2 Gbit	200 MB
GigE NIC	1 Gbit	80 MB

Table 2–1 (Cont.) Throughput Performance Conversion

Component	Bits	Bytes Per Second
Inf-2 Gbit	2 Gbit	160 MB

In addition to having sufficient components to ensure sufficient I/O bandwidth, the layout of data on the disk is also a key to success or failure. If you configured the system for sufficient throughput across all disk arrays, and all data a query is going to retrieve still resides on one disk, then you still will not be able to get the required throughput, because your one disk will be the bottleneck. In order to avoid such a situation, stripe data across as many disks as possible, ideally all disks. A stripe size of 256 KB to 1 MB provides a good balance between multiblock read operations and data spread across multiple disks.

About Automatic Storage Management (ASM)

ASM is a component of Oracle Database that you can use to stripe data across disks in a disk group. ASM ensures the data is balanced across all disks. Disks can be added or removed while ASM is operational, and ASM will automatically rebalance the storage across all available disks. ASM can also be used to mirror data on the file system, to avoid loss of data in case of disk failure. The default stripe size for ASM is 1 MB. You can lower the stripe size to 128 KB.

You can perform storage operations without ASM, but this increases the chances of making a mistake. Thus, Oracle recommends you use ASM whenever possible.

Verifying the Data Warehouse Hardware Configuration

Before you install Oracle Database, you should verify your setup on the hardware and operating-system level. The key point to understand is that if the operating system cannot deliver the performance and throughput you need, Oracle Database will never be able to perform according to your requirements. Two tools for verifying throughput are the `dd` utility and Orion, an Oracle-supplied tool.

About the `dd` Utility

A very basic way to validate the operating system throughput, on UNIX or Linux systems, is to use the `dd` utility. The `dd` utility is a very basic way to read data blocks directly from disk and, because there is almost no overhead involved, the output from the `dd` utility provides a reliable calibration. Oracle Database will reach a maximum throughput of approximately 90 percent of what the `dd` utility can achieve.

Example: Using the `dd` Utility

To use the `dd` utility:

First, the most important options for using `dd` are the following:

```
bs=BYTES: Read BYTES bytes at a time; use 1 MB
count=BLOCKS: copy only BLOCKS input blocks
if=FILE: read from FILE; set to your device
of=FILE: write to FILE; set to /dev/null to evaluate read performance;
write to disk would erase all existing data!!!
skip=BLOCKS: skip BLOCKS BYTES-sized blocks at start of input
```

To estimate the maximum throughput Oracle Database will be able to achieve, you can mimic a workload of a typical data warehouse application, which consists of large, random sequential disk access.

The following `dd` command performs random sequential disk access across two devices reading a total of 2 GB. The throughput is 2 GB divided by the time it takes to finish the following command:

```
dd bs=1048576 count=200 if=/raw/data_1 of=/dev/null &
dd bs=1048576 count=200 skip=200 if=/raw/data_1 of=/dev/null &
dd bs=1048576 count=200 skip=400 if=/raw/data_1 of=/dev/null &
dd bs=1048576 count=200 skip=600 if=/raw/data_1 of=/dev/null &
dd bs=1048576 count=200 skip=800 if=/raw/data_1 of=/dev/null &
dd bs=1048576 count=200 if=/raw/data_2 of=/dev/null &
dd bs=1048576 count=200 skip=200 if=/raw/data_2 of=/dev/null &
dd bs=1048576 count=200 skip=400 if=/raw/data_2 of=/dev/null &
dd bs=1048576 count=200 skip=600 if=/raw/data_2 of=/dev/null &
dd bs=1048576 count=200 skip=800 if=/raw/data_2 of=/dev/null &
```

In your test, you should include all the storage devices that you plan to include for your database storage. When you configure a clustered environment, you should run `dd` commands from every node.

About the Orion Utility

Orion is a tool that Oracle provides to mimic an Oracle-like workload on a system to calibrate the throughput. Compared to the `dd` utility, Orion provides the following advantages:

- Orion's simulation is closer to the workload the database will produce.
- Orion enables you to perform reliable write and read simulations within one simulation.

Oracle recommends you use Orion to verify the maximum achievable throughput, even if a database has already been installed.

The types of supported I/O workloads are as follows:

- small and random
- large and sequential
- large and random
- mixed workloads

For each type of workload, Orion can run tests at different levels of I/O load to measure performance metrics such as MB per second, I/O per second, and I/O latency. A data warehouse workload is typically characterized by sequential I/O throughput, issued by multiple processes. You can run different I/O simulations depending upon which type of system you plan to build. Examples are the following:

- daily workloads when users or applications query the system
- the data load when users may or may not access the system
- index and materialized view builds
- backup operations

To download Orion software, point your browser to the following:

<http://www.oracle.com/technology/software/tech/orion/index.html>

Note that Orion is Beta software, and unsupported.

Example: Using the Orion Utility

To invoke Orion:

```
$ orion -run simple -testname mytest -num_disks 8
```

Typical output is as follows:

```
Orion VERSION 10.2
```

```
Command line:
```

```
-run advanced -testname orion14 -matrix point -num_large 4 -size_large 1024  
-num_disks 4 -type seq -num_streamIO 8 -simulate raid0 -cache_size 0 -verbose
```

```
This maps to this test:
```

```
Test: orion14  
Small IO size: 8 KB  
Large IO size: 1024 KB  
IO Types: Small Random IOs, Large Sequential Streams  
Number of Concurrent IOs Per Stream: 8  
Force streams to separate disks: No  
Simulated Array Type: RAID 0  
Stripe Depth: 1024 KB  
Write: 0%  
Cache Size: 0 MB  
Duration for each Data Point: 60 seconds  
Small Columns:, 0  
Large Columns:, 4  
Total Data Points: 1
```

```
Name: /dev/vx/rdisk/asm_vol1_1500m   Size: 1572864000  
Name: /dev/vx/rdisk/asm_vol2_1500m   Size: 1573912576  
Name: /dev/vx/rdisk/asm_vol3_1500m   Size: 1573912576  
Name: /dev/vx.rdisk/asm_vol4_1500m   Size: 1573912576  
4 FILES found.
```

```
Maximum Large MBPS=57.30 @ Small=0 and Large=4
```

In this example, the maximum throughput for this particular workload is 57.30 MB per second.

Setting Up a Database for a Data Warehouse

After you set up your environment and install the Oracle Database software, ensure that you have the Database parameters set correctly. Note that there are not many database parameters that have to be set.

As a general guideline, avoid changing a database parameter unless you have good reason to do so. You can use Oracle Enterprise Manager to set up your data warehouse. To view various parameter settings, navigate to the Database page, then click **Server**. Under Database Configuration, click **Memory Parameters** or **All Initialization Parameters**.

How Should I Set the Memory Management Parameters?

On a high level, there are two memory segments:

- **Shared memory:** Also called the system global area (SGA), this is the memory used by the Oracle instance.

- Session-based memory: Also called program global area (PGA), this is the memory that is occupied by sessions in the database. It is used to perform database operations, such as sorts and aggregations.

Oracle Database can automatically tune the distribution of the memory components in two memory areas. As a result, you need to set only the following parameters:

- `SGA_TARGET`

The `SGA_TARGET` parameter is the amount of memory you want to allocate for shared memory. For a data warehouse, the SGA can be relatively small compared to the total memory consumed by the PGA. To get started, assign 25% of the total memory you allow Oracle Database to use to the SGA. The SGA, at a minimum, should be 100 MB.

- `PGA_AGGREGATE_TARGET`

The `PGA_AGGREGATE_TARGET` parameter is the target amount of memory that you want the total PGA across all sessions to consume. As a starting point, you can use the following formula to define the `PGA_AGGREGATE_TARGET` value:

- `PGA_AGGREGATE_TARGET = 3 * SGA_TARGET` If you do not have enough physical memory for the `PGA_AGGREGATE_TARGET` to fit in memory, then reduce `PGA_AGGREGATE_TARGET`.

- `MEMORY_TARGET` and `MEMORY_MAX_TARGET`

The `MEMORY_TARGET` parameter enables you to set a target memory size and the related initialization parameter, `MEMORY_MAX_TARGET`, sets a maximum target memory size. The database then tunes to the target memory size, redistributing memory as needed between the system global area (SGA) and aggregate program global area (PGA). Because the target memory initialization parameter is dynamic, you can change the target memory size at any time without restarting the database. The maximum memory size serves as an upper limit so that you cannot accidentally set the target memory size too high. Because certain SGA components either cannot easily shrink or must remain at a minimum size, the database also prevents you from setting the target memory size too low.

Example: Setting an Initialization Parameter

You can set an initialization parameter by issuing an `ALTER SYSTEM` statement, as illustrated by the following:

```
ALTER SYSTEM SET SGA_TARGET = 1024M;
```

What Other Initialization Parameter Settings Are Important?

A good starting point for a data warehouse is the data warehouse template database that you can select when you run the Database Configuration Assistant (DBCA). However, any database will be acceptable as long as you make sure you take the following initialization parameters into account:

- `COMPATIBLE`

The `COMPATIBLE` parameter identifies the level of compatibility that the database has with earlier releases. To benefit from the latest features, set the `COMPATIBLE` parameter to your database release number.

- `OPTIMIZER_FEATURES_ENABLE`

To benefit from advanced cost-based optimizer features such as query rewrite, make sure this parameter is set to the value of the current database version.

- `DB_BLOCK_SIZE`

The default value of 8 KB is appropriate for most data warehousing needs. If you intend to use table compression, consider a larger block size.

- `DB_FILE_MULTIBLOCK_READ_COUNT`

The `DB_FILE_MULTIBLOCK_READ_COUNT` parameter enables reading several database blocks in a single operating-system read call. Because a typical workload on a data warehouse consists of many sequential I/Os, make sure you can take advantage of fewer large I/Os as opposed to many small I/Os. When setting this parameter, take into account the block size as well as the maximum I/O size of the operating system, and use the following formula:

- $DB_FILE_MULTIBLOCK_READ_COUNT * DB_BLOCK_SIZE = <maximum\ operating\ system\ I/O\ size>$

Maximum operating-system I/O sizes vary between 64 KB and 1 MB.

- `PARALLEL_MAX_SERVERS`

The `PARALLEL_MAX_SERVERS` parameter sets a resource limit on the maximum number of processes available for parallel execution. Parallel operations need at most twice the number of query server processes as the maximum degree of parallelism (DOP) attributed to any table in the operation.

Oracle Database sets the `PARALLEL_MAX_SERVERS` parameter to a default value that is sufficient for most systems. The default value for `PARALLEL_MAX_SERVERS` is as follows:

- $(CPU_COUNT * PARALLEL_THREADS_PER_CPU * (2\ if\ PGA_AGGREGATE_TARGET > 0; otherwise\ 1) * 5)$

This might not be enough for parallel queries on tables with higher DOP attributes. Oracle recommends users who expect to run queries of higher DOP to set `PARALLEL_MAX_SERVERS` as follows:

- $2 * DOP * NUMBER_OF_CONCURRENT_USERS$

For example, setting the `PARALLEL_MAX_SERVERS` parameter to 64 will allow you to run four parallel queries simultaneously, assuming that each query is using two slave sets with a DOP of eight for each set.

If the hardware system is neither CPU-bound nor I/O bound, then you can increase the number of concurrent parallel execution users on the system by adding more query server processes. When the system becomes CPU- or I/O-bound, however, adding more concurrent users becomes detrimental to the overall performance. Careful setting of the `PARALLEL_MAX_SERVERS` parameter is an effective method of restricting the number of concurrent parallel operations.

- `PARALLEL_ADAPTIVE_MULTI_USER`

The `PARALLEL_ADAPTIVE_MULTI_USER` parameter, which can be `TRUE` or `FALSE`, defines whether or not the server will use an algorithm to dynamically determine the degree of parallelism for a particular statement depending on the current workload. To take advantage of this feature, set `PARALLEL_ADAPTIVE_MULTI_USER` to `TRUE`.

- `QUERY_REWRITE_ENABLED`

To take advantage of query rewrite against materialized views, you must set this parameter to `TRUE`. This parameter defaults to `TRUE`.

- `QUERY_REWRITE_INTEGRITY`

The default for the `QUERY_REWRITE_INTEGRITY` parameter is `ENFORCED`. This means that the database will rewrite queries against only fully up-to-date materialized views, if it can base itself on enabled and validated primary, unique, and foreign key constraints.

In `TRUSTED` mode, the optimizer trusts that the data in the materialized views is current and the hierarchical relationships declared in dimensions and `RELY` constraints are correct.

- `STAR_TRANSFORMATION_ENABLED`

To take advantage of highly optimized star transformations, make sure to set this parameter to `TRUE`.

Accessing Oracle Warehouse Builder

Oracle Warehouse Builder is a flexible tool that enables you to design and deploy various types of data management strategies, including traditional data warehouses.

To enable Warehouse Builder, complete the following steps:

1. Ensure that you have access to either a Enterprise or Standard Edition of the Oracle Database 11g.

Oracle Database 11g comes with the Warehouse Builder server components pre-installed. This includes a schema for the Warehouse Builder repository.

2. To utilize the default Warehouse Builder schema installed in Oracle Database 11g, first unlock the schema.

Connect to SQL*Plus as the `SYS` or `SYSDBA` user. Execute the following commands:

```
SQL> ALTER USER OWBSYS ACCOUNT UNLOCK;
```

```
SQL> ALTER USER OWBSYS IDENTIFIED BY owbsys_passwd;
```

3. Launch the Warehouse Builder Design Center.

For Windows, select **Start, Programs, Oracle, Warehouse Builder** and then select **Design Center**.

For UNIX, locate `owb_home/owb/bin/unix` and then execute `owbclient.sh`

4. Define a workspace and assign a user to the workspace.

In the single Warehouse Builder repository, you can define multiple workspaces with each workspace corresponding to a set of users working on related projects. For instance, you could create a workspace for each of the following environments: development, test, and production.

For simplicity, create one workspace `MY_WORKSPACE` and assign a user.

In the Design Center dialog box, click **Show Details** and then **Workspace Management**.

The Repository Assistant displays.

Following the prompts and accept the default settings in the Repository Assistant, you create a workspace and assign a user as the workspace owner.

5. Log into the Design Center with the user name and password you created.

Installing the Oracle Warehouse Builder Demonstration

In subsequent topics, this guide uses exercises to illustrate how to consolidate data from multiple flat file sources, transform the data, and load it into a new relational target. To execute the exercises presented in this guide, download the Warehouse Builder demonstration. To facilitate your learning of the product, the demonstration provides you with flat file data and scripts that create various Warehouse Builder objects.

To perform the Warehouse Builder exercises presented in this guide, complete the following steps:

1. Download the demonstration.

The demonstration is comprised of a set of files in a zip file called `owb_demo.zip`, which is available at the following link:

http://www.oracle.com/technology/obe/admin/owb10gr2_gs.htm

The zip file includes a SQL script, two source files in comma separated values format, and 19 scripts written in Tcl.

2. Edit the script `owbdemoinit.tcl`.

The script `owbdemoinit.tcl` defines and sets variables used by the other tcl scripts. Edit the following variables to match the values in your computer environment:

- `set temp space TEMP`
- `set owbclientpwd workspace_owner`
- `set sysuser sys`
- `set sypwd pwd`
- `set host hostname`
- `set port portnumber`
- `set service servicename`
- `set project owb_project_name`
- `set owbclient workspace_owner`
- `set sourcedir drive:/newowbdemo`
- `set indexspace USERS`
- `set dataspace USERS`
- `set snapspace USERS`
- `set sqlpath drive:/oracle/11.1.0/db_1/BIN`
- `set sid servicename`

3. Execute the Tcl scripts from the Warehouse Builder scripting utility, OMB Plus.

For Windows, select **Start, Programs, Oracle, Warehouse Builder** and then select **OMB Plus**.

For UNIX, locate `owb_home/owb/bin/unix` and then execute `OMBPlus.sh`

At the OMB+> prompt, type the following to change to the directory containing the scripts:

```
cd drive:\\newowbdemo\\
```

Execute all the Tcl scripts in desired sequence by typing the following command:

```
source loadall.tcl
```

4. Launch the Design Center and log into it as the workspace owner, using the credentials you specified in the script `owbdemoinit.tcl`.
5. Verify that you successfully set up the Warehouse Builder client to follow the demonstration.

In the Design Center, expand the Locations node which is on the right side and in the Connection Explorer. Expand **Databases** and then **Oracle**. The Oracle node should include the follow locations:

```
OWB_REPOSITORY
```

```
SALES_WH_LOCATION
```

When you successfully install the Warehouse Builder demonstration, the Design Center displays with an Oracle module named `EXPENSE_WH`.

Identifying Data Sources and Importing Metadata

In general, the source systems for a data warehouse are typically transaction processing applications. A sales analysis data warehouse, for instance, extracts data from an order entry system that records current order activities.

Designing the extraction process can be problematic. If the source system is complex and poorly documented, then determining which data to extract can be difficult. Moreover, it is usually not possible to modify the source system, nor adjust its performance or availability. To address these problems, first import the metadata.

Metadata is the data that describes the contents of a given object in a data set. The metadata for a table, for instance, indicates the data type for each column.

For Oracle Database customers, the recommended tool of choice for importing metadata is Oracle Warehouse Builder. After you import the metadata into Warehouse Builder, you can annotate the metadata and design an extraction strategy independently from the transaction processing application.

This section describes how to use Warehouse Builder to import metadata. It includes the following topics:

- [General Steps for Importing Metadata from Sources](#)
- [About Workspaces, Projects, and other Devices in Warehouse Builder](#)
- [Example: Importing Metadata from Flat Files](#)

General Steps for Importing Metadata from Sources

To import metadata, complete the following steps:

1. Complete the instructions for "[Accessing Oracle Warehouse Builder](#)" on page 2-9.
2. Download and install the Oracle Warehouse Builder demonstration as described in "[Installing the Oracle Warehouse Builder Demonstration](#)" on page 2-10.
3. Identify the Warehouse Builder project.

Refer to "[About Workspaces, Projects, and other Devices in Warehouse Builder](#)" on page 3-2.

4. Follow along with the "[Example: Importing Metadata from Flat Files](#)" on page 3-2.

The example explains how to specify where the source files are located and how to launch the Metadata Import Wizard. The process for importing data objects such as files, tables, and views is the same. Therefore, once you complete this example,

you gain a general understanding of how to import all data objects into Warehouse Builder.

Subsequent Steps

After you successfully define the sources by importing their metadata, next design the target schema as described in [Chapter 4, "Defining Warehouses in Oracle Warehouse Builder"](#).

Notice that, up to this point, you have only imported metadata and not extracted data. You design and implement a data extraction strategy in subsequent sections of this guide.

About Workspaces, Projects, and other Devices in Warehouse Builder

After you install the Warehouse Builder demonstration and launch the Design Center, you log into a workspace. Notice that the user name and workspace name display along the top of the Design Center.

For an illustration of this and subsequent points, see the following link:

http://www.oracle.com/technology/obe/10gr2_owb/owb10gr2_gs/owb/lesson1/start01.htm#s

Recall that a workspace includes a set of users working on related projects. Security is an important consideration for determining how many workspaces to create. A common model is to create separate workspaces for development, testing, and production. Using this model, you can allow users such as your developers access to the development and testing workspaces but restrict them from the production workspace.

You can optionally divide a workspace into projects. In practise, however, workspaces typically contain only one active project. This is because a project is simply a container and not suitable for implementing security or establishing subject-oriented groupings. Recall that security is implemented via workspaces. Establishing subject-oriented groupings can be accomplished via modules, as discussed later.

The purpose of a project is to contain the sets of metadata related to an initiative. For data warehousing, therefore, include all the metadata for sources and targets in the same project. Also include all the functions, procedures, transformations, mappings, and other objects required to implement your initiative. The project contain nodes for each type of object that you can either create or import into Warehouse Builder. Expand the different nodes to gain a general understanding of the types of objects you can create or import.

Notice that in the demonstration, the Project Explorer displays on the left side and includes two projects. `MY_PROJECT` is a default, pre-seeded project. You can use `MY_PROJECT` as your single active project in the workspace. For the purposes of the demonstration, we work in the `OWB_DEMO` project.

Example: Importing Metadata from Flat Files

This example illustrates how to import metadata from flat files. Specifically, our objective is to import the metadata into the `OWB_DEMO` project such that the two files, `export.csv` and `expense_categories.csv`, display in the Project Explorer under the Files node.

To import metadata from flat files:

1. Indicate where the flat files are located as described in "[Specifying Locations for the Flat Files](#)" on page 3-3.
2. Organize OWB_DEMO to receive the incoming flat file metadata as described in "[Creating Modules in the Project](#)" on page 3-3.
3. Indicate which files to import as described in "[Launching the Import Metadata Wizard](#)" on page 3-4.
4. Specify the metadata structure as described in "[Using the Flat File Sample Wizard](#)" on page 3-4.
5. Import the metadata for both flat files as described in "[Importing the Flat File Data](#)" on page 3-5.

Subsequent Steps

After you successfully define the sources by importing their metadata, next design the target schema as described in [Chapter 4, "Defining Warehouses in Oracle Warehouse Builder"](#).

Notice that, at this point, you have only imported metadata and not extracted data. You design and implement a data extraction strategy in subsequent sections of this guide.

Specifying Locations for the Flat Files

Indicate where the flat files are located.

Notice in the Design Center, on the right side is an explorer called **Connection Explorer** and it includes a node called **Locations**. Use the locations node to indicate where your source data resides.

Expand the **Location** node and the nodes within it to gain a general understanding of the types of source and targets you can access from Warehouse Builder

For this example, we right-click in the **Files** node and select **New** to define a location for the flat files.

Follow the prompts in the **Create File System Location** dialog box. Each location you define corresponds to a specific directory on your computer file system. Therefore, consider naming the location based on the drive and directory. For the purposes of this demonstration, name the location `C_NEWOWBDEMO_SOURCEFILES`.

Creating Modules in the Project

In the Project Explorer, organize OWB_DEMO to receive the incoming flat file metadata.

In a data warehousing implementation, you are likely to have numerous source and target objects. As a means of organizing these various objects, Warehouse Builder requires you to create modules. Each module corresponds to a location that you create in the Connection Explorer.

In this example, since you have only one location for the two flat files, you create one module in the Project Explorer. Right-click on the Files node under OWB_DEMO and select **New**. Name the new module `SALES_EXPENSES`. For its location, specify the location you defined in the previous step, `C_NEWOWBDEMO_SOURCEFILES`.

Launching the Import Metadata Wizard

Launch the Import Metadata Wizard.

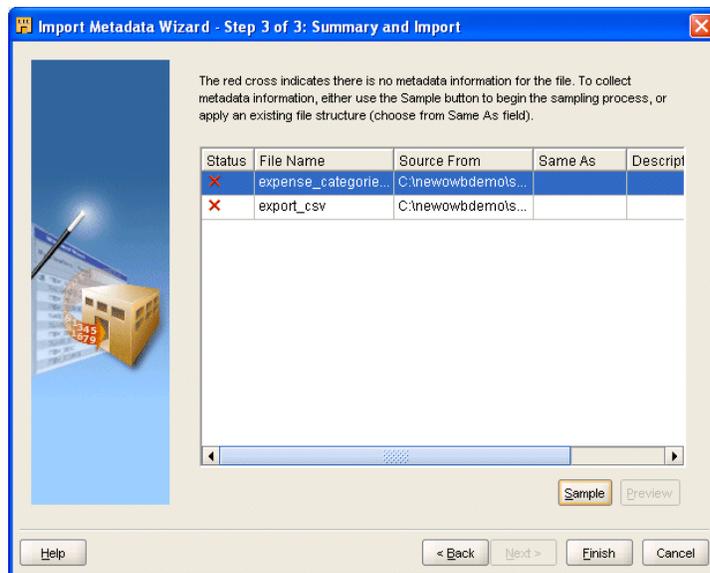
Right-click on the module SALES_EXPENSES, select **New**, and follow the prompts in the Import Metadata Wizard. The prompts in the wizard vary according to the type of module you selected and therefore the type of data object you are importing.

In this example, you selected to import two flat files. On the summary page of the Import Metadata Wizard, select one of the files and then select **Sample** to launch the Flat File Sample Wizard.

In the next steps, you sample each file in turn and then select **Finish** on this page to import the metadata.

Figure 3–1 shows **Sample** on the Summary and Import page of the Import Metadata Wizard.

Figure 3–1 Import Metadata Wizard



Using the Flat File Sample Wizard

Follow the prompts in the Flat File Sample Wizard to specify the metadata structure.

Based on the number of characters you specify to be sampled, the wizard reads the flat file data and provides you with suggestions as to the structure of the metadata. If the sample size is too small, the wizard may misinterpret the data and make invalid suggestions. Accordingly, you can modify and refine the settings in the wizard.

For the purposes of this example, the wizard correctly determines that the file is delimited, includes a single record type, and the character set is WE8MSWIN1252. Accept all the default settings presented in the Flat File Wizard.

To become familiar with the various types of files the wizard can sample, notice the options on the wizard pages and also select **Help** for additional insights.

After sampling the first flat file, return to the Summary and Import page of Metadata Import Wizard to sample the second file.

Accept the default setting in the Flat File Wizard as you did for the previous file.

Importing the Flat File Data

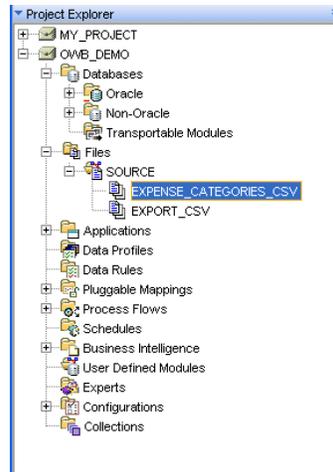
Import the metadata for both flat files.

Return again to the Summary and Import page and select **Finish**.

When you select **Finish**, the wizard imports the data based on the selections you made when sampling the data. The two comma separated files now display under the SALES_EXPENSES module which is under the Files node in OWB_DEMO project.

Figure 3–2 shows the two files after being imported into the SOURCE module.

Figure 3–2 The SOURCE Flat File Module



Defining Warehouses in Oracle Warehouse Builder

Using Oracle Warehouse Builder, you can design a warehouse that is either relational or dimensional.

Warehouse Builder explicitly separates dimensional design from physical implementation. You can choose either a relational implementation or a multidimensional implementation for the dimensional objects using a simple click operation. Therefore, you can implement the same dimensional object as a relational target warehouse or a multidimensional warehouse.

This section shows you how to design a dimensional model implemented as a relational target warehouse. You model a small data warehouse consisting of a cube and two dimensions. Although you can use Warehouse Builder to model complex snowflake schemas, for the purposes of this demonstration, you model a simple star schema consisting of a cube with foreign key references to the two dimensions.

This section includes the following topics:

- [General Steps for Defining a Relational Target Warehouse](#)
- [Identifying the Warehouse Target Schema](#)
- [About Flat File Sources in Warehouse Builder](#)
- [About Dimensions](#)

General Steps for Defining a Relational Target Warehouse

To define a relational target warehouse, complete the following steps:

1. Designate a schema as the warehouse target schema as described in "[Identifying the Warehouse Target Schema](#)" on page 4-2.
2. Define or import source and target objects into the warehouse target module.

In general, you can right-click on any node in the warehouse target module and select either **New** or **Import**. Warehouse Builder launches the appropriate wizard to guide you. Click **Help** for additional information.

The types of objects you add to the warehouse target module depend on the nature of your source data and what it is you want to accomplish with the data warehouse.

To continue with the exercises presented in this guide, see "[Exercise: Adding External Tables to the Target Module](#)" on page 4-3 and "[Exercise: Understanding Dimensions](#)" on page 4-4.

3. Configure the source and target objects as necessary.

Some objects require additional configuration. After you import or define an object in the warehouse module, right-click and select **Configure** to review the settings and make changes as necessary.

Subsequent Steps

After you successfully define the target warehouse, next design a strategy for extracting the source data, transforming it as necessary, and loading it into the warehouse.

Designing the ETL strategy is discussed in [Chapter 5, "Defining ETL Logic"](#).

Identifying the Warehouse Target Schema

In a traditional data warehousing implementation, there is typically only one target schema, which is the data warehouse target. In Warehouse Builder, however, any user that you define has the potential to become a target schema. Therefore, you need to indicate which of the schemas is to be the data warehouse target schema.

To designate a schema as the data warehouse target schema:

1. Register the schema in Warehouse Builder.

In the Global Explorer panel, expand the Security node. Right-click the Users node and select **New**.

On the Create User dialog box, select **Create DB User...** and follow the prompts. Click **Help** if you need more information.

For the purposes of the demonstration, create a new schema and call it `EXPENSE_WH`.

2. Specify the location information for the new schema.

In the Connection Explorer, right click and select **New** from **Locations** under the Oracle node.

Create a location named `EXPENSE_WH_LOCATION`. Select the option to test the connection.

3. In the Project Explorer, associate a module with the schema location.

Recall that in "[Example: Importing Metadata from Flat Files](#)" on page 3-2, you created a module to correspond to a location from which you import metadata. In a similar fashion, you need to create a module to correspond to the location for the target schema.

In the `OWB_DEMO` project, expand the Databases node, right-click the Oracle node, and select **New**. Follow the prompts in the Create Module wizard. Be sure to designate the module status as `Warehouse Target`.

For the purposes of the demonstration, name the module `EXPENSE_WH`.

4. Familiarize yourself with the new data warehouse target schema.

In the Project Explorer, expand the node for the newly defined warehouse target module. Notice the various types of objects listed under the node. These are the types of objects that you can either define in or import into the module.

About Flat File Sources in Warehouse Builder

The types of objects you add to the target module have implications on the ETL logic you subsequently design. If your source data originates from flat files, you can choose to generate either SQL*Loader code or SQL code. Each type of code has its own advantages. Stated concisely, SQL*Loader handles large volumes of data better while SQL enables a broader range of complex joins and transformations.

To utilize SQL*Loader in Warehouse Builder, import the flat files as described in ["Example: Importing Metadata from Flat Files"](#) on page 3-2. To utilize SQL, however, you must define an external table in the warehouse module as described in ["Exercise: Adding External Tables to the Target Module"](#) on page 4-3.

External tables are database objects in the Oracle Database, versions 9i and higher. You cannot use external tables with any other database type or any Oracle Database previous to the 9i release.

Exercise: Adding External Tables to the Target Module

External tables are tables that represent data from flat files in a relational format. They are read-only tables that act like regular source tables in Warehouse Builder. Each external table you create corresponds to a single record type in an existing flat file.

Our objective in this exercise is to create the necessary external tables for the two flat files we previously imported. Since both files have a single record type, we need to create only one external table for each file.

To add external tables to the target warehouse module:

1. In the Project Explorer, expand the Databases node and then the Oracle node.
2. Expand the target module where you want to create the external table.
That is, expand the `EXPENSE_WH` module.
3. Right-click the External Tables node and select **New**.

Warehouse Builder displays the Create External Table wizard. Follow the prompts.

Name the external table `EXPENSE_CATEGORIES`. When prompted to select a flat file, select `EXPENSE_CATEGORIES_CSV`.

4. Repeat the previous step to create an external table called `EXPENSE_DATA` to represent `EXPORT_CSV`.
5. Configure the physical file system details for the two external tables.

Right-click an external table from the module and select **Configure**. On the DataFiles node, right-click and select **Create**. Accept the default name, `NEW_DATAFILE_1`. Enter the name of the flat file from which the external table inherits data. Therefore, specify the data file name as `expense_categories.csv` for one external table and `export.csv` for the other.

About Dimensions

A dimension is a structure that organizes data. Examples of commonly used dimensions are Customers, Time, and Products.

For relational dimensions, using dimensions improves query performance because users often analyze data by drilling down on known hierarchies. An example of a hierarchy is the Time hierarchy of year, quarter, month, day. The Oracle Database uses

these defined hierarchies by rewriting queries that retrieve data from materialized views rather than detail tables.

Typical relational dimension tables have the following characteristics:

- A single column primary key populated with values called warehouse keys. Warehouse keys that provide administrative control over the dimension, support techniques that preserve dimension history, and reduce the size of cubes.
- One or more hierarchies that are explicitly defined as dimension objects. Hierarchies maximize the number of query rewrites by the Oracle server.
- Dimensions are the primary organizational unit of data in a star schema. Examples of some commonly used dimensions are Customer, Product, and Time.

A dimension consists of a set of levels and a set of hierarchies defined over these levels. When you create a dimension, you define the following:

- **Dimension Attributes:** A descriptive characteristic of a dimension member. It has a name and a data type.
- **Levels:** Defines the level of aggregation of data. For example, the dimension Products can have the following levels: Total, Groups, and Product.
- **Level attributes:** A descriptive characteristic of a level member. Each level in the dimension has a set of level attributes.
- **Hierarchies:** A logical structure that uses ordered levels or a set of data values (for a value-based hierarchy) as a means of organizing data. A hierarchy describes parent-child relationships among a set of levels

Exercise: Understanding Dimensions

To understand the basic concepts and design of a dimension, in this exercise you examine a pre-defined dimension.

To become familiar with the dimensions:

1. Open the PRODUCTS dimension in the Data Object Editor.

In the Project Explorer panel, navigate to `OWB_DEMO`, Databases, Oracle, `SALES_WH`, and then expand Dimensions. Double-click `PRODUCTS`.

Warehouse Builder launches the Data Object Editor. The Data Object Editor is the single interface where you can easily design, create, and manage a variety of database or dimensional objects.

2. Observe the dimension attributes.

A dimension attribute is a descriptive characteristic of a dimension member. It has a name and a data type. A dimension attribute is applicable to one or more levels in the dimension. They are implemented as level attributes to store data.

The list of dimension attributes must include all the attributes that you may need for any of the levels in the dimension.

For example, the Products dimension has a dimension attribute called Description. This attribute is applicable to all the levels Total, Groups, and Products and stores the description for each of the members of these levels.

3. Observe the levels.

The levels in a dimension represent the level of aggregation of data. A dimension must contain at least one level, except in the case of a dimension that contains a value-based hierarchy. Every level must have level attributes and a level identifier.

For example, the dimension Products can have the following levels: Total, Groups, and Product.

4. For additional exercises, you can create a time dimension and a regular dimension by following the Oracle By Example at

http://www.oracle.com/technology/obe/10gr2_owb/owb10gr2_gs/owb/lesson3/relational-wh-03.htm

About Levels

Every level must have two identifiers: a surrogate identifier and a business identifier. When you create a dimension, each level must implement the dimension attributes marked as the surrogate identifier and business identifier (attributes, in the case of a composite business identifier) of the dimension.

A surrogate identifier uniquely identifies each level record across all the levels of the dimension. It must be composed of a single attribute. Surrogate identifiers enable you to hook facts to any dimension level as opposed to the lowest dimension level only.

For a dimension that has a relational implementation, the surrogate identifier should be of the data type `NUMBER`. Because the value of the surrogate identifier must be unique across all dimension levels, you use the same sequence to generate the surrogate identifier of all the dimension levels.

For a relational implementation, the surrogate identifier serves the following purposes:

- If a child level is stored in a different table from the parent level, each child level record stores the surrogate identifier of the parent record.
- In a fact table, each cube record stores only the surrogate identifier of the dimension record to which it refers. By storing the surrogate identifier, the size of the fact table that implements the cube is reduced.

A business identifier consists of a user-selected list of attributes. The business identifier must be unique across the level and is always derived from the natural key of the data source. The business identifier uniquely identifies the member. For example, the business identifier of a Product level can be its Universal Product Code (UPC), which is a unique code for each product.

The business identifier does the following:

- Identifies a record in business terms.
- Provides a logical link between the fact and the dimension or between two levels.
- Enables the lookup of a surrogate key.

When you populate a child level in a dimension, you must specify the business identifier of its parent level. When you populate a cube, you must specify the business identifier of the dimension level to which the cube refers.

A parent identifier is used to annotate the parent reference in a value-based hierarchy.

For example, an `EMPLOYEE` dimension with a value-based hierarchy, has the following dimension attributes: `ID`, `FIRST_NAME`, `LAST_NAME`, `EMAIL`, `PHONE`, `JOB_ID`, `HIRE_DATE`, and `MANAGER_ID`. In this dimension, `ID` is the surrogate identifier and `MANAGER_ID` is the parent identifier.

Defining Level Attributes

A level attribute is a descriptive characteristic of a level member. Each level in the dimension has a set of level attributes. To define level attributes, you just select the dimension attributes that the level will implement. A level attribute has a distinct name and a data type. The data type is inherited from the dimension attribute that the level attribute implements. The name of the level attribute can be modified to be different from that of the dimension attribute that it implements.

Every level must implement the attribute marked as the surrogate identifier and the business identifier in the set of the dimension attributes.

Defining Hierarchies

A dimension hierarchy is a logical structure that uses ordered levels or a set of data values (for a value-based hierarchy) as a means of organizing data. A hierarchy describes parent-child relationships among a set of levels. A level-based hierarchy must have at least one level. A level can be part of more than one hierarchy.

For example, the Time dimension can have the following two hierarchies:

Fiscal Hierarchy: Fiscal Year > Fiscal Quarter > Fiscal Month > Fiscal Week > Day

Calendar Hierarchy: Calendar Year > Calendar Quarter > Calendar Month > Day

All hierarchies must be strict 1:n relationships. One record in a parent level corresponds to multiple records in a child level. But one record in a child level corresponds to only one parent record within a hierarchy.

Dimension Roles

A dimension role is an alias for a dimension. In a data warehouse, a cube can refer to the same dimension multiple times, without requiring the dimension to be stored multiple times. Multiple references to the same dimension may cause confusion. So you create an alias for each reference to the dimension, thus allowing the joins to be instantly understandable. In such cases, the same dimension performs different dimension roles in the cube.

For example, a sales record can have the following three time values:

- Time the order is booked
- Time the order is shipped
- Time the order is fulfilled

Instead of creating three time dimensions and populating them with data, you can use dimension roles. Model one time dimension and create the following three roles for the time dimension: order booked time, order shipped time, and order fulfillment time. The sales cube can refer to the order time, ship time, and fulfillment time dimensions.

When the dimension is stored in the database, only one dimension is created and each dimension role references this dimension. But when the dimension is stored in the OLAP catalog, Warehouse Builder creates a dimension for each dimension role. Thus, if a time dimension has three roles, three dimensions are created in the OLAP catalog. However, all three dimensions are mapped to the same underlying table. This is a workaround because the OLAP catalog does not support dimension roles.

Note: Dimension roles can be created for dimensions that have a relational implementation only.

Level Relationships

A level relationship is an association between levels in a dimension hierarchy. Level relationships are implemented using level attributes that store the reference to the parent level in the hierarchy.

For example, the Products dimension has the following hierarchy: Total > Groups > Product. Warehouse Builder creates two level relationships: Product to Groups and Groups to Total. Two new attributes implement this level relationship: one in the Product level and one in the Groups level. These attributes store the surrogate ID of the parent level.

Dimension Example

An example of a dimension is the Products dimension that you use to organize product data. [Table 4-1](#) lists the levels in the Products dimension and the surrogate identifier and business identifier for each of the levels in the dimension.

Table 4-1 Products Dimension Level Details

Level	Attribute Name	Identifier
Total	ID	Surrogate
	Name	Business
	Description	
Groups	ID	Surrogate
	Name	Business
	Description	
Product	ID	Surrogate
	UPC	Business
	Name	
	Description	
	Package Type	
	Package Size	

The Products dimension contains the following hierarchy:

Hierarchy 1: Total > Groups > Product

Control Rows

Warehouse Builder creates control rows that enable you to link fact data to a dimension at any level. For example, you may want to reuse a Time dimension in two different cubes to record the budget data at the month level and the actual data at the day level. Because of the way dimensions are loaded with control rows, you can perform this without any additional definitions. Each member in a dimension hierarchy is represented using a single record.

All control rows have negative dimension key values starting from -2. For each level value of higher levels, a row is generated that can act as a unique linking row to the fact table. All the lower levels in this linking or control rows are nulled out.

Consider the Products dimension described in [Dimension Example](#). You load data into this dimension from a table that contains four categories of products. Warehouse Builder inserts control rows in the dimension as shown in [Table 4-2](#). These rows

enable you to link to a cube at any dimension level. Note that the table does not contain all the dimension attribute values.

Table 4–2 Control Rows Created for the Products Dimension

Dimension Key	Total Name	Categories Name	Product Name
-3	TOTAL		
-9	TOTAL	Hardware	
-10	TOTAL	Software	
-11	TOTAL	Electronics	
-12	TOTAL	Peripherals	

To obtain the real number of rows in a dimension, count the number of rows by including a WHERE clause that excludes the NULL rows. For example, to obtain a count on Products, count the number of rows including a WHERE clause to exclude NULL rows in Product.

Implementing a Dimension

Implementing a dimension consists of specifying how the dimension and its data are physically stored. Warehouse Builder enables several types of implementations for dimensional objects, including multi-dimensional implementations. However, for the purposes of this guide, we discuss a relational implementation only.

Star Schema

In a star schema implementation, Warehouse Builder stores the dimension data in a single table. Because the same table or view stores data for more than one dimension level, you must specify a dimension key column in the table. The dimension key column is the primary key for the dimension. This column also forms the foreign key reference to the cube.

Each level implements a subset of dimension attributes. By default, the level attribute name is the same as the dimension attribute name. To avoid name conflicts caused by all level data being stored in the same table, Warehouse Builder uses the following guidelines for naming in a star table:

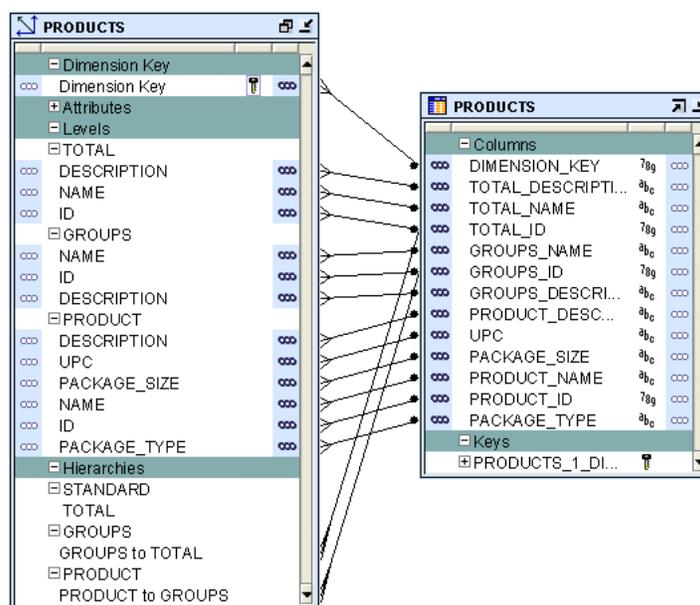
- If the level attribute name is not unique, Warehouse Builder prefixes it with the name of the level.
- If the level attribute name is unique, Warehouse Builder does not use any prefix.

Note: To ensure that no prefixes are used, you must explicitly change the level attribute name in the Create Dimension wizard or the Data Object Editor.

For example, if you implement the Products dimension using a star schema, Warehouse Builder uses a single table to implement all the levels in the dimension.

Figure 4–1 displays the star schema implementation of the Products dimension. The attributes in all the levels are mapped to different columns in a single table called PRODUCTS. The column called DIMENSION_KEY stores the surrogate ID for the dimension and is the primary key of the table.

Figure 4–1 Star Schema Implementation of Products Dimension



Binding

When you perform binding, you specify the database columns that will store the data of each attribute and level relationship in the dimension. You can perform either auto binding or manual binding for a dimension.

Auto Binding When you perform auto binding, Warehouse Builder binds the dimension object attributes to the database columns that store their data. When you perform auto binding for the first time, Warehouse Builder also creates the tables that are used to store the dimension data.

When you perform auto binding on a dimension that is already bound, Warehouse Builder uses the following rules:

- If the implementation method of the dimension remains the same, Warehouse Builder rebinds the dimensional object to the existing implementation objects.
For example, you create a Products dimension using the star schema implementation method and perform auto binding. The dimension data is stored in a table called Products. You modify the dimension definition at a later date but retain the implementation method as star. When you now auto bind the Products dimension, Warehouse Builder rebinds the Products dimension attributes to the same implementation tables.
- If the implementation method of a dimension is changed, Warehouse Builder deletes the old implementation objects and creates a new set of implementation tables. If you want to retain the old implementation objects, you must first unbind the dimensional object and then perform auto binding. For more information on implementation methods, see "Star Schema" on page 4-8.

For example, you create a Products dimension using the star schema implementation method and bind it to the implementation table. You now edit this dimension and change its implementation method to snowflake. When you now perform auto binding for the modified Products dimension, Warehouse Builder deletes the table that stores the dimension data, creates new implementation

tables, and binds the dimension attributes and relationships to the new implementation tables.

To perform auto binding:

1. In the Project Explorer, right-click the dimension and select **Open Editor**.

The Data Object Editor for this dimension is displayed.

2. On the Dimensional tab, right-click the Dimension node and select **Bind**.

Alternatively, select the dimension node on the canvas and from the **Object** menu choose **Bind**.

If the Bind option is not enabled, check if the dimension is a relational dimension and that the Manual options is not set in the Implementation section of the Storage tab.

Auto binding uses the implementation settings described in [Star Schema](#) on page 4-8.

Manual Binding You would typically use manual binding to bind existing tables to a dimension. Use manual binding if no auto binding or rebinding is required.

To perform manual binding for a dimension:

1. Create the implementation objects (tables or views) that you will use to store the dimension data.

In the case of relational dimensions, create the sequence used to load the surrogate identifier of the dimension. You can also choose to use an existing sequence.

2. In the Project Explorer, right-click the dimension and select **Open Editor**.

The Data Object Editor for the dimension opens. On the canvas, the Dimensional tab is active.

3. Right-click the dimension and select **Detail View**.

Warehouse Builder opens a new tab that has the same name as the dimension.

4. From the Palette, drag and drop the operator that represents the implementation object onto the canvas.

Warehouse Builder displays the Add a New or Existing <Object> dialog box. For example, if the dimension data is stored in a table, drag a Table operator from the Palette and drop it onto the canvas. The Add a New or Existing Table dialog box is displayed.

5. Choose the **Select an existing <Object>** option and then select the data object from the list of objects displayed in the selection tree.

6. Click **OK**.

A node representing the object that you just added is displayed on the canvas.

7. If more than one data object is used to store the dimension data, perform steps 4 to 6 for each data object.

8. Map the attributes in each level of the dimension to the columns that store their data. To do this, hold down your mouse on the dimension attribute, drag, and then drop on the column that stores the attribute value.

Also map the level relationships to the database column that store their data.

For example, for the Products dimension described in "[Dimension Example](#)" on page 4-7, the attribute Name in the Groups level of the Products dimension is

stored in the Group_name attribute of the Products_tab table. Hold down the mouse on the Name attribute, drag, and drop on the Group_name attribute of the Products_tab table.

About Cubes

Cubes contain measures and link to one or more dimensions. The axes of a cube contain dimension members and the body of the cube contains measure values. Most measures are additive. For example, sales data can be organized into a cube whose edges contain values for Time, Products, and Customers dimensions and whose body contains values from the measures Value sales, and Dollar sales.

A cube is linked to dimension tables over foreign key constraints. Since data integrity is vital, these constraints are critical in a data warehousing environment. The constraints enforce referential integrity during the daily operations of the data warehouse.

Data analysis applications typically aggregate data across many dimensions. This enables them to look for anomalies or unusual patterns in the data. Using cubes is the most efficient way of performing these type of operations. In a relational implementation, when you design dimensions with warehouse keys, the cube row length is usually reduced. This is because warehouse keys are shorter than their natural counterparts. This results in lesser amount of storage space needed for the cube data.

A typical cube contains:

- A primary key defined on a set of foreign key reference columns or, in the case of a data list, on an artificial key or a set of warehouse key columns. When the cube is a data list, the foreign key reference columns do not uniquely identify each row in the cube.
- A set of foreign key reference columns that link the table with its dimensions.

Defining a Cube

A cube consists of the set of measures defined over a set of dimensions. To create a cube, you must define the following:

- [Cube Measures](#)
- [Cube Dimensionality](#)

Cube Measures

A measure is data, usually numeric and additive, that can be examined and analyzed. Examples of measures include sales, cost, and profit. A cube must have one or more measures. You can also perform aggregation of measures. Only numeric measures can be aggregated.

Cube Dimensionality

A cube is defined by a set of dimensions. A cube can refer to a level that is not the lowest level in a dimension.

For cubes that use a pure relational implementation, you can reuse the same dimension multiple times with the help of dimension roles. For more information on dimension roles, see "[Dimension Roles](#)" on page 4-6.

Before you validate a cube, ensure that all the dimensions that the cube references are valid.

To define a dimension reference, specify the following:

- The dimension and the level within the dimension to which the cube refers.
 For a cube that uses a relational implementation, you can refer to intermediate levels in a dimension. Warehouse Builder supports a reference to the non surrogate identifier of a level, for example, the business keys.
- For dimensions that use a relational implementation, a dimension role for each dimension to indicate what role the dimension reference is performing in the cube. Specifying the dimension role is optional.

Cube Example

The Sales cube stores aggregated sales data. It contains the following two measures: Value_sales and Dollar_sales.

- Value_sales: Stores the amount of the sale in terms of the quantity sold.
- Dollar_sales: Stores the amount of the sale.

Table 4–3 describes the dimensionality of the Sales cube. It lists the name of the dimension and the dimension level that the cube references.

Table 4–3 Dimensionality of the Sales Cube

Dimension Name	Level Name
Products	Product
Customers	Customer
Times	Day

Implementing a Cube

When you implement a cube, you specify the physical storage details for the cube. As with dimensions, Warehouse Builder enables you to implement cubes in relational or multidimensional form. We address the relational implementation in this guide.

Relational Implementation of a Cube

The database object used to store the cube data is called a fact table. A cube must be implemented using only one fact table. The fact table contains columns for the cube measures and dimension references.

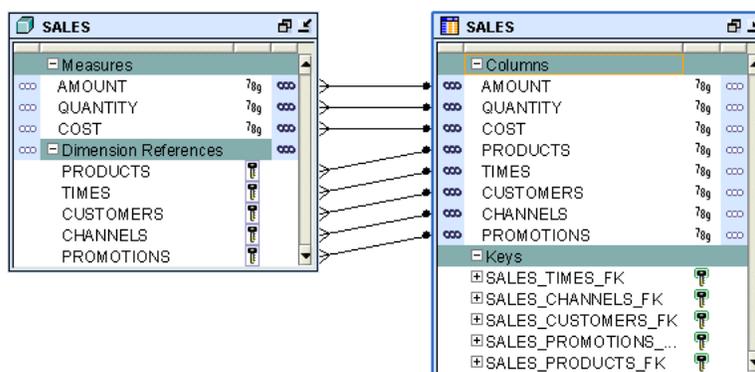
To implement a cube:

- Select a table or materialized view that will store the cube data.
- For each measure, select a column that will store the measure data.
- For each dimension reference, select a column that will store the dimension reference.

Each dimension reference corresponds to a column on the fact table and optionally a foreign key from the fact table to dimension table. The 1:n relationships from the fact tables to the dimension tables must be enforced.

Figure 4–2 displays the bindings for the relational implementation of the Sales cube. The data for the Sales cube is stored in a table called SALES.

Figure 4-2 Implementation of the Sales Cube



Binding

When you perform binding, you specify the database columns that will store the data of each measure and dimension reference of the cube. You can perform auto binding or manual binding for a cube.

Auto Binding When you perform auto binding, Warehouse Builder creates the table that stores the cube data and then binds the cube measures and references to the database columns. For detailed steps on performing auto binding, see ["Auto Binding"](#) on page 4-9.

When you perform auto binding for a cube, ensure that you auto bind the dimensions that a cube references before you auto bind the cube. You will not be able to deploy the cube if any dimension that the cube references has been auto bound after the cube was last auto bound.

For example, you create the SALES cube that references the TIMES and PRODUCTS dimensions and perform auto binding for the cube. You later modify the definition of the PRODUCTS dimension. If you now attempt to auto bind the SALES cube again, Warehouse Builder generates an error. You must first auto bind the PRODUCTS dimensions and then auto bind the cube.

Manual Binding In manual binding, you must first create the table or view that stores the cube data and then map the cube references and measures to the database columns that store their data. Alternatively, you can use an existing database table or view to store the cube data.

To perform manual binding for a cube:

1. Create the table or view that stores the cube data.
2. In the Project Explorer, right-click the cube and select **Open Editor**.
The Data Object Editor for the cube opens. On the canvas, the Dimensional tab is active.
3. Right-click the cube and select **Detail View**.
Warehouse Builder opens a new tab that has the same name as the cube.
4. From the Palette, drag and drop the operator that represents the implementation object onto the canvas.

Warehouse Builder displays the Add a New or Existing <Object> dialog box. For example, if the cube data is stored in a table, drag a Table operator from the Palette

and drop it onto the canvas. The Add a New or Existing Table dialog box is displayed.

5. Choose the **Select an existing <object>** option and then select the data object from the list of objects displayed in the selection tree.

6. Click **OK**.

A node representing the object that you just added is displayed on the canvas.

7. Map the measures and dimension references of the cube to the columns that store the cube data. To do this, hold down your mouse on the measure or dimension reference, drag, and then drop on the data object attribute that stores the measure or dimension reference.

Part II

Loading Data into Your Data Warehouse

This section discusses loading data into the data warehouse, and includes:

- [Chapter 5, "Defining ETL Logic"](#)
- [Chapter 6, "Improving Loading Performance"](#)
- [Chapter 7, "Deploying to Target Schemas and Executing ETL Logic"](#)

Defining ETL Logic

After you create and import data object definitions in Warehouse Builder, you can design extraction, transformation, and loading (ETL) operations that move data from sources to targets. In Warehouse Builder, you design these operations in a mapping.

This section contains the following topics that describe how to create, edit, and use mappings:

- [About Mappings and Operators](#) on page 5-1
- [Instructions for Defining Mappings](#) on page 5-2
- [Creating a Mapping](#) on page 5-2
- [Adding Operators](#) on page 5-4
- [Editing Operators](#) on page 5-6
- [Connecting Operators](#) on page 5-7
- [Setting Mapping Properties](#) on page 5-10
- [Setting Operator, Group, and Attribute Properties](#) on page 5-11
- [Synchronizing Operators and Workspace Objects](#) on page 5-12

About Mappings and Operators

Mappings describe a series of operations that extract data from sources, transform it, and load it into targets. They provide a visual representation of the flow of the data and the operations performed on the data. When you design a mapping in Warehouse Builder, you use the Mapping Editor interface.

The basic design element for a mapping is the operator. Use operators to represent sources and targets in the data flow. Also use operators to define how to transform the data from source to target. The operators you select as sources have an impact on how you design the mapping. Based on the operators you select, Warehouse Builder assigns the mapping to one of the following Mapping Generation Languages:

- PL/SQL
- SQL*Loader
- ABAP

Each of these code languages require you to adhere to certain rules when designing a mapping.

This guide illustrates how to define a PL/SQL mapping. To define the other types of mappings, refer to the Oracle Warehouse Builder User's Guide. A basic rules for

defining a PL/SQL mapping is as follows: PL/SQL mappings can contain any type of source operator other a flat file operator or a SAP/R3 source.

Instructions for Defining Mappings

To define a mapping, refer to the following sections:

1. [Creating a Mapping](#) on page 5-2
2. [Adding Operators](#) on page 5-4
3. [Editing Operators](#) on page 5-6
4. [Connecting Operators](#) on page 5-7
5. [Setting Mapping Properties](#) on page 5-10
6. [Setting Operator, Group, and Attribute Properties](#) on page 5-11
7. [Configuring Mappings Reference in the *Warehouse Builder Online Help*](#)
8. When you are satisfied with the mapping design, generate the code by selecting the Generate icon in the toolbar.

Subsequent Steps

After you design a mapping and generate its code, you can next create a process flow or proceed directly with deployment followed by execution.

Use process flows to interrelate mappings. For example, you can design a process flow such that the completion of one mapping triggers an email notification and launches another mapping.

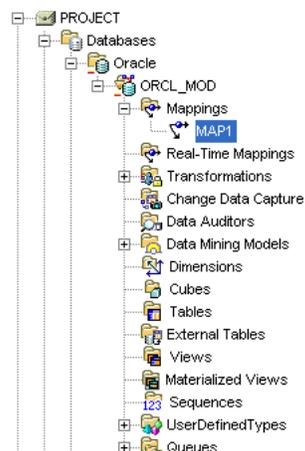
Deploy the mapping, and any associated process flows you created, and then execute the mapping.

Creating a Mapping

To create a mapping:

1. Navigate to the **Mappings** node in the Project Explorer. This node is located under a warehouse target module, under the Databases folder, under the Oracle folder.

[Figure 5–1, "Mappings Node on the Project Explorer"](#) shows the Mappings node containing maps MAP1. The warehouse target in this example is named ORCL_MOD.

Figure 5–1 Mappings Node on the Project Explorer

2. Right-click Mappings and then select **New**.

Warehouse Builder opens the Create Mapping dialog box.

3. Enter a name and an optional description for the new mapping.

Select **Help** to review the rules on naming and describing mappings.

4. Click **OK**.

Warehouse Builder stores the definition for the mapping and inserts its name in the Project Explorer. Warehouse Builder opens a mapping editor for the mapping and displays the name of the mapping in the title bar.

To open a previously created mapping:

1. From the Project Explorer, locate a warehouse target module under the Databases folder and then under the Oracle Database folder.
2. Expand the Mappings node.
3. Open the Mapping Editor in one of the following ways:
 - Double-click a mapping.
 - Select a mapping and then from the **Edit** menu, select **Open Editor**.
 - Select a mapping and press **Ctrl + O**.
 - Right-click a mapping, and select **Open Editor**.

Warehouse Builder displays the Mapping Editor.

Types of Operators

As you design a mapping, you select operators from the Mapping Editor palette and drag them onto the canvas.

- **Oracle source/target operators:** Use these operators represent Oracle Database objects such as Oracle tables, views, materialized views.
- **Remote and Non-Oracle Source and Target Operators:** The use of these operator have special requirements.
- **Data flow operators:** Use data flow operators to transform data.

- **Pre/Post Processing operators:** Use these operators to perform processing before or after executing a mapping. The Mapping parameter operator is used to provide values to and from a mapping.
- **Pluggable mapping operators:** A pluggable mapping is a reusable grouping of mapping operators that behaves as a single operator.

Adding Operators

The steps you take to add an operator to a mapping depend on the type of operator you select. This is because some operators are bound to workspace objects while others are not. As a general rule, when you add a data source or target operator, Warehouse Builder creates and maintains a version of that object in the Warehouse Builder workspace and a separate version for the Mapping Editor. For example, when you add a table operator to a mapping, Warehouse Builder maintains a separate copy of the table in the workspace. The separate versions are said to be *bound* together. That is, the version in the mapping is bound to the version in the workspace.

To distinguish between the two versions, this section refers to objects in the workspace either generically as *workspace objects* or specifically as *workspace tables*, *workspace views*, and so on. And this section refers to operators in the mapping as *table operators*, *view operators*, and so on. Therefore, when you add a dimension to a mapping, refer to the dimension in the mapping as the *dimension operator* and refer to the dimension in the workspace as the *workspace dimension*.

Warehouse Builder maintains separate workspace objects for some operators so that you can synchronize changing definitions of these objects. For example, when you re-import a new metadata definition for the workspace table, you may want to propagate those changes to the table operator in the mapping. Conversely, as you make changes to a table operator in a mapping, you may want to propagate those changes back to its associated workspace table. You can accomplish these tasks by a process known as synchronizing. In Warehouse Builder, you can synchronize automatically. Alternatively, synchronize manually from within the Mapping Editor.

To add an operator to a mapping:

1. Open the Mapping Editor.
2. From the **Mapping** menu, select **Add** and select an operator. Alternatively, you can drag an operator icon from the toolbox and drop it onto the Mapping Editor canvas.

If you select an operator that you can bind to a workspace object, the Mapping Editor displays the **Add Mapping <operator name>** dialog box. For details on how to use this dialog box, select **Help**.

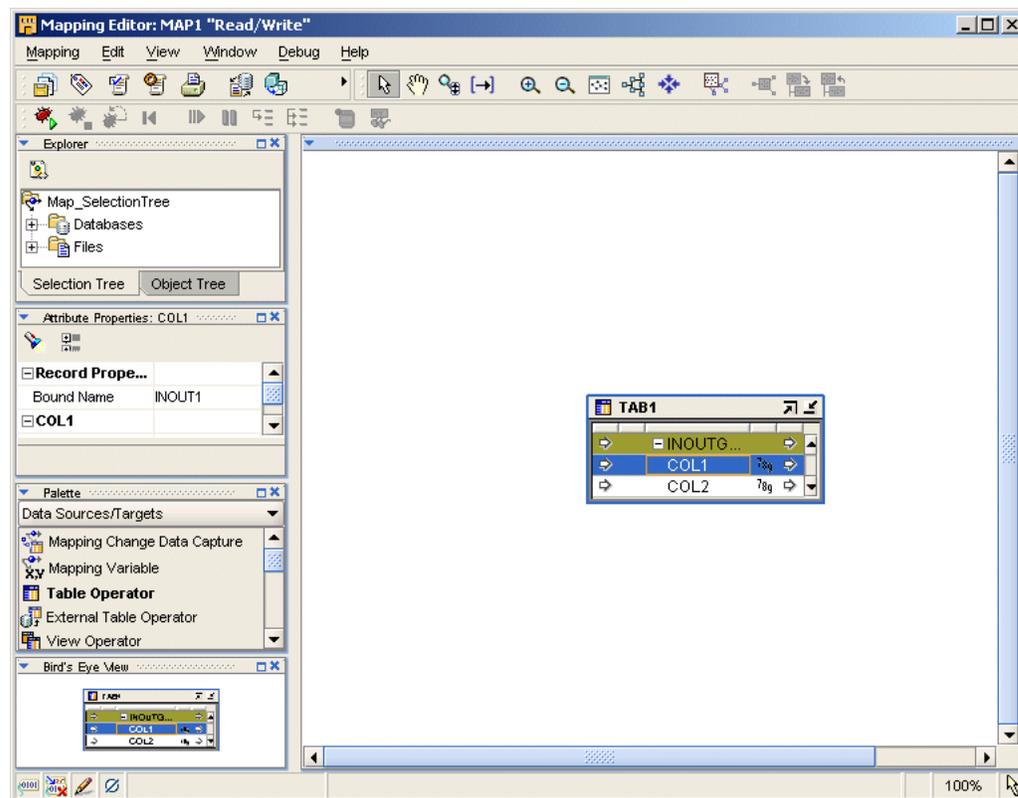
If you select an operator that you cannot bind to a workspace object, Warehouse Builder may display a wizard or dialog box to assist you in creating the operator.

3. Follow any prompts Warehouse Builder displays and click **OK**.

The Mapping Editor displays the operator maximized on the canvas. The operator name appears in the upper left corner. You can view each attribute name and data type.

If you want to minimize the operator, click the arrow in the upper right corner and the Mapping Editor displays the operator as an icon on the canvas.

Figure 5–2 Mapping Editor Showing a Table Operator Source



Adding Operators that Bind to Workspace Objects

When you add an operator that you can bind to a workspace object, the Mapping Editor displays the **Add Mapping <operator name>** dialog box. Select one of the following options:

- [Create Unbound Operator with No Attributes](#)
- [Select from Existing Workspace Object and Bind](#)

Create Unbound Operator with No Attributes

Use this option when you want to use the Mapping Editor to define a new workspace object such as a new staging area table or a new target table.

After you select **Create Unbound Operator with No Attributes**, type a name for the new object. Warehouse Builder displays the operator on the canvas without any attributes.

You can now add and define attributes for the operator as described in "[Editing Operators](#)" on page 5-6. Next, to create the new workspace object in a target module, right-click the operator and select **Create and Bind**.

For an example on how to use this option in a mapping design, see "[Example: Using the Mapping Editor to Create Staging Area Tables](#)" on page 5-8.

Select from Existing Workspace Object and Bind

Use this option when you want to add an operator based on an object you previously defined or imported into the workspace.

Either type the prefix to search for the object or select from the displayed list of objects within the selected module.

To select multiple items, press the Control key as you click each item. To select a group of items located in a series, click the first object in your selection range, press the Shift key, and then click the last object.

You can add operators based on workspace objects within the same module as the mapping or from other modules. If you select a workspace object from another module, the Mapping Editor creates a connector if one does not already exist. The connector establishes a path for moving data between the mapping location and the location of the workspace object.

Editing Operators

Each operator has an editor associated with it. Use the operator editor to specify general and structural information for operators, groups, and attributes. In the operator editor you can add, remove, or rename groups and attributes. You can also rename an operator.

Editing operators is different from assigning loading properties and conditional behaviors. To specify loading properties and conditional behaviors, use the properties windows as described in "[Setting Operator, Group, and Attribute Properties](#)" on page 5-11.

To edit an operator, group, or attribute:

1. Select an operator from the Mapping Editor canvas.

Or select any group or attribute within an operator.

2. Right-click and select **Open Details**.

The Mapping Editor displays the operator editor with the Name Tab, Groups Tab, and Input and Output Tabs for each type of group in the operator.

Some operators include additional tabs. For example, the Match Merge operator includes tabs for defining Match rules and Merge rules.

3. Follow the prompts on each tab and click **OK** when you are finished.

Select **Help** if you need additional information for completing a tab.

Connecting Operators

After you select mapping source operators, operators that transform data, and target operators, you are ready to connect them. Data flow connections graphically represent how the data flows from a source, through operators, and to a target.

You can connect operators by one of the following methods:

- **Connecting Attributes:** Connect individual operator attributes to each other one at a time.
- **Connecting Groups:** Define criteria for connecting all the attributes between two groups.
- **Using an Operator Wizard:** For operators such as the Pivot operator and Name-Address operator, you can use the wizard to define data flow connections.

Connecting Attributes

You can draw a line from a single output attribute of one operator to a single input attribute of another operator.

To connect attributes:

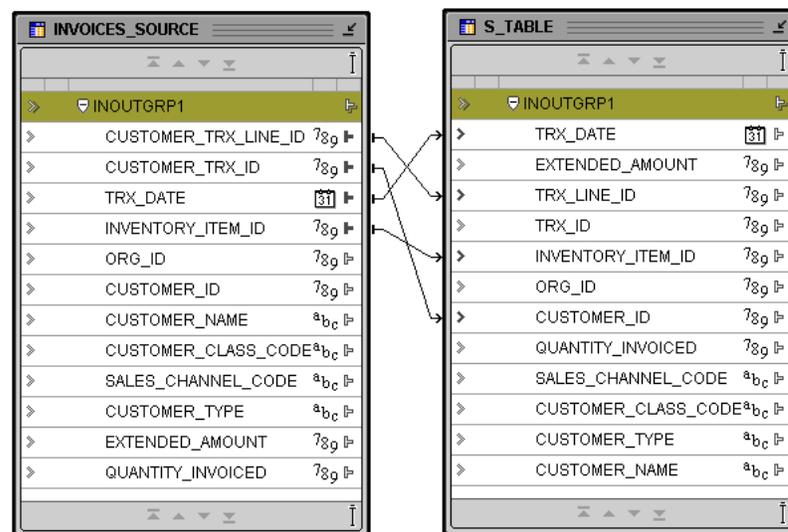
1. Click and hold down the mouse button while the pointer is positioned over an output attribute.
2. Drag the mouse away from the output attribute and toward the input attribute to which you want data to flow.

As you drag the mouse, a line appears on the Mapping Editor canvas to indicate a connection.

3. Release the mouse over the input attribute.
4. Repeat steps one through three until you create all the required data flow connections.

Figure 5–3 displays a mapping with attributes connected.

Figure 5–3 Connected Operators in a Mapping



When connecting attributes, keep the following rules in mind:

- You cannot connect to the same input attribute twice.
- You cannot connect attributes within the same operator.
- You cannot connect out of an input only attribute nor can you connect into an output only attribute.
- You cannot connect operators in such a way as to contradict an established cardinality. Instead, use a Joiner operator.

Connecting Groups

When you connect groups, the Mapping Editor assists you by either automatically copying the attributes or prompts you for more information.

If you connect from one operator group to a target group with no existing attributes, the Mapping Editor automatically copies the attributes and connects the attributes. This is useful for designing mappings such shown in "Example: Using the Mapping Editor to Create Staging Area Tables".

Example: Using the Mapping Editor to Create Staging Area Tables

You can use the Mapping Editor with an unbound table operator to quickly create staging area tables.

The following instructions describe how to create a staging table based on an existing source table. You can also use these instructions to create views, materialized views, flat files, and transformations.

To map a source table to a staging table:

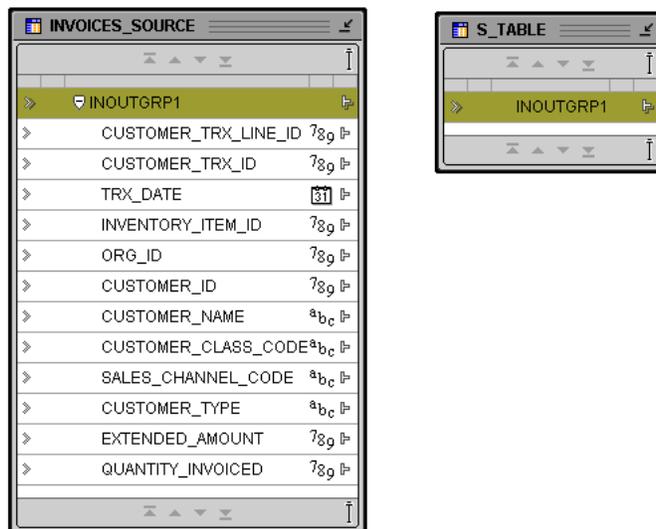
1. In the Mapping Editor, add a source table.

From the menu bar, select **Mapping**, select **Add**, then select **Data Sources/Targets**. In the **Data Sources/Targets** menu, select **Table Operator**.

2. Use the **Add Table Operator** dialog box to select and bind the source table operator in the mapping. From the Add Table Operator dialog box, select **Create unbound operator with no attributes**.

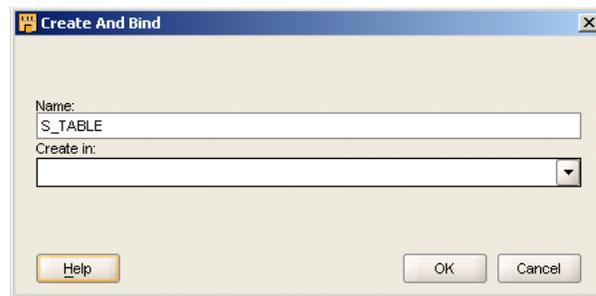
The mapping should now resemble [Figure 5-4](#) with one source table and one staging area table without attributes.

Figure 5-4 Unbound Staging Table without Attributes and Source Table



3. With the mouse button positioned over the group in the source operator, click and hold down the mouse button.
4. Drag the mouse to the staging area table group.
Warehouse Builder copies the source attributes to the staging area table and connects the two operators.
5. In the Mapping Editor, select the unbound table you added to the mapping. Right-click and select **Create and Bind**.

Warehouse Builder displays the dialog box shown in [Figure 5-5](#).

Figure 5–5 Create and Bind Dialog Box

6. In **Create in**, specify the target module in which to create the table.
Warehouse Builder creates the new table in the target module you specify.

Setting Mapping Properties

When you select white space on the mapping canvas, the editor displays the mapping properties in the property inspector along the left side. You can set the following property for the mapping:

- [Target Load Order](#) on page 5-10

Target Load Order

If your mapping includes only one target or is a SQL*Loader or ABAP mapping, target load ordering does not apply. Accept the default settings and continue with your mapping design.

When you design a PL/SQL mapping with multiple targets, Warehouse Builder calculates a default ordering for loading the targets. If you define foreign key relationships between targets, Warehouse Builder creates a default order that loads the parent and then the child. If you do not create foreign key relationships or if a target table has a recursive relationship, Warehouse Builder assigns a random ordering as the default.

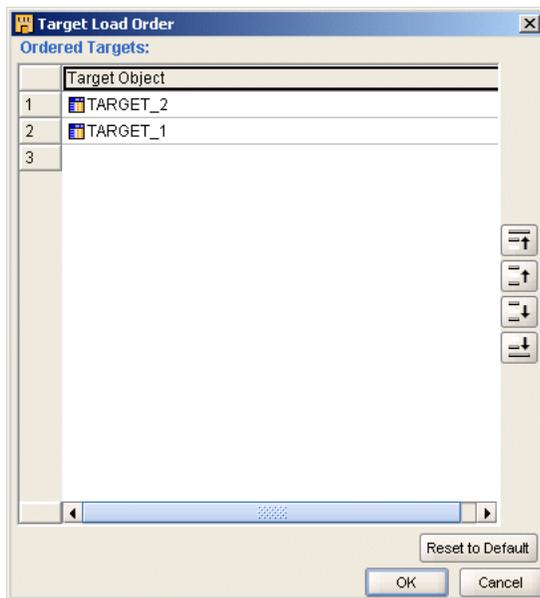
You can override the default load ordering by setting the Target Load Order property. If you make a mistake when reordering the targets, you can restore the default ordering by selecting the [Reset to Default](#) option.

To specify the loading order for multiple targets:

1. Click whitespace in the mapping canvas to view the mapping properties in the Property Window in the upper right corner.
2. Go to the Map Targets Load Order property and click the Ellipsis button on the right side.

Warehouse Builder displays the Targets Load Order dialog box, as shown in [Figure 5–6](#), which shows TARGET2 loading before TARGET1.

Figure 5–6 Target Load Order Dialog Box



3. To change the loading order, select a target and use the shuttle buttons on the right to move the target up or down on the list.

Reset to Default

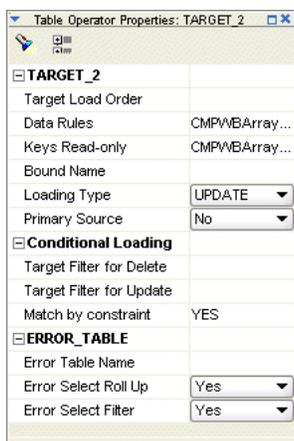
Use the Reset to Default button to instruct Warehouse Builder to recalculate the target loading order. You may want to recalculate if you made an error reordering the targets or if you assigned an order and later change the mapping design such that the original order became invalid.

Setting Operator, Group, and Attribute Properties

When you select an object on the canvas, the editor displays its associated properties in the property inspector along the left side.

Figure 5–7 displays the property inspector for a Table operator.

Figure 5–7 Property Inspector for a Table Operator



You can view and set the following types of properties:

- **Operator Properties:** Properties that affect the operator as a whole. The properties you can set depend upon the operator type.
- **Group Properties:** Properties that affect a group of attributes. Most operators do not have properties for their groups. Examples of operators that do have group properties include the splitter operator and the deduplicator.
- **Attribute Properties:** Properties that pertain to attributes in source and target operators. Examples of attribute properties are data type, precision, and scale.

Synchronizing Operators and Workspace Objects

Many of the operators you use in a mapping have corresponding definitions in the Warehouse Builder workspace. This is true of source and target operators such as table and view operators. This is also true of other operators such as sequence and transformation operators whose definitions you may want to use across multiple mappings. As you make changes to these operators, you may want to propagate those changes back to the workspace object.

You have the following choices in deciding the direction in which you propagate changes:

Synchronizing From a Workspace Object to an Operator: After you begin using mappings in a production environment, there may be changes to the sources or targets that impact your ETL designs. Typically, the best way to manage these changes is through the Warehouse Builder Dependency Manager described in the *Warehouse Builder Online Help*. Use the Dependency Manager to automatically evaluate the impact of changes and to synchronize all effected mappings at one time. Alternatively, in the Mapping Editor, you can manually synchronize objects as described in "Synchronizing From a Workspace Object to an Operator" on page 5-13.

Synchronizing from an Operator to a Workspace Object: When you make changes to an operator in a mapping, you may want to propagate those changes to its corresponding workspace definition. For example, the sources you imported and used in a mapping may have complex physical names for its attributes.

Note that synchronizing is different from refreshing. The refresh command ensures that you are up-to-date with changes made by other users in a multiuser environment. Synchronizing matches operators with their corresponding workspace objects.

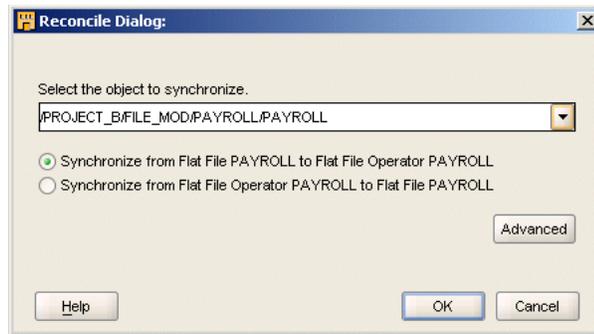
Synchronizing An Operator

To synchronize, select a single operator and synchronize it with the definition of a specified workspace object.

To synchronize an operator, complete the following steps:

1. Select an operator on the Mapping Editor canvas.
2. From the **Edit** menu, select **Synchronize** or right-click the header of the operator and select **Synchronize**.

The Synchronize Operator dialog box displays as shown in [Figure 5-8](#).

Figure 5–8 Synchronizing an Operator

3. By default, Warehouse Builder selects the option for you to synchronize your selected operator with its associated object in the workspace. You can accept the default or select another workspace object from the list box.

In this step you also specify either [Synchronizing From a Workspace Object to an Operator](#) or select the option for [Synchronizing from an Operator to a Workspace Object](#).

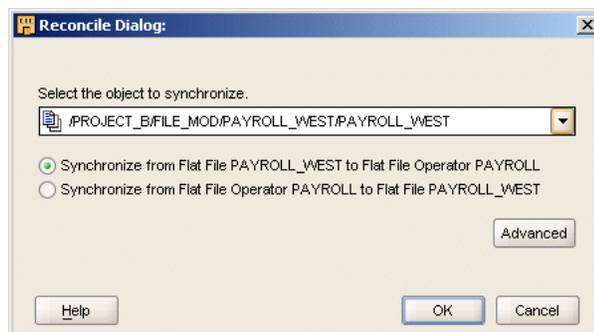
4. As an optional step, click **Advanced** to set the Matching Strategies.
Select **Help** for instruction on how to use the Matching Strategies.
5. Click **OK**.

Synchronizing From a Workspace Object to an Operator

In the Mapping Editor, you can synchronize from a workspace object for any of the following reasons:

- **Manually propagate changes:** Propagate changes you made in a workspace object to its associated operator. Changes to the workspace object can include structural changes, attribute name changes, attribute data type changes. To automatically propagate changes in a workspace object across multiple mappings, see in the *Warehouse Builder Online Help*.
- **Synchronize an operator with a new workspace object:** You can associate an operator with a new workspace object if, for example, you migrate mappings from one version of a data warehouse to a newer version and maintain different object definitions for each version.

[Figure 5–9](#) shows an example of synchronizing a flat file operator with a new workspace object.

Figure 5–9 Synchronizing from a Different Workspace Object

- **Prototype mappings using tables:** When working in the design environment, you could choose to design the ETL logic using tables. However, for production, you may want to the mappings to source other workspace object types such as views, materialized views, or cubes.

Synchronizing Operators based on Workspace Objects

Table 5–1 lists operators and the types of workspace objects from which you can synchronize.

Table 5–1 Operators Synchronized with Workspace Objects

To: Operator	From: Workspace Object Type
Cube Operator	Tables, Views, Materialized Views, Flat Files, Dimensions and Cubes
Dimension Operator	Tables, External Tables, Views, Materialized Views, Flat Files, Dimensions and Cubes
External Table Operator	Tables, External Tables, Views, Materialized Views, Flat Files, Dimensions and Cubes
Flat File Operator	Tables, External Tables, Views, Materialized Views, Flat Files, Dimensions and Cubes
Key Lookup Operator	Tables only
Materialized View Operator	Tables, External Tables, Views, Materialized Views, Files, Dimensions and Cubes
Post Mapping Process Operator	Transformations only
Pre Mapping Process Operator	Transformations only
Sequence Operator	Sequences only
Table Operator	Tables, External Tables, Views, Materialized Views, Flat Files, Dimensions and Cubes
Transformation Operator	Transformations only
View Operator	Tables, External Tables, Views, Materialized Views, Files, Dimensions and Cubes

Note that when you synchronize from an external table operator, Warehouse Builder updates the operator based on the workspace external table only and not its associated flat file.

Synchronizing from an Operator to a Workspace Object

As you make changes to operators in a mapping, you may want to propagate those changes back to a workspace object. By synchronizing, you can propagate changes from the following operators: tables, views, materialized views, transformations, and flat file operators.

Synchronize from the operator to a workspace object for any of the following reasons:

- **Propagate changes:** Propagate changes you made in an operator to its associated workspace object. When you rename the business name for an operator or attribute, Warehouse Builder propagates the first 30 characters of the business name as the bound name.

- **Replace workspace objects:** Synchronize to replace an existing workspace object. Synchronizing from an operator has no impact on the dependent relationship between other operators and the workspace objects.

Improving Loading Performance

Use this section as a guide for creating ETL logic that meets your performance expectations.

Improved Performance via Partition Exchange Loading

Data partitioning can improve performance when loading or purging data in a target system. This practice is known as Partition Exchange Loading (PEL).

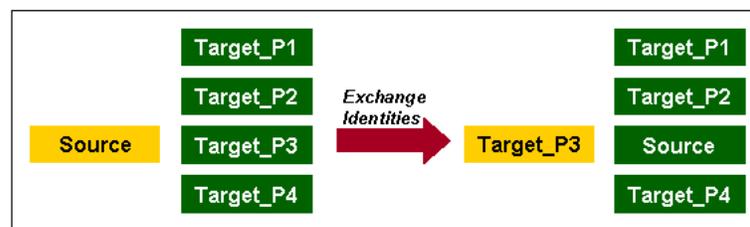
PEL is recommended when loading a relatively small amount of data into a target containing a much larger volume of historical data. The target can be a table, a dimension, or a cube in a data warehouse.

About Partition Exchange Loading

By manipulating partitions in your target system, you can use Partition Exchange Loading (PEL) to instantly add or delete data. When a table is exchanged with an empty partition, new data is added.

You can use Partition Exchange Loading to load new data by exchanging it into a target table as a partition. For example, a table that holds the new data assumes the identity of a partition from the target table and this partition assumes the identity of the source table. This exchange process is a DDL operation with no actual data movement.

Figure 6–1 Overview of Partition Exchange Loading



In Figure 5–8, data from a source table *Source* is inserted into a target table consisting of four partitions (*Target_P1*, *Target_P2*, *Target_P3*, and *Target_P4*). If the new data needs to be loaded into *Target_P3*, the partition exchange operation only exchanges the names on the data objects without moving the actual data. After the exchange, the formerly labeled *Source* is renamed to *Target_P3*, and the former *Target_P3* is now labeled as *Source*. The target table still contains four partitions: *Target_P1*, *Target_P2*, *Target_P3*, and *Target_P4*. The partition

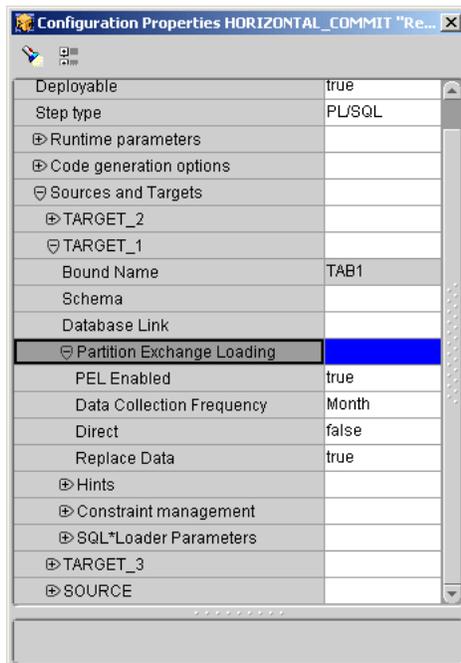
exchange operation available in Oracle 9i completes the loading process without data movement.

Configuring a Mapping for Partition Exchange Loading

To configure a mapping for partition exchange loading, complete the following steps:

1. In the Project Explorer, right-click a mapping and select **Configure**.
Warehouse Builder displays the Configuration Properties window.

Figure 6–2 PEL Configuration Properties



2. By default, Partition Exchange Loading is disabled for all mappings. Select **PEL Enabled** to use Partition Exchange Loading.
3. Use **Data Collection Frequency** to specify the amount of new data to be collected for each run of the mapping. Set this parameter to specify if you want the data collected by Year, Quarter, Month, Day, Hour, or Minute. This determines the number of partitions.
4. Select **Direct** if you want to create a temporary table to stage the collected data before performing the partition exchange. If you do not select this parameter, Warehouse Builder directly swaps the source table into the target table as a partition without creating a temporary table.
5. If you select **Replace Data**, Warehouse Builder replaces the existing data in the target partition with the newly collected data. If you do not select it, Warehouse Builder preserves the existing data in the target partition. The new data is inserted into a non-empty partition. This parameter affects the local partition and can be used to remove or swap a partition out of a target table. At the table level, you can set Truncate/Insert properties.

Direct and Indirect Partition Exchange Loading

When you use Warehouse Builder to load a target by exchanging partitions, you can load the target indirectly or directly.

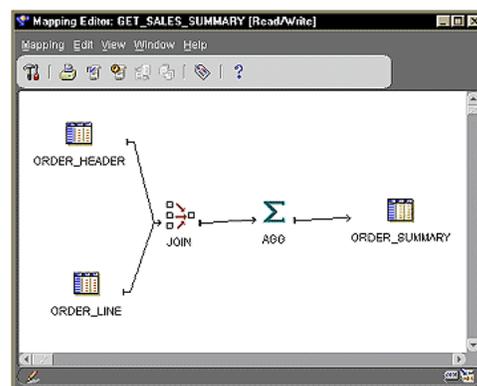
- **Indirect Partition Exchange Loading:** By default, Warehouse Builder creates and maintains a temporary table that stages the source data before initiating the partition exchange process. For example, use Indirect PEL when the mapping includes a remote source or a join of multiple sources.
- **Direct Partition Exchange Loading:** You design the source for the mapping to match the target structure. For example, use Direct PEL in a mapping to instantaneously publish fact tables that you loaded in a previously executed mapping.

Using Indirect Partition Exchange Loading

If you design a mapping using Partition Exchange Loading and it includes remote sources or a join of multiple sources, Warehouse Builder must perform source processing and stage the data before partition exchange can proceed. Therefore, configure such mappings with Direct PEL set to False. Warehouse Builder transparently creates and maintains a temporary table that stores the results from source processing. After performing the PEL, Warehouse Builder drops the table.

Figure 5–10 shows a mapping that joins two sources and performs an aggregation. If all new data loaded into the ORDER_SUMMARY table is always loaded into same partition, then you can use Indirect PEL on this mapping to improve load performance. In this case, Warehouse Builder transparently creates a temporary table after the Aggregator and before ORDER_SUMMARY.

Figure 6–3 Mapping with Multiple Sources



Warehouse Builder creates the temporary table using the same structure as the target table with the same columns, indexes, and constraints. For the fastest performance, Warehouse Builder loads the temporary table using parallel direct-path loading INSERT. After the INSERT, Warehouse Builder indexes and constrains the temporary table in parallel.

Example: Using Direct Partition Exchange Loading to Publish Fact Tables

Use Direct PEL when the source table is local and the data is of good quality. You must design the mapping such that the source and target are in the same database and have exactly the same structure. The source and target must have the same indexes and constraints, the same number of columns, and the same column types and lengths.

For example, assume that you have the same mapping from Figure 5–10 but would like greater control on when data is loaded into the target. Depending on the amount of data, it could take hours to load and you would not know precisely when the target table would be updated.

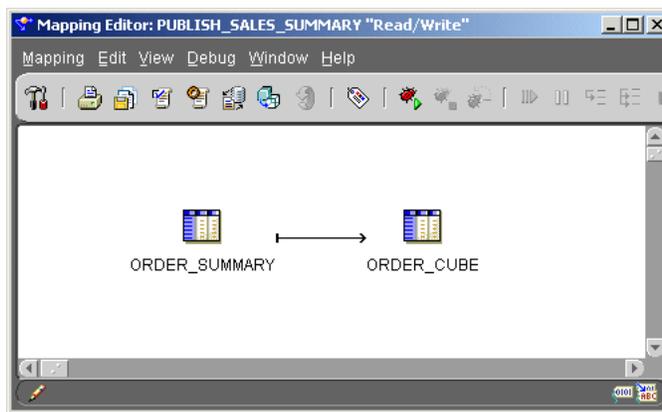
To instantly load data to a target using Direct PEL:

1. Design one mapping to join source data, if necessary, transform data, ensure data validity, and load it to a staging table. Do not configure this mapping to use PEL.

Design the staging table to exactly match the structure of the final target that you will load in a separate mapping. For example, the staging table in Figure 5–10 is ORDER_SUMMARY and should be of the same structure as the final target, ORDER_CUBE in Figure 5–11.

2. Create a second mapping that loads data from the staging table to the final target such as shown in Figure 5–11. Configure this mapping to use Direct PEL.

Figure 6–4 Publish_Sales_Summary Mapping



3. Use either the Warehouse Builder Process Flow Editor or Oracle Workflow to launch the second mapping after the completion of the first.

Using Partition Exchange Loading Effectively

You can use Partition Exchange Loading effectively for scalable loading performance if the following conditions are true:

- **Table partitioning and tablespace:** The target table must be Range partitioned by one DATE column. All partitions must be created in the same tablespace. All tables are created in the same tablespace.
- **Existing historical data:** The target table must contain a huge amount of historical data. An example use for PEL is for a click stream application where the target collects data every day from an OLTP database or Web log files. New data is transformed and loaded into the target that already contains historical data.
- **New data:** All new data must to be loaded into the same partition in a target table. For example, if the target table is partitioned by day, then the daily data should be loaded into one partition.
- **Loading Frequency:** The loading frequency should be equal to or less than the data collection frequency.
- **No global indexes:** There must be no global indexes on the target table.

Configuring Targets in a Mapping

To configure targets in a mapping for Partition Exchange Loading:

- Step 1: Create All Partitions
- Step 2: Create All Indexes Using the LOCAL Option
- Step 3: Primary/Unique Keys Use "USING INDEX" Option

Step 1: Create All Partitions

Warehouse Builder does not automatically create partitions during runtime. Before you can use Partition Exchange Loading, you must create all partitions.

For example, if you select Month as the frequency of new data collection, you need to create all the required partitions for each month of new data. Use the Data Object Editor to create partitions for a table, dimension, or cube.

Figure 6–5 shows the property inspector window for table ORDER_SUMMARY. This figure shows six partitions that have been added for this table.

To use Partition Exchange Loading, all partition names must follow a naming convention. For example, for a partition that will hold data for May 2002, the partition name must be in the format Y2002_Q2_M05.

For PEL to recognize a partition, its name must fit one of the following formats.

Ydddd

Ydddd_Qd

Ydddd_Qd_Mdd

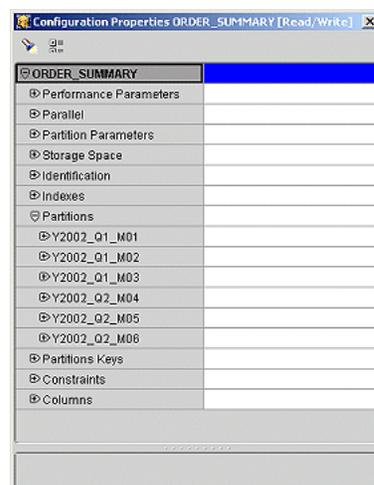
Ydddd_Qd_Mdd_Ddd

Ydddd_Qd_Mdd_Ddd_Hdd

Ydddd_Qd_Mdd_Ddd_Hdd_Mdd

Where d represents a decimal digit. All the letters must be in upper case. Lower case is not recognized.

Figure 6–5 Configuration Properties for Table ORDER_SUMMARY



If you correctly name each partition, Warehouse Builder automatically computes the Value Less Than property for each partition. Otherwise, you must manually configure Value Less Than for each partition for Warehouse Builder to generate a DDL statement. The following is an example of a DDL statement generated by Warehouse Builder:

```

. . .
PARTITION A_PARTITION_NAME
    VALUES LESS THAN (TO_DATE('01-06-2002', 'DD-MM-YYYY')),
. . .

```

Figure 5–13 shows automatically generated configuration values for the Value Less Than parameter.

Figure 6–6 Automatically Generated "Value Less Than" Setting



Step 2: Create All Indexes Using the LOCAL Option

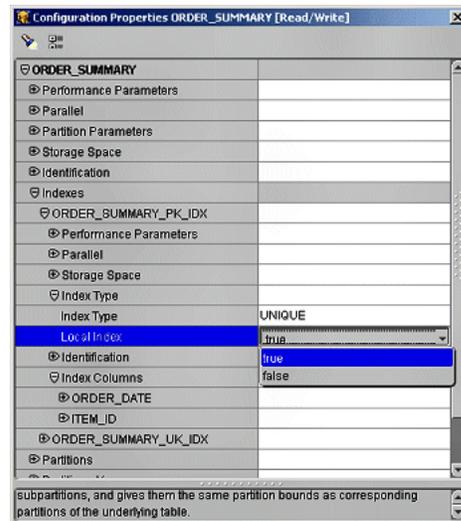
Figure 5–14 shows an index (ORDER_SUMMARY_PK_IDX) added to the ORDER_SUMMARY table. This index has two columns, ORDER_DATE and ITEM_ID.

Configure the following:

- Set the Index Type parameter to UNIQUE.
- Set the Local Index parameter to True.

Now Warehouse Builder can generate a DDL statement for a unique local index on table ORDER_SUMMARY.

Using local indexes provides the most important PEL performance benefit. Local indexes require all indexes to be partitioned in the same way as the table. When the temporary table is swapped into the target table using PEL, so are the identities of the index segments.

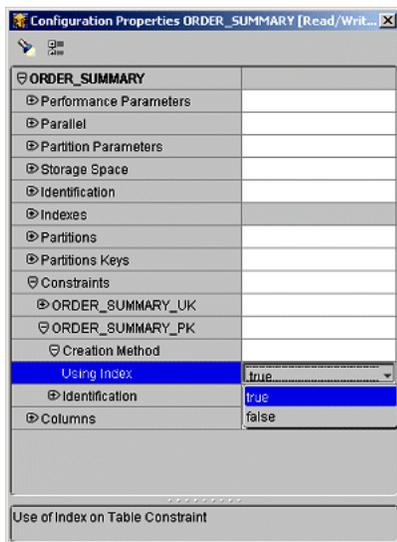
Figure 6–7 Configure an Index as a Local Index

If an index is created as a local index, the Oracle server requires that the partition key column must be the leading column of the index. In the preceding example, the partition key is ORDER_DATE and it is the leading column in the index ORDER_SUMMARY_PK_IDX.

Step 3: Primary/Unique Keys Use "USING INDEX" Option

In this step you must specify that all primary key and unique key constraints are created with the USING INDEX option. Figure 5–15 shows an example where the primary key constraint ORDER_SUMMARY_PK on the ORDER_SUMMARY table is specified with the USING INDEX option.

With the USING INDEX option, a constraint will not trigger automatic index creation when it is added to the table. The server will search existing indexes for an index with same column list as that of the constraint. Thus, each primary or unique key constraint must be backed by a user-defined unique local index. The index required by the constraint ORDER_SUMMARY_PK is ORDER_SUMMARY_PK_IDX which was created in "Step 2: Create All Indexes Using the LOCAL Option".

Figure 6–8 Specify a Constraint with USING INDEX option

Restrictions for Using Partition Exchange Loading in Warehouse Builder

These are the restrictions for using PEL in Warehouse Builder:

- Only One Date Partition Key:** Only one partition key column of DATE data type is allowed. Numeric partition keys are not supported in Warehouse Builder.
- Only Natural Calendar System:** The current PEL method supports only the natural calendar system adopted worldwide. Specific business calendar systems with user-defined fiscal and quarter endings are currently not supported.
- All Data Partitions Must Be In The Same Tablespace:** All partitions of a target (table, dimension, or cube) must be created in the same tablespace.
- All Index Partitions Must Be In The Same Tablespace:** All indexes of a target (table, dimension, or cube) must be created in the same tablespace. However, the index tablespace can be different from the data tablespace.

Deploying to Target Schemas and Executing ETL Logic

This section contains the following topics:

- [About Deployment](#)
- [About Schedules](#)
- [Process for Defining and Using Schedules](#)

About Deployment

Deployment is the process of creating physical objects in a target location from the logical objects in a Warehouse Builder repository.

Deploying a mapping or a process flow includes these steps:

- Generate the PL/SQL, SQL*Loader, or ABAP script, if necessary.
- Copy the script from the Design Center to the Control Center. Also copy SQL*Loader control files to the Control Center.
- Deploy any new connectors; that is, create the database links and database directories between locations.

After deploying a mapping or a process flow, you must explicitly start the scripts.

You can deploy only those objects for which you have the `COMPILE` privilege. By default, you have this privilege on all objects in the repository. However, the repository owner may have instituted a different security policy.

You can deploy directly from the Design Center navigation tree or using the Control Center Manager.

Note: Always maintain objects using Warehouse Builder. Do not modify the deployed, physical objects manually in SQL. Otherwise, the logical objects and the physical objects will not be synchronized, which may cause unpredictable results.

Note: Whenever you deploy an object, Warehouse Builder automatically saves all changes to all design objects to the repository. You can choose to display a warning message by selecting **Prompt for commit** on the Preferences dialog box.

What is a Control Center?

A Control Center stores detailed information about every deployment, which you can access either by object or by job, including:

- The current deployment status of each object.
- A history of all deployment attempts for each object.
- A history of all ETL start attempts for each mapping and process flow.
- A complete log of messages from all deployment jobs.

A Control Center is implemented as a schema in the same database as the target location. Each repository has a default Control Center, which was created in the schema of the repository owner. For example, the `REP_OWNER` repository owner has a schema named `REP_OWNER` that stores the metadata from both the Design Center and the default Control Center.

You can use the default Control Center to deploy to the local system, or you can create additional Control Centers for deploying to different systems. Only one Control Center is active at any time.

The Control Center Manager offers a comprehensive deployment console that enables you to view and manage all aspects of deployment. It provides access to the information stored in the active Control Center.

You can also access deployment data reports using the Repository Browser, as described in the *Warehouse Builder Online Help*.

To create a new Control Center:

1. In the Connection Explorer, right-click Control Centers and choose **New**.
The Create Control Center dialog box is displayed.
2. Complete the dialog box. Click the **Help** button for additional details.

You can also create a Control Center using the Create Configuration wizard.

To make a Control Center active:

1. Create or edit a configuration so that it uses the Control Center.
Refer to "[Creating a New Configuration](#)".
2. Activate that configuration.

Configuring the Physical Details of Deployment

Warehouse Builder separates the logical design of the objects from the physical details of the deployment. It creates this separation by storing the physical details in configuration parameters. An object called a **configuration** stores all of the configuration settings. You can create a different configuration for each deployment location, with different settings for the object parameters in each one.

Before deployment, be sure to check the configuration of the target objects, the mappings, and the modules.

For an object to be deployable:

- Its target location must be fully defined, valid, and selected for the object's module.
- Its Deployable parameter must be selected, which it is by default.

- It must validate and generate without errors.

Deployment Actions

As soon as you define a new object in the Design Center, the object is listed in the Control Center Manager. Each object has a default deployment action, which you can display. The default is set by the previous action and varies depending on the type of object. You can override the default by choosing a different deployment action in the Control Center Manager.

These are the deployment actions:

- **Create:** Creates the object in the target location. If an object with that name already exists, then an error may result.
- **Upgrade:** Modifies the object without losing data, if possible. You can undo and redo an upgrade. This action is not available for some object types, such as schedules.
- **Drop:** Deletes the object from the target location.
- **Replace:** Deletes and recreates the object. This action is quicker than Upgrade, but it deletes all data.

The Deployment Process

During the life cycle of a data system, you typically will take these steps in the deployment process:

1. Select a configuration with the object settings and the Control Center that you want to use.
2. Deploy objects to the target location. You can deploy them individually, in stages, or all at once.
3. Review the results of the deployment. If an object fails to deploy, then fix the problem and try again.
4. Start the ETL process.
5. Revise the design of target objects to accommodate user requests, changes to the source data, and so forth.
6. Set the deployment action on the modified objects to Upgrade or Replace.
7. Repeat these steps.

Note: Warehouse Builder automatically saves all changes to the repository before deployment.

About Schedules

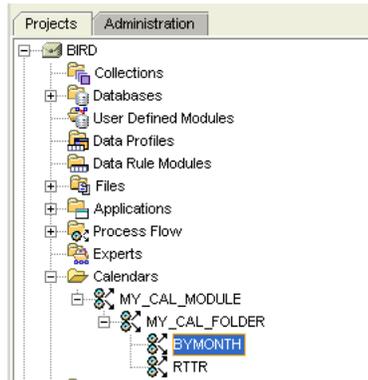
Use schedules to plan when and how often to execute operations that you designed within Warehouse Builder. You can apply schedules to mappings and process flows that you want to execute in an Oracle Database, version 10g or higher.

When you are in the development phase of using Warehouse Builder, you may not want to schedule mappings and process flows but rather start and stop them immediately from a Control Center.

You can define schedules to execute once or to execute repeatedly based on an interval you define in the user interface. For each schedule you define, Warehouse Builder generates codes that follows the iCal calendaring standards.

Schedules are defined in the context of projects and contained in schedule modules under the **Schedules** node on the Project Explorer as shown in [Figure 7-1](#).

Figure 7-1 Schedules on the Project Explorer



For every new project you create, Warehouse Builder creates a default schedule module, `MY_CAL_MODULE`. Create schedules under the default module or create a new module by right-clicking the **Schedules** node and selecting **New...**

Deploying Warehouse Builder Schedules to Oracle Workflow

To successfully deploy Warehouse Builder schedules to Oracle Workflow, ensure access to the correct version of Oracle Workflow as described in the Oracle Warehouse Builder Installation and Administration Guide.

Ensure that the target location has the 'CREATE SYNONYM' system privilege. If the Evaluation Location is specified or the deployment location references a different database instance from Control Center schema, then the deployment location must have the 'CREATE DATABASE LINK' system privilege.

Options for Scheduling Jobs in Warehouse Builder

This section describes the preferred method for scheduling Warehouse Builder ETL processes.

Process for Defining and Using Schedules

1. To create a module to contain schedules, right click on the Schedules node and select **New**.
2. To create a schedule, right click a schedule module and select **New**.

Warehouse Builder launches the Schedule Wizard.

3. On the **Name and Description** page, type a name for the schedule that is 24 characters or less.

The rules for most Warehouse Builder objects is that physical names can be 1 to 30 alphanumeric characters and business names can be 1 to 2000 alphanumeric characters.

4. Follow the instructions in the Schedule Wizard.

Use the wizard to specify values for [Start and End Dates and Times](#), [Frequency Unit](#), and [Repeat Every](#). When you complete the wizard, Warehouse Builder saves the schedule under the schedule module you selected.

5. On the Project Explorer, right click on the schedule you created with the wizard and select **Open Editor**.

Warehouse Builder displays the schedule editor. Review your selections and view the list of calculated execution times. For complex schedules, you can now enter values for the [By Clauses](#).

6. To apply a schedule to a mapping or process flow, right-click on the object in the Project Explorer and select **Configure**. In the Referred Calendar field, select the Ellipsis button to view a list of existing schedules.

For any mapping or process flow you want to schedule, the physical name must be 25 characters or less and the business name must be 1995 characters or less. This restriction enables Warehouse Builder to append to the mapping name the suffix `_job` and other internal characters required for deployment and execution.

7. Deploy the schedule.

Recall that when you deploy a mapping, for example, you also need to deploy any associated mappings and process flows and any new target data objects. Likewise, you should also deploy any associated schedules.

When properly deployed with its associated objects, the target schema executes the mapping or process flow based on the schedule you created.

Deploying Objects

Deployment is the process of creating physical objects in a target location from the metadata using your generated code. You can deploy an object from the Project Explorer or using the Control Center Manager. Warehouse Builder automatically validates and generates the object.

Deployment from the Project Explorer is restricted to the default action, which may be set to Create, Replace, Drop, or Update. To override the default action, use the Control Center Manager, which provides full control over the deployment process.

To deploy from the Project Explorer:

Select the object and click the Deploy icon on the toolbar.

Status messages appear at the bottom of the Design Center window. For notification that deployment is complete, select **Show Deployment Completion Messages** in your preferences before deploying.

To open the Control Center Manager:

1. Open a project.
2. Choose **Control Center Manager** from the Tools menu.

Starting ETL Jobs

ETL is the process of extracting data from its source location, transforming it as defined in a mapping, and loading it into target objects. When you start ETL, you submit it as a job to the Warehouse Builder job queue. The job can start immediately or at a scheduled time, if you use a scheduler such as the one in Oracle Enterprise Manager. Like deployment, you can start ETL from the Project Explorer or using the

Control Center Manager. You can also start ETL using tools outside of Warehouse Builder that execute SQL scripts.

To start ETL from the Project Explorer:

Select a mapping or a process flow, then choose **Start** from the Design menu.

Viewing the Data

After completing ETL, you can easily check any data object in Warehouse Builder to verify that the results are as you expected.

To view the data:

In the Project Explorer, right-click the object and select **Data**. The Data Viewer will open with the contents of the object.

Part III

Analyzing and Reporting on a Data Warehouse

This section discusses managing the data warehouse, and includes:

- [Chapter 8, "Analyzing Metadata Dependencies"](#)
- [Chapter 9, "SQL for Reporting and Analysis"](#)

Analyzing Metadata Dependencies

The Metadata Dependency Manager enables you to detect and resolve the impact of the changes made to the object definitions or the metadata in the Warehouse Builder repository.

This section contains the following topics:

- [Introduction to the Metadata Dependency Manager](#)
- [Usage Scenario](#)
- [What are Lineage and Impact Analysis Diagrams?](#)
- [Generating an LIA Diagram](#)
- [Modifying the Display of an LIA Diagram](#)
- [Modifying Objects in the Dependency Manager](#)

Introduction to the Metadata Dependency Manager

The Metadata Dependency Manager generates lineage and impact diagrams for any data object. A lineage diagram traces the process flows for an object back to the data source and displays all objects along that path. An impact diagram identifies all the objects that are derived from selected object.

This type of information can help you in many circumstances. For example, you can use these diagrams to:

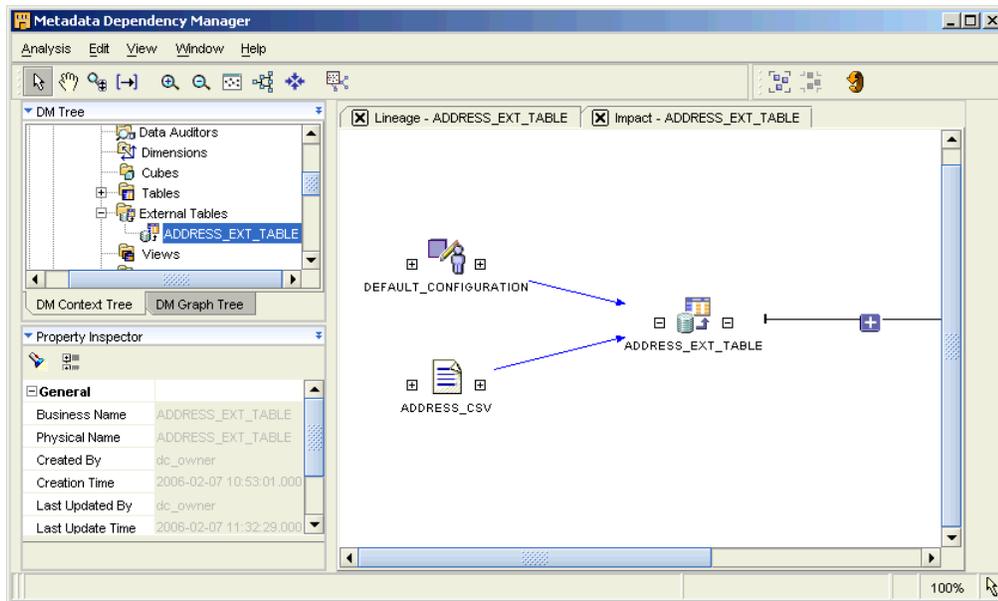
- Comply with government regulations or other audits that require tracking of financial data.
- Modify the system design because of changes in the source data.
- Modify the system design because of new requirements for reporting and analysis.
- Assess the impact of design changes in a pluggable map that is used throughout a project.

The Dependency Manager enables you to plan your project by previewing the impact of the changes or future changes. If you are planning to introduce changes to your source systems, you can use the Dependency Manager to gauge the impact of that change on your warehouse design. Or, if the change has already been introduced, then you can plan the time required to update your ETL design and rebuild your data warehouse.

[Figure 8–1](#) shows the Metadata Dependency Manager. Like other windows in Warehouse Builder, the Dependency Manager has menus, toolbars, a navigator, a property inspector, and a canvas. The canvas displays one or more diagrams.

For specific information about these components, choose **Topic** from the Help menu.

Figure 8–1 Metadata Dependency Manager



Usage Scenario

Source databases often undergo changes even after a data warehouse has been designed and built. These changes imply a corresponding change, or impact, in your Warehouse Builder design or metadata. Warehouse Builder enables you to reimport the modified source definitions into the repository. However, your original warehouse design may no longer remain valid with the re-imported definitions, and you may need to make changes to the design and fix the inconsistencies.

You need to first find out how the warehouse design is affected by the changes in source and then determine all the design objects that are dependent upon the sources must synchronize the metadata so that all the affected design objects are updated to reflect the changes in the source. After this process is complete, you can redeploy the updated design to rebuild your data warehouse and synchronize the data.

In this scenario, a company retrieves all its statistical reports from a flat file named CUSTOMERS. Over a period of time, the file definition needs to be changed to generate reports on additional parameters. The statistical analysis database has also recently migrated from Oracle 9i to Oracle Database 10g.

The designers at this company first need to synchronize the modified metadata definitions for this file in the design repository with all the design objects and mappings that are based on its data. The CUSTOMERS flat file is part of multiple mappings in the ETL design, and any changes to its definitions invalidate all them. Manually tracking all the areas impacted by the CUSTOMERS flat file and updating the definitions is a process prone to errors.

The Dependency Manager enables them to identify all of the objects that are affected by this change to the data source. They can be sure that each related design object is updated so that the ETL designs remain valid. After finishing this process, they can redeploy the affected tables and the updated ETL design objects to their data warehouse.

What are Lineage and Impact Analysis Diagrams?

Lineage and Impact Analysis (LIA) diagrams show the relationships among objects managed by Warehouse Builder. These relationships are constructed by mappings and process flows. The lineage diagram for a particular object shows its source objects, and the impact diagram shows its targets.

Lineage and impact are mirror images of each other. If Object A is part of the lineage diagram of Object B, then Object B is part of the impact diagram of Object A. When you read a diagram from left to right, you are seeing impact. When you read it from right to left, you are seeing lineage.

For example, you might have a mapping that extracts data from a file and loads it into a table by way of an external table. This is the relationship:

```
flat_file > external_table > table
```

Figure 8–2 shows a lineage diagram of an external table named ADDRESS_EXT_TABLE. ADDRESS_CSV is a flat file, and it is part of the lineage of ADDRESS_EXT_TABLE. Thus, any change to ADDRESS_CSV will impact ADDRESS_EXT_TABLE.

Figure 8–2 Lineage Analysis Diagram for ADDRESS_EXT_TABLE

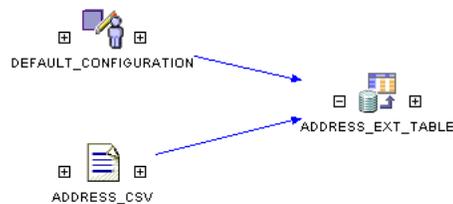
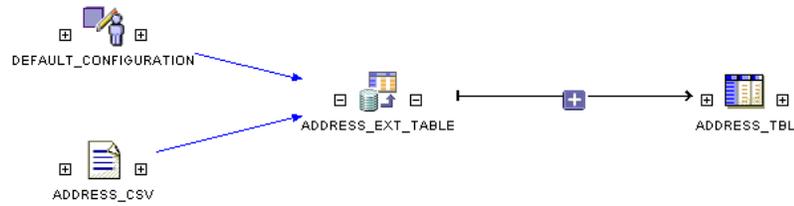


Figure 8–3 shows an impact diagram of ADDRESS_EXT_TABLE, which includes the ADDRESS_TBL. Any change to ADDRESS_EXT_TABLE will impact ADDRESS_TBL. ADDRESS_EXT_TABLE is part of the lineage of ADDRESS_TBL.

Figure 8–3 Impact Analysis Diagram for ADDRESS_EXT_TABLE



You can easily see both the lineage and the impact of an object just by clicking the plus signs (+) on either side of the object icon, as shown in Figure 8–4.

Figure 8–4 Lineage and Impact Analysis Diagram

Generating an LIA Diagram

You can generate an LIA diagram from the Project Explorer in the Design Center or by first opening the Metadata Dependency Manager.

To generate a diagram from the Design Center:

1. Expand the Project Explorer until you see the object that you want to analyze.
2. Right-click the object and choose **Lineage** or **Impact** from the popup menu.

The Metadata Dependency Manager will open with the diagram.

The Lineage and Impact commands are also available from the View menu.

To generate a diagram from the Metadata Dependency Manager:

1. From the Design Center Tools menu, choose **Metadata Dependency Manager**.
The Metadata Dependency Manager is displayed.
2. Expand the DM Context Tree until you see the object that you want to analyze.
3. Right-click the object and choose **Show Lineage** or **Show Impact** from the popup menu.

The diagram will be displayed on the canvas.

Modifying the Display of an LIA Diagram

Your initial selection of an object and a diagram type simply determine the initial starting point and the direction that the diagram branches from that object. You can modify an LIA diagram in the following ways:

- Drag-and-drop another object onto the diagram.
- Click the plus (+) and minus (-) signs next to an object icon to expand or collapse a branch of the diagram.
- Use the grouping tool to collapse a section of the diagram into a single icon, as described in ["Using Groups in an LIA Diagram"](#) on page 8-5.
- Double-click an object to display its attributes, as described in ["Displaying an Object's Attributes"](#) on page 8-5.

Using Groups in an LIA Diagram

Groups enable you to organize the objects in a complex diagram so that they are easier to locate and edit. By reducing the number of objects in a diagram, you can more easily focus on the objects currently of interest.

To create a group:

1. Select a group of objects by dragging and dropping a box around them.
2. Click the Group Selected Objects tool.

The Group Selected Data Objects dialog box is displayed.

3. Enter a name for the group.

The selected objects are collapsed into a single folder icon.

To display the individual objects in a group, double-click the folder icon. You can work on these objects in the same way as ungrouped objects.

To ungroup the objects, select the group and click the Ungroup Selected Object tool.

Displaying an Object's Attributes

You can expand an object icon in a diagram so that you can examine its attributes. [Figure 8–5](#) shows two expanded icons whose column attributes are connected by a mapping.

To expand an icon, double-click it. To reduce it to an icon, click the down arrow in the upper right corner.

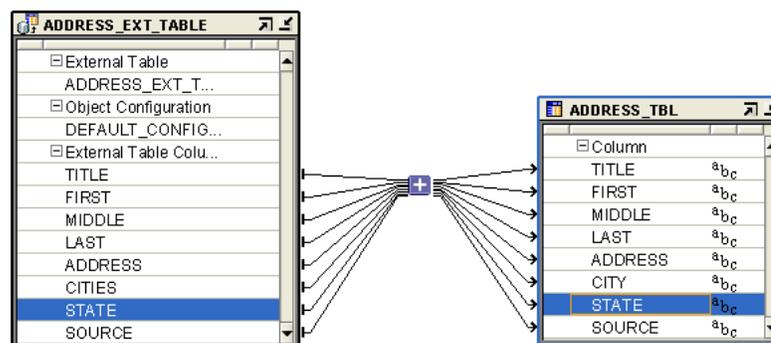
To generate an LIA diagram for an attribute:

1. Generate an LIA diagram for an object.
2. Double-click the icons to display their attributes.
3. Right-click an attribute and select **Show Lineage** or **Show Impact**.

The attributes along the lineage or impact path for the selected attribute are highlighted in a different color.

You may use this detailed information for auditing or when planning to propagate changes.

Figure 8–5 Expanded Icons in an LIA Diagram



Modifying Objects in the Dependency Manager

The LIA diagrams identify all of the objects that may be invalidated by a change to one or more objects. With this knowledge, you can examine the affected objects and modify them as necessary.

You can make these changes within the Dependency Manager, using either of these methods:

- **Object Editor:** Provides extensive editing capabilities by opening the standard editor for the object.
- **Propagate Changes tool:** Provides basic editing capabilities such as changing the name, data type, or precision for a single column or attribute.

To open the object editor:

Right-click the object icon in a diagram, and choose **Open Editor** from the popup menu.

To use the Propagate Changes tool:

1. Double-click the object icon in a diagram.
The icon will expand to show its attributes.
2. Right-click the attribute that you want to change, and choose **Propagate Change** from the popup menu.
The Propagate Change dialog box is displayed.
3. Make your changes to the attributes in the New Value column and select the Propagate box.
4. Click **OK**.

SQL for Reporting and Analysis

This section discusses how to produce effective business reports derived from business queries, and includes the following sections:

- [Use of SQL Analytic Capabilities to Answer Business Queries](#)
- [Use of Partition Outer Join to Handle Sparse Data](#)
- [Use of the WITH Clause to Simplify Business Queries](#)

Use of SQL Analytic Capabilities to Answer Business Queries

Oracle Database has enhanced SQL's analytical processing capabilities by introducing a family of aggregate and analytic SQL functions. These functions enable you to calculate ranking, percentiles, and moving averages, and allow you to answer queries such as the following:

- What are the top 10 products sold by country?
- What is the weekly moving average for products in stock?
- What percentage of total sales occurs during the fourth quarter?
- How much higher is the average discount in the fourth quarter than the discount for yearly average?
- What would be the profitability ranking of existing oil refineries if 20 percent of the refineries in a country were closed?

Aggregate functions are a fundamental part of data warehousing because they enable you to derive different types of totals, and then use these totals for additional calculations. To improve aggregate performance in your data warehouse, Oracle Database provides several extensions to the `GROUP BY` clause. The `CUBE`, `ROLLUP`, `GROUPING`, and `GROUPING SETS` functions make querying and reporting easier and faster. The `ROLLUP` function calculates aggregations such as `SUM`, `COUNT`, `MAX`, `MIN`, and `AVG` at increasing levels of aggregation, from the most individual detailed level up to a grand summary total. The `CUBE` function is an extension similar to `ROLLUP`, enabling a single statement to calculate all possible combinations of aggregations.

Analytic functions compute an aggregate value based on a group of rows. These functions differ from aggregate functions in that they return multiple rows for each group. This group of rows is called a **window**. This window enables calculations such as moving average or cumulative total. For each row, a window of rows is defined. This window determines the range of rows used to perform the calculations for the current row. Window sizes can be based on either a logical interval such as time, or a physical number of rows. Some functions are used only with windows and are often referred to as window functions.

To enhance performance, aggregate and analytic functions can each perform in parallel: multiple processes can simultaneously execute all of these functions. These capabilities make calculations, analysis, and reporting easier and more efficient, thereby enhancing data warehouse performance, scalability, and simplicity.

You can take advantage of the advanced SQL and PL/SQL capabilities Oracle Database offers to translate business queries into SQL. This section discusses these advanced capabilities, and includes the following topics:

- [How to Add Totals to Reports Using the ROLLUP Function](#)
- [How to Separate Totals at Different Levels Using the CUBE Function](#)
- [How to Add Subtotals Using the GROUPING Function](#)
- [How to Combine Aggregates Using the GROUPING SETS Function](#)
- [How to Calculate Rankings Using the RANK Function](#)
- [How to Calculate Relative Contributions to a Total](#)
- [How to Perform Interrow Calculations with Window Functions](#)
- [How to Calculate a Moving Average Using a Window Function](#)

How to Add Totals to Reports Using the ROLLUP Function

The ROLLUP function enables a SELECT statement to calculate multiple levels of subtotals across a specified group of dimensions, as well as a grand total. The ROLLUP function is a simple extension to the GROUP BY clause, so its syntax is extremely easy to use. The ROLLUP function is highly efficient, adding minimal overhead to a query.

The action of the ROLLUP function is straightforward: it creates subtotals that roll up from the most detailed level to a grand total, following a grouping list specified in the ROLLUP function. ROLLUP takes as its argument an ordered list of grouping columns. First, it calculates the standard aggregate values specified in the GROUP BY clause. Then, it creates progressively higher-level subtotals, moving from right to left through the list of grouping columns. Finally, it creates a grand total.

When to Use the ROLLUP Function

When your tasks involve subtotals, particularly when the subtotals are along a hierarchical dimension such as time or geography, use the ROLLUP function. Also, a ROLLUP function can simplify and speed up the maintenance of materialized views.

Example: Using the ROLLUP Function

A common request when preparing business reports is to find quarterly sales revenue across different product categories, in order by the amount of revenue. The following query achieves this, and is used for the starting point for building more complicated queries later:

```
SELECT t.calendar_quarter_desc quarter
, p.prod_category category
, TO_CHAR(SUM(s.amount_sold), 'L999G999G990D00') revenue
FROM times t
, products p
, sales s
WHERE t.time_id = s.time_id
AND p.prod_id = s.prod_id
AND s.time_id BETWEEN TO_DATE('01-JAN-2001', 'dd-MON-yyyy')
AND TO_DATE('31-DEC-2001', 'dd-MON-yyyy')
GROUP BY t.calendar_quarter_desc, p.prod_category
```

```
ORDER BY t.calendar_quarter_desc
, SUM(s.amount_sold);
```

QUARTER	CATEGORY	REVENUE
-----	-----	-----
2001-01	Software/Other	\$860,819.81
2001-01	Electronics	\$1,239,287.71
2001-01	Hardware	\$1,301,343.45
2001-01	Photo	\$1,370,706.38
2001-01	Peripherals and Accessories	\$1,774,940.09
2001-02	Software/Other	\$872,157.38
2001-02	Electronics	\$1,144,187.90
2001-02	Hardware	\$1,557,059.59
2001-02	Photo	\$1,563,475.51
2001-02	Peripherals and Accessories	\$1,785,588.01
2001-03	Software/Other	\$877,630.85
2001-03	Electronics	\$1,017,536.82
2001-03	Photo	\$1,607,315.63
2001-03	Hardware	\$1,651,454.29
2001-03	Peripherals and Accessories	\$2,042,061.04
2001-04	Software/Other	\$943,296.36
2001-04	Hardware	\$1,174,512.68
2001-04	Electronics	\$1,303,838.52
2001-04	Photo	\$1,792,131.39
2001-04	Peripherals and Accessories	\$2,257,118.57

This query is useful, but you may want to see the totals for different categories in the same report. The following example illustrates how you can use the ROLLUP function to add the totals to the original query:

```
SELECT t.calendar_quarter_desc quarter
, p.prod_category category
, TO_CHAR(SUM(s.amount_sold), 'L999G999G990D00') revenue
FROM times t
, products p
, sales s
WHERE t.time_id = s.time_id
AND p.prod_id = s.prod_id
AND s.time_id BETWEEN TO_DATE('01-JAN-2001', 'dd-MON-yyyy')
AND TO_DATE('31-DEC-2001', 'dd-MON-yyyy')
GROUP BY ROLLUP(t.calendar_quarter_desc, p.prod_category)
ORDER BY t.calendar_quarter_desc
, SUM(s.amount_sold);
```

QUARTER	CATEGORY	REVENUE
-----	-----	-----
2001-01	Software/Other	\$860,819.81
2001-01	Electronics	\$1,239,287.71
2001-01	Hardware	\$1,301,343.45
2001-01	Photo	\$1,370,706.38
2001-01	Peripherals and Accessories	\$1,774,940.09
2001-01		\$6,547,097.44
2001-02	Software/Other	\$872,157.38
2001-02	Electronics	\$1,144,187.90
2001-02	Hardware	\$1,557,059.59
2001-02	Photo	\$1,563,475.51
2001-02	Peripherals and Accessories	\$1,785,588.01
2001-02		\$6,922,468.39
2001-03	Software/Other	\$877,630.85
2001-03	Electronics	\$1,017,536.82

2001-03	Photo	\$1,607,315.63
2001-03	Hardware	\$1,651,454.29
2001-03	Peripherals and Accessories	\$2,042,061.04
2001-03		\$7,195,998.63
2001-04	Software/Other	\$943,296.36
2001-04	Hardware	\$1,174,512.68
2001-04	Electronics	\$1,303,838.52
2001-04	Photo	\$1,792,131.39
2001-04	Peripherals and Accessories	\$2,257,118.57
2001-04		\$7,470,897.52
		\$28,136,461.98

How to Separate Totals at Different Levels Using the CUBE Function

The `CUBE` function takes a specified set of grouping columns and creates subtotals for all of the possible combinations. In terms of multidimensional analysis, the `CUBE` function generates all the subtotals that can be calculated for a data cube with the specified dimensions. If you have specified `CUBE(time, region, department)`, the result set will include all the values that would be included in an equivalent `ROLLUP` function plus additional combinations.

When to Use the CUBE Function

Consider using the `CUBE` function in any situation requiring **cross-tabular reports**. The data needed for cross-tabular reports can be generated with a single `SELECT` statement using the `CUBE` function. Like `ROLLUP`, the `CUBE` function can be helpful in generating materialized views. Note that population of materialized views is even faster if the query containing a `CUBE` function executes in parallel.

Example: Using the CUBE Function

You may want to get not only quarterly totals, but also totals for the different product categories for the selected period. The `CUBE` function enables this calculation, as shown in the following example.

To use the CUBE function:

```
SELECT t.calendar_quarter_desc quarter
, p.prod_category category
, TO_CHAR(SUM(s.amount_sold), 'L999G999G990D00') revenue
FROM times t
, products p
, sales s
WHERE t.time_id = s.time_id
AND p.prod_id = s.prod_id
AND s.time_id BETWEEN TO_DATE('01-JAN-2001', 'dd-MON-yyyy')
AND TO_DATE('31-DEC-2001', 'dd-MON-yyyy')
GROUP BY CUBE(t.calendar_quarter_desc, p.prod_category)
ORDER BY t.calendar_quarter_desc
, SUM(s.amount_sold);
```

QUARTER	CATEGORY	REVENUE
-----	-----	-----
2001-01	Software/Other	\$860,819.81
2001-01	Electronics	\$1,239,287.71
2001-01	Hardware	\$1,301,343.45
2001-01	Photo	\$1,370,706.38
2001-01	Peripherals and Accessories	\$1,774,940.09
2001-01		\$6,547,097.44
2001-02	Software/Other	\$872,157.38

2001-02	Electronics	\$1,144,187.90
2001-02	Hardware	\$1,557,059.59
2001-02	Photo	\$1,563,475.51
2001-02	Peripherals and Accessories	\$1,785,588.01
2001-02		\$6,922,468.39
2001-03	Software/Other	\$877,630.85
2001-03	Electronics	\$1,017,536.82
2001-03	Photo	\$1,607,315.63
2001-03	Hardware	\$1,651,454.29
2001-03	Peripherals and Accessories	\$2,042,061.04
2001-03		\$7,195,998.63
2001-04	Software/Other	\$943,296.36
2001-04	Hardware	\$1,174,512.68
2001-04	Electronics	\$1,303,838.52
2001-04	Photo	\$1,792,131.39
2001-04	Peripherals and Accessories	\$2,257,118.57
2001-04		\$7,470,897.52
	Software/Other	\$3,553,904.40
	Electronics	\$4,704,850.95
	Hardware	\$5,684,370.01
	Photo	\$6,333,628.91
	Peripherals and Accessories	\$7,859,707.71
		\$28,136,461.98

How to Add Subtotals Using the GROUPING Function

Two challenges arise with the use of the ROLLUP and CUBE functions. How can you programmatically determine which result set rows are subtotals, and how do you find the exact level of aggregation for a given subtotal? You often need to use subtotals in calculations such as percent-of-totals, so you need an easy way to determine which rows are the subtotals. What happens if query results contain both stored NULL values and null values created by a ROLLUP or CUBE function? How can you differentiate between the two?

The GROUPING function handles this problem. Using a single column as its argument, the GROUPING function returns 1 when it encounters a null value created by a ROLLUP or CUBE function. That is, if the null value indicates the row is a subtotal, GROUPING returns a value of 1. Any other type of value, including a stored NULL value, returns a value of 0.

When to Use the GROUPING Function

When you need to handle NULL values or null values created by a ROLLUP or CUBE operation, use the GROUPING function. One reason you may want to work with null values is to put a description in null fields, for example, text describing that a number represents a total.

Example: Using the GROUPING Function

You might want more descriptive columns in your report because it is not always clear when a value represents a total. The GROUPING function enables you to insert labels showing totals in the results of the query as shown in the following example.

To use the GROUPING function:

```
SELECT DECODE(GROUPING(t.calendar_quarter_desc)
             , 0, t.calendar_quarter_desc
             , 1, 'TOTAL'
             ) quarter
, DECODE(GROUPING(p.prod_category)
```

```

        , 0, p.prod_category
        , 1, 'TOTAL'
    ) category
, TO_CHAR(SUM(s.amount_sold), 'L999G999G990D00') revenue
FROM times t
, products p
, sales s
WHERE t.time_id = s.time_id
AND p.prod_id = s.prod_id
AND s.time_id BETWEEN TO_DATE('01-JAN-2001', 'dd-MON-yyyy')
AND TO_DATE('31-DEC-2001', 'dd-MON-yyyy')
GROUP BY CUBE(t.calendar_quarter_desc, p.prod_category)
ORDER BY t.calendar_quarter_desc
, SUM(s.amount_sold);

```

QUARTER	CATEGORY	REVENUE
2001-01	Software/Other	\$860,819.81
2001-01	Electronics	\$1,239,287.71
2001-01	Hardware	\$1,301,343.45
2001-01	Photo	\$1,370,706.38
2001-01	Peripherals and Accessories	\$1,774,940.09
2001-01	TOTAL	\$6,547,097.44
2001-02	Software/Other	\$872,157.38
2001-02	Electronics	\$1,144,187.90
2001-02	Hardware	\$1,557,059.59
2001-02	Photo	\$1,563,475.51
2001-02	Peripherals and Accessories	\$1,785,588.01
2001-02	TOTAL	\$6,922,468.39
2001-03	Software/Other	\$877,630.85
2001-03	Electronics	\$1,017,536.82
2001-03	Photo	\$1,607,315.63
2001-03	Hardware	\$1,651,454.29
2001-03	Peripherals and Accessories	\$2,042,061.04
2001-03	TOTAL	\$7,195,998.63
2001-04	Software/Other	\$943,296.36
2001-04	Hardware	\$1,174,512.68
2001-04	Electronics	\$1,303,838.52
2001-04	Photo	\$1,792,131.39
2001-04	Peripherals and Accessories	\$2,257,118.57
2001-04	TOTAL	\$7,470,897.52
TOTAL	Software/Other	\$3,553,904.40
TOTAL	Electronics	\$4,704,850.95
TOTAL	Hardware	\$5,684,370.01
TOTAL	Photo	\$6,333,628.91
TOTAL	Peripherals and Accessories	\$7,859,707.71
TOTAL	TOTAL	\$28,136,461.98

How to Combine Aggregates Using the GROUPING SETS Function

You can selectively specify the set of groups that you want to create using a GROUPING SETS function within a GROUP BY clause. This allows precise specification across multiple dimensions without computing the whole data cube. In other words, not all dimension totals are needed.

When to Use the GROUPING SETS Function

When you want particular subtotals in a data cube, but not all that are possible, use the GROUPING SETS function.

Example: Using the GROUPING SETS Function

You may want to see the total sales numbers based on sales channel. Instead of adding a separate query to retrieve the totals per channel class, you can use the GROUPING SETS function as illustrated in the following example.

To use the GROUPING SETS function:

```
SELECT DECODE(GROUPING(t.calendar_quarter_desc)
              , 0, t.calendar_quarter_desc
              , 1, 'TOTAL'
              ) quarter
, DECODE(GROUPING(c.channel_class)
         , 0, c.channel_class
         , 1, '--all--'
         ) channel
, DECODE(GROUPING(p.prod_category)
         , 0, p.prod_category
         , 1, 'TOTAL'
         ) category
, TO_CHAR(SUM(s.amount_sold), 'L999G999G990D00') revenue
FROM times t
, products p
, channels c
, sales s
WHERE t.time_id = s.time_id
AND p.prod_id = s.prod_id
AND c.channel_id = s.channel_id
AND s.time_id BETWEEN TO_DATE('01-JAN-2001', 'dd-MON-yyyy')
AND TO_DATE('31-DEC-2001', 'dd-MON-yyyy')
GROUP BY GROUPING SETS(c.channel_class,
                        CUBE(t.calendar_quarter_desc, p.prod_category))
ORDER BY t.calendar_quarter_desc
, SUM(s.amount_sold);
```

QUARTER	CHANNEL	CATEGORY	REVENUE
2001-01	--all--	Software/Other	\$860,819.81
2001-01	--all--	Electronics	\$1,239,287.71
2001-01	--all--	Hardware	\$1,301,343.45
2001-01	--all--	Photo	\$1,370,706.38
2001-01	--all--	Peripherals and Accessories	\$1,774,940.09
2001-01	--all--	TOTAL	\$6,547,097.44
2001-02	--all--	Software/Other	\$872,157.38
2001-02	--all--	Electronics	\$1,144,187.90
2001-02	--all--	Hardware	\$1,557,059.59
2001-02	--all--	Photo	\$1,563,475.51
2001-02	--all--	Peripherals and Accessories	\$1,785,588.01
2001-02	--all--	TOTAL	\$6,922,468.39
2001-03	--all--	Software/Other	\$877,630.85
2001-03	--all--	Electronics	\$1,017,536.82
2001-03	--all--	Photo	\$1,607,315.63
2001-03	--all--	Hardware	\$1,651,454.29
2001-03	--all--	Peripherals and Accessories	\$2,042,061.04
2001-03	--all--	TOTAL	\$7,195,998.63
2001-04	--all--	Software/Other	\$943,296.36
2001-04	--all--	Hardware	\$1,174,512.68
2001-04	--all--	Electronics	\$1,303,838.52
2001-04	--all--	Photo	\$1,792,131.39
2001-04	--all--	Peripherals and Accessories	\$2,257,118.57
2001-04	--all--	TOTAL	\$7,470,897.52

TOTAL	--all--	Software/Other	\$3,553,904.40
TOTAL	--all--	Electronics	\$4,704,850.95
TOTAL	--all--	Hardware	\$5,684,370.01
TOTAL	--all--	Photo	\$6,333,628.91
TOTAL	Indirect	TOTAL	\$6,709,496.66
TOTAL	--all--	Peripherals and Accessories	\$7,859,707.71
TOTAL	Others	TOTAL	\$8,038,529.96
TOTAL	Direct	TOTAL	\$13,388,435.36
TOTAL	--all--	TOTAL	\$28,136,461.98

How to Calculate Rankings Using the RANK Function

Business information processing requires advanced calculations, including complex ranking, subtotals, moving averages, and lead/lag comparisons. These aggregation and analysis tasks are essential in creating business intelligence queries, and are accomplished by the use of window functions.

When to Use the RANK Function

When you want to perform complex queries and analyze the query results, use the RANK function.

Example: Using the RANK Function

Users would like to see an additional column that shows the rank of any revenue number within the quarter. The following example illustrates using the RANK function to achieve this.

To use the RANK function:

```
SELECT DECODE(GROUPING(t.calendar_quarter_desc)
              , 0, t.calendar_quarter_desc
              , 1, 'TOTAL'
              ) quarter
, DECODE(GROUPING(t.calendar_quarter_desc) + GROUPING(p.prod_category)
          , 0, RANK() OVER (PARTITION BY t.calendar_quarter_desc
                          ORDER BY SUM(s.amount_sold))
          , 1, null
          ) ranking
, DECODE(GROUPING(c.channel_class)
          , 0, c.channel_class
          , 1, '--all--'
          ) channel
, DECODE(GROUPING(p.prod_category)
          , 0, p.prod_category
          , 1, 'TOTAL'
          ) category
, TO_CHAR(SUM(s.amount_sold), 'L999G999G990D00') revenue
FROM times t
, products p
, channels c
, sales s
WHERE t.time_id = s.time_id
AND p.prod_id = s.prod_id
AND c.channel_id = s.channel_id
AND s.time_id BETWEEN TO_DATE('01-JAN-2001', 'dd-MON-yyyy')
AND TO_DATE('31-DEC-2001', 'dd-MON-yyyy')
GROUP BY GROUPING SETS(c.channel_class,
                       CUBE(t.calendar_quarter_desc, p.prod_category))
ORDER BY t.calendar_quarter_desc
```

```
, SUM(s.amount_sold);
```

QUARTER	RANKING	CHANNEL	CATEGORY	REVENUE
2001-01	1	--all--	Software/Other	\$860,819.81
2001-01	2	--all--	Electronics	\$1,239,287.71
2001-01	3	--all--	Hardware	\$1,301,343.45
2001-01	4	--all--	Photo	\$1,370,706.38
2001-01	5	--all--	Peripherals and Accessories	\$1,774,940.09
2001-01		--all--	TOTAL	\$6,547,097.44
2001-02	1	--all--	Software/Other	\$872,157.38
2001-02	2	--all--	Electronics	\$1,144,187.90
2001-02	3	--all--	Hardware	\$1,557,059.59
2001-02	4	--all--	Photo	\$1,563,475.51
2001-02	5	--all--	Peripherals and Accessories	\$1,785,588.01
2001-02		--all--	TOTAL	\$6,922,468.39
2001-03	1	--all--	Software/Other	\$877,630.85
2001-03	2	--all--	Electronics	\$1,017,536.82
2001-03	3	--all--	Photo	\$1,607,315.63
2001-03	4	--all--	Hardware	\$1,651,454.29
2001-03	5	--all--	Peripherals and Accessories	\$2,042,061.04
2001-03		--all--	TOTAL	\$7,195,998.63
2001-04	1	--all--	Software/Other	\$943,296.36
2001-04	2	--all--	Hardware	\$1,174,512.68
2001-04	3	--all--	Electronics	\$1,303,838.52
2001-04	4	--all--	Photo	\$1,792,131.39
2001-04	5	--all--	Peripherals and Accessories	\$2,257,118.57
2001-04		--all--	TOTAL	\$7,470,897.52
TOTAL		--all--	Software/Other	\$3,553,904.40
TOTAL		--all--	Electronics	\$4,704,850.95
TOTAL		--all--	Hardware	\$5,684,370.01
TOTAL		--all--	Photo	\$6,333,628.91
TOTAL		Indirect	TOTAL	\$6,709,496.66
TOTAL		--all--	Peripherals and Accessories	\$7,859,707.71
TOTAL		Others	TOTAL	\$8,038,529.96
TOTAL		Direct	TOTAL	\$13,388,435.36
TOTAL		--all--	TOTAL	\$28,136,461.98

In this example, the PARTITION BY clause defines the boundaries for the RANK function.

How to Calculate Relative Contributions to a Total

A common business intelligence request is to calculate the contribution of every product category to the total revenue based on a given time period.

Example: Calculating Relative Contributions to a Total

You want to get the differences for revenue numbers on a quarter-by-quarter basis. As illustrated in the following example, you can use a window function with a PARTITION BY product category to achieve this.

To calculate relative contributions to a total:

```
SELECT DECODE(GROUPING(t.calendar_quarter_desc)
             , 0, t.calendar_quarter_desc
             , 1, 'TOTAL'
             ) quarter
, DECODE(GROUPING(t.calendar_quarter_desc) + GROUPING(p.prod_category)
        , 0, RANK() OVER (PARTITION BY t.calendar_quarter_desc
```

```

ORDER BY SUM(s.amount_sold)
      , 1, null
      ) RANKING
, DECODE(GROUPING(c.channel_class)
      , 0, c.channel_class
      , 1 , '--all--'
      ) channel
, DECODE(GROUPING(p.prod_category)
      , 0, p.prod_category
      , 1, 'TOTAL'
      ) category
, TO_CHAR(SUM(s.amount_sold), 'L999G999G990D00') revenue
, TO_CHAR(100 * RATIO_TO_REPORT(SUM(s.amount_sold))
OVER (PARTITION BY (TO_CHAR(GROUPING(p.prod_category) ||
t.calendar_quarter_desc)), '990D0') percent
FROM times t
, products p
, channels c
, sales s
WHERE t.time_id = s.time_id
AND p.prod_id = s.prod_id
AND c.channel_id = s.channel_id
AND s.time_id BETWEEN TO_DATE('01-JAN-2001', 'dd-MON-yyyy')
AND TO_DATE('31-DEC-2001', 'dd-MON-yyyy')
GROUP BY GROUPING SETS(c.channel_class,
      CUBE(t.calendar_quarter_desc, p.prod_category))
ORDER BY t.calendar_quarter_desc
, SUM(s.amount_sold);

```

QUARTER	RANKING	CHANNEL	CATEGORY	REVENUE	PERC
2001-01	1	--all--	Software/Other	\$860,819.81	13.1
2001-01	2	--all--	Electronics	\$1,239,287.71	18.9
2001-01	3	--all--	Hardware	\$1,301,343.45	19.9
2001-01	4	--all--	Photo	\$1,370,706.38	20.9
2001-01	5	--all--	Peripherals	\$1,774,940.09	27.1
2001-01		--all--	TOTAL	\$6,547,097.44	100.0
2001-02	1	--all--	Software/Other	\$872,157.38	12.6
2001-02	2	--all--	Electronics	\$1,144,187.90	16.5
2001-02	3	--all--	Hardware	\$1,557,059.59	22.5
2001-02	4	--all--	Photo	\$1,563,475.51	22.6
2001-02	5	--all--	Peripherals	\$1,785,588.01	25.8
2001-02		--all--	TOTAL	\$6,922,468.39	100.0
2001-03	1	--all--	Software/Other	\$877,630.85	12.2
2001-03	2	--all--	Electronics	\$1,017,536.82	14.1
2001-03	3	--all--	Photo	\$1,607,315.63	22.3
2001-03	4	--all--	Hardware	\$1,651,454.29	22.9
2001-03	5	--all--	Peripherals	\$2,042,061.04	28.4
2001-03		--all--	TOTAL	\$7,195,998.63	100.0
2001-04	1	--all--	Software/Other	\$943,296.36	12.6
2001-04	2	--all--	Hardware	\$1,174,512.68	15.7
2001-04	3	--all--	Electronics	\$1,303,838.52	17.5
2001-04	4	--all--	Photo	\$1,792,131.39	24.0
2001-04	5	--all--	Peripherals	\$2,257,118.57	30.2
2001-04		--all--	TOTAL	\$7,470,897.52	100.0
TOTAL		--all--	Software/Other	\$3,553,904.40	12.6
TOTAL		--all--	Electronics	\$4,704,850.95	16.7
TOTAL		--all--	Hardware	\$5,684,370.01	20.2
TOTAL		--all--	Photo	\$6,333,628.91	22.5
TOTAL		Indirect	TOTAL	\$6,709,496.66	11.9

TOTAL	--all--	Peripherals	\$7,859,707.71	27.9
TOTAL		Others TOTAL	\$8,038,529.96	14.3
TOTAL		Direct TOTAL	\$13,388,435.36	23.8
TOTAL	--all--	TOTAL	\$28,136,461.98	50.0

"Peripherals" was used instead of "Peripherals and Accessories" to save space.

How to Perform Interrow Calculations with Window Functions

A common business intelligence question is how a particular result relates to another result. To do this in a single query, you can use window functions and perform interrow calculations in a single statement.

Example: Performing Interrow Calculations

You may want to know the contribution of every product category to the total revenue for each quarter. You can use the window function `RATIO_TO_REPORT` to achieve this result, as illustrated in the following example. Note that you must use concatenation with `GROUPING(p.prod_category)` to preclude the total from the `RATIO_TO_REPORT` per quarter.

To perform interrow calculations:

```
SELECT DECODE(GROUPING(t.calendar_quarter_desc)
              , 0, t.calendar_quarter_desc
              , 1, 'TOTAL'
              ) quarter
, DECODE(GROUPING(t.calendar_quarter_desc) + GROUPING(p.prod_category)
        , 0, RANK() OVER (PARTITION BY t.calendar_quarter_desc
                        ORDER BY SUM(s.amount_sold)
                        , 1, null
                        ) RANKING
        , DECODE(GROUPING(c.channel_class)
                , 0, c.channel_class
                , 1, '--all--'
                ) channel
        , DECODE(GROUPING(p.prod_category)
                , 0, p.prod_category
                , 1, 'TOTAL'
                ) category
        , TO_CHAR(SUM(s.amount_sold), 'L999G999G990D00') revenue
        , TO_CHAR(100 * RATIO_TO_REPORT(SUM(s.amount_sold))
                  OVER (PARTITION BY (TO_CHAR(GROUPING(p.prod_category) ||
t.calendar_quarter_desc)), '990D0') percent
        , DECODE(GROUPING(t.calendar_quarter_desc) + GROUPING(p.prod_category)
                , 0, TO_CHAR(SUM(s.amount_sold) - LAG(SUM(s.amount_sold), 1)
                            OVER (PARTITION BY p.prod_category
                                ORDER BY t.calendar_quarter_desc), 'L999G990D00')
                , 1, null
                ) q_q_diff
FROM times t
, products p
, channels c
, sales s
WHERE t.time_id = s.time_id
AND p.prod_id = s.prod_id
AND c.channel_id = s.channel_id
AND s.time_id BETWEEN TO_DATE('01-JAN-2001', 'dd-MON-yyyy')
AND TO_DATE('31-DEC-2001', 'dd-MON-yyyy')
GROUP BY GROUPING SETS(c.channel_class,
```

```

        CUBE(t.calendar_quarter_desc, p.prod_category))
ORDER BY t.calendar_quarter_desc
, SUM(s.amount_sold);

```

QUARTER	RANKING	CHANNEL	CATEGORY	REVENUE	PERC	Q_Q_DIFF
2001-01	1	--all--	Software/Other	\$860,819.81	13.1	
2001-01	2	--all--	Electronics	\$1,239,287.71	18.9	
2001-01	3	--all--	Hardware	\$1,301,343.45	19.9	
2001-01	4	--all--	Photo	\$1,370,706.38	20.9	
2001-01	5	--all--	Peripherals	\$1,774,940.09	27.1	
2001-01		--all--	TOTAL	\$6,547,097.44	100.0	
2001-02	1	--all--	Software/Other	\$872,157.38	12.6	\$11,337.57
2001-02	2	--all--	Electronics	\$1,144,187.90	16.5	-\$95,099.81
2001-02	3	--all--	Hardware	\$1,557,059.59	22.5	\$255,716.14
2001-02	4	--all--	Photo	\$1,563,475.51	22.6	\$192,769.13
2001-02	5	--all--	Peripherals	\$1,785,588.01	25.8	\$10,647.92
2001-02		--all--	TOTAL	\$6,922,468.39	100.0	
2001-03	1	--all--	Software/Other	\$877,630.85	12.2	\$5,473.47
2001-03	2	--all--	Electronics	\$1,017,536.82	14.1	-\$126,651.08
2001-03	3	--all--	Photo	\$1,607,315.63	22.3	\$43,840.12
2001-03	4	--all--	Hardware	\$1,651,454.29	22.9	\$94,394.70
2001-03	5	--all--	Peripherals	\$2,042,061.04	28.4	\$256,473.03
2001-03		--all--	TOTAL	\$7,195,998.63	100.0	
2001-04	1	--all--	Software/Other	\$943,296.36	12.6	\$65,665.51
2001-04	2	--all--	Hardware	\$1,174,512.68	15.7	-\$476,941.61
2001-04	3	--all--	Electronics	\$1,303,838.52	17.5	\$286,301.70
2001-04	4	--all--	Photo	\$1,792,131.39	24.0	\$184,815.76
2001-04	5	--all--	Peripherals	\$2,257,118.57	30.2	\$215,057.53
2001-04		--all--	TOTAL	\$7,470,897.52	100.0	
TOTAL		--all--	Software/Other	\$3,553,904.40	12.6	
TOTAL		--all--	Electronics	\$4,704,850.95	16.7	
TOTAL		--all--	Hardware	\$5,684,370.01	20.2	
TOTAL		--all--	Photo	\$6,333,628.91	22.5	
TOTAL		Indirect	TOTAL	\$6,709,496.66	11.9	
TOTAL		--all--	Peripherals	\$7,859,707.71	27.9	
TOTAL		Others	TOTAL	\$8,038,529.96	14.3	
TOTAL		Direct	TOTAL	\$13,388,435.36	23.8	
TOTAL		--all--	TOTAL	\$28,136,461.98	50.0	

"Peripherals" was used instead of "Peripherals and Accessories" to save space.

How to Calculate a Moving Average Using a Window Function

You can create moving aggregations with window functions. A moving aggregation can be based on a number of physical rows or it can be a logical time period. Window functions use the PARTITION keyword, and, for each row in a partition, you can define a sliding window of data. This window determines the range of rows used to perform the calculations for the current row. Window sizes can be based on either a physical number of rows or a logical interval such as time. The window has a starting row and an ending row. Depending on its definition, the window may move at one or both ends. For instance, a window defined for a cumulative SUM function would have its starting row fixed at the first row of its partition, and its ending row would slide from the starting point all the way to the last row of the partition. In contrast, a window defined for a moving average would have both its starting and ending points slide so that they maintain a constant physical or logical range.

Window functions are commonly used to calculate moving and cumulative versions of SUM, AVERAGE, COUNT, MAX, MIN, and many more functions. They can be used only in

the `SELECT` and `ORDER BY` clauses of the query. Window functions include the `FIRST_VALUE` function, which returns the first value in the window; and the `LAST_VALUE` function, which returns the last value in the window. These functions provide access to more than one row of a table without requiring a self-join.

Example: Calculating a Moving Average

The following example shows a query that retrieves a 7-day moving average of product revenue per product, using a logical time interval.

To calculate a moving average:

```
SELECT time_id
, prod_name
, TO_CHAR(revenue,'L999G990D00') revenue
, TO_CHAR(AVG(revenue) OVER (PARTITION BY prod_name ORDER BY time_id
RANGE INTERVAL '7' DAY PRECEDING),'L999G990D00') mv_7day_avg
FROM
( SELECT s.time_id, p.prod_name, SUM(s.amount_sold) revenue
  FROM products p
    , sales s
  WHERE p.prod_id = s.prod_id
    AND s.time_id BETWEEN TO_DATE('25-JUN-2001','dd-MON-yyyy')
                        AND TO_DATE('16-JUL-2001','dd-MON-yyyy')
    AND p.prod_name LIKE '%Memory%'
    AND p.prod_category = 'Photo'
  GROUP BY s.time_id, p.prod_name
)
ORDER BY time_id, prod_name;
```

TIME_ID	PROD_NAME	REVENUE	MV_7DAY_AVG
26-JUN-01	256MB Memory Card	\$560.15	\$560.15
30-JUN-01	256MB Memory Card	\$844.00	\$702.08
02-JUL-01	128MB Memory Card	\$3,283.74	\$3,283.74
02-JUL-01	256MB Memory Card	\$3,903.32	\$1,769.16
03-JUL-01	256MB Memory Card	\$699.37	\$1,501.71
08-JUL-01	128MB Memory Card	\$3,283.74	\$3,283.74
08-JUL-01	256MB Memory Card	\$3,903.32	\$2,835.34
10-JUL-01	256MB Memory Card	\$138.82	\$1,580.50

Use of Partition Outer Join to Handle Sparse Data

Data is usually stored in sparse form. That is, if no value exists for a given combination of dimension values, no row exists in the **fact table** (the table in a data warehouse that contains the important facts, frequently sales). However, a reader of a business report may want to view the data in **dense** form, with rows for all combinations of dimension values displayed even when no fact table data exists for them. For example, if a product did not sell during a particular time period, you may still want to see the product for that time period with zero sales value next to it. Moreover, time series calculations can be performed most easily when data is dense along the time dimension. This is because dense data will fill a consistent number of rows for each period, which in turn makes it simple to use window functions with physical offsets.

Data **densification** is the process of converting sparse data into dense form. To overcome the problem of sparsity, you can use a partition outer join to fill the gaps in a time series or any dimension. Such a join extends the conventional outer join syntax by applying the outer join to each logical partition defined in a query. Oracle Database logically partitions the rows in your query based on the expression you specify in the

PARTITION BY clause. The result of a partition outer join is a UNION operation of the outer joins of each of the partitions in the logically partitioned table with the table on the other side of the join. Note that you can use this type of join to fill the gaps in any dimension, not just the time dimension.

When to Use Partition Outer Join

When you want to fill in missing rows in a result set or perform time series calculations, use a partition outer join.

Example: Using Partition Outer Join

You may want to see how a particular product sold over the duration of a number of weeks. In this example, memory cards from the Photo category are used. Because these products are not sold very frequently, there may be weeks that a product is not sold at all. To make convenient comparisons, you have to make the data dense using the partition outer join as illustrated in the following example.

To use partition outer join:

```
SELECT tim.week_ending_day
, rev.prod_name product
, nvl(SUM(rev.amount_sold),0) revenue
FROM (SELECT p.prod_name, s.time_id, s.amount_sold
      FROM products p
      , sales s
      WHERE s.prod_id = p.prod_id
      AND p.prod_category = 'Photo'
      AND p.prod_name LIKE '%Memory%'
      AND s.time_id BETWEEN TO_DATE('25-JUN-2001','dd-MON-yyyy')
                        AND TO_DATE('16-JUL-2001','dd-MON-yyyy')
      ) rev
PARTITION BY (prod_name)
RIGHT OUTER JOIN (SELECT time_id, week_ending_day FROM times
                 WHERE week_ending_day
                 BETWEEN TO_DATE('01-JUL-2001','dd-MON-yyyy')
                 AND TO_DATE('16-JUL-2001','dd-MON-yyyy')
                 ) tim
ON (rev.time_id = tim.time_id)
GROUP BY tim.week_ending_day
, rev.prod_name
ORDER BY tim.week_ending_day
, rev.prod_name;
```

WEEK_ENDI	PRODUCT	REVENUE
-----	-----	-----
01-JUL-01	128MB Memory Card	0
01-JUL-01	256MB Memory Card	1404.15
08-JUL-01	128MB Memory Card	6567.48
08-JUL-01	256MB Memory Card	8506.01
15-JUL-01	128MB Memory Card	0
15-JUL-01	256MB Memory Card	138.82

Use of the WITH Clause to Simplify Business Queries

Queries that make extensive use of window functions as well as different types of joins and access many tables can become quite complex. The WITH clause allows you to eliminate much of this complexity by incrementally building up the query. It lets you reuse the same query block in a SELECT statement when it occurs more than once

within a complex query. Oracle Database retrieves the results of a query block and stores them in the user's temporary tablespace.

When to Use the WITH Clause

When a query has multiple references to the same query block and there are joins and aggregations, use the WITH clause.

Example: Using the WITH Clause

Assume you want to compare the sales of memory card products in the Photo category for the first three week endings in July 2001. The following query takes into account that some products may not have sold at all in that period, and it returns the increase or decrease in revenue relative to the week before. Finally, the query retrieves the percentage contribution of the memory card sales for that particular week. Due to the use of the WITH clause, individual sections of the query are not very complex.

To use the WITH clause:

```
WITH sales_numbers AS
( SELECT s.prod_id, s.amount_sold, t.week_ending_day
  FROM sales s
    , times t
    , products p
  WHERE s.time_id = t.time_id
    AND s.prod_id = p.prod_id
    AND p.prod_category = 'Photo'
    AND p.prod_name LIKE '%Memory%'
    AND t.week_ending_day BETWEEN TO_DATE('01-JUL-2001','dd-MON-yyyy')
                                AND TO_DATE('16-JUL-2001','dd-MON-yyyy')
)
, product_revenue AS
( SELECT p.prod_name product, s.week_ending_day, SUM(s.amount_sold) revenue
  FROM products p
    LEFT OUTER JOIN (SELECT prod_id, amount_sold, week_ending_day
                     FROM sales_numbers) s
    ON (s.prod_id = p.prod_id)
  WHERE p.prod_category = 'Photo'
    AND p.prod_name LIKE '%Memory%'
  GROUP BY p.prod_name, s.week_ending_day
)
, weeks AS
( SELECT distinct week_ending_day week FROM times WHERE week_ending_day
  BETWEEN TO_DATE('01-JUL-2001','dd-MON-yyyy')
    AND TO_DATE('16-JUL-2001','dd-MON-yyyy')
)
, complete_product_revenue AS
( SELECT w.week, pr.product, nvl(pr.revenue,0) revenue
  FROM product_revenue pr
    PARTITION BY (product)
    RIGHT OUTER JOIN weeks w
    ON (w.week = pr.week_ending_day)
)
SELECT week
, product
, TO_CHAR(revenue,'L999G990D00') revenue
, TO_CHAR(revenue - lag(revenue,1) OVER (PARTITION BY product
  ORDER BY week),'L999G990D00') w_w_diff
, TO_CHAR(100 * RATIO_TO_REPORT(revenue) OVER (PARTITION BY week),'990D0')
percentage
```

```
FROM complete_product_revenue
ORDER BY week, product;
```

WEEK	PRODUCT	REVENUE	W_W_DIFF	PERCENT
-----	-----	-----	-----	-----
01-JUL-01	128MB Memory Card	\$0.00		0.0
01-JUL-01	256MB Memory Card	\$1,404.15		100.0
01-JUL-01	64MB Memory Card	\$0.00		0.0
08-JUL-01	128MB Memory Card	\$6,567.48	\$6,567.48	43.6
08-JUL-01	256MB Memory Card	\$8,506.01	\$7,101.86	56.4
08-JUL-01	64MB Memory Card	\$0.00	\$0.00	0.0
15-JUL-01	128MB Memory Card	\$0.00	-\$6,567.48	0.0
15-JUL-01	256MB Memory Card	\$138.82	-\$8,367.19	100.0
15-JUL-01	64MB Memory Card	\$0.00	\$0.00	0.0

Part IV

Managing a Data Warehouse

This section discusses maintaining the data warehouse, and includes:

- [Chapter 10, "Refreshing a Data Warehouse"](#)
- [Chapter 11, "Optimizing Data Warehouse Operations"](#)
- [Chapter 12, "Eliminating Performance Bottlenecks"](#)
- [Chapter 13, "Backing up and Recovering a Data Warehouse"](#)
- [Chapter 14, "Securing a Data Warehouse"](#)

Refreshing a Data Warehouse

This section includes the following topics:

- [About Refreshing Your Data Warehouse](#)
- [Using Rolling Windows to Offload Data](#)

About Refreshing Your Data Warehouse

You must update your data warehouse on a regular basis to ensure that the information derived from it is current. This process of updating the data is called the **refresh process**.

Extraction, transformation and loading (ETL) is done on a scheduled basis to reflect changes made to the original source system. During this step, you physically insert the new, updated data into the production data warehouse schema, and take all the other steps necessary (such as building indexes, validating constraints, making backup copies) to make this new data available to the users. Once all of this data has been loaded into the data warehouse, the materialized views must be updated to reflect the latest data.

The partitioning scheme of the data warehouse is often crucial in determining the efficiency of refresh operations in the data warehouse loading process. The loading process is often considered when choosing the partitioning scheme of data warehouse tables.

Most data warehouses are loaded with new data on a regular schedule. For example, every night, week, or month, new data is brought into the data warehouse. The data being loaded at the end of the week or month typically corresponds to the transactions for the week or month. In this very common scenario, the data warehouse is being loaded by time. This suggests that the data warehouse tables should be partitioned on a date column. In the data warehouse example, suppose the new data is loaded into the `sales` table every month. Furthermore, the `sales` table has been partitioned by month. These steps show how the load process will proceed to add the data for a new month (Q1 2006) to the table `sales`.

Example: Refreshing Your Data Warehouse

Many queries request few columns from the `products`, `customers`, and `sales` tables, restricting the query by date. Take advantage of a materialized view that will speed up the majority of the queries against the three tables. Use a pre-built table on top of which the materialized view will be created. Choose the partitioning strategy of the materialized view in sync with the `sales` table's partitioning strategy.

The following example illustrates the refreshing of a materialized view. It uses a partition exchange loading operation. The example is based on the `sales` table in the `sh` schema.

To refresh a materialized view:

1. Create a table that will be the basis for the materialized view.

```
CREATE TABLE sales_prod_cust_mv
( time_id DATE
, prod_id NUMBER
, prod_name VARCHAR2(50)
, cust_id NUMBER
, cust_first_name VARCHAR2(20)
, cust_last_name VARCHAR2(40)
, amount_sold NUMBER
, quantity_sold NUMBER
)
PARTITION BY RANGE (time_id)
( PARTITION p1999 VALUES LESS THAN (TO_DATE('01-JAN-2000', 'DD-MON-YYYY'))
, PARTITION p2000 VALUES LESS THAN (TO_DATE('01-JAN-2001', 'DD-MON-YYYY'))
, PARTITION p2001h1 VALUES LESS THAN (TO_DATE('01-JUL-2001', 'DD-MON-YYYY'))
, PARTITION p2001h2 VALUES LESS THAN (TO_DATE('01-JAN-2002', 'DD-MON-YYYY'))
, PARTITION p2001q1 VALUES LESS THAN (TO_DATE('01-APR-2002', 'DD-MON-YYYY'))
, PARTITION p2002q2 VALUES LESS THAN (TO_DATE('01-JUL-2002', 'DD-MON-YYYY'))
, PARTITION p2002q3 VALUES LESS THAN (TO_DATE('01-OCT-2002', 'DD-MON-YYYY'))
, PARTITION p2002q4 VALUES LESS THAN (TO_DATE('01-JAN-2003', 'DD-MON-YYYY'))
, PARTITION p2003q1 VALUES LESS THAN (TO_DATE('01-APR-2003', 'DD-MON-YYYY'))
, PARTITION p2003q2 VALUES LESS THAN (TO_DATE('01-JUL-2003', 'DD-MON-YYYY'))
, PARTITION p2003q3 VALUES LESS THAN (TO_DATE('01-OCT-2003', 'DD-MON-YYYY'))
, PARTITION p2003q4 VALUES LESS THAN (TO_DATE('01-JAN-2004', 'DD-MON-YYYY'))
, PARTITION p2004q1 VALUES LESS THAN (TO_DATE('01-APR-2004', 'DD-MON-YYYY'))
, PARTITION p2004q2 VALUES LESS THAN (TO_DATE('01-JUL-2004', 'DD-MON-YYYY'))
, PARTITION p2004q3 VALUES LESS THAN (TO_DATE('01-OCT-2004', 'DD-MON-YYYY'))
, PARTITION p2004q4 VALUES LESS THAN (TO_DATE('01-JAN-2005', 'DD-MON-YYYY'))
, PARTITION p2005q1 VALUES LESS THAN (TO_DATE('01-APR-2005', 'DD-MON-YYYY'))
, PARTITION p2005q2 VALUES LESS THAN (TO_DATE('01-JUL-2005', 'DD-MON-YYYY'))
, PARTITION p2005q3 VALUES LESS THAN (TO_DATE('01-OCT-2005', 'DD-MON-YYYY'))
, PARTITION p2005q4 VALUES LESS THAN (TO_DATE('01-JAN-2006', 'DD-MON-YYYY'))
, PARTITION p2006q1 VALUES LESS THAN (TO_DATE('01-APR-2006', 'DD-MON-YYYY'))
) PARALLEL COMPRESS;
```

2. Load the initial table from the `sales` table.

```
ALTER SESSION ENABLE PARALLEL DML;
INSERT /*+ PARALLEL smv */ INTO sales_prod_cust_mv smv
SELECT /*+ PARALLEL s PARALLEL c */ s.time_id
, s.prod_id
, p.prod_name
, s.cust_id
, cust_first_name
, c.cust_last_name
, SUM(s.amount_sold)
, SUM(s.quantity_sold)
FROM sales s
, products p
, customers c
WHERE s.cust_id = c.cust_id
AND s.prod_id = p.prod_id
GROUP BY s.time_id
, s.prod_id
```

```

, p.prod_name
, s.cust_id
, c.cust_first_name
, c.cust_last_name;
COMMIT;

```

3. Create a materialized view.

```

CREATE MATERIALIZED VIEW sales_prod_cust_mv
ON PREBUILT TABLE
ENABLE QUERY REWRITE
AS SELECT s.time_id
, s.prod_id
, p.prod_name
, s.cust_id
, c.cust_first_name
, c.cust_last_name
, SUM(s.amount_sold) amount_sold
, SUM(s.quantity_sold) quantity_sold
FROM sales s
, products p
, customers c
WHERE s.cust_id = c.cust_id
AND s.prod_id = p.prod_id
GROUP BY s.time_id
, s.prod_id
, p.prod_name
, s.cust_id
, c.cust_first_name
, c.cust_last_name;

```

4. Load a separate table to be exchanged with the new partition.

```

CREATE TABLE sales_q1_2006 PARALLEL COMPRESS
AS SELECT * FROM sales
WHERE 0 = 1;

/* This would be the regular ETL job */

ALTER SESSION ENABLE PARALLEL DML;

INSERT /* PARALLEL qs */ INTO sales_q1_2006 qs
SELECT /* PARALLEL s */ prod_id
, cust_id
, add_months(time_id,3)
, channel_id
, promo_id
, quantity_sold
, amount_sold
FROM sales PARTITION(sales_q4_2005) s;

COMMIT;

CREATE BITMAP INDEX bmp_indx_prod_id ON sales_q1_2006 (prod_id);
CREATE BITMAP INDEX bmp_indx_cust_id ON sales_q1_2006 (cust_id);
CREATE BITMAP INDEX bmp_indx_time_id ON sales_q1_2006 (time_id);
CREATE BITMAP INDEX bmp_indx_channel_id ON sales_q1_2006 (channel_id);
CREATE BITMAP INDEX bmp_indx_promo_id ON sales_q1_2006 (promo_id);

ALTER TABLE sales_q1_2006 ADD CONSTRAINT sales_q_prod_fk
FOREIGN KEY (prod_id) REFERENCES products(prod_id) ENABLE NOVALIDATE;

```

```

ALTER TABLE sales_q1_2006 ADD CONSTRAINT sales_q_cust_fk
FOREIGN KEY (cust_id) REFERENCES customers(cust_id) ENABLE NOVALIDATE;

ALTER TABLE sales_q1_2006 ADD CONSTRAINT sales_q_time_fk
FOREIGN KEY (time_id) REFERENCES times(time_id) ENABLE NOVALIDATE;

ALTER TABLE sales_q1_2006 ADD CONSTRAINT sales_q_channel_fk
FOREIGN KEY (channel_id) REFERENCES channels(channel_id) ENABLE NOVALIDATE;

ALTER TABLE sales_q1_2006 ADD CONSTRAINT sales_q_promo_fk
FOREIGN KEY (promo_id) REFERENCES promotions(promo_id) ENABLE NOVALIDATE;

BEGIN
  DBMS_STATS.GATHER_TABLE_STATS('SH', 'SALES_Q1_2006');
END;
/

```

5. Create and load a separate table to be exchanged with a partition in the materialized view.

```

CREATE TABLE sales_mv_q1_2006 PARALLEL COMPRESS
AS SELECT * FROM sales_prod_cust_mv
WHERE 1 = 0;

ALTER SESSION ENABLE PARALLEL DML;

INSERT /*+ PARALLEL smv */ INTO sales_mv_q1_2006 smv
SELECT /*+ PARALLEL s PARALLEL c */ s.time_id
, s.prod_id
, p.prod_name
, s.cust_id
, cust_first_name
, c.cust_last_name
, SUM(s.amount_sold)
, SUM(s.quantity_sold)
FROM sales_q1_2006 s
, products p
, customers c
WHERE s.cust_id = c.cust_id
AND s.prod_id = p.prod_id
GROUP BY s.time_id
, s.prod_id
, p.prod_name
, s.cust_id
, c.cust_first_name
, c.cust_last_name;

COMMIT;

```

6. Gather statistics.

```

BEGIN
  DBMS_STATS.GATHER_TABLE_STATS('SH', 'SALES_MV_Q1_2006');
END;

```

7. Exchange the partitions.

```

ALTER TABLE sales
EXCHANGE PARTITION sales_q1_2006
WITH TABLE sales_q1_2006

```

```
INCLUDING INDEXES WITHOUT VALIDATION;

ALTER TABLE sales_prod_cust_mv
EXCHANGE PARTITION p2006q1
WITH TABLE sales_mv_q1_2006
INCLUDING INDEXES WITHOUT VALIDATION;
```

8. Inform the database that the materialized view is fresh again.

```
ALTER MATERIALIZED VIEW sales_prod_cust_mv CONSIDER FRESH;
```

Note that because this scenario uses a prebuilt table and, because the constraints are not RELY constraints, the query rewrite feature will work only with `query_rewrite_integrity` set to `STALE_TOLERATED`.

Using Rolling Windows to Offload Data

A particularly effective way of removing and archiving your data is through the use of a rolling window. An example of using a rolling window is when the data warehouse stores the most recent 36 months of sales data. A new partition can be added to the `sales` table for each new month, and an old partition can be removed from the `sales` table. This way, you will always maintain 36 months of data in the warehouse.

Example: Using a Rolling Window

The following example illustrates a rolling window for the `sales` table in the `sh` schema.

To use a rolling window:

1. Add the sales for December 2005.

```
ALTER TABLE sales
ADD PARTITION sales_12_2005 VALUES LESS THAN ('01-JAN-2006');
```

Note that you need to rebuild any existing indexes.

2. Drop the partition for 1999.

```
ALTER TABLE sales
DROP PARTITION sales_1999;
```

Optimizing Data Warehouse Operations

This section discusses how to optimize your data warehouse's performance, and includes the following sections:

- [Avoiding System Overload](#)
- [Optimizing the Use of Indexes and Materialized Views](#)
- [Optimizing Storage Requirements](#)

Avoiding System Overload

This section discusses how to identify and avoid system overload. In general, you should use the automatic diagnostic feature Automatic Database Diagnostic Monitor (ADDM) to identify performance problems with the database, as described in *Oracle Database 2 Day + Performance Tuning Guide*. This section discusses additional methods for avoiding performance problems in your system, and includes the following sections:

- [Monitoring System Performance](#)
- [Using Database Resource Manager](#)

Monitoring System Performance

This section provides information on how to avoid system overload by regularly monitoring important metrics. You can monitor these metrics through the use of the Database Performance page in Oracle Enterprise Manager. This section includes the following topics:

- [Monitoring Parallel Execution Performance](#)
- [Monitoring I/O](#)

Monitoring Parallel Execution Performance

This section describes how to monitor parallel execution performance. Suppose that you see many parallel statements are being downgraded. This may indicate a performance problem. Statements that run with a degree of parallelism lower than expected can take much longer, and users may experience different execution times depending on whether statements got downgraded. Possible causes for downgraded parallel statements include the following:

- The initial degree of parallelism is higher than it should be and should be lowered.
- There are not enough parallel servers available, which may indicate the system is overloaded.

To monitor parallel execution performance:

1. On the Database Home page, click **Performance**.

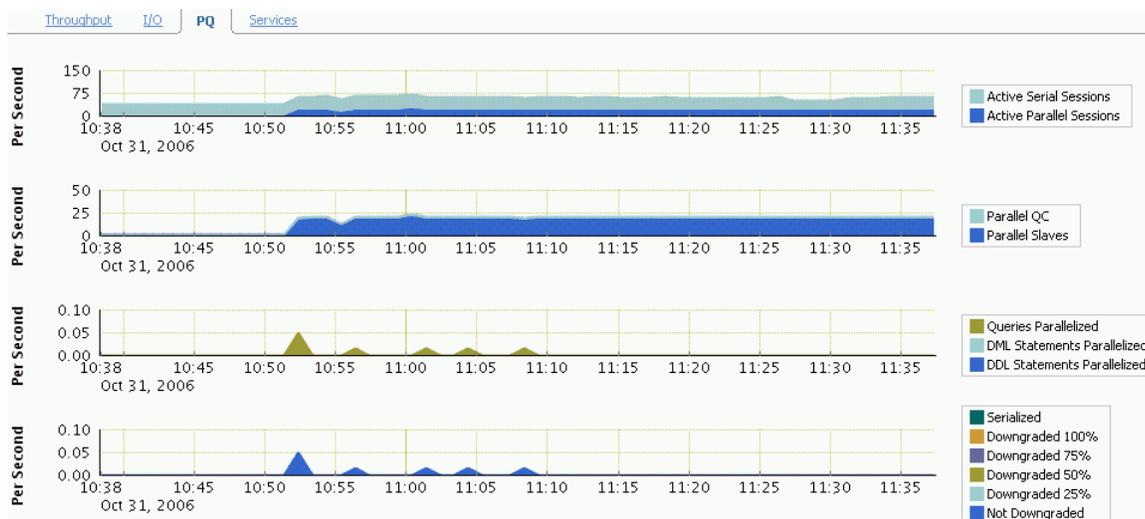
The Performance page is displayed.

2. Scroll down the page. Under the list of links, click **PQ**.

The PQ page is displayed. Parallel query performance characteristics are shown for:

- Parallel sessions
- Parallel slaves
- DML and DDL parallelization
- Serialization and statement downgrades

Figure 11–1 Monitoring Parallel Execution



Monitoring I/O

This section describes how to monitor I/O performance. If the throughput on your system is significantly lower than what you expect based on the system configuration (see [Chapter 2, "Setting Up Your Data Warehouse System"](#)), and your users complain about performance issues, then there could be a performance problem. In a well-configured system that runs a typical data warehouse workload, you would expect a large portion of large I/Os and a relatively low latency (lower than 30ms) for a single block I/O.

To monitor I/O performance:

1. On the Database Home page, click **Performance**.

The Performance page is displayed.

2. Scroll down the page. Under the list of links, click **I/O**.

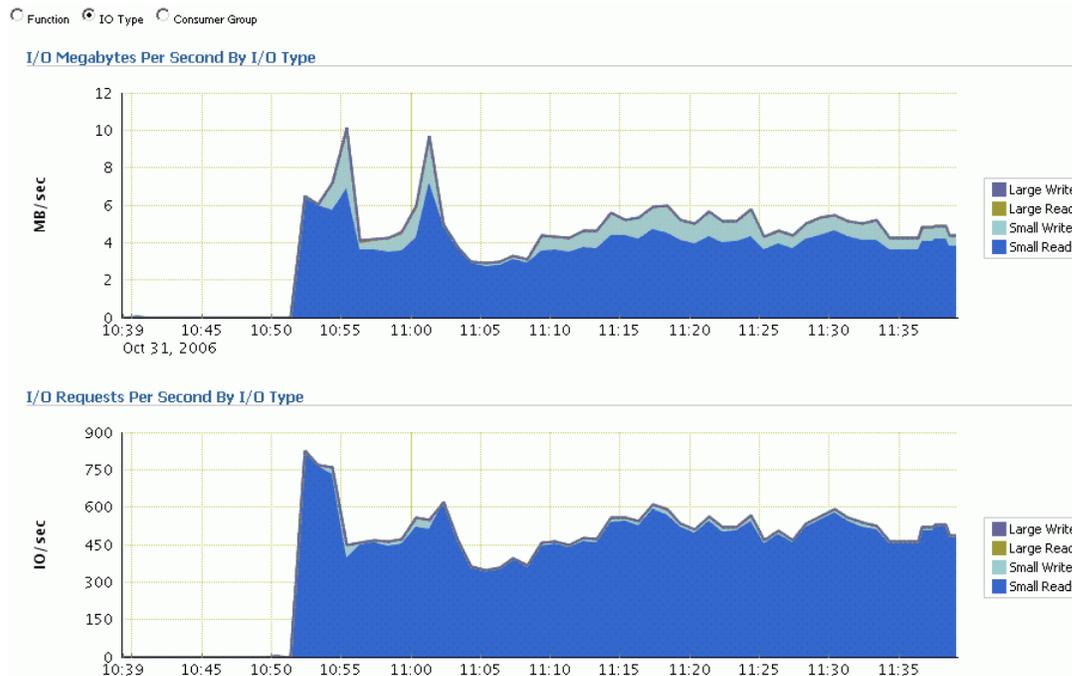
The I/O page is displayed, displaying I/O Megabytes per Second by Function and I/O Requests per Second by Function.

3. For details regarding read and write operations, select **IO Type**.

I/O details are shown for the following:

- Large Writes
- Large Reads
- Small Writes
- Small Reads

Figure 11–2 Monitoring I/O



Using Database Resource Manager

The Database Resource Manager provides the ability to prioritize work within the Oracle system. Users with higher priority jobs get resources in order to minimize response time for online work, for example, while users with lower priority jobs, such as batch jobs or reports, might encounter slower response times. This priority assignment allows for more granular control over resources and provides features such as automatic consumer group switching, maximum active sessions control, query execution time estimation and undo pool quotas for consumer groups.

You can specify the maximum number of concurrently active sessions for each consumer group. When this limit is reached, the Database Resource Manager queues all subsequent requests and runs them only after existing active sessions complete.

The Database Resource Manager is part of the Oracle Database and can distinguish different processes inside the database. As a result, the Database Resource Manager can assign priorities to individual operations running inside the database.

With the Database Resource Manager, you can do the following:

- Guarantee certain users a minimum amount of processing resources regardless of the load on the system and the number of users.
- Distribute available processing resources by allocating percentages of CPU time to different users and applications. In a data warehouse, a higher percentage may be given to ROLAP (relational online analytical processing) applications than to batch jobs.

- Allow automatic switching of users from one group to another based on administrator-defined criteria. If a member of a particular group of users creates a session that runs for longer than a specified amount of time, that session can be automatically switched to another group of users with different resource requirements.
- Configure an instance to use a particular method of allocating resources. You can dynamically change the method, for example, from a daytime setup to a nighttime setup, without having to shut down and restart the instance.

Optimizing the Use of Indexes and Materialized Views

You can improve the performance of your data warehouse by the proper use of indexes and materialized views. A key benefit of the SQL Access Advisor is its capability to use the current workload as the basis for the recommendations.

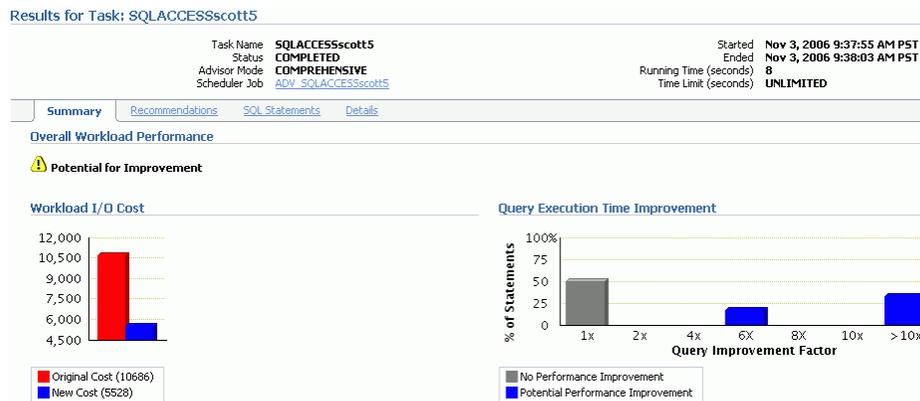
Example: Optimizing Indexes and Materialized Views Using the SQL Access Advisor

For this example, assume you have a workload running on the system that may benefit from certain indexes or materialized views.

To optimize an index and materialized view:

1. From the Advisor Central page, click **SQL Advisors**.
The Advisors page is displayed.
2. From the Advisors page, click **SQL Access Advisor**.
The SQL Access Advisor page is displayed.
3. Select **Use Default Options** and click **Continue**.
The Workload Source page is displayed.
4. Select your workload source as **Use an Existing SQL Tuning Set**. Go to a SQL Tuning Set and click **Select**. Then click **Next**.
The Recommendation Options page is displayed.
5. Select **Indexes, Materialized Views, and Comprehensive Mode**. Click **Next**.
The Schedule page is displayed.
6. Click **Submit**.
The Recommendations page is displayed.
7. Enter a name in the **Name** field and select **Immediately** for when it should start. Then click **Next**.
The Review page is displayed.
8. Click **Submit**.
The Confirmation page is displayed.
9. Select your task name and click **View Result**.
The Results for Task page is displayed. It shows a possible improvement in costs. You can view additional information under Recommendations, SQL Statements, or Details.

Figure 11–3 Suggested Improvements



Optimizing Storage Requirements

You can reduce your storage requirements by compressing data, which is achieved by eliminating duplicate values in a database block. Database objects that can be compressed include tables and materialized views. For partitioned tables, you can choose to compress some or all partitions. Compression attributes can be declared for a tablespace, a table, or a partition of a table. If declared at the tablespace level, then all tables created in that tablespace are compressed by default. You can alter the compression attribute for a table (or a partition or tablespace), and the change only applies only to new data going into that table. As a result, a single table or partition may contain some compressed blocks and some regular blocks. This guarantees that data size will not increase as a result of compression; in cases where compression could increase the size of a block, it is not applied to that block.

Using Data Compression to Improve Storage

You can compress several partitions or a complete partitioned heap-organized table. You do this either by defining a complete partitioned table as being compressed, or by defining it on a per-partition level. Partitions without a specific declaration inherit the attribute from the table definition or, if nothing is specified on table level, from the tablespace definition.

The decision about whether or not a partition should be compressed or stay uncompressed adheres to the same rules as a nonpartitioned table. However, due to the ability of range and composite partitioning to separate data logically into distinct partitions, such a partitioned table is an ideal candidate for compressing parts of the data (partitions) that are mainly read-only. It is, for example, beneficial in all rolling window operations as a kind of intermediate stage before aging out old data. With data compression, you can keep more old data online, minimizing the burden of additional storage consumption.

You can also change any existing uncompressed table partition later, add new compressed and uncompressed partitions, or change the compression attribute as part of any partition maintenance operation that requires data movement, such as MERGE PARTITION, SPLIT PARTITION, or MOVE PARTITION. The partitions can contain data or they can be empty.

The access and maintenance of a partially or fully compressed partitioned table are the same as for a fully uncompressed partitioned table. All rules that apply to fully uncompressed partitioned tables are also valid for partially or fully compressed partitioned tables.

To use data compression:

The following example creates a range-partitioned table with one compressed partition `costs_old`. The compression attribute for the table and all other partitions is inherited from the tablespace level.

```
CREATE TABLE costs_demo (  
    prod_id    NUMBER(6),    time_id    DATE,  
    unit_cost  NUMBER(10,2), unit_price NUMBER(10,2)  
PARTITION BY RANGE (time_id)  
    (PARTITION costs_old  
        VALUES LESS THAN (TO_DATE('01-JAN-2003', 'DD-MON-YYYY')) COMPRESS,  
    PARTITION costs_q1_2003  
        VALUES LESS THAN (TO_DATE('01-APR-2003', 'DD-MON-YYYY')),  
    PARTITION costs_q2_2003  
        VALUES LESS THAN (TO_DATE('01-JUN-2003', 'DD-MON-YYYY')),  
    PARTITION costs_recent VALUES LESS THAN (MAXVALUE));
```

Eliminating Performance Bottlenecks

This section discusses how to identify and reduce performance issues, and includes the following sections:

- [Verifying That SQL Runs Efficiently](#)
- [Improving Performance by Minimizing Resource Consumption](#)
- [Using Resources Optimally](#)

Verifying That SQL Runs Efficiently

An important aspect of ensuring that your system performs well is to eliminate performance problems. This section describes some methods of finding and eliminating these bottlenecks, and contains the following topics:

- [Analyzing Optimizer Statistics](#)
- [Analyzing an Execution Plan](#)
- [Using Hints to Improve Data Warehouse Performance](#)
- [Using Advisors to Verify SQL Performance](#)

Analyzing Optimizer Statistics

Optimizer statistics are a collection of data that describes more details about the database and the objects in the database. These statistics are stored in the data dictionary, and are used by the query optimizer to choose the best execution plan for each SQL statement. Optimizer statistics include the following:

- Table statistics (number of rows, blocks, and the average row length)
- Column statistics (number of distinct values in a column, number of null values in a column, and data distribution)
- Index statistics (number of leaf blocks, levels, and clustering factor)
- System statistics (CPU and I/O performance and utilization)

The optimizer statistics are stored in the data dictionary. They can be viewed using data dictionary views similar to the following:

```
SELECT * FROM DBA_SCHEDULER_JOBS WHERE JOB_NAME = 'GATHER_STATS_JOB';
```

Because the objects in a database can constantly change, statistics must be regularly updated so that they accurately describe these database objects. Statistics are maintained automatically by Oracle Database or you can maintain the optimizer statistics manually using the `DBMS_STATS` package.

Analyzing an Execution Plan

To execute a SQL statement, Oracle Database might need to perform many steps. Each of these steps either retrieves rows of data physically from the database or prepares them in some way for the user issuing the statement. The combination of the steps Oracle Database uses to execute a statement is called an execution plan. An execution plan includes an access path for each table that the statement accesses and an ordering of the tables (the join order) with the appropriate join method.

You can examine the execution plan chosen by the optimizer for a SQL statement by using the `EXPLAIN PLAN` statement. When the statement is issued, the optimizer chooses an execution plan and then inserts data describing the plan into a database table. Simply issue the `EXPLAIN PLAN` statement and then query the output table.

General guidelines for using the `EXPLAIN PLAN` statement are:

- To use the SQL script `UTLXPLAN.SQL` to create a sample output table called `PLAN_TABLE` in your schema.
- To include the `EXPLAIN PLAN FOR` clause prior to the SQL statement.
- After issuing the `EXPLAIN PLAN` statement, to use one of the scripts or packages provided by Oracle Database to display the most recent plan table output.
- The execution order in `EXPLAIN PLAN` output begins with the line that is indented farthest to the right. If two lines are indented equally, then the top line is normally executed first.

Example: Analyzing Explain Plan Output

The following statement illustrates the output of two `EXPLAIN PLAN` statements, one with dynamic pruning and one with static pruning.

To analyze EXPLAIN PLAN output:

```
EXPLAIN PLAN FOR
SELECT p.prod_name
, c.channel_desc
, SUM(s.amount_sold) revenue
FROM products p
, channels c
, sales s
WHERE s.prod_id = p.prod_id
AND s.channel_id = c.channel_id
AND s.time_id BETWEEN '01-12-2001' AND '31-12-2001'
GROUP BY p.prod_name
, c.channel_desc;
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

WITHOUT TO_DATE

Id	Operation	Name	Rows	Bytes	Cost	Time	Pstart	Pstop
(%CPU)								
0	SELECT STATEMENT		252	15876	305 (1)	00:00:06		
1	HASH GROUP BY		252	15876	305 (1)	00:00:06		
*2	FILTER							
*3	HASH JOIN		2255	138K	304 (1)	00:00:06		
4	TABLE ACCESS FULL	PRODUCTS	72	2160	2 (0)	00:00:01		
5	MERGE JOIN		2286	75438	302 (1)	00:00:06		
6	TABLE ACCESS BY INDEX ROWID	CHANNELS	5	65	2 (0)	00:00:01		
7	INDEX FULL SCAN	CHANNELS_PK	5		1 (0)	00:00:01		

*8	SORT JOIN			2286	45720	299(1)	00:00:06		
9	PARTITION RANGE ITERATOR			2286	45720	298(0)	00:00:06	KEY	KEY
10	TABLE ACCESS BY LOCAL INDEX ROWID	SALES		2286	45720	298(0)	00:00:06	KEY	KEY
11	BITMAP CONVERSION TO ROWIDS								
*12	BITMAP INDEX RANGE SCAN	SALES_TIME_BIX						KEY	KEY

Predicate Information (identified by operation id):

```

2 - filter(TO_DATE('01-12-2001')<=TO_DATE('31-12-2001'))
3 - access("S"."PROD_ID"="P"."PROD_ID")
8 - access("S"."CHANNEL_ID"="C"."CHANNEL_ID")
   filter("S"."CHANNEL_ID"="C"."CHANNEL_ID")
12 - access("S"."TIME_ID">='01-12-2001' AND "S"."TIME_ID"<='31-12-2001')

```

Note the values of KEY KEY for Pstart and Pstop.

WITH TO_DATE

Id	Operation	Name	Rows	Bytes	Cost(%CPU)	Time	Pstart	Pstop
0	SELECT STATEMENT		252	15876	31 (20)	00:00:01		
1	HASH GROUP BY		252	15876	31 (20)	00:00:01		
*2	HASH JOIN		21717	1336K	28 (11)	00:00:01		
3	TABLE ACCESS FULL	PRODUCTS	72	2160	2 (0)	00:00:01		
*4	HASH JOIN		21717	699K	26 (12)	00:00:01		
5	TABLE ACCESS FULL	CHANNELS	5	65	3 (0)	00:00:01		
6	PARTITION RANGE SINGLE		21717	424K	22 (10)	00:00:01	20	20
*7	TABLE ACCESS FULL	SALES	21717	424K	22 (10)	00:00:01	20	20

Predicate Information (identified by operation id):

```

2 - access("S"."PROD_ID"="P"."PROD_ID")
4 - access("S"."CHANNEL_ID"="C"."CHANNEL_ID")
7 - filter("S"."TIME_ID">=TO_DATE('2001-12-01 00:00:00', 'yyyy-mm-dd hh24:mi:ss') AND
          "S"."TIME_ID"<=TO_DATE('2001-12-31 00:00:00', 'yyyy-mm-dd hh24:mi:ss'))

```

Note the values of 20 20 for Pstart and Pstop.

The first execution plan shows dynamic pruning, using the KEY KEY values for Pstart and Pstop respectively. Dynamic pruning means that the database will have to figure out at execution time which partition or partitions to access. In the case of static pruning, the database knows at parse time which partition or partitions to access, which leads to more efficient execution.

You can frequently improve the execution plan by using explicit date conversions. Using explicit date conversions is a *best practice* for optimal partition pruning and index usage.

Using Hints to Improve Data Warehouse Performance

Hints let you make decisions usually made by the optimizer. As an application developer, you might know information about your data that the optimizer does not know. Hints provide a mechanism to instruct the optimizer to choose a certain query execution plan based on the specific criteria.

For example, you might know that a certain index is more selective for certain queries. Based on this information, you might be able to choose a more efficient execution plan

than the optimizer. In such a case, use hints to instruct the optimizer to use the optimal execution plan.

By default, Oracle Warehouse Builder includes hints to optimize a typical data load. See *Oracle Warehouse Builder User's Guide* for more information.

Example: Using Hints to Improve Data Warehouse Performance

Suppose you want to very quickly run a summary across the sales table for last year while the system is otherwise idle. In this case, you could issue the following statement.

To use a hint to improve data warehouse performance:

```
SELECT /*+ PARALLEL(s,16) */ SUM(amount_sold)
FROM sales s
WHERE s.time_id BETWEEN TO_DATE('01-JAN-2005', 'DD-MON-YYYY')
      AND TO_DATE('31-DEC-2005', 'DD-MON-YYYY');
```

Another common use for hints in data warehouses is to ensure that records are efficiently loaded using compression. For this, you use the APPEND hint, as shown in the following SQL:

```
...
INSERT /*+APPEND */ INTO my_materialized_view
...
```

Using Advisors to Verify SQL Performance

Using the SQL Tuning Advisor and SQL Access Advisor, you can invoke the query optimizer in advisory mode to examine a given SQL statement, or set of SQL statements, and provide recommendations to improve their efficiency. The SQL Tuning Advisor and SQL Access Advisor can make various types of recommendations, such as creating SQL profiles, restructuring SQL statements, creating additional indexes or materialized views, and refreshing optimizer statistics. Additionally, Oracle Enterprise Manager enables you to accept and implement many of these recommendations in very few steps.

The SQL Access Advisor is primarily responsible for making schema modification recommendations, such as adding or dropping indexes and materialized views. It also recommends a partitioning strategy. The SQL Tuning Advisor makes other types of recommendations, such as creating SQL profiles and restructuring SQL statements. In some cases where significant performance improvements can be gained by creating a new index, the SQL Tuning Advisor may recommend doing so. However, such recommendations should be verified by running the SQL Access Advisor with a SQL workload that contains a set of representative SQL statements.

Example: Using the SQL Tuning Advisor to Verify SQL Performance

You can use the SQL Tuning Advisor to tune a single or multiple SQL statements. When tuning multiple SQL statements, keep in the mind that the SQL Tuning Advisor does not recognize interdependencies between the SQL statements. Instead, it is just meant to be a convenient way for you to run the SQL Tuning Advisor for a large number of SQL statements.

To run the SQL Tuning Advisor to verify SQL performance:

1. Go to the Advisor Central page, then click **SQL Advisors**.

The SQL Advisors page is displayed.

2. Click **Schedule SQL Tuning Advisor**.

The Schedule SQL Tuning Advisor page is displayed. A suggested name will be in the **Name** field, which you can modify. Then select **Comprehensive** to have a comprehensive analysis performed. Select **Immediately** for the Schedule. Choose a proper SQL Tuning Set, and then click **OK**.

3. The Processing page is displayed. Then the Recommendations page shows the recommendations for improving performance. Click **View Recommendations**.

The Recommendations page is displayed.

Figure 12–1 SQL Tuning Advisor: Recommendations

Only one recommendation should be implemented.

SQL Text
`select sum(s.amount_sold) from sales_copy s, customers c where s.cust_id = c.cust_id and c.cust_last_name = 'Sixton'`

Select Recommendation
 Original Explain Plan (Annotated)

Implement

Select	Type	Findings	Recommendations	Rationale	New Benefit (%)	Explain Plan
1	Index	The execution plan of this statement can be improved by creating one or more indices.	Consider running the Access Advisor to improve the physical schema design or creating the recommended index: SH.CUSTOMERS("CUST_LAST_NAME") SH.SALES_COPY("CUST_ID")	Creating the recommended indices significantly improves the execution plan of this statement. However, it might be preferable to run "Access Advisor" using a representative SQL workload as opposed to a single statement. This will allow to get comprehensive index recommendations which takes into account index maintenance overhead and additional space consumption.	100.0	

4. The recommendation is to create an index, which you can implement by clicking **Implement**. Alternatively, you may want to run the SQL Access Advisor as well.

Improving Performance by Minimizing Resource Consumption

You can minimize resource consumption, and thus improve your data warehouse's performance through the use of the following capabilities:

- [Improving Performance: Partitioning](#)
- [Improving Performance: Query Rewrite and Materialized Views](#)
- [Improving Performance: Indexes](#)

Improving Performance: Partitioning

Data warehouses often contain large tables and require techniques both for managing these large tables and for providing good query performance across these large tables. This section discusses partitioning, a key method for addressing these requirements. Two capabilities relevant for query performance in a data warehouse are partition pruning and partitionwise joins.

Improving Performance: Partition Pruning

Partition pruning is an essential performance feature for data warehouses. In partition pruning, the optimizer analyzes `FROM` and `WHERE` clauses in SQL statements to eliminate unneeded partitions when building the partition access list. This enables Oracle Database to perform operations only on those partitions that are relevant to the SQL statement. Oracle Database prunes partitions when you use range, `LIKE`, equality, and `IN`-list predicates on the range or list partitioning columns, and when you use equality and `IN`-list predicates on the hash partitioning columns.

Partition pruning dramatically reduces the amount of data retrieved from disk and shortens the use of processing time, thus improving query performance and resource utilization. If you partition the index and table on different columns (with a global

partitioned index), partition pruning also eliminates index partitions even when the partitions of the underlying table cannot be eliminated.

Depending upon the actual SQL statement, Oracle Database may use static or dynamic pruning. Static pruning occurs at compiletime, with the information about the partitions accessed beforehand while dynamic pruning occurs at runtime, meaning that the exact partitions to be accessed by a statement are not known beforehand. A sample scenario for static pruning would be a SQL statement containing a `WHERE` condition with a constant literal on the partition key column. An example of dynamic pruning is the use of operators or functions in the `WHERE` condition.

Partition pruning affects the statistics of the objects where pruning will occur and will therefore also affect the execution plan of a statement.

Improving Performance: Partitionwise Joins

Partitionwise joins reduce query response time by minimizing the amount of data exchanged among parallel execution servers when joins execute in parallel. This significantly reduces response time and improves the use of both CPU and memory resources. In Oracle Real Application Clusters environments, partitionwise joins also avoid or at least limit the data traffic over the interconnection, which is the key to achieving good scalability for massive join operations.

Partitionwise joins can be full or partial. Oracle Database decides which type of join to use.

Example: Evaluating Partitioning with the SQL Access Advisor

You should always consider partitioning in data warehousing environments.

To evaluate partitioning:

- 1. In the Advisor Central page, click `SQL Advisors`.**
The SQL Advisors page is displayed.
- 2. Click `SQL Access Advisor`.**
The SQL Access Advisor page is displayed.
- 3. From the Initial Options, select `Use Default Options` and click `Continue`.**
- 4. From the Workload Sources, select `Current and Recent SQL Activity` and click `Next`.**
The Recommendation Options page is displayed.
- 5. Select `Partitioning` and then `Comprehensive Mode`, then click `Next`.**
The Schedule page is displayed.
- 6. Enter `SQLACCESStest1` into the `Task Name` field and click `Next`**
The Review page is displayed. Click `Submit`.

Figure 12–2 Evaluating Partitioning

SQL Access Advisor: Review
Database **database** Cancel Show SQL Back Step 4 of 4 Submit

Please review the SQL Access Advisor options and values you have selected.

Task Name **SQLACCESS_partition**
Task Description **SQL Access Advisor**
Scheduled Start Time **Run Immediately**

Options
Show All Options

Modified Option	Value	Description
<input checked="" type="checkbox"/> Analysis Scope	Partitionings on Base Tables	The type of recommendations that are allowed
<input checked="" type="checkbox"/> Workload SQL Limit	25	Specifies the number of SQL statements to be analyzed

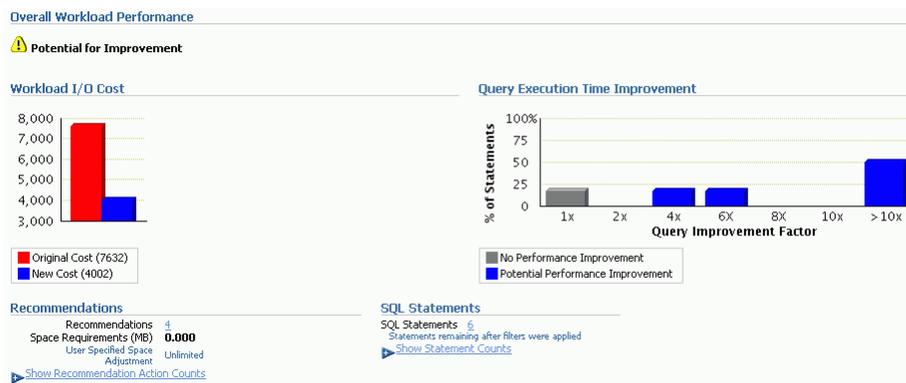
Cancel Show SQL Back Step 4 of 4 Submit

7. Click **Submit**.

The Confirmation page is displayed.

8. Select your task and click **View Result**. The Results for Task page is displayed, illustrating possible improvements as a result of partitioning.

Figure 12–3 Partitioning Results



Improving Performance: Query Rewrite and Materialized Views

In data warehouses, you can use materialized views to compute and store aggregated data such as the sum of sales. You can also use them to compute joins with or without aggregations, and they are very useful for frequently executed expensive joins between large tables as well as expensive calculations. A materialized view eliminates the overhead associated with expensive joins and aggregations for a large or important class of queries because it computes and stores summarized data before processing large joins or queries. Materialized views in these environments are often referred to as summaries.

One of the major benefits of creating and maintaining materialized views is the ability to take advantage of the query rewrite feature, which transforms a SQL statement expressed in terms of tables or views into a statement accessing one or more materialized views that are defined on the detail tables. The transformation is transparent to the user or application, requiring no intervention and no reference to the materialized view in the SQL statement. Because the query rewrite feature is transparent, materialized views can be added or dropped just like indexes without invalidating the SQL in the application code.

When underlying tables contain large amount of data, it is an expensive and time-consuming process to compute the required aggregates or to compute joins between these tables. In such cases, queries can take minutes or even hours to return

the answer. Because materialized views contain already computed aggregates and joins, Oracle Database uses the powerful query rewrite process to quickly answer the query using materialized views.

Improving Performance: Indexes

Bitmap indexes are widely used in data warehousing environments. The environments typically have large amounts of data and ad hoc queries, but a low level of concurrent DML transactions. Fully indexing a large table with a traditional B-tree index can be prohibitively expensive in terms of disk space because the indexes can be several times larger than the data in the table. Bitmap indexes are typically only a fraction of the size of the indexed data in the table. For such applications, bitmap indexing provides the following:

- Reduced response time for large classes of ad hoc queries
- Reduced storage requirements compared to other indexing techniques
- Dramatic performance gains even on hardware with a relatively small number of CPUs or a small amount of memory
- Efficient maintenance during parallel DML and loads

Using Resources Optimally

You can maximize how resources are used in your system by ensuring that operations run in parallel whenever possible. Any database operation would run faster if it were not constrained by a resource at any point in time. The operation may be constrained by CPU resources, I/O capacity, memory, or interconnection traffic (in a cluster). To improve the performance of database operations, you focus on the performance problem and try to eliminate it (so that the problem might shift to another resource). Oracle Database provides functions to optimize the use of available resources, but also to avoid using unnecessary resources.

Optimizing Performance with Parallel Execution

Parallel execution dramatically reduces response time for data-intensive operations on large databases typically associated with a decision support system (DSS) and data warehouses. You can also implement parallel execution on certain types of online transaction processing (OLTP) and hybrid systems. Parallel execution is sometimes called parallelism. Simply expressed, parallelism is the idea of breaking down a task so that, instead of one process doing all the work in a query, many processes do part of the work at the same time. An example of this is when four processes handle four different quarters in a year instead of one process handling all four quarters by itself. The improvement in performance can be quite high. Parallel execution improves processing for the following:

- Queries requiring large table scans, joins, or partitioned index scans
- Creation of large indexes
- Creation of large tables (including materialized views)
- Bulk insert, update, merge, and delete operations

You can also use parallel execution to access object types within an Oracle database. For example, you can use parallel execution to access large objects (LOBs).

Parallel execution benefits systems with all of the following characteristics:

- Symmetric multiprocessors (SMPs), clusters, or massively parallel systems
- Sufficient I/O bandwidth
- Underutilized or intermittently used CPUs (for example, systems where CPU usage is typically less than 30 percent)
- Sufficient memory to support additional memory-intensive processes, such as sorts, hashing, and I/O buffers

If your system lacks any of these characteristics, parallel execution might not significantly improve performance. In fact, parallel execution might reduce system performance on overutilized systems or systems with small I/O bandwidth.

How Parallel Execution Works

Parallel execution divides the task of executing a SQL statement into multiple small units, each of which is executed by a separate process. Also, the incoming data (tables, indexes, partitions) can be divided into parts called **granules**. The user shadow process that wants to execute a query in parallel takes on the role as parallel execution coordinator or query coordinator. The query coordinator does the following:

- Parses the query and determines the degree of parallelism
- Allocates one or two sets of slaves (threads or processes)
- Controls the query and sends instructions to the parallel query slaves
- Determines which tables or indexes need to be scanned by the parallel query slaves
- Produces the final output to the user

Setting the Degree of Parallelism

The parallel execution coordinator may enlist two or more of the instance's parallel execution servers to process a SQL statement. The number of parallel execution servers associated with a single operation is known as the degree of parallelism.

A single operation is a part of a SQL statement, such as an `ORDER BY` operation or a full table scan to perform a join on a nonindexed column table.

The degree of parallelism (DOP) is specified in the following ways:

- At the statement level with `PARALLEL` hints
- At the session level by issuing the `ALTER SESSION FORCE PARALLEL` statement
- At the table level in the table's definition
- At the index level in the index's definition

Example: Setting the Degree of Parallelism

Suppose that you want to set the DOP to 4 on a table.

To set the degree of parallelism:

Issue the following statement:

```
ALTER TABLE orders PARALLEL 4;
```

About Wait Events

Wait events are statistics that are incremented by a server process to indicate that the server process had to wait for an event to complete before being able to continue processing. A session could wait for a variety of reasons, including waiting for more input, waiting for the operating system to complete a service such as a disk write operation, or it could wait for a lock or latch.

When a session is waiting for resources, it is not doing any useful work. A large number of wait events is a source of concern. Wait event data reveals various symptoms of problems that might be affecting performance, such as latch contention, buffer contention, and I/O contention.

Backing up and Recovering a Data Warehouse

This section discusses some considerations for data warehouse backup and recovery, and includes:

- [How Should I Handle Backup and Recovery for a Data Warehouse?](#)
- [Strategies and Best Practices for Backup and Recovery](#)

How Should I Handle Backup and Recovery for a Data Warehouse?

Backup and recovery are among the most important tasks for an administrator, and data warehouses are no different. However, because of the sheer size of the database, data warehouses introduce new challenges for an administrator in the backup and recovery area.

Data warehouses are unique in that the data can come from a myriad of resources and it is transformed before finally being inserted into the database; but mostly because it can be very large. Managing the recovery of a large data warehouse can be a daunting task and traditional OLTP backup and recovery strategies may not meet the needs of a data warehouse.

Data warehouses differ from OLTP systems in the following ways:

- Data warehouses are typically much larger.
- A data warehouse may have different availability requirements than an operational system. Even though business decisions do rely on information from the data warehouse, a situation in which for example a service desk cannot operate is much worse. Also, due to the size of data warehouses, there is a much higher cost involved in guaranteeing the same level of availability for a data warehouse.
- Data warehouses are typically populated through more controlled processes, usually referred to as ETL (Extraction, Transformation, and Loading). As a result, updates in a data warehouse are better known and may be reproducible from data sources.
- A data warehouse typically stores a lot of historical data, that is often not subject to change. Data that does not change only needs to be backed up once.

You should plan a backup strategy as part of your system design and consider what to back up and how frequently to back up. The most important variables in your backup design are the amount of resources you have to perform a backup or recovery and the recovery time objective (the amount of time you can afford the system or part of the system to be unavailable).

NOLOGGING operations must be taken into account when planning a backup and recovery strategy. Traditional recovery, restoring a backup and applying the changes from the archive log, does not apply for NOLOGGING operations. On top of that, subsequent operations that rely on the data that was manipulated by the nologging operation fail. The NOLOGGING operations must be taken into account when designing a backup and recovery strategy.

Never make a backup when a NOLOGGING operation is taking place.

Plan for one of the following or a combination of the following strategies:

- The ETL strategy. Recover a backup that does not contain non-recoverable transactions and replay the ETL that has taken place between the backup and the failure.
- The incremental backup strategy. Perform a backup immediately after an otherwise non-recoverable transaction has taken place. Oracle provides a tracking file feature that enables incremental backups based on changed data blocks. RMAN leverages the tracking file feature.

Strategies and Best Practices for Backup and Recovery

Devising a backup and recovery strategy can be a daunting task. And when you have hundreds of gigabytes of data that must be protected and recovered in the case of a failure, the strategy can be very complex.

The following best practices can help you implement your warehouse's backup and recovery strategy:

- [Best Practice A: Use ARCHIVELOG Mode](#)
- [Best Practice B: Use RMAN](#)
- [Best Practice C: Use Read-Only Tablespaces](#)
- [Best Practice D: Plan for NOLOGGING Operations](#)
- [Best Practice E: Not All Tablespaces are Equally Important](#)

Best Practice A: Use ARCHIVELOG Mode

Archived redo logs are crucial for recovery when no data can be lost, because they constitute a record of changes to the database. Oracle can be run in either of two modes:

- ARCHIVELOG -- Oracle archives the filled online redo log files before reusing them in the cycle.
- NOARCHIVELOG -- Oracle does not archive the filled online redo log files before reusing them in the cycle.

Running the database in ARCHIVELOG mode has the following benefits:

- The database can be completely recovered from both instance and media failure.
- The user can perform backups while the database is open and available for use.
- Oracle supports multiplexed archive logs to avoid any possible single point of failure on the archive logs
- The user has more recovery options, such as the ability to perform tablespace-point-in-time recovery (TSPITR).

- Archived redo logs can be transmitted and applied to the physical standby database, which is an exact replica of the primary database.
- The database can be completely recovered from both instance and media failure.

Running the database in `NOARCHIVELOG` mode has the following consequences:

- The user can only back up the database while it is completely closed after a clean shutdown.
- Typically, the only media recovery option is to restore the whole database, which causes the loss of all transactions since the last backup.

Is Downtime Acceptable?

Oracle database backups can be made while the database is open or closed. Planned downtime of the database can be disruptive to operations, especially in global enterprises that support users in multiple time zones, up to 24-hours per day. In these cases it is important to design a backup plan to minimize database interruptions.

Depending on your business, some enterprises can afford downtime. If your overall business strategy requires little or no downtime, then your backup strategy should implement an online backup. The database needs never to be taken down for a backup. An online backup requires the database to be in `ARCHIVELOG` mode.

There is essentially no reason not to use `ARCHIVELOG` mode. All data warehouses (and for that matter, all mission-critical databases) should use `ARCHIVELOG` mode. Specifically, given the size of a data warehouse (and consequently the amount of time to back up a data warehouse), it is generally not viable to make an offline backup of a data warehouse, which would be necessitated if one were using `NOARCHIVELOG` mode.

Of course, large-scale data warehouses may undergo large amounts of data-modification, which in turn will generate large volumes of log files. To accommodate the management of large volumes of archived log files, RMAN provides the option to compress log files as they are archived. This will enable you to keep more archive logs on disk for faster accessibility for recovery.

In summary, a best practice is to put the database in archive log mode to provide online backups and point-in-time recovery options.

Best Practice B: Use RMAN

There are many reasons to adopt RMAN. Some of the reasons to integrate RMAN into your backup and recovery strategy are that it offers:

- extensive reporting
- incremental backups
- downtime free backups
- backup and restore validation
- backup and restore optimization
- easily integrates with media managers
- block media recovery
- archive log validation and management
- corrupt block detection

Best Practice C: Use Read-Only Tablespaces

One of the biggest issues facing a data warehouse is sheer size of a typical data warehouse. Even with powerful backup hardware, backups may still take several hours. Thus, one important consideration in improving backup performance is minimizing the amount of data to be backed up. Read-only tablespaces are the simplest mechanism to reduce the amount of data to be backed up in a data warehouse.

The advantage of a read-only tablespace is that the data only need to be backed up once. So, if a data warehouse contains five years of historical data, and the first four years of data can be made read-only. Theoretically, the regular backup of the database would only back up 20% of the data. This can dramatically reduce the amount of time required to back up the data warehouse.

Most data warehouses store their data in tables that have been range-partitioned by time. In a typical data warehouse, data is generally 'active' for a period ranging anywhere from 30 days to one year. During this period, the historical data can still be updated and changed (for example, a retailer may accept returns up to 30 days beyond the date of purchase, so that sales data records could change during this period). However, once a data has reached a certain date, it is often known to be static.

By taking advantage of partitioning, users can make the static portions of their data read-only. RMAN supports read-only tablespaces rather than read-only partitions or tables. To take advantage of the read-only tablespaces and reduce the backup window, a strategy of storing constant data partitions in a read-only tablespace should be devised. Two strategies for implementing a rolling window are as follows:

- Implement a regularly scheduled process to move partitions from a read/write tablespace to a read-only tablespace when the data matures to the point where it is entirely static.

The best practice in this case is to put the database in ARCHIVELOG mode to provide online backups and point-in-time recovery options.

- Create a series of tablespaces, each containing a small number of partitions and regularly modify one tablespace from read/write to read-only as the data in that tablespaces ages.

One consideration is that backing up data is only half of the recovery process. If you configure a tape system so that it can backup the read/write portions of a data warehouse in 4 hours, the corollary is that a tape system might take 20 hours to recover the database if a complete recovery is necessary when 80% of the database is read-only.

In summary, a best practice is to place static tables and partitions into read-only tablespaces. A read-only tablespace needs to be backed up only once.

Best Practice D: Plan for NOLOGGING Operations

In general, one of the highest priorities for a data warehouse is performance. Not only must the data warehouse provide good query performance for online users, but the data warehouse must also be efficient during the ETL process so that large amount of data can be loaded in the shortest amount of time.

One common optimization leveraged by data warehouses is to execute bulk-data operations using the NOLOGGING mode. The database operations which support NOLOGGING modes are direct-path loads and inserts, index creation, and table creation. When an operation runs in NOLOGGING mode, data is not written to the redo log (or more precisely, only a small set of metadata is written to the redo log). This

mode is widely used within data warehouses and can improve the performance of bulk data operations by up to 50%.

However, the trade-off is that a NOLOGGING operation cannot be recovered using conventional recovery mechanisms, since the necessary data to support the recovery was never written to the log file. Moreover, subsequent operations to the data upon which a nologging operation has occurred also cannot be recovered even if those operations were not using nologging mode. Because of the performance gains provided by NOLOGGING operations, it is generally recommended that data warehouses utilize nologging mode in their ETL process.

The presence of NOLOGGING operations must be taken into account when devising the backup and recovery strategy. When a database relies on NOLOGGING operations, the conventional recovery strategy (of recovering from the latest tape backup and applying the archived logfiles) is no longer applicable because the log files cannot recover the NOLOGGING operation.

The first principle to remember is, do not make a backup when a NOLOGGING operation is occurring. Oracle does not currently enforce this rule, so the DBA must schedule the backup jobs and the ETL jobs such that the nologging operations do not overlap with backup operations.

There are two approaches to backup and recovery in the presence of NOLOGGING operations: ETL or incremental backups. If you are not using NOLOGGING operations in your data warehouse, then you do not have to choose either of the following options: you can recover your data warehouse using archived logs. However, the following options may offer some performance benefits over an archive-log-based approach in the event of recovery.

- [Extraction, Transformation, and Loading](#)
- [Incremental Backup](#)

Extraction, Transformation, and Loading

The ETL process uses several Oracle features or tools and a combination of methods to load (re-load) data into a data. These features or tools may consist of:

- **Transportable Tablespaces.** The Oracle Transportable Tablespace feature enables users to quickly move a tablespace across Oracle databases. It is the most efficient way to move bulk data between databases. Oracle Database provides the ability to transport tablespaces across platforms. If the source platform and the target platform are of different endianness, then RMAN will convert the tablespace being transported to the target format.
- **SQL*Loader.** SQL*Loader loads data from external flat files into tables of an Oracle database. It has a powerful data-parsing engine that puts little limitation on the format of the data in the datafile.
- **Data Pump (export/import).** Oracle Database offers the Oracle Data Pump technology, which enables very high-speed movement of data and metadata from one database to another. This technology is the basis for Oracle's data movement utilities, Data Pump Export and Data Pump Import.
- **External Tables.** The external tables feature is a complement to existing SQL*Loader functionality. It enables you to access data in external sources as if it were in a table in the database.

The ETL Strategy and NOLOGGING Operations One approach is to take regular database backups and store the necessary data files to re-create the ETL process for that entire week. If a recovery is necessary, the data warehouse could be recovered from the most

recent backup. Then, instead of rolling forward by applying the archived redo logs (as would be done in a conventional recovery scenario), the data warehouse could be rolled forward by re-running the ETL processes. This paradigm assumes that the ETL processes can be easily replayed, which would typically involve storing a set of extract files for each ETL process (many data warehouses do this already as a best practice, in order to be able to identify repair a bad data feed for example).

A sample implementation of this approach is make a backup of the data warehouse every weekend, and then store the necessary files to support the ETL process for each night. Thus, at most, seven days of ETL processing would need to be re-applied in order to recover a database. The data warehouse administrator can easily project the length of time to recover the data warehouse, based upon the recovery speeds from tape and performance data from previous ETL runs.

Essentially, the data warehouse administrator is gaining better performance in the ETL process through nologging operations, at a price of slight more complex and less-automated recovery process. Many data warehouse administrators have found that this is a desirable trade-off.

One downside to this approach is that the burden is upon the data warehouse administrator to track all of the relevant changes that have occurred in the data warehouse. This approach will not capture changes that fall outside of the ETL process. For example, in some data warehouses, end-users may create their own tables and data structures. Those changes will be lost in the event of a recovery. This restriction needs to be conveyed to the end-users. Alternatively, one could also mandate that end-users create all of private database objects in a separate tablespace, and during recovery, the DBA could recover this tablespace using conventional recovery while recovering the rest of the database using the approach of replaying the ETL process.

In summary, a best practice is to restore a backup that does not contain non-recoverable (nologging) transactions. Then replay the ETL process to reload the data.

Sizing the Block Change Tracking File The size of the block change tracking file is proportional to:

- Database size in bytes. The block change tracking file contains data representing data file blocks in the database. The data is approximately 1/250000 of the total size of the database.
- The number of enabled threads. All Real Application Cluster (RAC) instances have access to the same block change tracking file, however, the instances update different areas of the tracking file without any locking or inter-node block swapping. You enable block change tracking for the entire database and not for individual instances.
- Changed Block Metadata. The block change tracking file keeps a record of all changes between previous backups, in addition to the modifications since the last backup. The tracking file retains the change history for a maximum of eight backups. If the tracking file contains the change history for eight backups then the Oracle database overwrites the oldest change history information.

Let us take an example of a 500 GB database, with only one thread, and having eight backups kept in the RMAN repository will require a block change tracking file of 20 MB.

$$\frac{((\text{Threads} * 2) + \text{number of old backups}) * (\text{database size in bytes})}{250000} = 20\text{MB}$$

Incremental Backup

A more automated backup and recovery strategy in the presence of nologging operations leverages RMAN's incremental backup capability. Incremental backups have been part of RMAN since it was first released. Incremental backups provide the capability to backup only the changed blocks since the previous backup. Incremental backups of datafiles capture data changes on a block-by-block basis, rather than requiring the backup of all used blocks in a datafile. The resulting backups set are generally smaller and more efficient than full datafile backups, unless every block in the datafile is change.

Oracle Database delivers the ability for faster incrementals with the implementation of the change tracking file feature. When you enable block change tracking, Oracle tracks the physical location of all database changes. RMAN automatically uses the change tracking file to determine which blocks need to be read during an incremental backup and directly accesses that block to back it up.

The Incremental Approach A typical backup and recovery strategy using this approach is to backup the data warehouse every weekend, and then take incremental backups of the data warehouse every night following the completion of the ETL process. Note that incremental backups, like conventional backups, must not be run concurrently with nologging operations. In order to recover the data warehouse, the database backup would be restored, and then each night's incremental backups would be re-applied. Although the nologging operations were not captured in the archive logs, the data from the nologging operations is present in the incremental backups. Moreover, unlike the previous approach, this backup and recovery strategy can be completely managed using RMAN.

The replay ETL approach and the incremental backup approach are both recommended solutions to efficiently and safely backing up and recovering a database which is a workload consisting of many `NOLOGGING` operations. The most important consideration is that your backup and recovery strategy must take these `NOLOGGING` operations into account.

In summary, a best practice is to implement Block Change Tracking functionality and make an incremental backup after a direct load that leaves objects unrecoverable due to `NOLOGGING` operations.

Best Practice E: Not All Tablespaces are Equally Important

While the simplest backup and recovery scenario is to treat every tablespace in the database the same, Oracle provides the flexibility for a DBA to devise a backup and recovery scenario for each tablespace as needed.

Not all of the tablespaces in a data warehouse are equally significant from a backup and recovery perspective. DBAs can leverage this information to devise more efficient backup and recovery strategies when necessary. The basic granularity of backup and recovery is a tablespace, so different tablespaces can potentially have different backup and recovery strategies. On the most basic level, temporary tablespaces never need to be backed up (a rule that RMAN enforces).

Moreover, in some data warehouses, there may be tablespaces that are not explicit temporary tablespaces but are essentially functioning as temporary tablespaces as they are dedicated to scratch space for end-users to store temporary tables and incremental results. Depending upon the business requirements, these tablespaces may not need to be backed up and restored. Instead, in the case of a loss of these tablespaces, the end users would re-create their own data objects.

In many data warehouses, some data is more important than other data. For example, the sales data in a data warehouse may be crucial and, in a recovery situation, this data must be online as soon as possible. But, in the same data warehouse, a table storing clickstream data from the corporate Web site may be much less critical. The business may tolerate this data being offline for a few days or may even be able to accommodate the loss of several days of clickstream data in the event of a loss of database files. In this scenario, the tablespaces containing sales data must be backed up often, while the tablespaces containing clickstream data need only to be backed up once every week or two.

Securing a Data Warehouse

This section discusses some preliminary considerations for data warehouse security, and includes the following topics:

- [Overview of Data Warehouse Security](#)
- [Using Roles and Privileges for Data Warehouse Security](#)
- [Using Virtual Private Database in Data Warehouses](#)
- [Overview of Oracle Label Security](#)
- [Overview of Fine-Grained Auditing in Data Warehouses](#)
- [Overview of Transparent Data Encryption in Data Warehouses](#)

Overview of Data Warehouse Security

Data warehousing poses its own set of challenges for security. One major challenge is that enterprise data warehouses are often very large systems, serving many user communities with varying security needs. Thus, while data warehouses require a flexible and powerful security infrastructure, the security capabilities must operate in an environment that has stringent performance and scalability requirements.

Why is Security Necessary for a Data Warehouse?

Many of the basic requirements for security are well-known, and apply equally to a data warehouse as they would to any other system: The applications must prevent unauthorized users from accessing or modifying data; the applications and underlying data must not be susceptible to data theft by hackers; the data must be available to the right users at the right time; and the system must keep a record of activities performed by its users.

These requirements are perhaps even more important in a data warehouse because a warehouse contains data consolidated from multiple sources, and therefore, from the perspective of an individual trying to steal information, a data warehouse can be one of the most lucrative targets in an enterprise. In addition, a robust security infrastructure can often vastly improve the effectiveness or reduce the costs of a data warehouse environment.

Some typical customer scenarios for data warehouse security include the following:

- An enterprise is managing a data warehouse that will be widely used by many divisions and subsidiaries. This enterprise needs a security infrastructure that ensures the employees of each division be able to view only the data that is relevant to their own division, while also allowing employees in its corporate offices to view data for all divisions and subsidiaries.

- An enterprise's data warehouse stores personal information. Privacy laws may govern the use of such personal information. The data warehouse must handle data so as to adhere to these laws.
- An enterprise sells data from a data warehouse to its clients. Those clients may view only the data to which they have purchased or to which they have subscribed. They should never be permitted to see the data of other clients.

Using Roles and Privileges for Data Warehouse Security

System privileges, object privileges, and roles provide a basic level of database security. They are designed to control user access to data and to limit the kinds of SQL statements that users can execute. Roles are groupings of privileges that you can use to create different levels of database access. For example, you can create a role for application developers that enables users to create tables and programs.

You can grant privileges and roles to other users only when you possess the necessary privilege. The granting of roles and privileges starts at the administrator level. At database creation, the administrative user `SYS` is created and granted all system privileges and predefined Oracle roles. User `SYS` can then grant privileges and roles to other users and also grant those users the right to grant specific privileges to others. Without explicitly granted privileges, a user cannot access any information in the database.

Roles and privileges enforce security on the data itself, and their use is essential to a data warehouse because users access data through a number of applications and tools.

Using Virtual Private Database in Data Warehouses

Virtual private database (VPD) enables you to enforce security, to a fine level of granularity, directly on tables, views, or synonyms. Because security policies are attached directly to tables, views, or synonyms and automatically applied whenever a user accesses data, there is no way to bypass security. By dynamically appending SQL statements with a predicate, VPD limits access to data at the row level and ties a security policy to the database object itself. It enables multiple users to have secure direct access to critical data within a single database server, with the assurance of complete data separation. VPD can ensure that banking customers see only their own account history, and that an enterprise serving multiple companies' data (who may be competitors) can do so from the same data warehouse, and allow each company to see only its own data. In addition to control at the row level, VPD offers controlled access to security-relevant columns so that employees could see their own salaries, but not one else's.

VPD is application-transparent. Security is enforced at the database layer and takes into account application-specific logic used to limit data access within the database. Both standard and custom-built applications can take advantage of the fine-grained access control, without the need to change a single line of application code.

Within an enterprise, VPD results in a lower cost of ownership in deploying applications. Security can be built once, in the warehouse, rather than in every application that accesses data. Security is stronger, because it is enforced by the database, no matter how a user accesses data. Security cannot be bypassed by a user accessing data through an ad hoc query tool or new report writer. In an enterprise data warehouse, which often supports dozens of different applications as well as many end-user tools, Virtual private database feature is key technology.

How Virtual Private Database Works

Virtual private database is enabled by associating a security policy with a table, view, or synonym. An administrator uses the PL/SQL `DBMS_RLS` package to bind a policy function with a database object. Direct or indirect access to the object with an attached security policy causes the database to consult a function implementing the policy. The policy function returns a predicate (a `WHERE` clause) that the database appends to the user's SQL statement, thus transparently and dynamically modifying the user's data access.

An application context enables access conditions to be based on virtually any attribute a database administrator deems significant, such as organization, subscriber number, account number, or position. For example, a warehouse of sales data can enforce access based on customer number, and whether the user is a customer, a sales representative or a marketing analyst. In this way, customers can view their order history over the Web (but only for their own orders), while sales representatives can view multiple orders, but only for their own customers, and analysts can analyze all sales from the previous two quarters.

An application context acts as a secure cache of data that may be applied to a fine-grained access control policy on a particular object. Upon user login to the database, Oracle Database sets up an application context to cache information in the user's session. Information in the application context is defined by a developer based on information relevant to the particular application. For example, a reporting application that will query regional sales data can base its access control on the user's position and division. The application, in this case, could initially set up an application context for each user as he logs in and populate the context with data queried from the employees and departments tables for the user's position and division, respectively. The package implementing the VPD policy on the regional sales table references this application context to populate the user's position and division for each query. As such, an application context makes executing subqueries unnecessary, which might otherwise hinder performance.

Overview of Oracle Label Security

Oracle Label Security, a security option for Oracle Database, extends the Virtual Private Database (VPD) to enforce label-based access control. Oracle Label Security is a complete, VPD-enabled application, augmenting VPD with labeled data management. It increases the ease of deploying secure data warehouses and provides row-level security out-of-the-box.

Label-based access control lets you assign sensitivity labels to rows in a table, control access to that data based on those labels, and ensure that data is marked with the appropriate security label. For example, an organization may differentiate between company confidential information and partner information. Furthermore, there may be some confidential information that can be shared with certain key partners, and some that is only accessible by certain subsets of internal groups, such as the finance or sales divisions. The ability to manage labeled data is a great advantage for organizations to provide proper information to the appropriate people, at the proper data access level.

How Oracle Label Security Works

Oracle Label Security uses **policies**, which are collections of labels, user authorizations and security enforcement options. Once created, policies can be applied to entire application schemas or specific application tables. Oracle Label Security supports multiple policy definitions within a single data warehouse. Label definitions, user

authorizations and enforcement options are defined on a per-policy basis. For example, a marketing policy might have labels such as marketing-only, manager, and senior vice president.

Oracle Label Security mediates access to rows in database tables based on a label contained in the row, a label associated with each database session, and Oracle Label Security privileges assigned to the session. It provides access mediation on an application table after a user has been granted the standard database system and object privileges. For example, suppose a user has `SELECT` privilege on a table. If the user executes a `SELECT` statement on the table, Oracle Label Security will evaluate the selected rows and determine if the user can access them based on the privileges and access labels assigned to the user. Oracle Label Security also performs such security checks on `UPDATE`, `DELETE`, and `INSERT` statements. Labels can be applied to tables as well as to materialized views, where the materialized views increase performance and labels increase security, thus ensuring the flexibility, speed and scalability desired in data warehouse environments.

How Data Warehouses Benefit from Labels

Oracle Label Security lets you consolidate information from multiple sources into one, very large system, with the convenience and manageability and security of centralized administration. Because this security option is an application on its own, there is no need to do any PL/SQL programming. It enables consolidation, minimizes risk by enforcing security on the data itself, and provides fine-grained access security by controlling access to data down to the row level.

Overview of Fine-Grained Auditing in Data Warehouses

Fine-grained auditing allows the monitoring of data access based on content. It enables you to specify the columns and conditions that you want audit records for. Conditions can include limiting the audit to specific types of DML statements used in connection with the columns that you specify. You can also provide the name of the routine you want called when an audit event occurs. This routine can notify or alert administrators or handle errors and anomalies. An example of fine-grained auditing would be a central tax authority tracking access to tax returns to guard against employee snooping. It is insufficient to know that a specific user issued a `SELECT` statement on a particular table. What is necessary for robust security is auditing at the finer level of when a user tries to access information that is not needed to perform his normal duties, in this case, a `SELECT` statement on a column or row containing non-work related information. Fine-grained auditing offers this capability.

Fine-grained auditing can be implemented in user applications using the `DBMS_FGA` package or by using database triggers.

Overview of Transparent Data Encryption in Data Warehouses

Transparent data encryption enables encryption of sensitive data in database columns as the data is stored in the operating system files. It provides for secure storage and management of encryption keys in a security module external to the database. It eliminates the need to embed encryption routines in existing applications and dramatically lowers the cost and complexity of encryption. With a few simple commands, sensitive application data can be encrypted.

Most encryption solutions require specific calls to encryption functions within the application code. This is expensive because it typically requires extensive understanding of an application as well as the ability to write and maintain software.

In general, most organizations do not have the time or expertise to modify existing applications to make calls to encryption routines. Transparent data encryption addresses the encryption problem by deeply embedding encryption in Oracle Database. Note, however, that it works with direct-path loading, but not with SQL*Loader.

Application logic performed through SQL will continue to work without modification. In other words, applications can use the same syntax to insert data into an application table and Oracle Database will automatically encrypt the data before writing the information to disk. Subsequent `SELECT` operations will have the data transparently decrypted so the application will continue to work normally.

A

adding
 mapping operators, 5-4
analytic capabilities, 9-1
analytic SQL, 9-1
ARCHIVELOG mode
 as a best practice, 13-2
ASM, 2-4
attribute properties, setting, 5-11
attributes
 connecting, 5-7
 level attributes, 4-6
 setting properties, 5-11
auto binding rules, 4-9
Automatic Storage Management (ASM), 2-4

B

backup, 13-1
backup and recovery, 13-1
 best practices, 13-2
best practices
 ARCHIVELOG mode, 13-2
 backup and recovery, 13-2
 NOLOGGING, 13-4
 read-only tablespaces, 13-4
 RMAN, 13-3
 tablespace differences, 13-7
binding
 auto binding, rules, 4-9

C

calculating moving averages, 9-12
calculating relative contributions, 9-9
common tasks
 in data warehouses, 1-2
connecting
 attributes, 5-7
 groups, 5-8
 operators, 5-7
CPUs, 2-2
Create External Table Wizard, 4-3
creating
 mappings, 5-1

CUBE function, 9-4
cubes, 4-11
 dimensionality, 4-11
 example, 4-12
 measures, 4-11
 relational implementation, 4-12

D

data compression, 11-5
data warehouses
 common tasks, 1-2
 key characteristics of, 1-2
 setting up, 2-1, 2-6
 tools, 1-4
data warehousing security, 14-1
dd utility, 2-4
defining
 mappings, 5-1
 schedules, 7-4
degree of parallelism (DOP), 12-9
densification, 9-13
diagrams, 8-3
 impact analysis, 8-3, 8-5
 lineage, 8-3, 8-5
dimensions
 dimension roles, 4-6
 example, 4-7
 hierarchies, 4-6
 implementing, 4-8
 level attributes, 4-6
 level relationships, 4-7
 star schema implementation, 4-8
disks
 needed, 2-3

E

ETL
 improving runtime performance, 6-1
explain plan
 analyzing, 12-2
explain plans, 12-2
external tables
 creating a new definition, 4-3
 defined, 4-3

wizard for creating, 4-3
See also flat files

F

fine-grained auditing
 in data warehouses, 14-4
flat files
 See also external tables

G

group properties, 5-11
GROUPING function, 9-5
GROUPING SETS function, 9-6
groups
 connecting, 5-8
 in LIA diagrams, 8-5
 setting properties, 5-11

H

hardware configuration, 2-2
 verifying, 2-4
hierarchies
 about, 4-6
hints, 12-3

I

impact analysis diagrams, 8-3, 8-5
implementing
 relational cubes, 4-12
 star schema, 4-8
improving runtime performance, 6-1
indexes, 12-8
 optimizing, 11-4
 optimizing with SQL Access Advisor, 11-4
initialization parameters
 setting, 2-7
interrow calculations
 performing, 9-11
I/O, 2-3, 11-2
 monitoring, 11-2

L

lineage diagrams, 8-3, 8-5

M

mapping operators
 about, 5-1
 adding, 5-4
 connecting, 5-7
 editing, 5-6
 synchronizing with Repository objects, 5-12
 types of, 5-3
mapping operators, setting, 5-11
mappings
 about, 5-1

 creating, 5-1
 defining, 5-1
 for Partition Exchange Loading, 6-2
 setting properties, 5-10
materialized views, 12-7
memory, 2-2
memory management
 parameters, 2-6
memory needed, 2-2
metadata
 dependencies, 8-1
metadata dependencies, diagrams of, 8-1
Metadata Dependency Manager, 8-1

N

NOLOGGING
 as a best practice, 13-4

O

offloading data
 with rolling windows, 10-5
operator attributes, 5-7
operator properties
 setting, 5-11
operators
 connecting, 5-7
 editing, 5-6
 synchronizing with Repository objects, 5-12
operators, mapping, 5-1
 adding, 5-4
 connecting, 5-7
 editing, 5-6
 types of, 5-3
optimizer statistics, 12-1
Oracle Label Security, 14-3
 in data warehouses, 14-2
Orion utility, 2-5

P

parallel execution, 12-8
 how it works, 12-9
parallel query, 11-1, 12-8
 monitoring, 11-1
parameters
 initialization, 2-7
 memory management, 2-6
Partition Exchange Loading (PEL), 6-1
 about, 6-1
 configuring targets for, 6-5
 mappings for, 6-2
 performance considerations, 6-4
 restrictions on, 6-5, 6-8
partition outer join, 9-13
partition pruning, 12-5
partitioning, 12-5
partitionwise joins, 12-6
performance
 using hints, 12-3

- using query rewrite, 12-7
- using SQL Access Advisor, 12-4
- using SQL Tuning Advisor, 12-4
- privileges
 - in data warehouse security, 14-2
- properties
 - mapping, 5-10
 - setting, 5-11

Q

- QUERY, 10-3
- Query, 12-7
- query rewrite, 2-8, 2-9, 4-4, 10-5, 12-7

R

- RANK function, 9-8
- read-only tablespaces
 - as a best practice, 13-4
- refresh, 10-1
- refreshing
 - data warehouse, 10-1
- resource consumption, 12-5
 - minimizing, 12-5
- Resource Manager, 11-3
- rewrite
 - query, 2-8, 2-9
- RMAN
 - as a best practice, 13-3
- roles
 - dimension roles, 4-6
 - in data warehouse security, 14-2
- rolling windows, 10-5
- ROLLUP function, 9-2
- runtime performance, improving, 6-1

S

- schedules
 - creating, 7-4
 - defining, 7-4
 - using, 7-4
- scheduling
 - about, 7-3
- security
 - data warehouse, 14-1
- setting
 - mapping properties, 5-10
- sparsity, 9-13
- SQL Access Advisor, 12-4
 - evaluating partitioning, 12-6
 - optimizing indexes, 11-4
- SQL Tuning Advisor, 12-4
- statistics
 - optimizer, 12-1
- storage
 - optimizing, 11-5
- storage management
 - manual, 2-4
- synchronizing

- operators and Repository objects, 5-12

T

- tables
 - See also* external tables
- tools
 - for data warehouses, 1-4
- transparent data encryption
 - in data warehouses, 14-4

V

- virtual private database
 - in data warehouses, 14-2
- VPD
 - how it works, 14-3

W

- wait events, 12-10
- WITH clause, 9-14
- wizards
 - Create External Table Wizard, 4-3

