Oracle® Warehouse Builder™

The Oracle Data Mart Cookbook

Release 2.1.1

March 1, 2001

Part No. A86690-01

ORACLE®

The Oracle Data Mart Cookbook, Release 2.1.1

Part No. A86690-01

Release 2.1.1

Copyright © 2001, Oracle Corporation. All rights reserved.

Authors: Oracle Warehouse Builder Documentation Team

Contributors: Eric Cao, Bill Friend, Daniel Gallagher, Yu Gong, Gary Glenn Graham, Dirk Grissett, Olaf Fermum, Josephine Ho, Gary Hostetler, Jeff Hutchins, Qiming Huang, Gregg Jackson, James Jin, Josef Hasenberger, James Jin, Jim Lorenz, Tim MacDondald, Mac McDermid, Edwin Meijer, Paul Mitchell, Sridhar Nanjundeswaran, Zhenhai Lin, Xuelin Lu, Ganesh Seetharaman, Jaime Singson, Elina Sternik, VeenaVani Revanuru, Ramesh Uppala, Hua Yao, Joseph Zheng

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

This Program contains proprietary information of Oracle Corporation; it is provided under a license agreement containing restrictions on use and disclosure and is also protected by copyright, patent and other intellectual property law. Reverse engineering of the software 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. Oracle Corporation does not warrant that this document is error free. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Oracle Corporation

If this Program is delivered to a U.S. Government Agency of the Department of Defense, then it is delivered with Restricted Rights and the following legend is applicable:

Restricted Rights Legend Programs delivered subject to the DOD FAR Supplement are "commercial computer software" and use, duplication and disclosure of the Programs shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement. Otherwise, Programs delivered subject to the Federal Acquisition Regulations are "restricted computer software" and use, duplication and disclosure of the Programs shall be subject to the restrictions in FAR 52.227-14, Rights in Data -- General, including Alternate III (June 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

Oracle, Oracle Call Interface, Oracle Developer, Oracle Forms, Oracle Reports, and SQL*Plus are registered trademarks of Oracle Corporation. Net8, Oracle8*i*, PL/SQL, Pro*C, Pro*C/C++ are trademarks or registered trademarks of Oracle Corporation. All other company or product names mentioned are used for identification purposes only and may be trademarks of their respective owners.

Contents

Send Us Your Comments xv

About the Cookbook xvii

Part I The Preliminaries

1 The Recipe

The Recipe 1-2
Ingredients 1-3
One Star Schema 1-3
Data Mart or Data Warehouse 1-3
Dependent and Independent Data Marts 1-4
Star Schemas 1-5
A Complete Set of Data Sources 1-8
Data Sources 1-8
Data Audit 1-8
A Set of Mapping Rules 1-9
An Oracle8*i* Database Configured for Data Warehousing 1-9
Configuration 1-9
Required Tools 1-10
Oracle Warehouse Builder 1-10
Builder Client 1-11
Repository Assistant 1-11
Runtime Assistant and Audit Viewer 1-11
Oracle Enterprise Manager 1-11
Enterprise Manager Console 1-12
Job Library 1-12
Publication Tools 1-13
Oracle Reports 1-13
Oracle Discoverer 1-13

The Instructions 1-14
The Case Study 1-14
Create Definitions 1-14
Configure, Validate, and Generate Scripts 1-15
Deploy the Data Mart 1-16
Publish the Data Mart 1-16
The Next Step 1-17

2 The Case Study

The User Requirements 2-2
The General to the Specifics 2-2
The Typical Questions 2-3
The Indispensable Reports 2-3
Iterative Process 2-5
The Star Schema 2-6
Basic Dimensional Design 2-6
Identify the Business Process 2-6
State the Grain 2-7
Describe the Dimensions 2-8
Describe the Facts 2-13
The Sources of Data 2-15
The Principal Data Source 2-15
The Order Entry System 2-15
Initial Inventory 2-18
The Secondary Sources 2-19
The Final Inventory 2-19
High Level Maps 2-20
The Three Steps 2-20
Load the Dimensions 2-20
Load the Staging Table 2-20
Load the Fact Table 2-20
The Basic Configuration 2-22
The Next Step 2-24

Part II The Definitions

3 The Containers

The Basic Steps 3-2
Logon to a Project 3-3
Logon Sequence 3-3
Quick Tour of the Builder Console 3-4
Project Environment 3-5
Commit Button 3-5
A Container for the Target Schema 3-6
Notes on Wizards, Navigation Trees, and Names 3-8
Navigation and Help 3-9
Search a Navigation Trees 3-10
Logical and Physical Names 3-11
A Container for the Database Source Definitions 3-12
The Example Source Database 3-12
Create the Source Module 3-13
A Checkpoint 3-19
A Minor Diversion 3-20
A Container for the Flat File Source Definitions 3-21
Current State of the Project 3-23
The Next Step 3-24

4 Target Definitions

The Basic Steps 4-2
Dimensions 4-3
The Products Dimension 4-3
Create the Definition 4-4
Update the Definition 4-10
Review the Definition 4-13
Create Definitions for the Remaining Dimensions 4-16
The Days Dimension 4-16
The Customers Dimension 4-17
The Channels Dimension 4-17

Fact Tables 4-18
The Sales Fact Table 4-18
Create the Definition 4-19
Update the Fact Table Definition 4-22
Materialized Views 4-24
The Product_Summaries Materialized View 4-24
Create the Definition 4-25
Update the Definition 4-28
The Customer_Summaries Materialized View 4-29
Conventional Views 4-30
The DeskTops View 4-30
Create the View 4-31
Sequence Objects 4-33
Current State of the Project 4-34
The Next Step 4-35

5 Source Definitions

The Basic Steps 5-2
The Database Sources 5-3
Import the Definitions 5-3
Update the Source Definitions 5-5
The Flat File Sources 5-9
Create Format Definitions 5-9
Update the Format Definitions 5-18
Current Status of the Project 5-20
The Next Step 5-21

6 The Mappings

The Basic Steps 6-2
Two Step Process 6-2
The Case Study Mappings 6-3
Mappings for Dimensions 6-4
The Customers Mapping 6-4
Create the Definition 6-4
Specify a Primary Source 6-5

Describe the Extract Transformations 6-8
Update the Definition 6-13
The Products Mapping 6-14
Create the Definition 6-14
Describe the Extract Transformations 6-15
Mapping Definitions for the Other Dimensions 6-18
Mapping for the Staging Table 6-18
The Basic Sequence 6-18
Clone the Table Definition 6-19
Complete the Table Definition 6-20
Complete the Mapping Definition 6-23
Two Steps Out of Three 6-27
Mapping for the Sales Table 6-28
Five Basic Steps 6-29
Create Definitions for the Custom Transforms 6-30
Create the Transformation Categories 6-30
The Subtract Transform 6-31
The Truncate_Table Transform 6-32
Create the Mapping 6-33
Create Definition for the High-Level Mapping 6-33
Specify the Primary Source and the Target 6-34
Define the Copy Transforms 6-35
Include the Subtract Transform 6-37
Include the Post Processing Cleanup Transform 6-40
Update a Mapping 6-43
Detail Mapping Property Sheet 6-43
Detail Mapping Editor 6-43
Current State of the Project 6-44
The Next Step 6-45

Part III Deploy the Data Mart

7 The Physical Environment

The Basic Steps 7-2
The Basic Operating Environment 7-2

The Components ... 7-3
The Schema Objects ... 7-4
Batch User Name for OEM ... 7-5
Create Windows NT Batch User ... 7-5
Configure Preferred Credentials ... 7-7
The Schema and Runtime Tables ... 7-9
Tablespace Allocation ... 7-9
Create the Schema Object ... 7-9
Current State of the Project ... 7-13
The Next Step ... 7-14

8 Configure a Physical Instance

The Basic Steps ... 8-2
Three Steps ... 8-2
Access Configuration Parameters ... 8-2
Configure the Instance ... 8-4
OEM Registration ... 8-4
Oracle Management Server ... 8-4
Intelligent Agent ... 8-5
Target Schema ... 8-6
Generation Target Directories ... 8-7
Configure the Database Links ... 8-8
Create Additional Indexes ... 8-9
Case Study Indexes ... 8-9
Define Bitmap Indexes ... 8-10
Configure Bitmap Indexes ... 8-12
Validate the Definitions ... 8-15
Validation Results ... 8-15
Valid and Invalid Definitions ... 8-17
Valid Definition ... 8-17
Invalid Definition ... 8-18
Validate a Set of Definitions ... 8-18
Current State of the Project ... 8-22
The Next Step ... 8-23

9 Generate and Deploy

The Basic Steps 9-2
Generate the Scripts 9-2
Script Generation 9-2
The Generated Scripts 9-4
Database Links 9-4
Dimensions 9-5
Loader Mappings 9-7
Materialized Views 9-7
PL/SQL Libraries 9-8
PL/SQL Mappings 9-8
Sequences 9-8
Tables 9-9
Configured Indexes 9-9
Deploy the Scripts 9-10
The Top Directory 9-10
Deploy All the Scripts 9-11
Deploy the Target Objects 9-13
Deploy All the Objects 9-13
Register the Jobs 9-15
Current State of the Project 9-16
The Next Step 9-16

Part IV Publication

10 Populate the Data Mart

The Basic Steps 10-2
Initial Population of the Data Mart 10-2
The Job Sequence 10-2
Run the Jobs 10-4
Background Services 10-4
Schedule 10-5
Builder Runtime Audit Viewer 10-7
Two Kinds of Jobs 10-8

Complete the Initial Load 10-11
The Next Step 10-12

11 Standard Reports

The Role of Oracle Reports 11-2
Scalable 11-2
Components 11-3
Build a Report 11-3
The Report Builder 11-4
Call the Report Builder Wizard 11-4
Identify the Data Sources 11-6
Quarterly Aggregates 11-9
Identify the Quarter and Sales Threshold 11-12
Order the Products Ascending 11-13
Display Columns 11-14
Generated Query Text 11-15
Complete the Report 11-16
Format the Report 11-18
Update the Report 11-20
Deploy the Report 11-22
The Next Step 11-22

12 Ad Hoc Analysis with Discoverer

Oracle Discoverer and the Data Mart 12-2
Discoverer User Edition 12-3
End User Layer 12-3
Discoverer Administration Edition 12-3
Discoverer Viewer 12-4
Design a Business Area 12-4
Create and Manage a Business Area 12-4
Manage End-User Access 12-5
Oracle Discoverer and Query Performance 12-6
Query Prediction 12-6
Query Governor 12-6
Client-Side Cubic Cache 12-6

The End User Layer 12-7
Install the End User Layer 12-7
Create and Transfer a Business Area 12-10
Create the Business Area 12-10
Transfer the Business Area 12-12
Import the Business Area 12-16
Grant Access to the Business Area 12-18
Grant Access 12-18
Grant Discoverer Privileges 12-19
Customize the Business Area 12-20
Rename Folders and Add Descriptions 12-20
Rename an Item 12-21
Hide Items in the Business Area 12-23
Format Currency 12-25
Create a New Item for Calculations 12-25
Defining a New Item Class 12-27
Create a Condition 12-28
Create a Business Area 12-30
Analyze the Data 12-34
Types of Reports 12-34
Table Format 12-34
Page-Detail Table Format 12-35
Crosstab Format 12-35
Page-Detail Crosstab Format 12-36
Graphical Format 12-37
Modifying Formats 12-37
Drill Down and Drill Up 12-37
Investigate the Data 12-38
Where To Begin 12-39
Access the Data Mart 12-39
Create a Report 12-40
Reformat a Report 12-42
Create a Page-Detail Crosstab Report 12-44
Drill Down 12-45
Create Graphs 12-47

Drill to Detail 12-49
Drill Out 12-49
Custom Formats 12-49
Publish Your Work for Others to View 12-50
Export to Excel 12-50
Export to HTML 12-50
Export to Other File Formats 12-50
Schedule a Workbook 12-51
The Next Step 12-52

13 Manage the Data Mart

The Basic Steps 13-2
Periodic Updates 13-3
Data Mart Refreshes 13-3
Refresh a Dimension 13-3
Basic Methods 13-4
Refresh the Products Dimension 13-5
After the Refresh 13-10
Fact Table Updates 13-11
Schedule Jobs 13-12
Physical Storage Growth 13-13
DBA Studio 13-13
Space Utilization 13-14
Add a Datafile 13-14
Data Mart Performance 13-17
Implementation of Indexes 13-17
Dimension Table Indexes 13-17
Fact Table Indexes 13-18
Utilization of Materialized Views 13-18
Query Rewrite 13-19
Enterprise Manager Summary Advisor 13-20
Configuration of Oracle8*i* 13-21
The Initialization File 13-21
Obvious Omissions 13-23
The Next Step 13-24

14 Preserve Data Mart History

The Basic Method 14-2
The Key Change 14-2
Implementation Steps 14-3
Change the Unique Key Constraint 14-4
Create and Load prodSTAGE 14-4
Create the Lookup Table 14-4
Create a Transformation to Load the Lookup Table 14-6
Create a Sequence for the Products Warehouse Key 14-7
Create a Mapping to Refresh the Dimension 14-7
Update stageONE Constraints 14-9
Update the Cleanup Transformation 14-10
Create a Mapping that Updates the Fact Table 14-10
Generation Steps 14-12
Deploy the Data Mart 14-12
Load the Data Mart 14-12
The Final Step 14-13

A The Case Study Environment

The Windows NT Host A-2
The Base Oracle8i Database Instance A-3
The Default Home A-4
The ora816 Home A-5
Typical Installation A-5
Oracle Initialization Parameters A-6
The owb21 Home A-7
The Repositories A-8
Oracle Enterprise Manager Repository A-9
Oracle Warehouse Builder Repository A-11
Create a Tablespace for the Repository A-11
Create the Builder Repository A-11
Create the Builder Project A-13
Install Publication Products A-15
Oracle Reports 6i A-15
Oracle Discoverer 3.1 ..... A-15

The Database Source A-16

B Materials for the Case Study

The Data Sources B-2
The GCC Database Application B-2
The Flat File Sources B-6
The Target Schema B-8
The Data Mart Objects B-8
Dimension Objects and Tables B-10
Channels Dimension B-10
Customers Dimension B-11
Days Dimension B-12
Products Dimension B-13
Fact and Staging Tables B-14
Sales Fact Table B-14
stageONE Table B-16
Sequence Objects B-17
Views B-18
Customer_Summaries Materialized View B-19
DeskTops Conventional View B-20
Product_Summaries Materialized View B-21

Glossary

Index

Send Us Your Comments

Oracle Warehouse Builder User's Guide, Release 2.1.1

Part No. A86690-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail - dwhdoc@us.oracle.com
- Postal service:
  Oracle Corporation
  Data Warehousing Program Office
  MS 2op7
  500 Oracle Parkway
  Redwood Shores, California 94065
  U.S.A.

If you would like a reply, please give your name, address, and telephone number below.

If you have problems with the software, please contact your local Oracle World Wide Support Center.

About the Cookbook

This cookbook shows you how to build a complete data mart using Oracle Warehouse Builder.

A complete data mart includes more than a star schema designed for query performance and a set of jobs designed to extract, transform, and load data from legacy sources. A complete data mart also includes the infrastructure necessary to operate the data mart on a daily basis and keep it refreshed with current data. Finally, a complete data mart provides access to standard reports and to analytic tools that allow users to query the contents of the data mart in an ad hoc fashion.

This cookbook uses a single case study to illustrate the design and implementation of a typical data mart. The examples step you through a proven recipe based on known best practices which are used daily in the field. As you work through these examples, you learn how to develop a logical model for a data mart, how to generate a set of scripts that create and run the data mart, how to create all the schema objects for a data mart including dimensions, facts, and materialized views, and how to manage the data mart using Oracle Enterprise Manager. You also learn how to publish the data mart's contents by setting up a reporting and ad hoc query environment. The approach is strictly hands-on.

The Cookbook Audience

This book was developed primarily for anyone who would like to develop and implement a data mart using Oracle Warehouse Builder. This covers a wide variety of individuals and includes:

- Business analysts on the look-out for solutions to data retrieval.
- MIS professionals such as application programers and developers, database administrators, systems administrators, and network administrators
- Data mart developers including data analysts and those who develop extract, transform, and load routines
- System Integrators and developers of large-scale products based on data marts

Generally speaking, this book defines and illustrates most of the technical terms and concepts when they first occur in the text. The book does assume, however, that you understand the underlying operating system, that you are familiar with basic system administration tasks, and that you have some knowledge of relational database systems.

Related Documents

For additional information on data marts and how to build and operate them, see the following Oracle guides:

- *Oracle Warehouse Builder User's Guide Release 2.1.1*
- *Oracle8i Data Warehousing Guide Release 2 (8.1.6)*

Oracle provides additional information sources, including other documentation, training, and support services that can increase your knowledge and understanding of Oracle Warehouse Builder.

You can order Oracle documentation by phone or using the World Wide Web:

Phone:

Call 800-252-0303 or 650-506-1500 to order documentation or request a fax listing of orderable Oracle documentation.

Oracle Documentation Sales Web site:

http://www.oracle.com/documentation/sales/

Oracle Customer Support Web site:

http://www.oracle.com/siteadmin/html/ask_oracle.html

How the Cookbook Is Organized

The first part of the *Data Mart Cookbook* describes a proven recipe and a case study. The recipe has three instructions, and the remaining three parts of the cookbook show you how to follow these three instructions respectively.

Part I, "The Preliminaries" Chapters in this part introduce you to a recipe for building a data mart using Oracle Warehouse Builder and a case study that serves as the basis for a progressive set of examples.

Chapter 1, "The Recipe" This chapter introduces a basic recipe for developing and building a data mart using Oracle Warehouse Builder. The cookbook follows this recipe and uses a series of examples to walk you through the development and construction of a complete data mart. Once you understand the recipe, you can use Oracle Warehouse Builder to build nearly any kind of data mart.

Chapter 2, "The Case Study" This chapter introduces the case study which is the basis for all the examples. The case study is based on the "Global Computing Company," a fictitious company that markets computer hardware and software components around the world through several sales channels including the Internet. The company tracks and manages sales with an online order entry system called "GCCAPPS" that resides on an Oracle8*i* database instance.

Part II, "The Definitions" The recipe's first instruction directs you to create a set of formal definitions that completely describe your data mart. This part of the cookbook shows you how to create definitions for the Global Computing Company Data Mart using Oracle Warehouse Builder. The definitions are stored and maintained in a Builder repository which is an Oracle8*i* schema object.

Chapter 3, "The Containers" This chapter shows you how to create three containers in a Builder Project where you can store and maintain the definitions for the case study data mart. This step is a necessary prerequisite to using Oracle Warehouse Builder.

Chapter 4, "Target Definitions" This chapter shows you how to create, store, and update formal definitions for the case study's star schema. This schema was developed and described in Chapter 2.

Chapter 5, "Source Definitions" This chapter shows you how to create definitions for the case study's data sources using Builder wizards. These sources were summarized in Chapter 2.

Chapter 6, "The Mappings" This chapter shows you how to create and update definitions for operations that extract data from a source, transform that data as necessary, and then load the data into the target schema. These definitions are called "mappings," and each operation within a mapping is called a "transformation." The mappings are stored in the warehouse module that defines the target schema.

Part III, "Deploy the Data Mart" This part of the cookbook follows the recipe's second instruction and shows you how to deploy a physical instance of the data mart from the logical model defined by the gccDM warehouse module.

Chapter 7, "The Physical Environment" This part of the cookbook follows the recipe's second instruction and shows you how to deploy a physical instance of the data mart from the logical model defined by the gccDM warehouse module.

Chapter 8, "Configure a Physical Instance" This chapter shows you how to configure a physical instance of a data mart from the definitions stored in the gccDM warehouse module. These definitions currently describe a logical model for a data mart which can be physically realized in several ways. The examples in this chapter show you how to configure a physical instance for the operating environment described in the previous chapter. The chapter also shows you how to validate a set of definitions stored in a warehouse module.

Chapter 9, "Generate and Deploy" This chapter shows you how to generate a set of scripts to create and load the physical instance of the Global Computing Company Data Mart. Builder generates these scripts from the set of configured definitions that reside in the gccDM warehouse module. The chapter also shows you how to deploy the generated scripts to a local directory, register the scripts with Oracle Enterprise Manager, and then create all the target schema objects.

Part IV, "Publication" This part of the cookbook, the concluding part, shows you how to publish the contents of the Global Computing Company Data Mart. This is accomplished in four steps. First, you run the jobs that initially populate the data mart. Second, you create standard reports and make them available to the users. Third, you provide query access to the data mart with query tools. Finally, you set up the processes and scripts necessary to run the data mart on a daily basis.

Chapter 10, "Populate the Data Mart" This chapter shows you how to set up and schedule jobs that initially populate the data mart. You must run these jobs according to a strict sequence and, after each job terminates, you must verify the completion status of each job. If a job terminates abnormally, you must take corrective action before you allow users access to the data mart.

Chapter 11, "Standard Reports" This chapter shows you how to generate, store, and distribute standard reports using Oracle Reports.

Chapter 12, "Ad Hoc Analysis with Discoverer" This chapter shows you how to set up Oracle Discoverer so that users can access the data mart using Discoverer End User Edition, a powerful query and analysis tool. The Chapter describes Oracle Discoverer, shows you how to create an End User Layer using Discoverer Administration Edition, reviews the fundamentals of querying and reporting, and then walks you through a typical exploration of data stored in the case study data mart.

Chapter 13, "Manage the Data Mart" This chapter gives you a few tips on how to manage the data mart. This includes showing you how to develop a set of jobs to periodically refresh the data mart, how to schedule those jobs in the correct sequence, how to keep an eye on the growth of physical storage, and how to keep a handle on performance. Several of these tasks are illustrated using Oracle Enterprise Manager.

Chapter 14, "Preserve Data Mart History" This chapter shows you how to revise the case study data mart so that it preserves the historical record over time. These revisions allow the Products dimension to track changes in the product descriptions by inserting a row in the dimension each time a description changes. The latest row inserted always contains the current descriptions of the product.

Appendix A, "The Case Study Environment" This appendix shows you how to set up a Windows NT system so that you can work through all the examples in the cookbook. The directions for the product installations follow the simple and straightforward path rather than the path that would install a complete system optimized for a data mart or a data warehouse. For example, the section on the Oracle8*i* Database Instance directs you to perform a "Typical" installation. In practise, you would more likely select the "Data Warehousing" button for the type of installation.

Appendix B, "Materials for the Case Study" This cookbook uses a set of examples that progressively walk you through the construction of a data mart from start to finish. These examples show you how to use Builder for specific tasks (such as create a dimension) but do not show you how to create every dimension required to implement the GCC Data Mart. Thus, many tasks are intentionally left as exercises for the reader. Moreover, many other examples remain at a summary level and omit many details. This appendix is intended for those readers who would like to actually build the entire data mart using Oracle Warehouse Builder.

Part I

The Preliminaries

Chapters in this part introduce you to a recipe for building a data mart using Oracle Warehouse Builder and a case study that serves as the basis for a progressive set of examples.

This part includes two chapters:

- Chapter 1, "The Recipe"
- Chapter 2, "The Case Study"

1

The Recipe

This chapter introduces a basic recipe for developing and building a data mart using Oracle Warehouse Builder. The cookbook follows this recipe and uses a series of examples to walk you through the development and construction of a complete data mart. Once you understand the recipe, you can use Oracle Warehouse Builder to build nearly any kind of data mart.

The recipe shows you how to build a *complete* data mart, one that includes the schema, the scripts required to populate the mart initially as well as update it periodically, and tools required to manage the data mart after it goes into operation. A complete data mart also includes the tools users need to access the published data. These client or web browser tools allow users to query the data in an ad hoc fashion or analyze standard reports.

This chapter covers the following topics:

- The Recipe
- Ingredients
- Required Tools
- The Instructions

The Recipe

The recipe contains a list of ingredients, a set of tools, and a set of instructions. The remaining sections of this chapter develop the proper context for the recipe.

Table 1–1 Recipe for a Data Mart

INGREDIENTS
■ One star schema for a data mart
■ A complete set of data sources
■ A set of mapping rules from the source data to the data mart
■ An Oracle8*i* database server configured and equipped for running a data warehouse (Enterprise Edition preferably)
TOOLS
■ Oracle Warehouse Builder Client and a Warehouse Builder repository Repository Assistant Runtime Assistant Runtime Audit Viewer
■ Oracle Enterprise Manager Console and Enterprise Manager repository Oracle Management Server
■ Publication Tools: Oracle Reports, Discoverer, and Portal
INSTRUCTIONS
1. Create definitions for the star schema, the individual data sources, and the extract, transform and load routines. These definitions constitute a logical model of the data mart.
2. Configure, validate, and generate scripts from these definitions and then use the generated scripts to deploy the data mart.
3. Populate the data mart and publish its contents.

Ingredients

The list of ingredients calls for a star schema for the data mart, a complete set of data sources, a complete set of extract, transform, and load routines ("ETL routines"), and an Oracle8*i* database configured and equipped as described under "Tools." This section describes each ingredient in a little more detail.

One Star Schema

The recipe describes how to build a data mart, not a data warehouse. Additionally, the series of examples that show you how to use the recipe to build an *independent* data mart, not a *dependent* one. Moreover, the schema is a specific kind of schema, a star schema commonly used for decision support applications.

Data Mart or Data Warehouse

Conceptually, a data mart can be seen as a subset of a data warehouse, a piece of the overall pie. Whereas a data warehouse supports the analysis of multiple subject areas such as financial analysis, sales analysis, marketing analysis, and clickstream analysis, a data mart restricts the analysis to a single subject area. Say, clickstream.

A data mart tends to be easier to design, build, and manage than a data warehouse because its design rests on a simple star schema. The data mart usually pulls its data from a few sources, requires a basic management infrastructure, and supports a set of users whose needs are, relatively speaking, clearly defined.

For example, you could deploy a data mart to support the analysis of clickstream data fairly quickly because its data could pulled from a single, well-defined source.

Table 1–2 Data Mart Compared with Data Warehouse

Category	Data Mart	Data Warehouse
Scope	Line-of-Business	Corporate
Subjects	Single subject	Multiple
Data Sources	Few	Many
Size (*typical*)	< 100 GB	100 GB - TB+
Implementation Time	Months	Months to years

Although a data mart tends to be easier to develop and implement that a data warehouse, the basic principles of design and implementation are the same.

Also, to say that a data mart is a subset of a data warehouse is not to say that it contains small volumes of data nor that it can not support complex forms of analysis. Data marts often turn into gigantic databases, and its stored data appears as the ultimate ground for many different kinds of analytical studies. The complexity of the analysis usually depends on the analysts and their tools.

For a comprehensive discussion of these topics, see the *Oracle8i Data Warehouse Guide, Release 2 (8.1.6)* and any of Ralph Kimball's *Data Warehouse Toolkit* books (published by John Wiley).

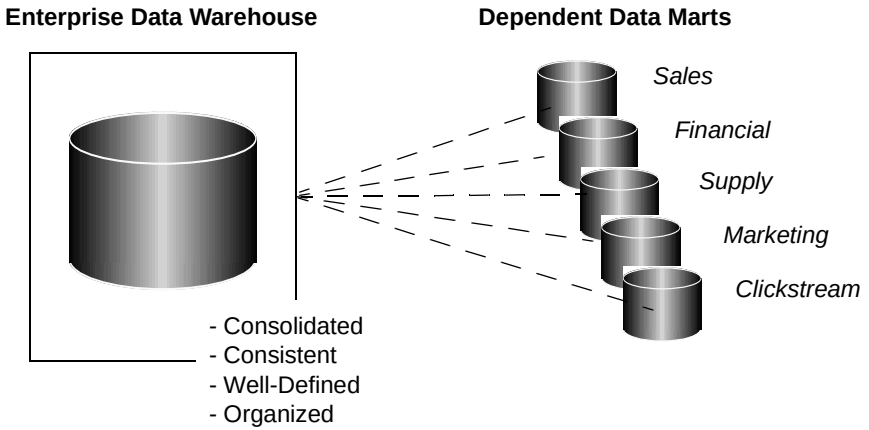
Dependent and Independent Data Marts

Data marts can be divided according to their data sources into three categories. A *dependent* data mart pulls its data from an existing data warehouse. An *independent* data mart pulls its data from legacy and syndicated sources. A *hybrid* data mart pulls its data from a data warehouse and additional sources.

These distinctions are important because most of the effort required to build a data mart or a data warehouse is spent on understanding, reconciling, and formally defining the sources of data. According to most practitioners, this work typically represents approximately seventy to eighty percent of the total effort.

A dependent data mart is usually the easiest to build and maintain because it pulls clean and consistent data directly from an existing data warehouse. All the work required to sanitize the legacy data has already been completed.

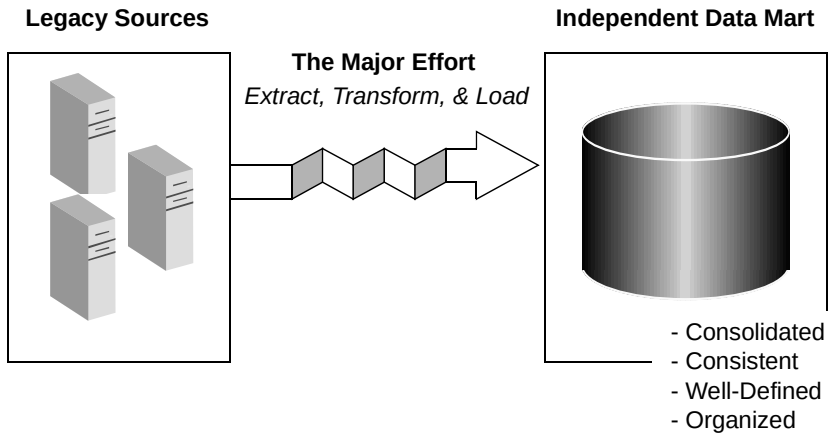
Figure 1–1 Dependent Data Mart



A hybrid data mart requires additional work because it pulls from external sources, and this data must be cleaned and integrated with the data pulled from a data warehouse. The difficulty varies with the number and diversity of external sources.

An independent data mart presents the greatest challenge because each of its data sources must be understood, reconciled, and formally defined. This kind of data mart faces the same major effort as a data warehouse except there are typically fewer, less diverse data sources, and a data mart's schema is less complex.

Figure 1–2 Independent Data Mart



The example in this cookbook shows you how to build an independent data mart, and you can use the same process to build dependent and hybrid data marts, or an enterprise-level data warehouse.

Star Schemas

Most data marts and data warehouses organize their data using a star schema. This kind of schema design is common because:

- Query performance against a star schema is excellent. The design minimizes join operations, and Oracle8*i* recognizes and exploits the star structure.
- Design of a star schema is based on the very measures a user needs to perform an analysis. The design is also readily extensible.

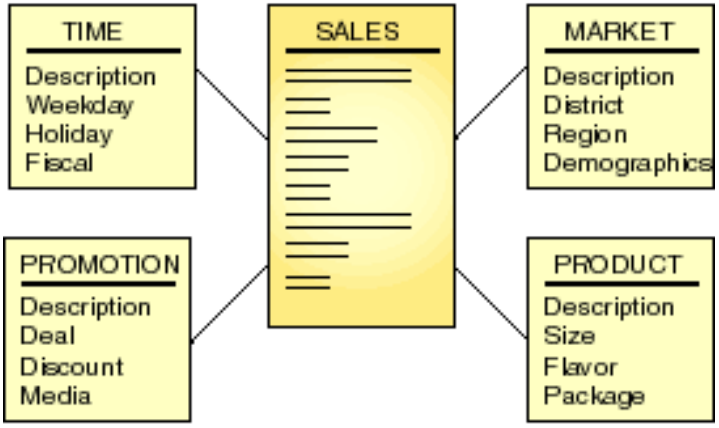
A star schema that meets the basic performance objectives of a user community can be quickly sketched from an understanding of the required measurements, and from this sketch, a good practitioner can readily develop a final schema.

The user measures and what they mean immediately establish a direct connection between the design and the users which minimizes misunderstandings. The design is couched in the terms understood by all members of the user community.

The Design Process The design process for a star schema usually differs from those used to develop other schemas because you already know what the star schema is going to look like, and what gives it a decided performance edge for query processing. Ralph Kimball likes to say that star schemas are all boringly the same. The structure is known up front, only the details are missing.

Star Schema Components The diagram for a star schema has several small dimension tables radiating from a central fact table. The central table usually contains measures *additive* across all dimensions and a set of foreign key references which point to its dimensions. Dimension tables provide the context for the measures stored in the fact table. Fact tables usually contain nothing but numbers; the dimensions give meaning to that meaningless mass of numbers.

The example below includes a single fact table named Sales with four dimensions. Lines radiating from the fact table represent foreign key references.



You can see how the dimensions give meaning to the numbers by looking at a single fact table row.

TIME_fkr	PROM_fkr	MKT_fkr	PROD_fkr	Sales	Units	Cost
107	003	275	1001	45000	100	15000

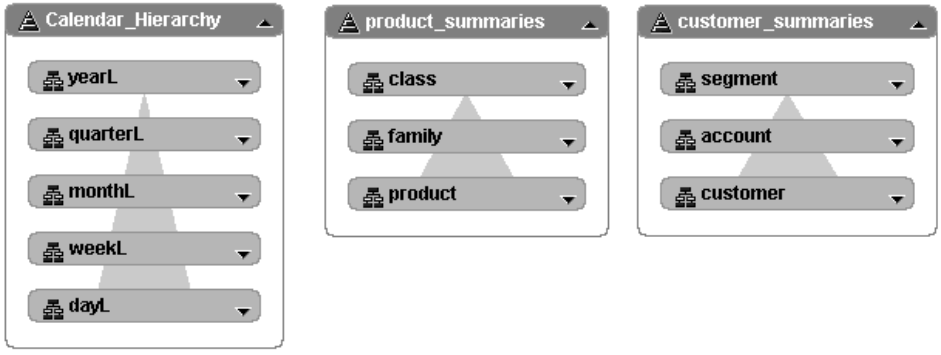
The values in the foreign key reference columns (PROD_fkr) link the row to meaningful descriptions. For example, the above row points to row 1,001 of the

Product dimension table, a row that completely defines a specific product and all of its attributes (size, color, model, features, …). Similarly, the value in the TIME_fkr column completely defines the date on which the sale took place.

Fact tables contain measures and references to a set of dimension tables. For the most part, a fact table's measures are *additive* across all dimensions. Common measures include dollars, units, and cost. Some fact tables, however, include measures that are not additive across all dimensions but only across a few. Semi-additive measures, as they are called, include margins, averages, and balances. Some fact tables contain no measures whatsoever, only references to a set of dimensions. An event table is often given as an example.

Dimension tables contain a rich set of descriptive details that completely describe their objects. Dimensions also determine the fastest access path to the measures stored in a fact table. For example, a user looking for the sales of cordless vacuum cleaners in Miami during the summer of '97 can pick out a vacuum cleaner type from the *Products* dimension, a city 'Miami' from the *Customers* dimension, and a year from the *Days* dimension. Oracle8*i* can then return the corresponding answer set quickly because the query plan for attacking a dimensional model is utterly predictable: constrain the dimensions, exploit the indexes, and parallelize the operation when optimal.

Query performance can be further improved in many cases because users often analyze data by drilling down known hierarchies using slice-and-dice techniques. For example drilling down a temporal hierarchy Year > Quarter > Month > Day and a brand hierarchy Category > Brand > Product. When these hierarchies obtain in the data, you can create *dimension objects* to codify the relationships as levels of aggregation. Oracle8*i* can then exploit defined hierarchies by rewriting queries to fetch data from summary rather than detail tables. The rewritten queries run much faster. In browser land, *speed* makes the difference.



A Complete Set of Data Sources

Dependent and hybrid data marts are usually built to take advantage of distributed processing: better local performance, reduced telecommunication costs, improved availability, and a greater degree of local control. Identifying the data for these kinds of marts is simple and straightforward because they pull their data from an existing data warehouse.

An independent data mart is usually built because a set of users can not analyze data that resides in an existing operational or legacy system. The queries required by the analysis are either too difficult to write, take too long to run, or impact the performance of the operational system. Often, the operational data is incomplete because a few data elements reside in flat files. The independent data mart solves this problem of access by pulling data out of the operational system and flat files, and then organizing it using a star schema.

Data Sources

Typical data sources include:

- Operational systems that reside on relational database systems
- Systems that depend on a series of flat (sequential) files
- Legacy mainframe systems
- Syndicated data providers
- Web server logs (Apache)

Data marts may pull their data from external sources such as spreadsheets or other localized data although seldom is this the case.

Data Audit

The data sources for a data mart must be identified, analyzed, thoroughly understood and well-documented. The data sources must also be complete. That is, you must be able to define the source for each column in the data mart whether that source is a raw source item or a derived value (transformed).

Take Note:

If you can not identify a source for a data item in the data mart, you must return to the user community and modify the set of user requirements or identify a source.

A Set of Mapping Rules

Once a star schema for the data mart has been finalized and the data audit is complete, you sketch a set of mapping rules which explicitly associate each data item in the data mart with its source.

The source for an item in the data mart can be a raw item in a defined source or an item derived from previously defined items. A product name in a source table, for example, can be mapped directly to a product name defined in the data mart. Or, the profit margin can be calculated from sales and cost columns defined in the data mart. Margin could also be defined using raw source fields.

These informally stated rules confirm the results of the data audit and are used to create formal mapping rules with Oracle Warehouse Builder when you are ready develop formal definitions for the data mart.

An Oracle8*i* Database Configured for Data Warehousing

The recipe calls for an Oracle8*i* database server, preferably the Enterprise Edition, equipped with a specific set of tools and configured for data warehousing. The data mart and its infrastructure are built on this server.

Configuration

Configure the Oracle8*i* server using the "Data Warehouse" option. This sets the blocksize larger for improved query performance and enables query rewrite, materialized views, and star transformation parameters. These parameters are specific to a data warehousing versus a transaction application.

Block Size The data block size should be at least 16k to account for the expected transfer of large amounts of data during query processing.

Query Rewrite When you enable query rewrite, the Oracle8*i* optimizer includes rewritten queries in its list of alternatives. The original query is rewritten to retrieve data from summary tables (materialized views). Often, the rewritten query is much faster than the query that references detail tables (QUERY_REWRITE_ENABLED & OPTIMIZER_MODE).

Star Transformation The Oracle8*i* optimizer includes a star transformation alternative when it detects that a query references a star schema. This algorithm can speed up query processing considerably (STAR_TRANSFORMATION_ENABLED). This algorithm and bitmap indexes are available only in the Enterprise Edition of Oracle8*i*.

Parallel Operation When the data mart is large and supports numerous users, configure the Oracle8*i* server for parallel operation. As this can be quite complicated, refer to the *Oracle8i Data Warehouse Guide, Release 2 (8.1.6)* for specific and detailed instructions.

Required Tools

The recipe calls for three sets of tools: one set to build the data mart, another to load and manage the data mart, and a third to publish its contents. The first two sets are tools shipped with Oracle8*i*, the third set consists of three Oracle products: Oracle Reports, Oracle Discoverer, and Oracle Portal. The latter set of tools *are* one part of a complete data mart.

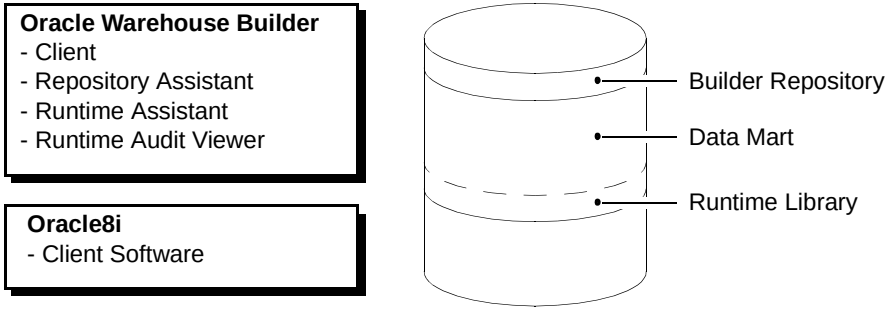
Oracle Warehouse Builder

Oracle Warehouse Builder is a comprehensive and powerful set of tools which are used to complete several interrelated tasks:

- Define and maintain logical models for data warehouses of any scale, from a dependent data mart to an enterprise-level data warehouse.
- Configure and validate physical instances of the logical models.
- Generate scripts to create, load, and manage a physical instance.
- Deploy a physical instance and the job scripts required for its operation.

Warehouse Builder consists of four tools: the Oracle Warehouse Builder client, the Repository Assistant, the Runtime Assistant, and the Runtime Audit Viewer.

Figure 1–3 Set of Tools and Repository



Builder Client

The Builder client is used to define, store, and update formal definitions of logical data warehouse models in an Oracle Warehouse Builder Repository. After you create a logical model, you can use the Builder client to configure and validate a physical instance of the model, and then generate scripts that can be used to create and manage the physical instance.

Of course, before you can create and store definitions for the logical warehouse model, you must first create a Warehouse Builder Repository.

Repository Assistant

The Oracle Warehouse Builder Repository is a schema object that resides in an Oracle8*i* database instance. When you create a repository, you must supply the schema owner's username and password.

Use the Oracle Warehouse Builder Repository Assistant to create and drop a Builder Repository. When called, this utility returns a multiple-page Repository Wizard that steps you through the entire process.

Runtime Assistant and Audit Viewer

The Oracle Warehouse Builder Runtime Assistant creates a set of tables that are used to trap runtime statistics (informational, warning, and error) during a load of the data mart. These tables must be deployed to the same Oracle8*i* database instance that hosts the data mart, preferably within the same schema object.

You view and interrogate the contents of these tables using the Oracle Warehouse Builder Audit Viewer.

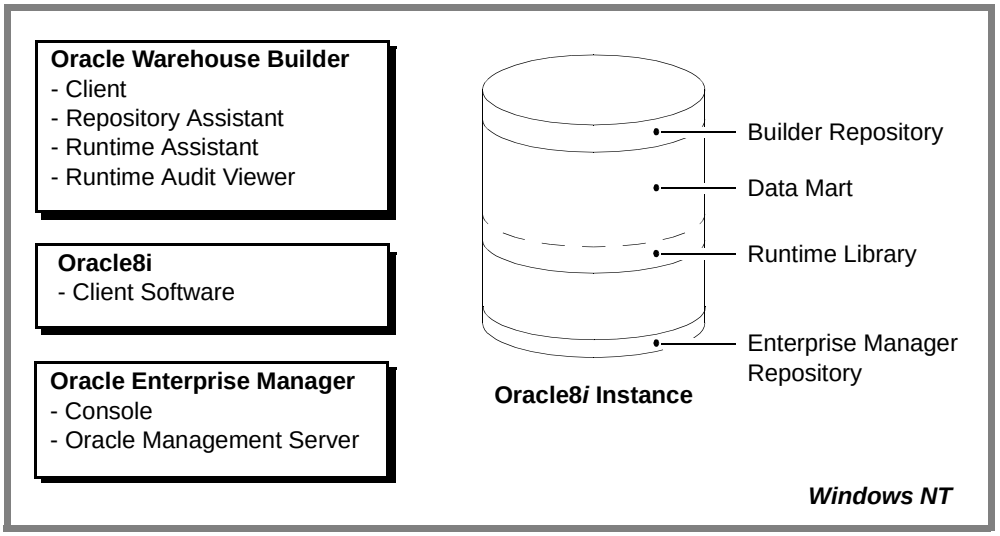
Oracle Enterprise Manager

Oracle Enterprise Manager is a set of database administrative tools which are used to manage small and large Oracle8*i* databases. You can manage a data mart without installing these tools but they greatly simplify and improve data mart management.

To use Oracle Enterprise Manager, you need to install the Oracle Management Server (OMS) and the Enterprise Manager Console (a client). Additionally, you must create, or have access to, an Oracle Enterprise Manager repository.

The data mart built in this cookbook runs on a single Windows NT system as illustrated in Figure 1–4. Appendix A, "The Case Study Environment" summarizes the installation of all the necessary components.

Figure 1–4 Example Data Mart Configuration



Enterprise Manager Console

You can perform the following kinds of tasks from the Enterprise Manager console:

- Start and stop an Oracle8*i* instance
- Create and manage objects within a database schema
- Create and manage database users
- Create and manage physical storage for an Oracle8*i* instance
- Access a schema using the SQL*Plus utility

The Oracle Management Server (OMS) manages these tasks and maintains detailed information about the current state and status of the Oracle8*i* host.

You access the OMS using the Enterprise Manager Console client or using a Browser.

Job Library

Oracle Enterprise Manager maintains a job library within its repository, and you can schedule jobs within this library to run immediately, at a specified time, or on a periodic basis. When you deploy a data mart with Warehouse Builder, you can also deploy a set of job scripts to this library, and use the Enterprise Manager console to schedule these jobs. These jobs initially load and periodically refresh the data mart.

Publication Tools

Publication tools include report generators used to create and distribute standardized reports and query tools to support ad hoc analysis. These tools should be installed early in the implementation process and should be accessible with a common web browser. By installing these tools early in the process, users can have access to the prototype versions of the data mart and provide valuable feedback.

The recipe calls for two publication tools, Oracle Reports and Oracle Discoverer, and a tool that supports browser access, Oracle Portal. The latter tool supports Reports and Discoverer client functionality from a web browser.

Oracle Reports

Oracle Reports supports the rapid development and publication of a wide variety of standardized reports. After you develop and publish standardized reports, users can access the published reports over the web using a web browser.

Oracle Discoverer

Oracle Discoverer is an end-user query tool that allows users to access and analyze information stored in a data mart. This kind of analysis is typically ad hoc.

The Instructions

The recipe's instructions describes how to develop and implement a complete data mart from an existing star schema design using Oracle Warehouse Builder:

1. Create Definitions
2. Configure, Validate, and Generate Scripts
3. Deploy the Data Mart
4. Perform the Initial Load
5. Publish the Data Mart

These step-by-step instructions are formulated at a high level, and each step actually covers a large number of individual actions. For example, the first step directs you to create definitions for each data mart object, each source item, and each mapping and transformation operation.

The subsequent chapters of this Cookbook describe and illustrate the individual actions for each step in detail using, as a case study, the Global Computing Company Data Mart. The next chapter describes the case study and the chapters that follow show you how to implement the case study data mart by following each step of the instructions.

The Case Study

The case study is an independent data mart that pulls its data from two sources: an application system based on an Oracle8*i* database and two flat files. The data mart is based on a simple star schema that contains four dimensions, a fact table, and two materialized views. The data mart is managed by Oracle Enterprise Manager and publishes its contents using Oracle Reports and Oracle Discoverer.

Create Definitions

During the first step you create a set of formal definitions for the data mart using Oracle Warehouse Builder and store these definitions in a Warehouse Builder repository. Before you can create the definitions, you must first create the repository using the Oracle Warehouse Repository Assistant.

You must create definitions for all of the data mart's objects (dimensions, facts, tables, materialized views, sequences, and views), definitions for all of its data sources whether database or flat file sources, and definitions for all of its mapping operations which pull data from the sources, transform the data, and load it into the data mart. The latter set of definitions include those for transformation operations.

This task is iterative and numerous iterations are sometimes required before the definitions converge on a final set. Once definitions are created and stored in the repository, they can be easily modified using Builder.

Configure, Validate, and Generate Scripts

The formal definitions stored in the Builder repository define a logical data mart which can be realized physically in various ways. You can, for example, configure a physical version of the data mart that contains no materialized views and another that contains several materialized views.

During this step you configure a physical instance of the logical data mart, validate the set of definitions, and then generate a set of scripts from the definitions. Builder generates the following kinds of scripts:

Script Language	Description
DDL	Create data mart objects which include database links, dimensions, facts, tables, materialized views, sequence objects, and views.
PLS	Create PL/SQL packages that pull data from database sources, transform the data, and store it in the data mart.
SQL*Loader	Control file that contain a set of directives used by the SQL*Loader to extract data from a file and then load it into the data mart.
TCL	Scripts used to register load operations with Oracle Enterprise Manager's Job Library. You can schedule these jobs using Oracle Enterprise Manager.

These scripts are relative to a given configuration. When you deploy different physical instances, you must configure the instance first, validate the definitions, and then generate a set of scripts.

Deploy the Data Mart

After a set of scripts have been generated for an instance of the data mart, you can use the scripts to deploy the data mart, an operation that requires several steps:

1. Create a database user who will own the data mart schema.
2. Create a runtime library in the data mart schema using the Oracle Warehouse Builder Runtime Assistant.
3. Deploy the data mart objects to the data mart schema.
   - Database links, dimensions, facts, materialized views, sequence objects, tables, and views.
   - PL/SQL packages that implement transformation libraries
   - PL/SQL packages that implement mapping operations
4. Deploy the SQL*Loader and TCL scripts to a configured file system.
5. Register the TCL scripts with Oracle Enterprise Manager's job library.
6. Deploy the Business Areas to the data mart.

These steps assume that the host database for the data mart, an Oracle8*i* instance, is configured for data warehousing and includes the Oracle Enterprise Manager console. This instance must also include or have access to an Oracle Management Server and an Oracle Enterprise Manager repository.

Publish the Data Mart

After the data mart has been deployed, you can perform an initial load of the data mart and then create a set of scripts to refresh the data mart on a periodic basis. The scripts that refresh the data mart depend on how the data mart is to be updated.

The contents of the data mart is delivered through Oracle Reports and Oracle Discoverer using stand alone client tools or a browser. You can publish all the standardized reports required by the user community using Oracle Reports and provide ad hoc query access using Oracle Discoverer.

Of course, the standarized reports must be designed, defined, and created, and an End User Layer must be established before ad hoc access is possible with Oracle Discoverer. You can work on these tasks during the development of the data mart and complete them after the initial load of data.

The Next Step

You can go build a data mart directly from the recipe, or you can continue working through the examples in this cookbook that show you how to build a simple data mart using Oracle Warehouse Builder.

The next chapter, "The Case Study," summarizes how a group of users settled on a star schema for their data mart, and then describes the star schema and the data sources for that data mart.

The remaining chapters of the cookbook develop and implement the this data mart by following the recipe steps. The data mart is called the "Global Computing Company Data Mart," which is abbreviated as gccDM.

2

The Case Study

This chapter introduces the case study which is the basis for all the examples. The case study is based on the "Global Computing Company," a fictitious company that markets computer hardware and software components around the world through several sales channels including the Internet. The company tracks and manages sales with an online order entry system called "GCCAPPS" that resides on an Oracle8*i* database instance.

This chapter begins with a set of requirements for a data mart that were developed by a group of sales analysts at the Global Computing Company. From these requirements, the chapter develops a star schema that meets these requirements. The chapter concludes with a description of the source data required to populate the star schema, initially as well as on a periodic basis.

This chapter follows the dimensional modeling method Ralph Kimball describes in his three "Toolkit" books and uses the vocabulary he adopts. See page 152 of his second book for a brief history of this method and vocabulary that dates to 1982.

This chapter covers the following topics:

- The User Requirements
- The Star Schema
- The Sources of Data
- High Level Maps
- The Basic Configuration
- The Next Step

The User Requirements

The Global Computing Company users would like to analyze product sales using the hard numbers stored in the company's online order entry system. This application system, GCCAPPS, manages and tracks product shipments to all customers. With respect to an individual order, the system stores the number of units of each product sold, the sales amount for each product in dollars, and the estimated cost to manufacture each product.

Currently, the operations group strictly limits access to this system for reasons of performance and security. The access for the sales analysts is so restricted that by the time they get the numbers they need, it is usually too late.

This kind of generally expressed need for access to quantitative data which is "locked-up" in an operational system is probably the most common reason that users are driven to build a data mart.

The General to the Specifics

Before you can develop a star schema for this data mart, you need to translate the general need to access hard sales numbers into a more specific set of user requirements. Which hard numbers do these users specifically need, for example, and *at what granularity*? Do they need daily sales totals by product and customer or weekly? Do they need sales channel too? Or, do they need numbers at a much finer granularity, such as at the individual sales invoice level?

Questions regarding the granularity of the hard numbers are fundamental to star schema design. As you will see, they essentially drive the design process. For example, consider the question:

> Why were product sales in Chicago much greater than elsewhere during June but way below average during September?

This question speaks volumes about the kind of hard numbers and granularity the users need to ground an analysis of sales, and you need to develop a set of questions that users typically ask.

These questions should be augmented with reports, actual or imagined, which are often used during an analysis, tabular reports and graphs that suggest areas for investigation or simply report obvious trends.

The project manager leading the data mart implementation usually collects this kind of information through a series of interviews with small groups and brain-storming sessions with larger ones. These groups include the actual users, the keepers of the corporate data, and members of the information technology staff.

These meetings should produce two kinds of information:

- A representative set of questions which are asked during a typical analysis.
- A set of standardized reports considered indispensable for analysis or for routine reporting. The reports need not exist but can be clearly defined.

This information is highly-suggestive and carries at least the germ of a star schema design (or, a dimensional model).

Later in the project when a proposed star schema is available as a sketch, these questions can serve as a rough measure of the proposed schema's adequacy. If one of the typical questions can not be translated into a query against the proposed schema, then the schema is (obviously) inadequate.

The Typical Questions

After several meetings and numerous scenarios, the Global Computing users settled on a short of list of typical questions:

- What are the top five best-selling products this week, month, quarter, and year?
- What are the five worst-selling products this week, month, quarter, and year?
- Which customers are buying the top five products and through which sales channels for the last fifteen days?
- Do any Zero-Coupon promotions correlate with the best-sellers?
- How do the daily sales of the top-five products during July compare with the bottom-five during the same month?
- How do the weekly sales of the top-five sellers in Chicago compare with the top-five in Seattle during the summer months?

These questions were culled from the large number of candidate questions proposed during the interviews and brain-storming sessions, and everyone agreed that they were typical and representative.

The Indispensable Reports

During the same meetings and brain-storming sessions, the users provided a few reports that were currently in use or would be required for a complete solution.

Figure 2–1, "Best and Worst Sellers" is a tabular report that lists the top and bottom sellers according to product name, units sold, and sales dollar.

Figure 2–1 Best and Worst Sellers



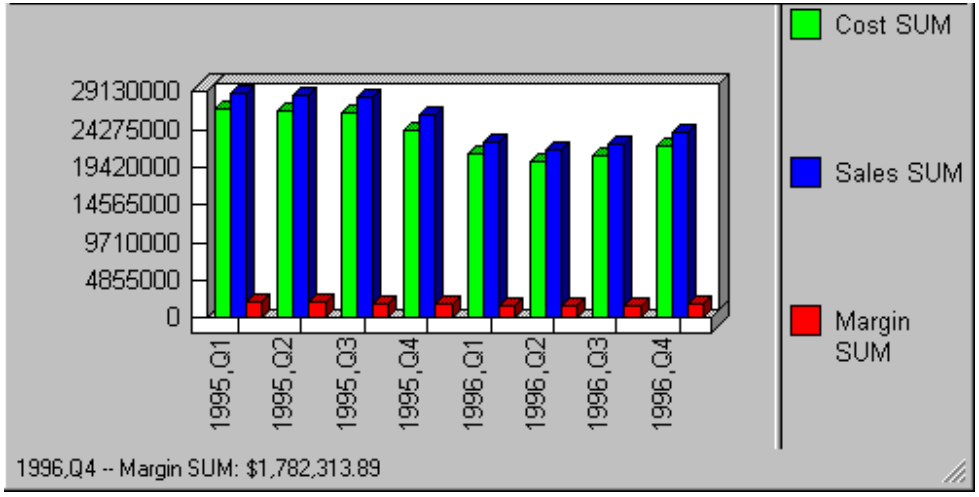
Best & Worst Sellers

Report run on: February 2, 1999 10:46 AM

Item Description	Sum Units	Su
Keyboard Wrist Rest	81263	
Mouse Pad	149220	
Envoy External Keyboard	1302	
O/S Documentation Set - Italian	1792	
O/S Documentation Set - Spanish	3038	
O/S Documentation Set - French	6315	
Standard Mouse	38162	
Internal 6X CD-ROM	4059	
O/S Documentation Set - Kanji	8036	
O/S Documentation Set - German	9787	
Envoy External 6X CD-ROM	3110	
Unix/Windows 5-user pack	15366	

Figure 2–2, "Sales versus Margin Bar Chart" is a bar chart that displays margin versus sales by quarter for two years.

Figure 2–2 Sales versus Margin Bar Chart



In practice, there would probably be a number of standardized reports that would be classified into "Required," "Needed," and "Nice-to-Have." All of these reports regardless of category need to be defined and included as part of the requirements.

Table 2–1

Fitzgerald Worldwide Sales					3/5/99
Eastern Regional Sales Projections					
Northeast	**State**	**FY1996**	**FY1997**	**FY1998**	**YTD1999**
	New York	$10,111	$13,400	$20,900	$12,090
	New Jersey	$22,100	$24,050	$27,890	$14,099
	Rhode Island	$13,270	$15,670	$19,850	$5,671
	Connecticut	$10,800	$21,500	$28,970	$8,277
	Massachusetts	$23,400	$25,600	$26,500	$7,571
	Vermont	$11,700	$12,285	$12,899	$3,686
	New Hampshire	$5,850	$6,143	$6,450	$1,843
	Maine	$4,095	$4,586	$5,504	$1,572
Total		$101,326	$123,234	$148,963	$54,809

Iterative Process

Once the meetings and brain-storming sessions yield a set of typical questions and indispensable reports, the developers can move forward and sketch a star schema that meets these basic needs.

All the users who have attended the interviews and brain-storming sessions must agree that the typical questions and standardized reports represent their fundamental needs for a data mart. While the user requirements process is iterative, the fundamentals will probably not vary greatly, and they mark a sound starting point for the early star schema design.

The developers at this point should move forward with a preliminary star schema design, make a rough sketch of the schema, and circulate the sketch to promote communication and gather feedback during the meetings. A simple sketch will do.

The Star Schema

The process of designing a star schema from a set of user requirements is usually referred to as "dimensional design." This process is driven by the need to quickly access to a set of *measured facts*, such as dollars sold. The star schema that results places those facts in a meaningful context where they can be efficiently retrieved and analyzed. The facts are stored in a central fact table and a set of dimensions couched in user terms place the measures in a meaningful and familiar context. Dimensions are tied to measures over the fact table's foreign key reference columns.

Basic Dimensional Design

Ralph Kimball, in all three of his "Toolkit" books, has consistently summarized four basic steps in the dimensional design process. Depending on the project, these steps can sound artificial, but they always serve as a sound guide whether they are followed strictly in order or not. Although data mart projects vary considerably and are highly iterative, you can always identify these steps.

The steps are common-sense guides:

1. Identify a business process and the operational system that supports it.
2. Formally state the fact table's grain.
3. Describe the fact table's dimensions.
4. Describe the fact table record.

The following example follows these four steps.

Identify the Business Process

In the present case the business process is the order entry system that tracks and manages the shipments of products to customers over multiple sales channels.

Business Process: The online order entry system that manages and tracks the shipment of products to customers. This system stores shipment information on the basis of daily orders.

This system tracks *daily* shipments according to order numbers. A customer can have at most one order per day, and that order can include multiple line items. A line item is for a specific product and it includes the number of units shipped, the total dollar sales, and a value representing the cost for these units.

The measures in a line item are all aggregate values and there is at most one line item per product:

- Units sold
- Sales dollars (units sold multiplied by the price per unit)
- Total cost (units sold multiplied by the cost per unit)

According to the typical questions and the indispensable reports, these are the measures of principal interest to the sales analyst.

State the Grain

The second step requires that you formally describe the principal measures stored in a fact table and the granularity of its record. The measures stored in the order entry system drives the current design and limits the granularity to daily aggregates of individual products sold to customers.

> **Fact Table Grain:** A fact record corresponds with a line item in the Global Computing Company's order entry system. Specifically, this is aggregate sales by day, product, customer and sales channel.

You can determine the required granularity from the typical questions and standardized reports. For example, the measures in the fact table must at minimum be at a daily level. Otherwise, the resulting design could not support the analysis of data required by questions that include "daily."

The grain could be at a finer level if the source data were available but not at a coarser level. If the grain were at a finer level, queries would then need to aggregate values using a group by clause and an aggregate function. Also, the data mart would occupy more physical storage. Nevertheless, the finer granularity would support a greater depth for drill-down analysis and, if the granularity were fine enough, could also support data mining. Plans for the future have an important role to play at this point. Some practitioners argue that you should implement at the next finest grain. In the present case, this is not possible because of the data sources.

Limits on Measures When you include all the measures in a fact table record during the fourth step, these measures must be consistent with the grain statement. For example, the grain statement says that the record contains daily aggregates. Thus, you can not add a measure to this fact table that represents a monthly aggregate: that would strictly violate the grain statement.

Describe the Dimensions

The grain statement implies four principal dimensions: day, product, customer, and channel. Each record in a dimension table should represent a single day, product, customer, or channel and include a rich set of attributes to describe their object.

> **Principal Dimensions:** Days, Product, Customers, and Channels

You can now use the typical questions and indispensable reports to describe and document each dimension in detail:

- Attributes
- Hierarchies
- Primary key

After the dimensions have been completely described, you can step back and assess with the users whether additional dimensions are required.

Attributes Dimension attributes describe the properties of individual objects (the color of a standard mouse). These attributes are mostly textual descriptions taken directly from the user vocabulary. Common examples include product description, package description, product identifier, or month name. Invariably, a few attributes turn out to be numeric such as quarter, age, and size.

Dimensions typically include one or more attributes that uniquely identify each object described by the dimension. These attributes are called *natural keys* because they are commonly used to identify objects. For example, the natural key for the Products dimension is *item ID* (Universal Product Code). A good practise is to include the text "Unknown" (or a code) for cases where a natural key is missing from the data.

> **Natural keys:** Source values that identify each object described by a dimension. For example, each product distributed by Global Computing Company is identified by a unique product UPC.

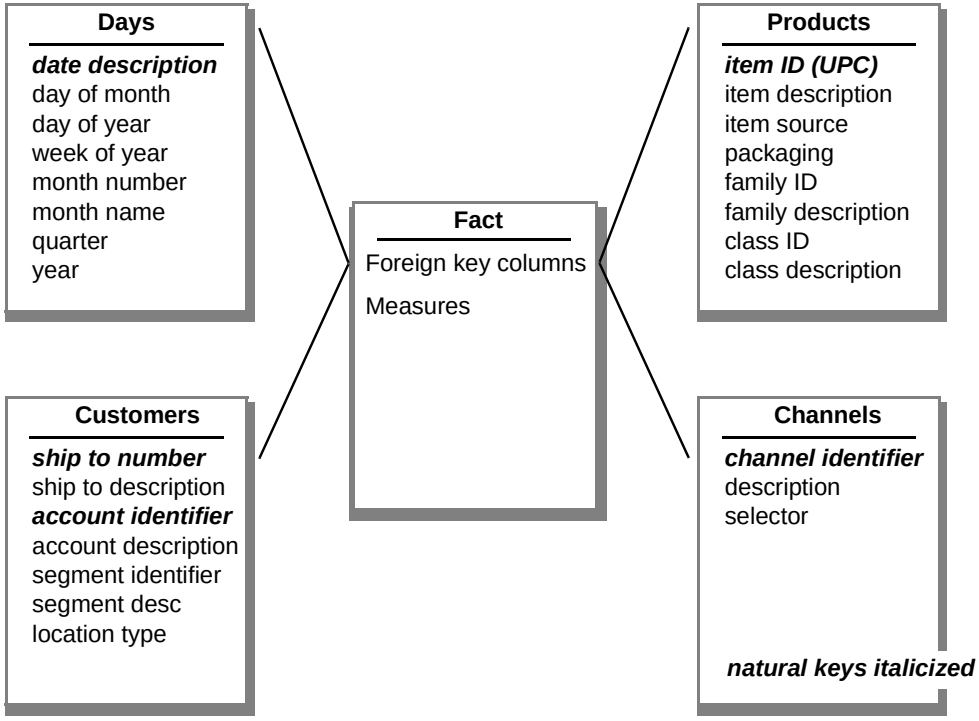
You can develop the initial set of attributes for a dimension from the typical questions and indispensable reports. For example, the attributes for the Days dimension table must include days, weeks, months, and quarters. Otherwise, the questions listed on page 2-3 could not be expressed as a query.

After you identify the attributes, sketch them out in a simple diagram and circulate the diagram among the user community. Afterwards, you can probe further to determine whether the set is complete and everyone agrees on the meanings. For example, is there an implied reference to calendar or fiscal attributes? When someone says quarter, *exactly* what do they mean? The precise meaning of these attributes are extremely important to the success of the data mart.

You can start taking advantage of Oracle Warehouse Builder at this point in the process. Use Builder to create tentative definitions for the dimensions, facts, and hierarchies, then generate and distribute diagrams from these definitions. As you move forward, you can modify the definitions as necessary. This helps to promote communication during the development and implementation process.

This kind of process affords a good opportunity to publish an internal web site available to everyone involved in the data mart project. All of the comments collected and documents produced could be posted on this site.

Figure 2–3 Principal Dimensions



The example dimensions have relatively few attributes compared with an actual dimensions, but they do suggest the kind of textual data that characterizes the contents of a dimension.

> **Dimension Attributes:** Dimension tables are highly *denormalized* and typically contain numerous textual attributes (one hundred is not uncommon) that richly describe their object.

Hierarchies As you fill-in the attributes for each dimension, the hierarchies implicit in the denormalized table will emerge. They in fact will be difficult to ignore. For example, the Day > Month > Quarter > Year hierarchy in the Days dimension stands out. Each element within a hierarchy represents a level of aggregation, and the lower levels roll-up into the higher levels.

The Oracle8*i* server can exploit hierarchies to improve query performance by rewriting queries that summarize data. For example, if a query rolls daily values into monthly subtotals, Oracle8*i* can *rewrite* the query to retrieve the subtotals from a summary table (a materialized view) instead of calculating the subtotals from the detail values. The performance improvement can be considerable and the rewrite operation is completely transparent to the user.

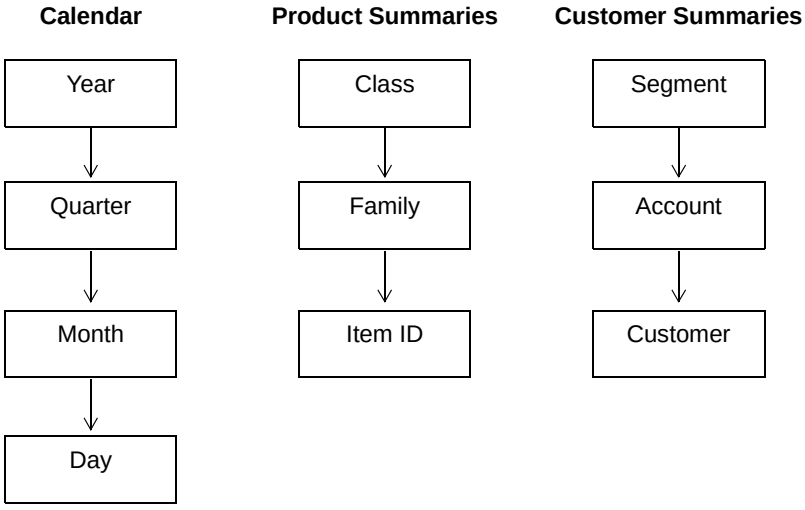
You can take advantage of query rewrite by identifying frequently traversed hierarchies, codifying these hierarchies in a dimension object, and creating materialized views that correspond with each level of aggregation. At this point, you need only identify and document the hierarchies. Later on when you formally define the data mart schema using Oracle Warehouse Builder, you will formally define the dimension objects for each dimension.

To document a hierarchy you need three pieces of information:

- Name of the hierarchy
- Name for each level of aggregation
- Ranking of the levels according to parent-child relationships

The lowest level of aggregation is actually detail data rather than summarized data, and for that reason is typically referred to as the degenerate case.

Figure 2–4 Hierarchies for the Global Computing Data Mart



Primary Keys The primary key for each dimension table is a single, numeric column which is populated with meaningless values. The values for each primary key are generated anonymously by an Oracle sequence object. The meaningless values are called *warehouse keys* to emphasize their non-descriptive role. This kind of key is variously called an anonymous, artificial, surrogate, or synthetic key.

Warehouse keys afford distinct advantages over natural keys:

Manage Unknown Values	When a natural key does not yet exist for a product, you can still include the product in the dimension yet maintain the data mart's integrity.
Support Update Strategies	Dimensions typically change slowly over time, and warehouse keys support the four basic methods used to manage the dimensions.
Reduce Fact Table Size	Fact tables reference dimensions over these compact numeric values rather than natural key values which can be large. This reduces fact table size considerably.
Improve Query Performance	The reduction in fact table size implies more fact table rows per block which translates into improved data transfer rates.

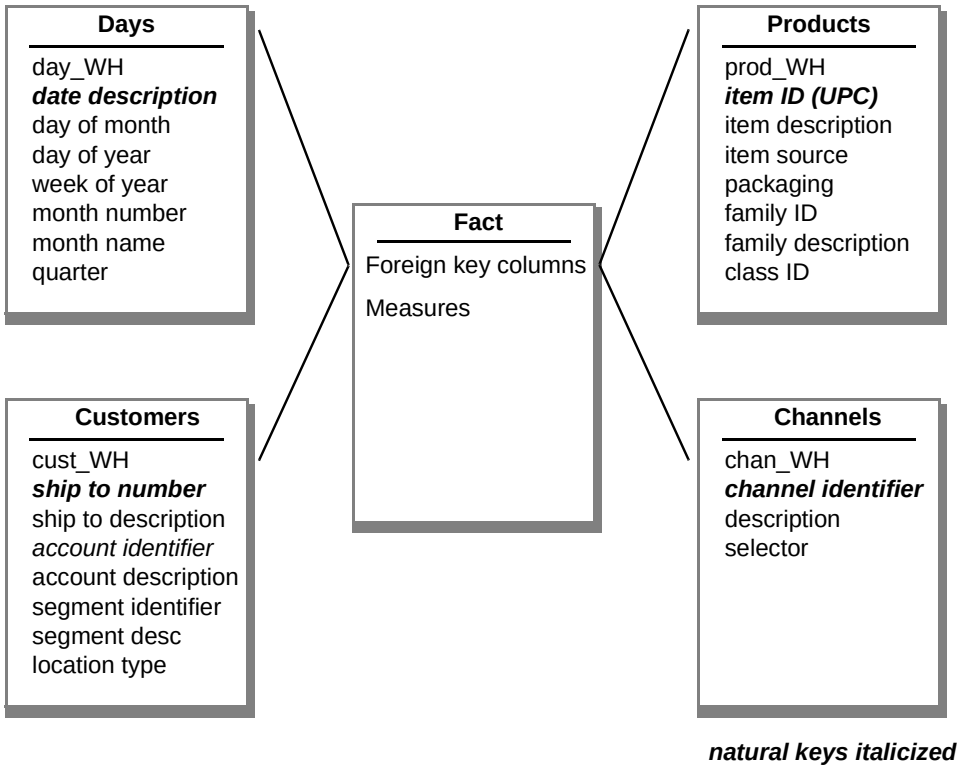
Note: Warehouse keys are used to efficiently join a fact table to its dimensions but should otherwise be transparent to users.

The names of warehouse key columns in the Global Computing Company data mart have the suffix "WH" and are of datatype NUMBER:

Column Name	Dimension
day_WH	Days
prod_WH	Products
cust_WH	Customers
chan_WH	Channels

These column names are the targets of the foreign key references made by the data mart fact table.

Figure 2–5 Natural and Warehouse Keys



Describe the Facts

Fact tables usually contain two kinds of columns: those for foreign key references and those for measures. Occasionally, a fact table contains only foreign key reference columns. For example, event and coverage fact tables contains no measures. For examples of these variations, see Kimball's Toolkit books.

The measures stored in a fact table are typically *additive* so that they can sustain any form of slice and dice attack: Additive measures can be summed across all of the fact table's dimensions.

A developer may sometimes include a *semi-additive* or *non-additive* measure to a fact table. A semi-additive measure can be summed across only a subset of the fact's dimensions. A checking account balance is an example of a semi-additive measure. You can not correctly sum balances across the time dimensions. A non-additive measure can not be summed across any dimension. These kinds of measures should be included only after careful thought and with caution.

The Global Computing Company fact table is named "Sales," it references four dimensions, and it includes four measures.

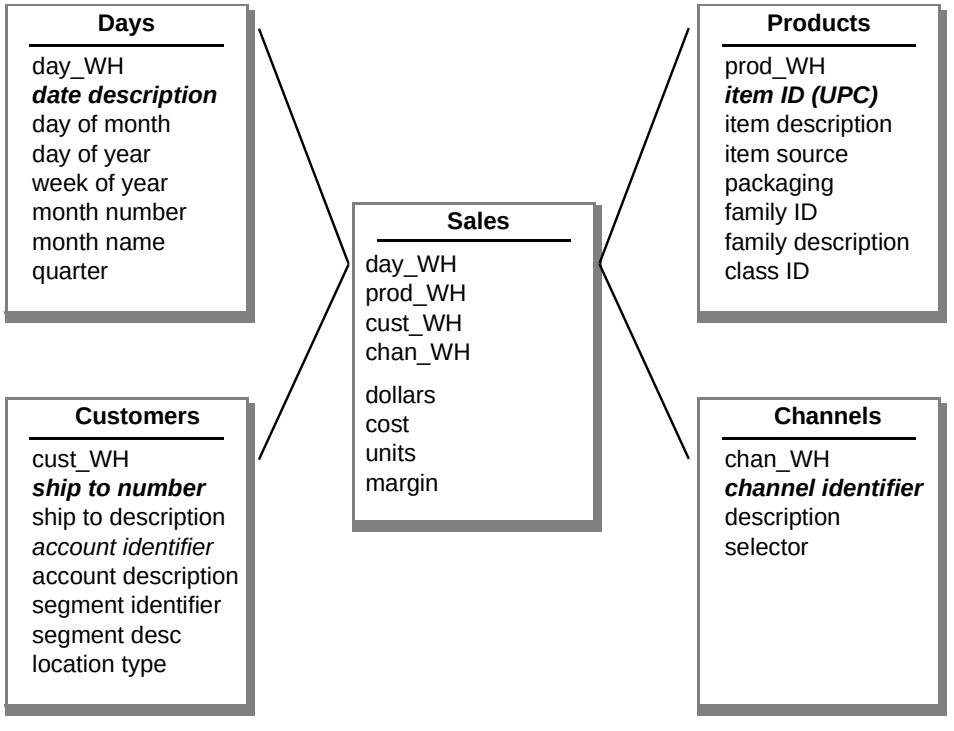
Dimension References The Sales fact table references four dimensions and thus includes four foreign key reference columns. These columns use the same names as the warehouse key columns (*day_WH*, *prod_WH*, *cust_WH*, and *chan_WH*) of the referenced dimensions. This convention reflects the source of values (surrogate keys) for the foreign key reference columns.

The Measures The Sales fact table includes four measures which were selected because they are required to answer the typical questions or they occur in an indispensable report. The measures are additive across all dimensions.

Column	Description
dollars	Total sales in dollars for a product on a line item
units	Total number of units of the product sold on a line item
cost	Total cost of the products on a line item (unit_cost*units)
margin	dollars - cost

These measures all accord with the grain statement for the fact table. Any additional measures that might be considered for inclusion in the fact table must also accord with the grain.

Figure 2–6 Initial Star Schema



This is the initial design of a star schema, and it meets all the user needs. The diagram does not specify the data types of each column. For a complete description of the final star schema, see "The Target Schema" on page B-8.

Once everyone agrees on the basic dimensions and facts, the developers begin their search for a source data item to map onto each column of the star schema. The next section covers this task.

The Sources of Data

At this point, you can begin your search for source items to map onto each column in the target schema. During this search, you can informally define a set of mappings which describe the source and any required transformations for each target column. The product of this task is a *complete* set of informal mappings. This latter condition completes the data audit.

The data for a target column can be a raw source item or a derived item. A derived item can be the output of one or more transformations performed on raw or other derived data items. The informal mappings must define the raw data items, derived data items and their transformation operations, and the target column data types.

This task is best carried out with the keepers of the source data: those professional data analysts who understand their source systems thoroughly, can identify the highest quality sources, and understand the frequency and risks associated with the update cycle. These underground professionals are essential to the successful completion of this task.

Like all the previous tasks, this one is highly iterative and aims to converge on a complete set of mappings. Often, a data item can not be found for one or more target columns. Or, an item is ambiguous. In that case, the developers return to the users to determine whether another item can be substituted, another data source can be found, or the star schema modified.

The Principal Data Source

The search for source data items is simplified in the Global Computing Company case because the rationale for developing a data mart was to unlock the measures stored in the online order system GCCAPPS. This application does contain all the basic measures and most of the other items. Unfortunately, it does not the contain product and channel descriptions. This data is stored in two flat files.

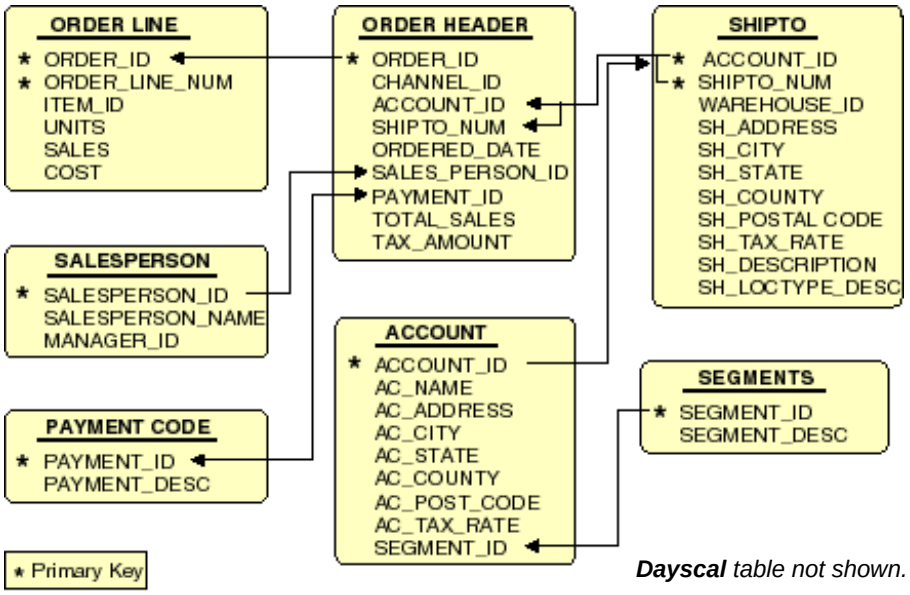
The Order Entry System

The GCCAPPS order entry system is an application that tracks product shipments to customers. This application runs on an Oracle8*i* database server, and the server resides on another Windows NT system. The entire data mart resides on one Windows NT system and the source database resides on another.

The schema for this application contains seven principal tables plus a generic table that uniformly defines the calendar items for all Global Computing Company applications. Figure 2–7 diagrams the principal tables but omits the calendar table.

Order_Header Table The application tracks shipments according to an assigned order number which is stored in the Order_Header table. This table contains information about the customer, the ordered date, and the sales channel. The application identifies a customer with a two part key: account_id and shipto_num. The primary key for a production source system is usually called a "natural" or "production" key. The table also contains a pointer to the Order_Line table which contains most of the facts required by the data mart.

Figure 2–7 GCCAPPS Order Entry Schema



Order_Line Table The Order_Line table contains one or more line items, and each line item identifies a product sold, the aggregate dollars for the sale, the number of units sold, and the aggregate cost. The product on each line item is identified as Item_ID. To understand the data mart's facts, you must understand that each line item measure represents an aggregate total: the total number of units by product shipped, total sales revenue in dollars for that product, and total cost of the product to the company. The cost column is calculated as total units times a unit cost.

Table 2–2 describes the line entry items for the shipment number 60023. The table includes a description for the product which is not included in the Line_Item table. This information is required by the data mart's Products table but must be obtained from a source external to the order entry system.

Table 2–2 Order Line Items

Order_ID	Order_Line_Num	Item_ID	Units	Sales	Cost	*Product*
60023	01	A471245-LX	10	300	125	*Deluxe Mouse*
	02	A461245	50	1000	100	*Standard Mouse*
	03	A301245	90	180	30	*Mouse Pad*

The units, sales, and cost source items from the Order_Line table represent the raw source items for the Sales table. As it turns out, these items are extracted, and moved first to a staging table before they are stored in the data mart schema. The only remaining measure is the margin which can be calculated from the raw data.

ShipTo, Account, and Segments Tables The ShipTo table contains specific information about the customer. The natural key for this table consists of two columns: accout_id and shipto_num. This table contains a specific account description, the shipping address, and a few additional descriptions.

Specific information about the customer's account is identified as an account identifier. For descriptive information about the account and its corresponding segment, you must use the customer's account_id which points to a row in the Account table. This table includes a general description of the account and a pointer, segment_id, to the Segments table. The Segments table contains a descriptive information about the segment (segment_desc).

Dayscal Table This table standardizes the calendar items used by all Global Computing Company applications. The columns in this table are all easily understood from their column names. The table includes the following:

Table 2-3 Dayscal Columns

Days	Weeks	Months	Quarters	Years
date_desc	week_ID	month_ID	quarter_ID	year
day_of_week	week_of_year	month_number	quarter	
day_of_month		month_descritpion		
day_of_year				

The granularity of this table is day and its natural primary key is the date_desc column. The data type of this column is DATE.

OLTP versus Star The order entry schema differs from the star schema primarily in its normalization and organization of data. The order entry schema minimizes repetition of descriptive fields such as a market segment's description by storing this information in a look-up table. The star schema, by contrast, repeats the description in its Customer table. This simplifies browsing, makes queries easier and more intuitive to write, and most importantly improves query performance.

Initial Inventory

The columns in the star schema drive the inventory. You can take this inventory using a simple spreadsheet that identifies candidates for each column in the target schema. The present case is highly-simplified and the choices are few.

The description column in Table 2–4 should include the primary source for the target column, any associated update risks, and alternative candidates.

Table 2–4 Current Inventory

Target column	Source Item	Datatype	Description
sales.dollars	order_line.sales	Number(9,2)	Daily sales by …
sales.units	order_line.units	Number	Daily units by …
sales.cost	order_line.cost	Number(9,2)	Cost by …
sales.margin	calculated	Number(9,2)	Sales margin …
days.date_description	order_header.ordered_date	Date	
days.day_of_week	dayscal.day_of_week	Number	
…			
products.item_id	order_line.item_id	Number	
products.item_description	*N/A*		
products.item_source	*N/A*		
products.packaging	*N/A*		
…			

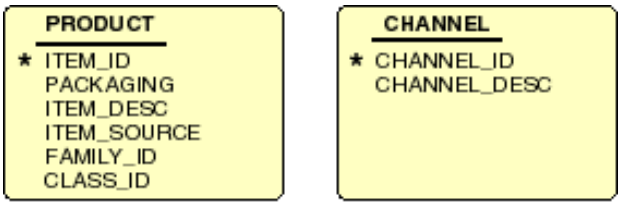
Table 2–4 illustrates a partial inventory but includes several data items that are not available in the source data. The developers will need to search secondary data sources for the unavailable items.

As you will see, you can easily capture and document all of this information and more using Oracle Warehouse Builder and display the information graphically.

The Secondary Sources

The descriptive information required for the Products and Channels dimensions reside in a couple of flat files: product.fix and channel.csv. Figure 2–8 summarizes the data items stored in these flat files.

Figure 2–8 Flat File Data Sources



Appendix B, "Materials for the Case Study" includes a complete description of these data items and their datatypes. The descriptions begins on page B-1.

The Final Inventory

Given these data sources, the inventory spreadsheet can be completely filled-out and the data audit finished. At this point, the developers should circulate the inventory to the entire group, step back, and then take a final look before moving on to the next major phase of the project.

Table 2–5 Current Inventory

Target column	Source Item	Datatype	Description
sales.dollars	order_line.sales	Number(9,2)	Daily sales by …
sales.units	order_line.units	Number	Daily units by …
…			
days.date_description	order_header.ordered_date	Date	
…			
products.item_id	order_line.item_id	Number	
products.item_description	product_fix.item_desc	Varchar(35)	
products.item_source	product.fix.item_source	Varchar(30)	
products.packaging	product.fix.packaging	Varchar(20)	
…			

High Level Maps

Given the star schema and a complete inventory of the data sources, you can now describe how the raw data will be extracted, transformed, and stored in the data mart. This informal mapping provides a rough guide to the flow of data during a typical extract and load operation.

If the data mart didn't use warehouse keys, the extract and load operation could be accomplished in two steps. The introduction of warehouse keys, however, necessitates another step and an additional table to facilitate the generation and management of warehouse keys. The additional table is a staging table named "stageONE". This table looks like the Sales fact table except that it contains natural keys at each position where the Sales table contains warehouse keys, and is joined to the dimension tables over it natural keys.

The Three Steps

The general strategy is to extract and load data for the dimensions first, then extract and load the measures into the staging table second, and finally load the Sales fact table with measures and generated warehouse keys.

Load the Dimensions

During this step, you informally describe four mappings: one for each dimension. Each mapping describes how to extract data from the raw sources, generate values for the dimension's warehouse key column, and then load the extracted data into the dimension table.

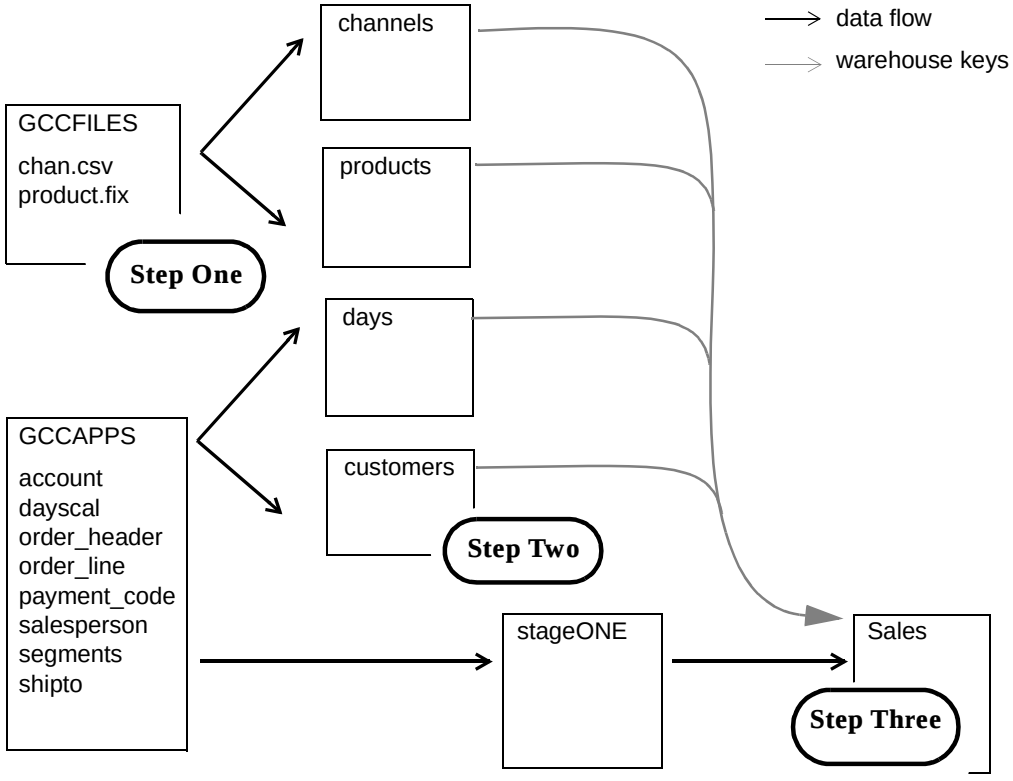
Load the Staging Table

During this step, you informally describe how to extract measures and natural keys from the raw sources, and then load this data into the staging table. The staging table contains measures and natural keys but *no* warehouse keys. This implementation of a staging table within the target schema minimizes the number of database links required to transport data and is very efficient.

Load the Fact Table

During this final step, you informally describe how to join the *stageONE* table to the dimensions over natural key columns, extract measures from the *stageONE* table and corresponding warehouse key values from each dimension, and then load this data into the Sales fact table. At the completion of this load, the data mart will be completely populated with data.

Figure 2–9 High-Level Mappings



While this description roughly sketches the flow of data from the raw sources to the data mart, it will serve you well as a faithful roadmap when you formally describe each of these high level mappings in great detail using Oracle Warehouse Builder.

The Basic Configuration

The example data mart resides on a single Windows NT server and most of the raw data sources reside on a separate Windows NT server.

The Windows NT system that hosts the data mart supports a single Oracle8*i* database instance with the following components:

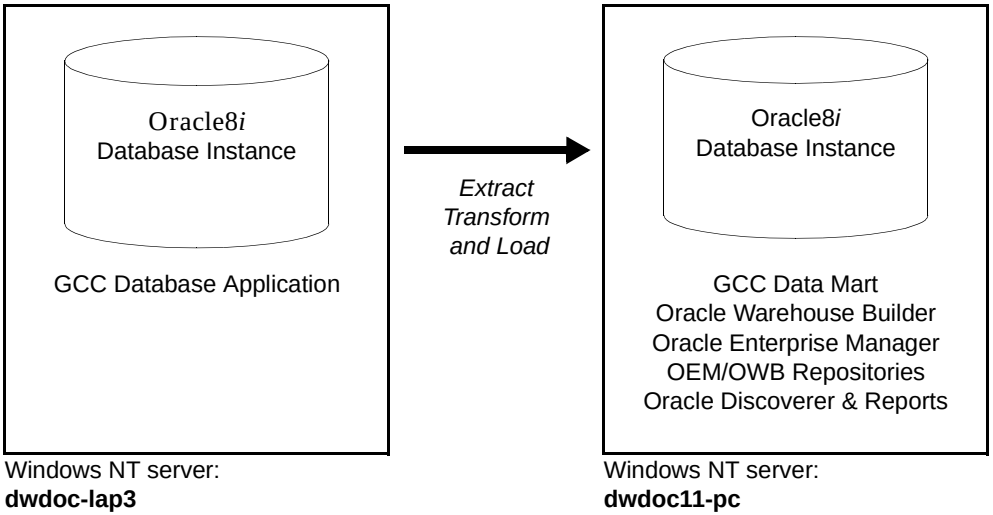
- Oracle Enterprise Manager Console
- Oracle Management Server
- Oracle Warehouse Builder
- Oracle Discoverer 3*i*
- Oracle Reports 6*i*

The Oracle8*i* instance includes three schema objects:

- Oracle Enterprise Manager Repository
- Oracle Warehouse Builder Repository
- Global Computing Company Data Mart (Global Computing Company Data Mart)

The two flat files product.fix and channels.csv also reside on the data mart system.

The Windows NT system that hosts the principal data sources supports a single Oracle8*i* database instance.



The Next Step

This chapter introduced a case study based on a fictitious company, the Global Computing Company. This company tracks and manages their product sales with an online order entry system. Users from the sales and marketing staffs wanted to analyze the quantitative data stored in this system. Unfortunately, their access to the application was so limited that by the time they go the numbers it was too late. This situation led directly to the design and implementation of a data mart that would allow access to the numbers and put their analysis of sales and marketing efforts on a sound foundation.

This chapter summarized their actual requirements in more specific terms, how they developed a star schema using a dimensional modeling technique, how they took a complete inventory of their source data to verify they could populate the data mart completely, and then sketched a high-level mapping that would extract, transform and load the raw source data and store it in the data mart.

You can now develop and implement a data mart for these users using Oracle Warehouse Builder. Just follow the recipe. The next part of this cookbook shows you how to carry out the recipe's first instruction.

INSTRUCTIONS

1. *Create definitions for the star schema, the individual data sources, and the extract, transform and load routines. These definitions constitute a logical model of the data mart.*
2. Configure, validate, and generate scripts from these definitions and then use the generated scripts to deploy the data mart.
3. Populate and publish the contents of the data mart.

After you define a complete logical model of the data mart using Builder, you can configure and deploy a physical instance of the logical model quickly.

Part II

The Definitions

The recipe's first instruction directs you to create a set of formal definitions that *completely* describe your data mart. This part of the cookbook shows you how to create definitions for the Global Computing Company Data Mart using Oracle Warehouse Builder. The definitions are stored and maintained in a Builder repository which is an Oracle8*i* schema object.

This part includes four chapters:

- Chapter 3, "The Containers"
- Chapter 4, "Target Definitions"
- Chapter 5, "Source Definitions"
- Chapter 6, "The Mappings"

The first chapter shows you how to create containers in a Builder repository where you can store the formal definitions; the remaining three chapters show you how to create and update definitions for the target schema, the data sources, and the mapping operations, respectively.

3

The Containers

This chapter shows you how to create three containers in a Builder Project where you can store and maintain the definitions for the case study data mart. This step is a necessary prerequisite to using Oracle Warehouse Builder.

Builder stores definitions in two kinds of containers: warehouse modules and source modules. Definitions for a target schema are stored in a warehouse module and definitions for a data source in a source module.

For the case study project, you need to create one warehouse module and two source modules. One source module is required to store and maintain definitions of the database source and the other for definitions of the flat file source.

This chapter assumes that a Builder repository named "GCCREP" already exists and contains a Builder Project named "GCC Data Mart." Appendix A, "The Case Study Environment" shows you how to create a Builder repository and a Project within that repository named "GCC Data Mart."

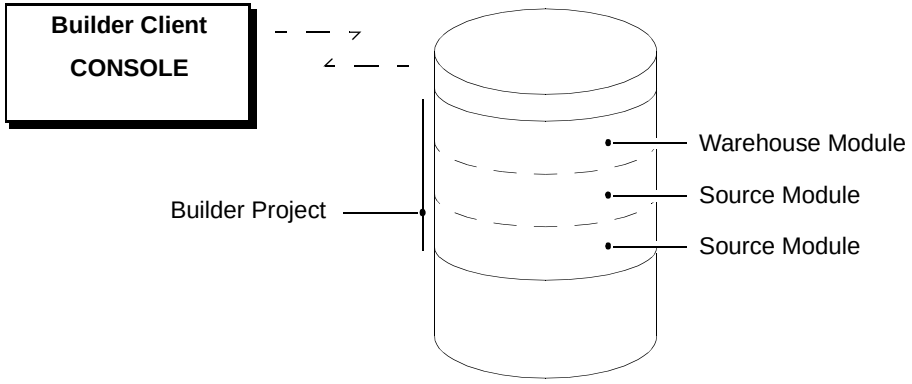
This chapter covers the following topics:

- The Basic Steps
- Logon to a Project
- A Container for the Target Schema
- A Container for the Database Source Definitions
- A Container for the Flat File Source Definitions
- Current State of the Project

The Basic Steps

This chapter shows you how to create three modules where you can store and maintain definitions for the case study data mart. Warehouse and source modules reside in a high-level repository structure called a Builder Project. The repository itself is an Oracle8*i* schema object.

To create a module, you must first logon to a specific Builder Project using the Oracle Warehouse Builder client, and then call the New Module Wizard. This wizard then steps you through the entire process one page at a time.



Builder Repository

Oracle8*i* schema object. For the case study, owner is *GCCREP*

This chapter shows you how to logon to a specific Builder Project using the Oracle Warehouse Builder client and then create three modules using the New Project Wizard. There are four steps:

1. Logon to the Builder Project *GCC Data Mart*
2. Create the warehouse module *gccDM*
3. Create the source module *GCCAPPS*
4. Create the source module *GCCFILES*

The chapter also describes each module's property sheet and shows you how to update the module by editing its property sheet.

Logon to a Project

This section shows you how to logon to a Builder repository and then familiarizes you with the Builder client's user interface. During the logon sequence, Builder returns a dialog box that lists all the existing Projects within the repository. You must select a Project to complete the logon sequence.

Logon Sequence

The Builder repository is an Oracle8*i* schema object and you need to provide the usual information required to logon to a database:

- Username / password *gccrep/gccrep*
- Host Name *dwdoc11-pc*
- Port Number *1521*
- Oracle SID *ora816*

The italicized entries correspond with the case study repository.

To logon to a project:

1. Select **Start > Oracle > Builder Client**

   Builder returns its logon dialog box.

2. Enter the logon information.

3. Click **Logon**.

   Builder returns the Project dialog box unless the Oracle8*i* instance is inactive. In that case, Builder returns a connection error message.

4. Click the down arrow and the select ***GCC Data Mart*** from the list.

   The case study repository, *gccrep*, currently contains two projects: "My Project" and "GCC Data Mart." When you initially create a Project, it is always seeded with the shell Project "My Project" which supports the initial logon. After you create additional Projects in the repository, you can drop the shell Project.

5. Click **OK**.

   Builder returns its Console in a window. This window is the principal path to the Builder client and remains open and available during the entire session.

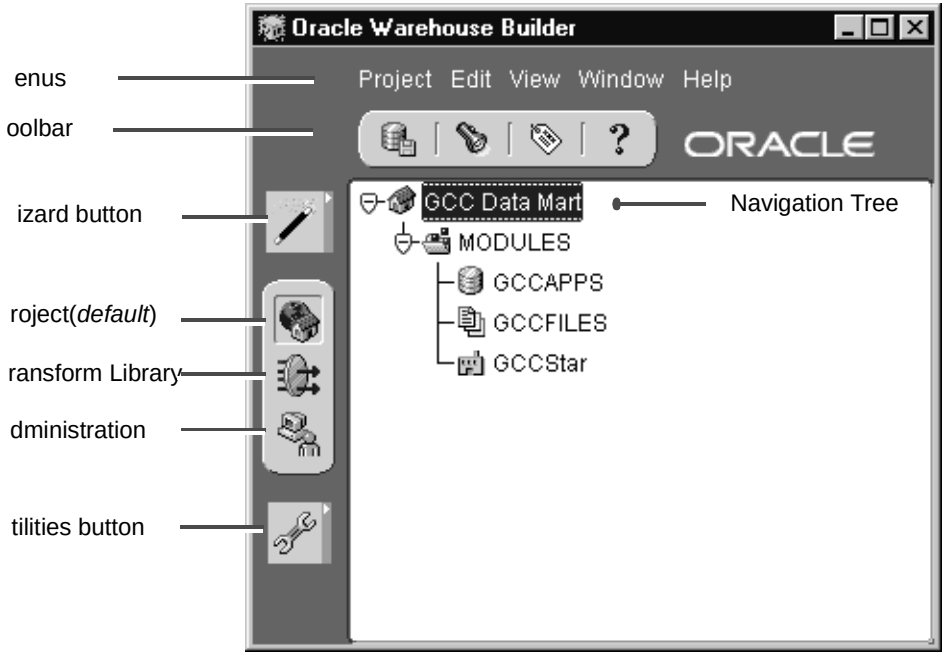
Quick Tour of the Builder Console

The Builder console has six major components:

- Menus
- Toolbar
- Wizard button
- Icons to switch context
- Utilities button
- Navigation tree

Initially, the Console is in the Project environment. This environment includes the Navigation Tree for the open Project and a specific set of menu selections related to operations on Builder Projects. You can switch the Console to operate in the Transform Library and Administration environments. When you switch to another environment, the icons remain constant, but menu selections and navigation trees change. Most of the operations in this cookbook require the Project environment.

Figure 3–1 The Builder Console



Project Environment

When you are in the Project environment, you can:

- Create, import, and modify definitions for schema objects such as dimensions, facts, relational tables, materialized views, sequences, and views
- Create, modify, and import definitions for data sources such as relational schema objects and flat files
- Create and modify transformation operations
- Create, modify, and import maps to direct the flow of data from sources to transformation operations and on to target schemas
- Configure and validate definitions
- Generate, deploy, and run scripts based on the definitions to create, load, and operate target warehouse or staging area schemas
- Export metadata for an entire Project, its modules, or individual definitions
- Generate standardized reports for objects in the Project

Warehouse developers operate mostly within the Project environment.

The illustration for the *GCC Data Mart* Project on the previous page shows the navigation tree for the *completed* Project. The navigation tree includes:

Table 3-1 Modules in the GCC Data Mart Project

Module Name	Type	Contents
GCCAPPS	Source Module	Definitions for objects in GCC Application
GCCFILES	Source Module	Definitions for flat files
GCCDM	Warehouse Module	Definitions for the target schema

Commit Button



Builder does not automatically commit your work to the repository; that is your responsibility. You can commit your work immediately or defer this action until you exit a Builder session. To commit your work immediately, click the Commit icon on the toolbar. Builder returns a Commit Confirmation panel: click **Yes**. You can also defer this action until you exit a Builder session. On exit, Builder always returns an Exit Confirmation panel: click **Yes** to commit your work.

A Container for the Target Schema

This section shows you how to create a warehouse module in the GCC Data Mart Project to store definitions for the target schema. The module's name is *gccDM*. To create this module, logon to the repository, open the GCC Data Mart Project, expand its navigation tree, and call the New Module Wizard directly from the MODULES branch of this tree.

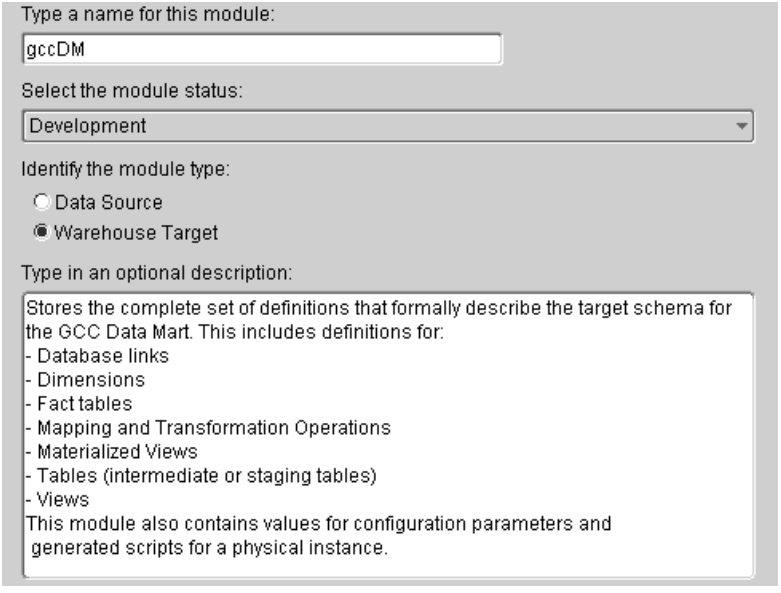
To create a warehouse module:

1. Expand the navigation tree (right-click the plus sign for the navigation tree).
2. Select **MODULES > Create Module**
3. Click **Next**.

   The wizard returns its Name and Other Information page which solicits:

   - Name of the module *gccDM*
   - Status of the module *Development*
   - Type of module *Warehouse Target*
   - Description of module (Optional but strongly advised)

4. Enter the solicited information.

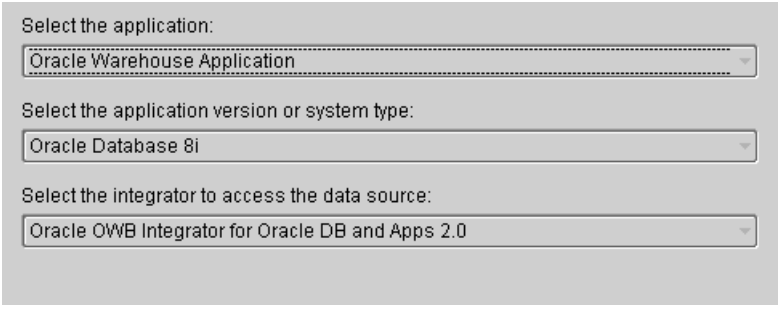


5. Click **Next**.

   The wizard returns the Target page which solicits:

   - Application *Oracle Warehouse Application*
   - Application Version *Oracle Database 8i*
   - Integrator *Oracle OWB Integrator for Oracle DB and Apps 2.0*

6. Select the Application, Type, and Integrator using the drop-down list activated by down-arrow buttons. Currently, only the default entries are available.



> **Note:** A target schema can reside only on an Oracle8*i* instance.

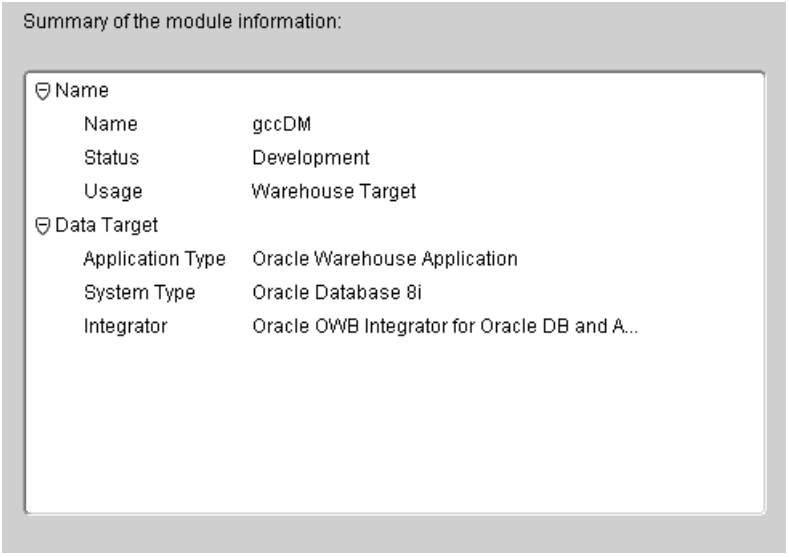
7. Click **Next**.

   The wizard returns the Connection Information page. This page solicits information required to create a database link which Builder can use to import definitions from another schema or from an Oracle Designer repository.

   The case study does not import definitions for the target schema from external sources, so the database link is not needed. Skip this page.

8. Click **Next**.

   The wizard returns the Finish page. This page summarizes the information you entered on each of the wizard's pages.



If any of the information displayed on the summary page is incorrect, navigate back through the wizard pages using the Back button and make any necessary corrections before you click Finish and exit the wizard.

9. Click **Finish**.

   Builder creates the module in the GCC Data Mart Project and inserts its name in the MODULES branch of the navigation tree.



Builder does not automatically commit your works to the repository, that is your responsibility. You can commit your work immediately or defer this action until you exit a Builder session. To commit your work immediately, click the Commit icon on the toolbar. Builder returns a Commit Confirmation panel: click **Yes**. You can also defer this action until you exit a Builder session. On exit, Builder always returns an Exit Confirmation panel: click **Yes** to commit your work.

Notes on Wizards, Navigation Trees, and Names

Oracle Warehouse Builder provides numerous wizards to simplify the design, creation, and implementation process. These wizards can be invoked in a number of ways but mostly from a branch of a navigation tree.

Each wizard is divided into one or more pages. The first page is always the welcome page. This page lists the number of pages in a specific wizard and describes the task performed by each page.

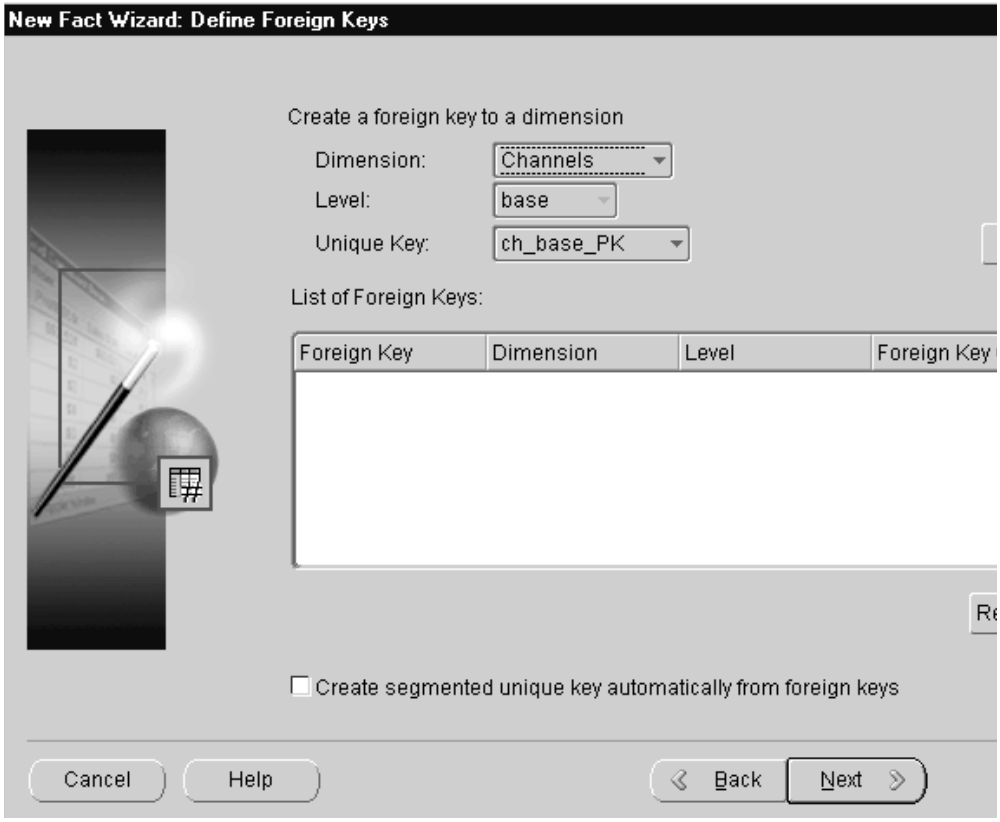
You can configure individual wizards so that they do not return their welcome pages using the check box. To re-enable this behavior, you must open the Preferences sheets and click the button "Display Welcome page on all wizards."

Navigation and Help

To navigate through individual pages of a wizard, click the Back and Next buttons. When you define a warehouse object, you may need to traverse these pages to make corrections or additions. Click Cancel to exit without saving your work.

For help regarding a specific topic within a wizard, click the Help button at the bottom of the wizard page. This invokes context-sensitive help instead of calling the general Help system.

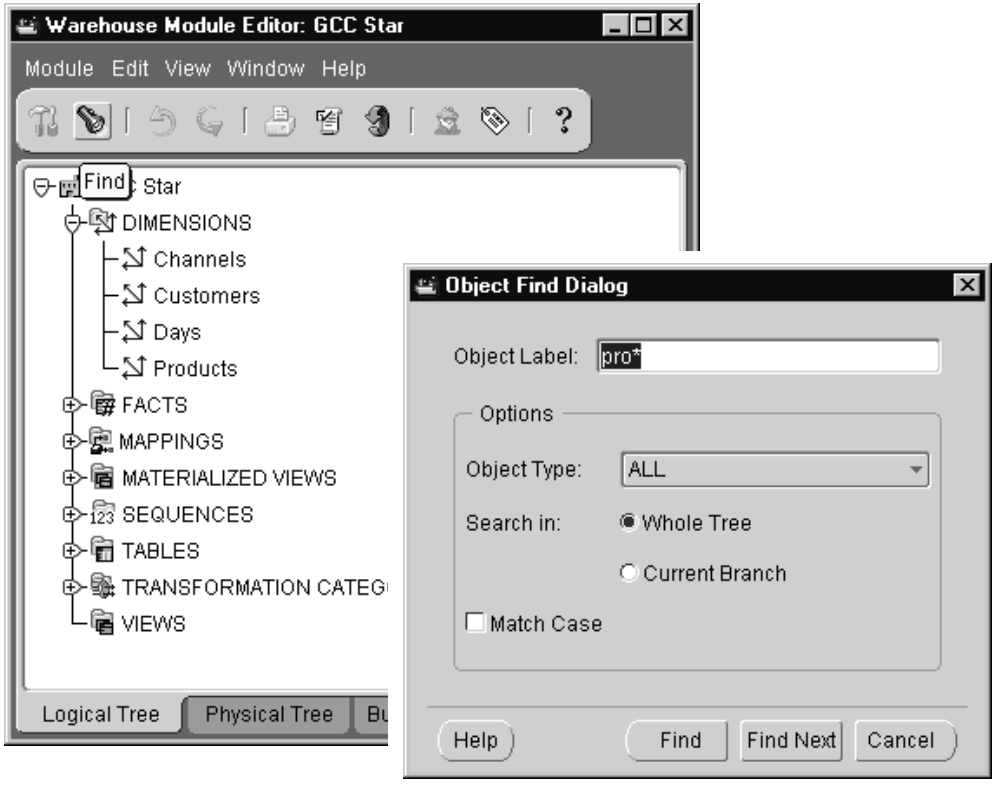
The wizard pages solicit information with down-arrow lists, text boxes, check boxes, and radio and function buttons (Add, Update, and Remove).



You can move around a specific page using a mouse or using the Tab key. When you tab to the last text box, you must then use the CTL-Tab combination to move the cursor down to and across the control buttons.

Search a Navigation Trees

Builder organizes repository objects using a navigation tree. To find an object in the repository, you can manually search for the object or use the flashlight icon to call the Object Find Dialog panel. To manually search for an object, expand each branch of the navigation tree until you find it (click the plus sign). To use the Object Find Dialog panel, click the Find button and enter a search string.



You can include the complete name of an object or use the asterisk wildcard character to match zero or more characters. You can attach the wildcard character as a prefix or suffix to any substring. The wildcard character matches any character.

Logical and Physical Names

Data mart and warehouse developers often use long descriptive names for dimensions, fact tables, materialized views, and other objects to promote communication. Using business names, the developer can make a data mart schema resemble the business being modeled. The Builder technical term for business names is "logical names."

Before you can generate the scripts that create a named object, however, all the logical names used to define objects must be translated into physical names. The physical names must conform to the rules of syntax which are completely defined in the *Oracle8i SQL Reference.*

Builder maintains a logical and a physical name for each object stored in its repository. When you define an object using a logical name, Builder *automatically* creates a legal physical name for that object. And, vice-versa, when you create an object and give it a physical name, Builder validates the physical name and automatically creates a logical counterpart. Thus, no translation is required before you can generate scripts to create a physical instance of the data mart.

Naming Mode You can run the Builder client in either the logical or physical name mode, but to create or modify *logical* or long descriptive names, you must configure the Builder client to run in the Logical Name mode. The Builder client's default naming mode is *physical*.

To set Builder in the logical naming mode from the Console toolbar:

1. Select **Project > Preferences**

   Builder returns its Preferences panel.

2. Click the **Naming** tab.

3. Click the button for the **Logical Name Mode**.

   You can also determine whether logical names are propagated to physical names, or vice versa; and, whether to preserve lower case letters.

4. Click **OK**.

Builder stores this preference in the repository so that the naming mode is preserved from session to session.

The Case Study The examples in the case study were defined with the Builder client in the Logical Name mode although most of the names qualify as physical names.

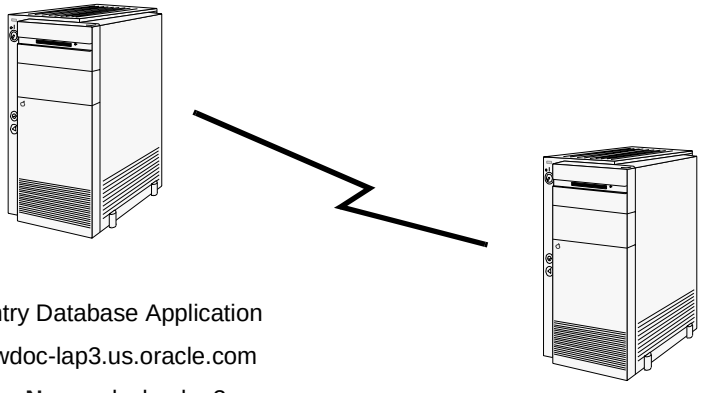
A Container for the Database Source Definitions

This section shows you how to create a source module in the GCC Data Mart Project to store definitions for the order entry database which resides on an Oracle8*i* database instance. You can easily modify the module for other Oracle database versions, a non-Oracle database, or an Oracle Designer repository.

The source module's name is *GCCAPPS*, and you create this module using the New Module Wizard, the same wizard you used to create the *gccDM* warehouse module. When you create a source module, however, you must supply specific information about the application and provide additional information that allows the Builder client to import table, view, and sequence definitions from the database.

The Example Source Database

The database source for the case study resides on an Oracle8*i* database instance, and that instance resides on a Windows NT system whose name is *dwdoc-lap3*. When you create the source module, you must provide information that allows Builder to connect with this database instance. Builder uses this information to create and store a database link in the *GCCREP* repository.



Order Entry Database Application

Node: dwdoc-lap3.us.oracle.com

Computer Name: dwdoc-lap3

Connect: dwdoc-lap3:1521:gcc816

Global Computing Data Mart

Node: dwdoc11-pc.us.oracle.com

Computer Name: dwdoc11-pc

Connect: dwdoc11-pc:1521:ora816

You must provide the following information:

- **Host Name** Name of the computer host for Oracle8*i* (*dwdoc-lap3)*
- **Port Number** Port number for the Oracle8*i* Listener (*1521*)
- **Oracle SID** The SID for the Oracle8*i* instance (*gcc816*)

The source database can reside on the same system as the data mart, or on a different system. The following examples assume that the source data resides on one Windows NT system (*dwdoc-lap3*) and the data mart on another (*dwdoc11-pc*).

See Appendix A, "The Case Study Environment", for information about how to configure a Windows NT system for the case study. "The Database Source" on page A-16 describes how to install the source tables used in the following examples.

Create the Source Module

The following example shows you how to create the *GCCAPPS* source module.

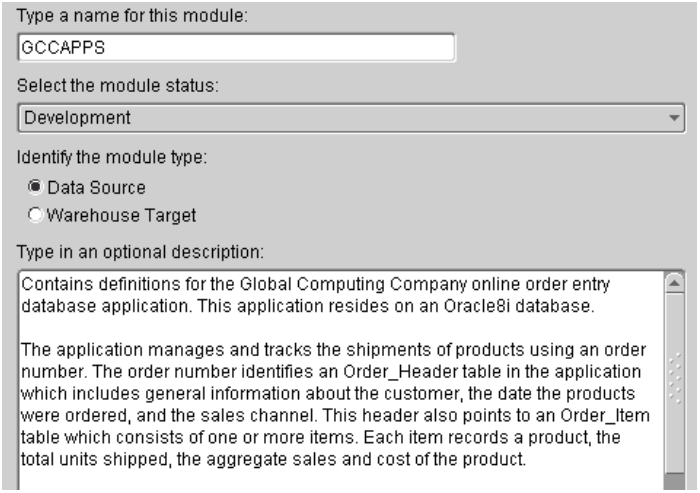
To create the source module:

1. Select **MODULES > Create Module** and click **Next** to skip the Welcome page.

   The wizard returns its Name and Other Information page which solicits:

   - Name of the module *GCCAPPS*
   - Status of the module *Development*
   - Type of module *Data Source*
   - Description of module *Optional but strongly advised*

2. Enter the requested information.

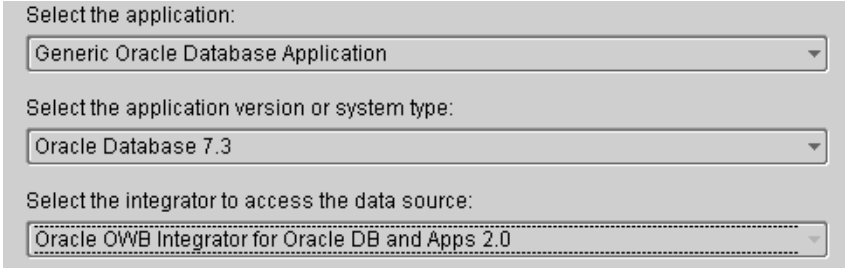


3. Click **Next**.

   The wizard returns the Target page which solicits:

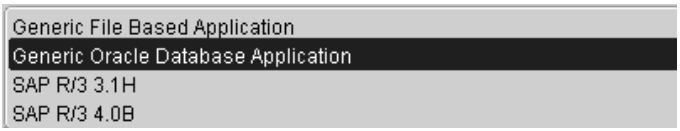
   - Application *Generic Oracle Database Application*
   - Application Version *Oracle Database 8i*
   - Integrator *Oracle OWB Integrator for Oracle DB and Apps 2.0*

4. Select the Application, Type, and Integrator using the drop-down list activated by down-arrow buttons.

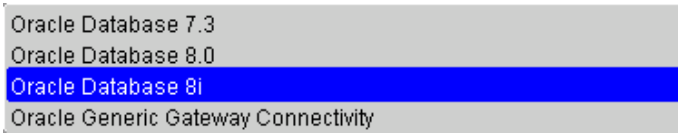


Builder currently provides software integrators to support several Application Types and Versions:

Application Type



Application Version or System Type



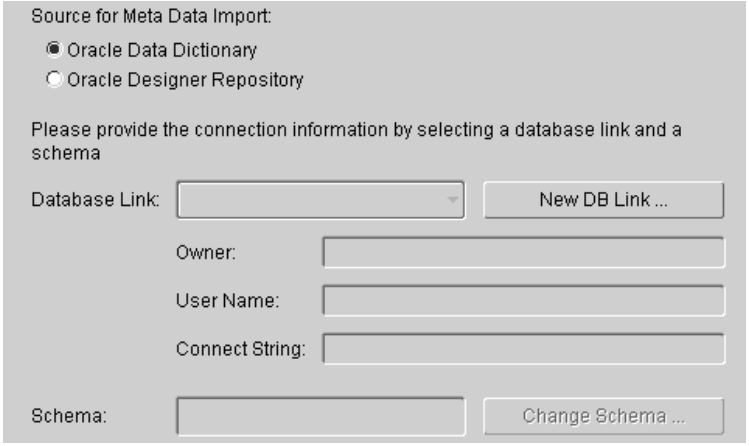
The wizard selects the corresponding Oracle Integrator



If your database source resides on a non-Oracle database, select *Oracle Generic Gateway Connectivity*. This allows Builder to connect and pull data from the source using a heterogeneous services agent (Transparent Gateway). Several examples in the *Oracle Warehouse Builder User's Guide Release 2.1.1* illustrate how to configure a source module for non-Oracle database sources.

5. Click **Next**.

The wizard returns the Connections Information page.



Use this page to define a database link so that Builder can connect with the source database. Builder establishes a connection with the source database,

reads its data dictionary, and the imports definitions of its tables, views, and sequences into the source module.

Note: You must specify a database link for each source module.

Currently, no database links exist in the repository and you must create one for the *GCCAPPS* database source.

To create a database link:

1. Gather the connection information:

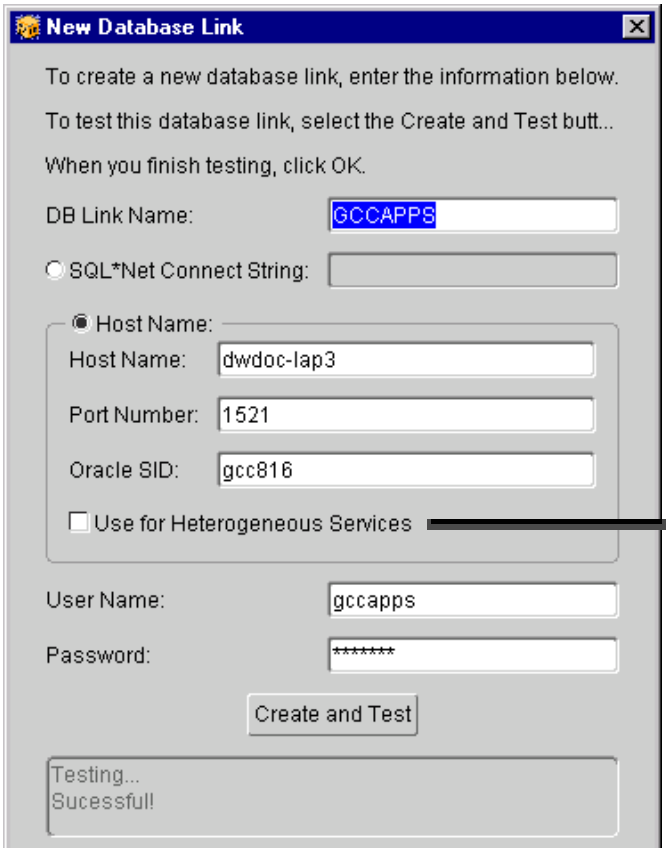
   - Username / password *gccapps/gccapps*
   - Host Name *dwdoc-lap3*
   - Port Number *1521*
   - Oracle SID *gcc816*

   See "The Database Source" on page A-16 for information about creating the gccapps username.

2. Click **New DBLink**.

   The wizard returns the New Database Link panel which solicits the connection information summarized above.

   You can create a database link manually or use the New Database Link panel. The New Database Link panel can be called from the wizard's Connection Information page. Database links raise security issues and database administrators usually prefer to create and control them manually.



3. Enter the connection information and then click **Create and Test.**

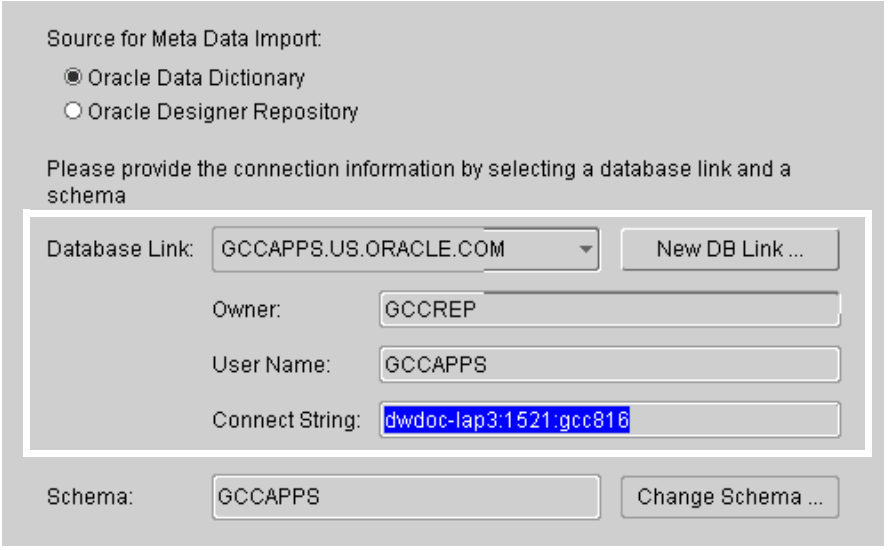
   The wizard attempts to connect with the source using the information supplied. If the test succeeds, then Builder stores the link and its definition in its repository; otherwise, Builder returns an error message.

Note: You can specify the connection information or the name of an SQL*Net connect string that exists in the Tnsnames.ora file. See the *Oracle Warehouse Builder User's Guide* for examples.

---

4. Click **OK** to close the panel.

The wizard returns control to an updated version of the New Module Wizard's Connections page. This page now has an entry in the Database Link down-arrow box:



When you access this source module in the future, Builder can use this database link to connect with the *GCCAPPS* database, read its data dictionary, and import any of its definitions.

Note: Builder does not currently support the **Change Schema** button for Oracle Database systems.

---

6. Click **Next**.

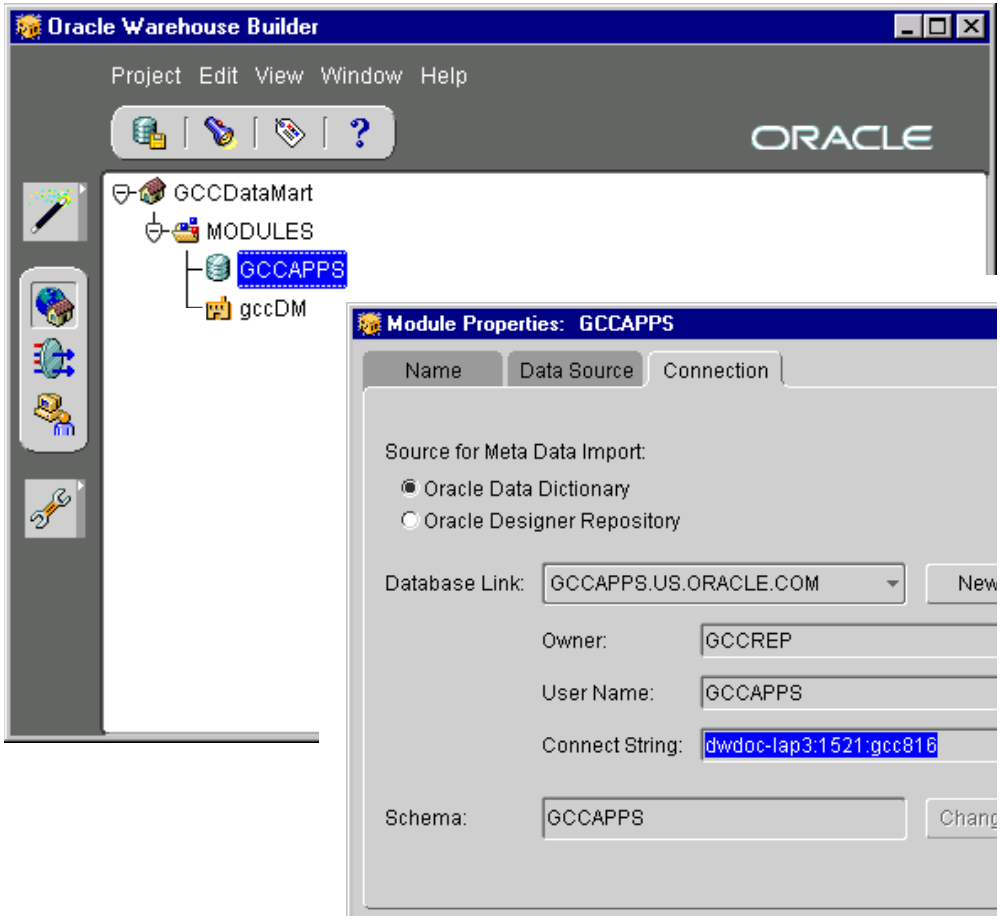
   The wizard returns its Finish page which summarizes the module's properties. If any of the information on this page is incorrect, navigate back through the wizard pages using the Back button and make any necessary corrections before you click Finish and exit the wizard.

7. Click **Finish**.

   Builder creates the *GCCAPPS* source module in the GCC Data Mart Project and inserts its name in the MODULES branch of the navigation tree.

A Checkpoint

The GCC Data Mart Project should now contain two modules. You can verify this by fully expanding the Project's navigation tree. The two modules should occur as branches of the MODULE branch.



To review the property sheets for one of these modules, select the module's name and then click the Property button. Module property sheets have three pages: Name, Data Source, and Connection.

You can always display an repository object's property sheet in the same fashion: select the object's name and the click the Property icon. The number of pages in a Property sheet varies by object.

A Minor Diversion

Suppose you only have a single Windows NT system but would still like to work through the examples in this cookbook. This would be easy. The source database could reside on the same Windows NT host as the target schema, and this would have almost no effect on the remaining examples in the cookbook.

You could load the source database on the data mart host so that source and target databases reside on the same Windows NT system. In that case, however, when you create the *GCCAPPS* source module, you would supply the connection information for the target system:

- **Host Name** Name of the computer host for Oracle8*i* (*dwdoc11-pc)*
- **Port Number** Port number for the Oracle8*i* Listener (*1521*)
- **Oracle SID** The SID for the Oracle8*i* instance (*ora816*)

When it comes down to it, the distinction is transparent to Builder. Builder just uses the information you provide to establish a connection with the source database instance. Builder doesn't really "know" or for that matter "care" whether the source and target databases reside on the same system.

Of course, if you set up your own version of the case study, the name of your host, your listener port number, and the SID for your Oracle8*i* database instance will most likely differ from those that occur in the cookbook. For the most part, however, these physical details show up in only a few places.

If you have problems setting up the environment, see your database administrator for help. Most of these physical issues can be resolved quickly. See Appendix A, "The Case Study Environment" for additional information on the physical characteristics of the case study environment.

A Container for the Flat File Source Definitions

This section shows you how to create a source module to store definitions for flat files. These files reside on the system that hosts the Oracle Warehouse Builder client.

The source module's name is *GCCFILES*, and as in the previous two cases, you create this module using the New Module Wizard.

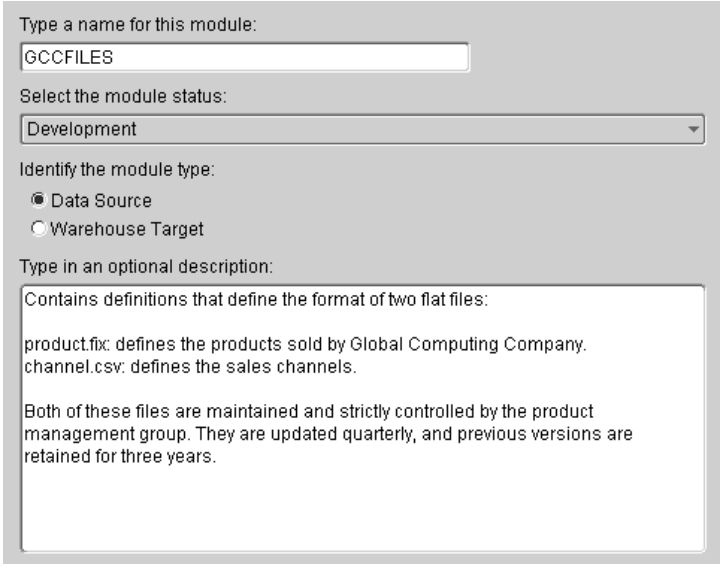
To create a source module:

1. Select **MODULES > Create Module** and click **Next** to skip the Welcome page.

   The wizard returns its Name and Other Information page which solicits:

   - Name of the module *GCCFILES*
   - Status of the module *Development*
   - Type of module *Data Source*
   - Description of module *Optional but strongly advised*

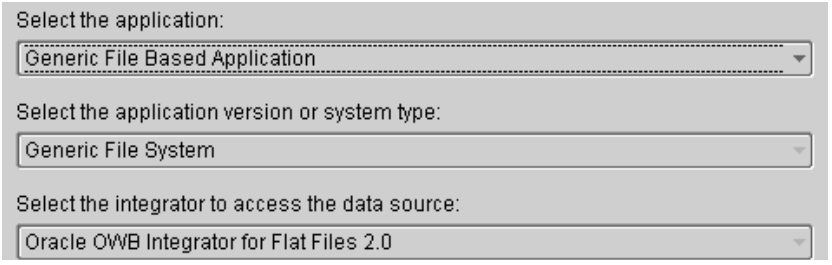
2. Enter the requested information.



3. Click **Next**.

The wizard returns its Target page which solicits:

- Application *Generic File Based Application*
- Application Version *Generic File System*
- Integrator *Oracle OWB Integrator for Flat Files 2.0*

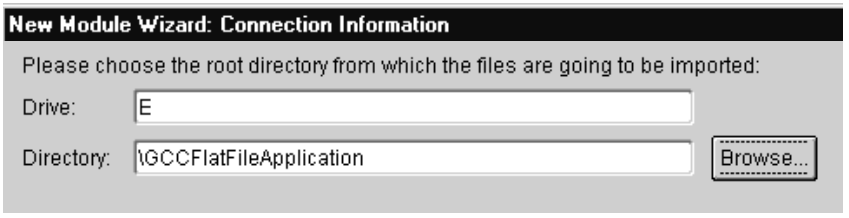


4. Select **Generic File Based Application**.

   When you select the Application Type as *Generic File Based Application*, the wizard determines the remaining two selections.

5. Click **Next**.

   The wizard returns its Connection Information page.



   This page contains two text boxes, Drive and Directory. Specify the full path name of the directory where the two flat files reside.

6. Click **Next**.

   The wizard returns its Finish page which summarizes the information you entered. Verify this information before you proceed.

7. Click **Finish**.

   The wizard creates the *GCCFILES* module and inserts its name in the navigation tree for the *GCC Data Mart* Project. (A warning might appear about the source's NLS character set being incompatible with that of the repository. You will be

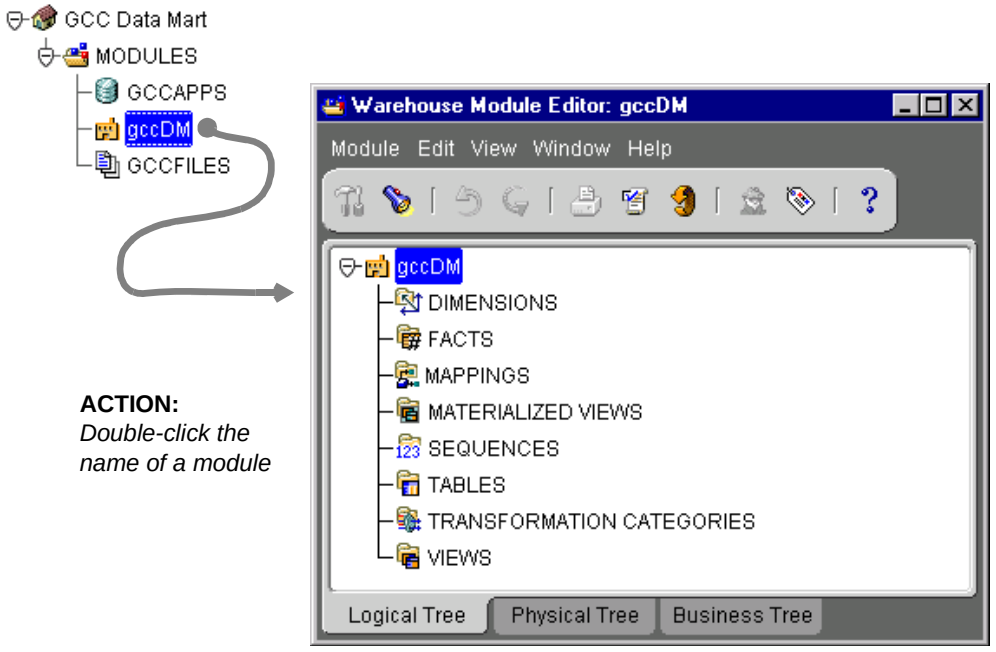
able to specify an NLS character set when you import the data; therefore you can safely click **OK** now to close the warning window. See Step 3 on page 5-11.)

Current State of the Project

At this stage, the *GCC Data Mart* Project contains one warehouse module and two source modules. To verify the contents of this Project, expand its navigation tree and you should find three branches under MODULES.

Although none of the modules contain definitions, each one contains a subdirectory to organize definitions. A warehouse module, for example, includes directories for dimensions, facts, mappings, and other definitions. To view a module's directory structure, double-click the module's name. Builder calls a module editor and displays the navigation tree for the module in a separate window. Figure 3–1 shows the fully expanded navigation tree for the *gccDM* warehouse module.

Figure 3–1 Warehouse Module Editor



The Warehouse Module Editor is the starting point for the next chapter which shows you how to create and store definitions for the *gccDM* star schema.

The Next Step

This chapter showed you how to create three containers for the data mart's definitions in a Builder Project. Each container was created using the New Module Wizard. The chapter also introduced the Builder console and it basic operation.

You can now create the definitions that completely define the data mart. The cookbook creates these definitions in three steps:

1. Create the definitions for the target schema.
2. Create the definitions for the source data.
3. Create the definitions for the mapping operations.

The next chapter shows you how to create the definitions for the target schema using the Warehouse Module Editor and its associated wizards. This chapter also shows you how to update these definitions by editing their property sheets.

4

Target Definitions

This chapter shows you how to create, store, and update formal definitions for the case study's star schema. This schema was developed and described in Chapter 2.

A series of examples show you how to create definitions for each object using a Builder wizard. The chapter also shows you how to update these definitions by editing their property sheets or using a Builder editor. The Builder editors can also display and print diagrams as well as detailed reports for each object.

This chapter covers the following topics:

- The Basic Steps
- Dimensions
- Fact Tables
- Materialized Views
- Conventional Views
- Sequence Objects
- Current State of the Project

The Basic Steps

This chapter shows you how to create definitions for the case study's schema objects using Builder wizards. There is a different wizard for each kind of object. For example, the New Dimension Wizard creates definitions for a dimension.

The examples follow a general order which simplifies keeping track of all the names for columns, foreign key references, primary keys, and unique keys. To simplify the process, definitions are usually created for a referenced object before they are created for objects that make the references. For example, a fact table references its dimensions, so definitions for the dimensions are created first. While this sequence is not strictly necessary, it takes advantage of Builder's pick-lists and pre-validation routines. The definitions are created in the following order: Dimensions, the stageONE table, the Sales fact table, and then Materialized and conventional views.

To create a definition for an object in the *gccDM* warehouse module:

1. Double-click the module's name (*gccDM*) in the Project navigation tree.

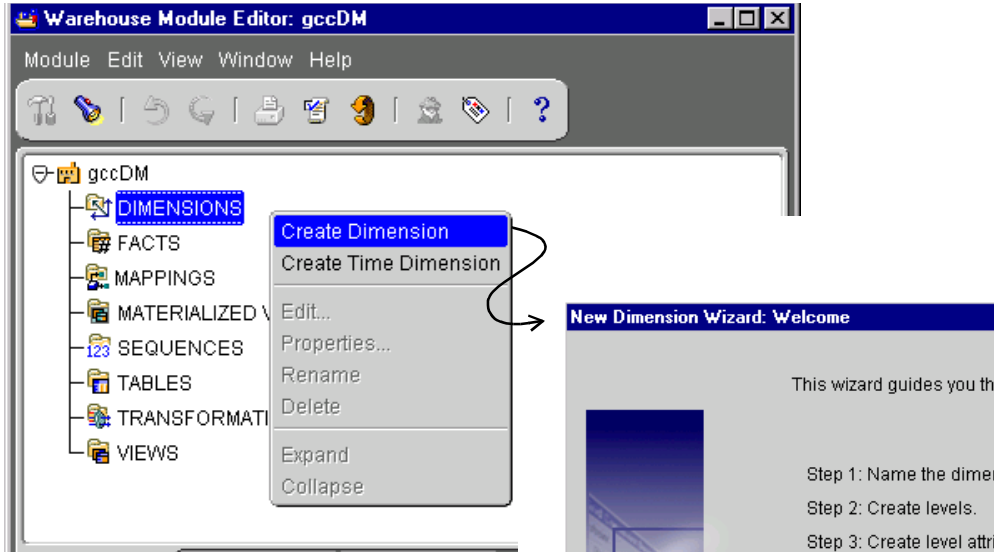
   Builder opens the module with the Warehouse Module Editor.

2. Expand the navigation tree for the module.

3. Right-click a category (**DIMENSIONS**) and select **CREATE** from the pop-up list.

   Builder calls the wizard that corresponds with the selected category.

4. Provide the information solicited by the wizard, one page at a time.



Dimensions

This section shows you how to create a definition for the Products dimension using the New Dimension Wizard and how to update the dimension by editing its property sheet. For the examples in this cookbook, *you must update each dimension's definition* so that it includes a primary key constraint on the warehouse key column and a unique key constraint on the natural key columns.

The New Dimension Wizard walks you through a five-step process one page at a time. After working through this example, you can use this same wizard to create the remaining dimensions as an exercise.

The Products Dimension

The Products dimension describes all products sold by Global Computing Company. To create a definition for this dimension, you need the following details. The table is named 'Products' and is assigned a prefix of 'pd'. This table has a primary key constraint defined on its warehouse key column. The dimension contains a single hierarchy named 'Product_Summaries' which defines three levels of aggregation.

Table 4–1 describes the specific information you need: each level of aggregation, a prefix for each level, and attributes defined on each level. The levels occur in parent to child order: class is the parent of family, and family the parent of product. If you do not specify a prefix, then the wizard uses the dimension's name as a default.

Table 4–1 Products Dimension Object

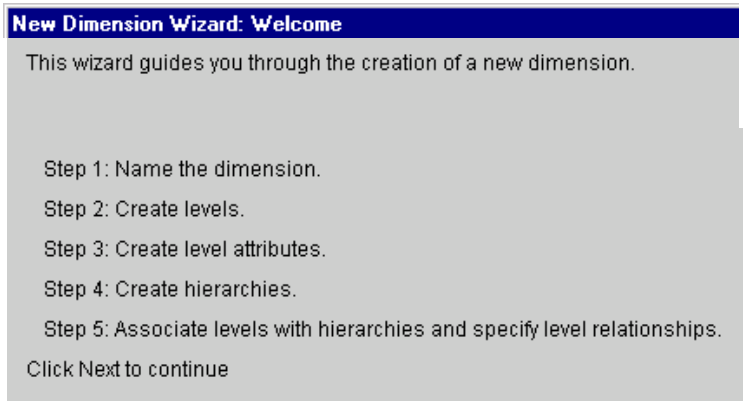
Level	Prefix	Attribute	Data Type	Description
class	cl	class_id	number	Level identifier or key
		class_desc	varchar2(20)	Description of product class
family	fa	family_id	number	Level identifier or key
		family_desc	varchar2(20)	Description of product family
product	pro	prod_WH	number	Base level (*warehouse key*)
		item_desc	varchar2(35)	Description of the product
		product_upc	varchar2(11)	Universal product code (*natural key*)
		item_source	varchar2(30)	Supplier for product
		packaging	varchar2(20)	Packaging for the product

Create the Definition

Open the *gccDM* module with the Warehouse Module Editor, and then call the New Dimension Wizard from module's navigation tree:

1. Right-click **DIMENSIONS > Create Dimension**.

   Builder returns the Welcome page for the New Dimension Wizard which summarizes each of the five steps required to create the definition:

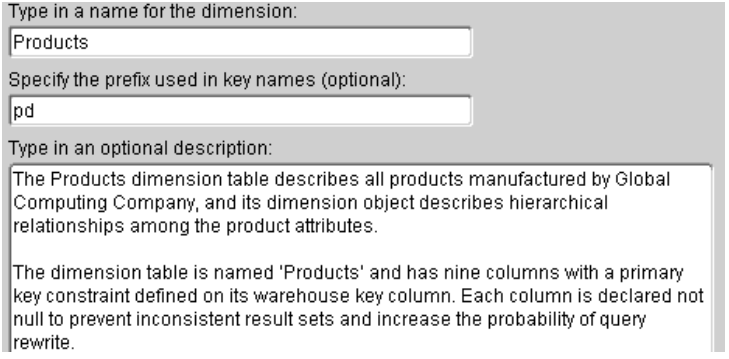


2. Click **Next**.

   The wizard returns its Name page:

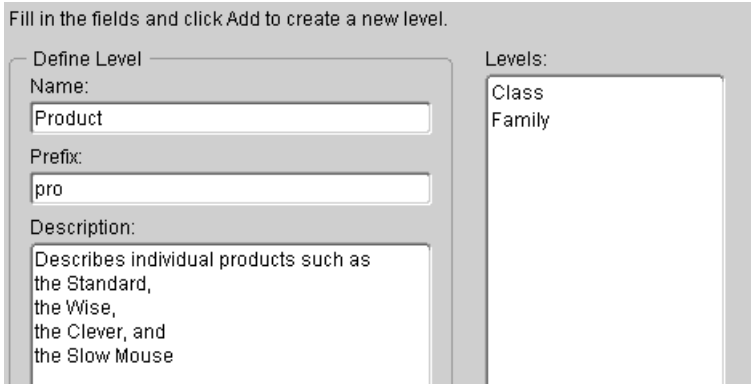
   - The name of the underlying table.
   - A prefix (*pd*).
   - A description of the dimension.

     An optional field that should be used to fully document the dimension.



3. Enter the dimension's name (*Products*), its prefix (*pd*), and a concise description, and then click **Next**.

   The wizard returns its Levels page. Use this page to explicitly name the levels of aggregation. During a subsequent step, you use these levels to define the Product_Summaries hierarchy.



   Note: Dimensions must have at least one level of aggregation. You can define a dummy level to satisfy this requirement.

   To define a level of aggregation, enter the following information:

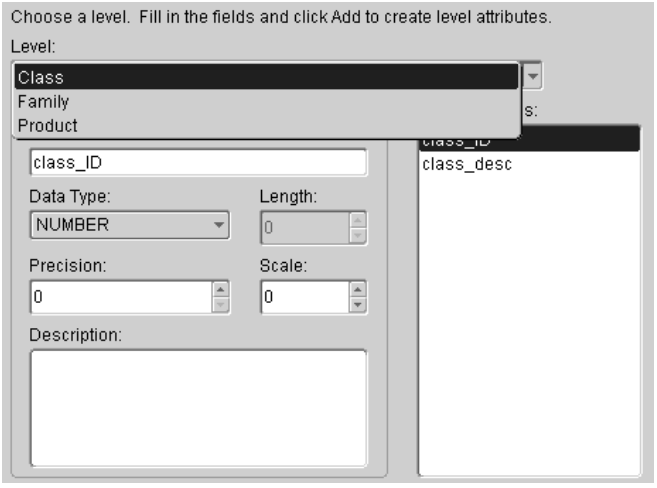
   1. The name of the level: *Product*, *Family*, or *Class*
   2. A prefix for the level: *pro*, *fa*, or *cl* (the default is the name of the level)
   3. A brief description of the level
   4. After you enter the information for a specific level, click **Add**.

      The wizard stores the information about the level and clears the text boxes.

   Continue this process until you have defined *each* level of aggregation.

4. Click **Next**.

   The wizard returns its Level Attributes page. Use this page to define the names and datatypes of all the dimension's attributes. These attributes are defined according to their respective levels.



Defining the attributes is an iterative process: select a level from the down-arrow list, define its attributes, and then continue this process until all the attributes for each level have been defined.

To define the attributes for the active level

1. Enter a name for the attribute.
2. Select a datatype for the attribute from the drop-down list under Data Type.

   Builder supports the following Oracle8*i* data types:

   * CHAR
   * DATE
   * FLOAT, NUMBER
   * VARCHAR, VARCHAR2
3. Click **Add** or **Update**.

   The wizard records the information and clears all the text boxes.

You can now define another attribute for the active level by repeating these steps, or change to another level and define its attributes.

The wizard *always* generates an ID attribute for each level. The lowest level corresponds with the dimension's base granularity, and the wizard generates a unique key constraint on this attribute. Each of the case study's dimensions use this column as the dimension's primary key column by changing the unique

key constraint to a primary key constraint (see the example in "Update a Constraint" on page 4-11).

The generated name for ID attributes has the form *prefix*_ID. For example, the generated name for the class level is *cl_ID*. You can change this default name to a more meaningful name using the Update button.

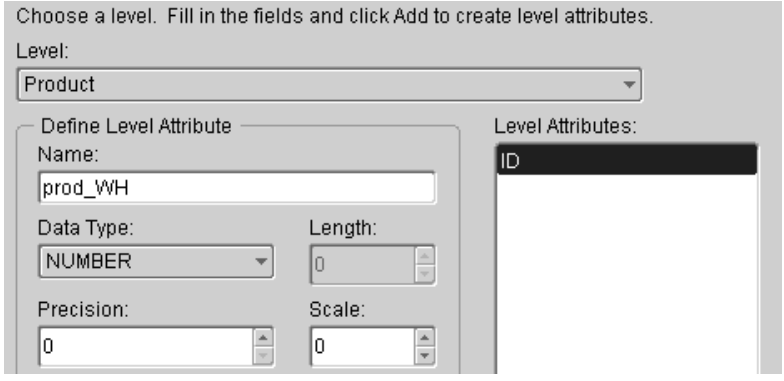
1. Select **ID** in the Levels Attributes
2. Enter a meaningful name in the **Name** text box
3. Click **Update**.

The table below lists the generated name and a more meaningful counterpart for the Products dimension:

Prefix	Generated Name	Meaningful Name	Level of Aggregation
cl	cl_ID	cl_class	Highest
fa	fa_ID	fa_family	
pro	pro_ID	pro_product_WH	Base

To rename the ID column for the active level:

1. Highlight **ID** in the Level Attributes box.
2. Enter a name in the Name text box (*prod_WH*, for example).
3. Enter a brief description.
4. Click **Update**.



At this point, all of the attributes for each level should be defined. Builder uses these definitions at a later stage to generate DDL scripts to create the dimension: one script for the dimension table and another to generate the dimension object which codifies the levels and hierarchies.

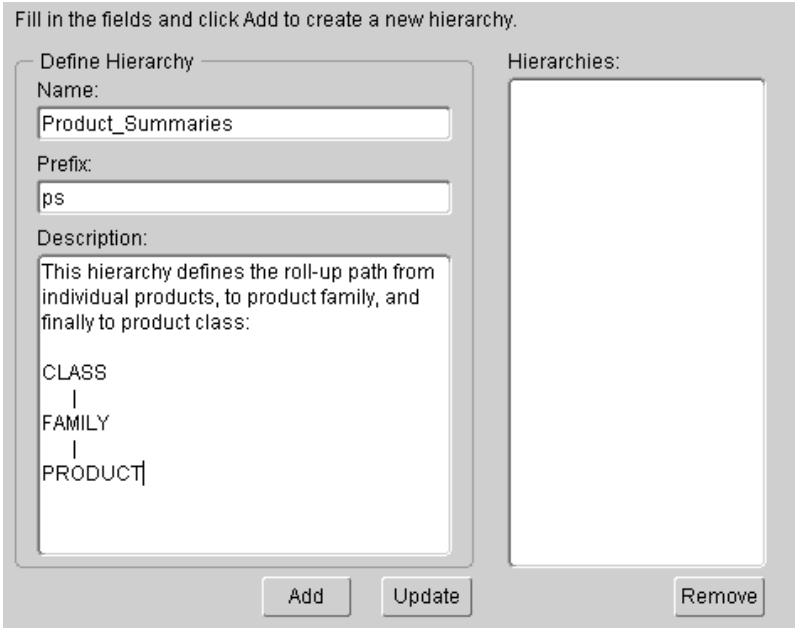
The column names generated for each attribute that occurs in these scripts have the form *prefix_name*. The table below lists the names generated by Builder for the Products dimension.

Prefix	Generated Column Name
cl	cl_class_ID, cl_class_desc
fa	fa_family_ID, fa_family_desc
pro	pro_prod_WH, pro_item_desc, pro_product_upc, pro_item_source, pro_packaging

You can now move on to the wizard's next page.

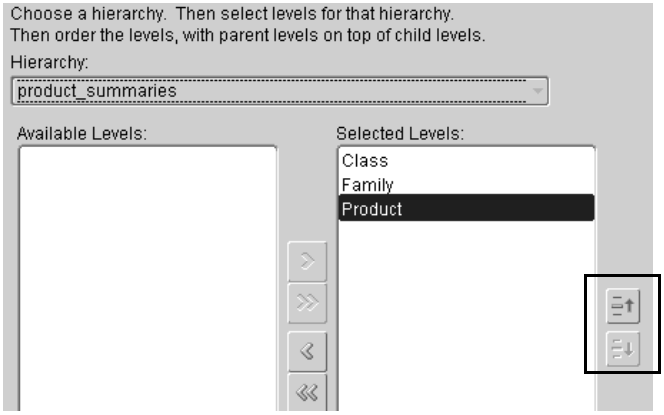
5. Click **Next**.

The wizard returns its Hierarchies page which solicits a name and a prefix for each of the dimension's hierarchies. The Products dimension has only one hierarchy: *Product_Summaries*.



6. Enter a name (*Product_Summaries*), a prefix (*ps*), and a concise description for the hierarchy.
7. Click **Next**.

   The wizard returns its Levels Relationships page.



   Use this page to order the levels that participate within each hierarchy.

   To order the levels within a hierarchy:

   1. Click the **down-arrow** and select a hierarchy from the drop-down list.
   2. Move the names of levels that participate in a selected hierarchy from Available Levels to Selected Levels.
   3. Order the levels so that they reflect the parent to child order.

8. Click ***Next***.

   The wizard returns its Finish page. Verify the description and make any necessary corrections before you click Finish.

9. Click **Finish**.

   The wizard creates a definition for the Products dimension, stores the definition in the *gccDM* warehouse module, and inserts its name in the navigation tree for the warehouse module.

A definition for the Products dimension now resides in the Builder repository but it is incomplete. The constraint on the dimension's warehouse key column must be changed from a unique key to a primary key constraint. To make this change, you can update the definition by editing its property sheet.

Update the Definition

Each definition stored in a Builder repository has an associated property sheet, and you can update a definition by editing its property sheet. The following example shows you how to update the definition for the Products dimension so that it includes a primary key constraint on the warehouse key column (*pro_prod_WH*) and a unique key constraint on the natural key column (*pro_product_upc*).

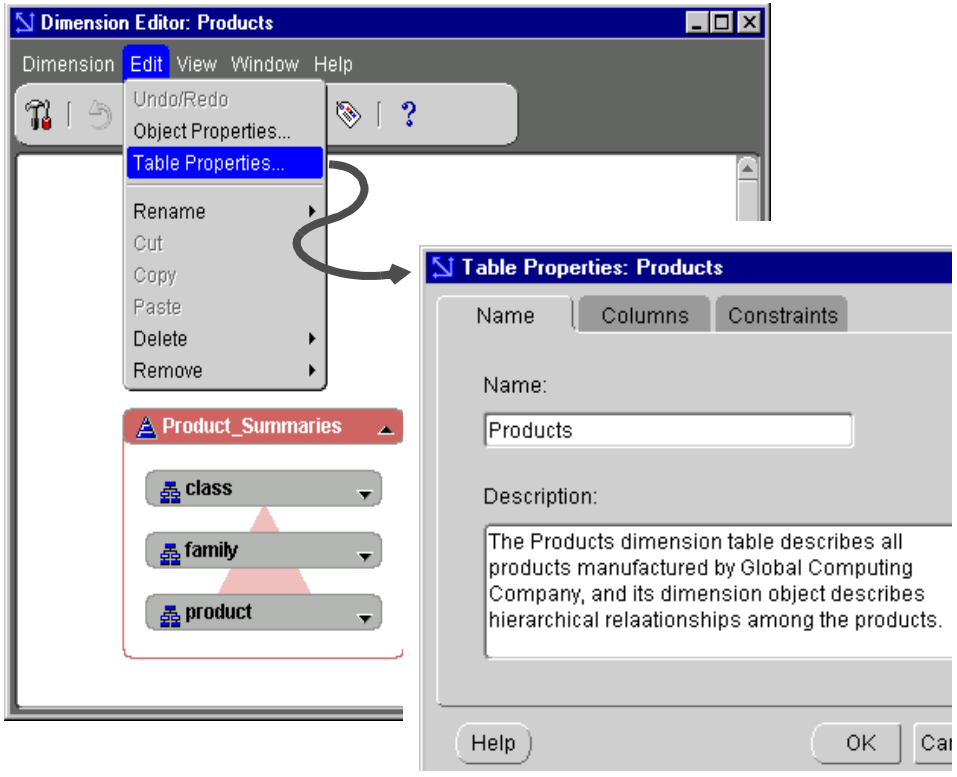
Two Property Sheets per Dimension When you create a definition for a dimension, Builder creates two property sheets: one for the dimension *object* and one for the dimension *table*. The dimension object's property sheet defines the levels and hierarchies, and its table property sheet defines the columns and constraints.

To display the property sheet for a dimension table:

1. From the navigation tree for the warehouse module, double-click **Products**.

   Builder returns the Dimension Editor in a separate window.

2. Select **Edit** on the toolbar and then select **Table Properties** from the pop-up list.

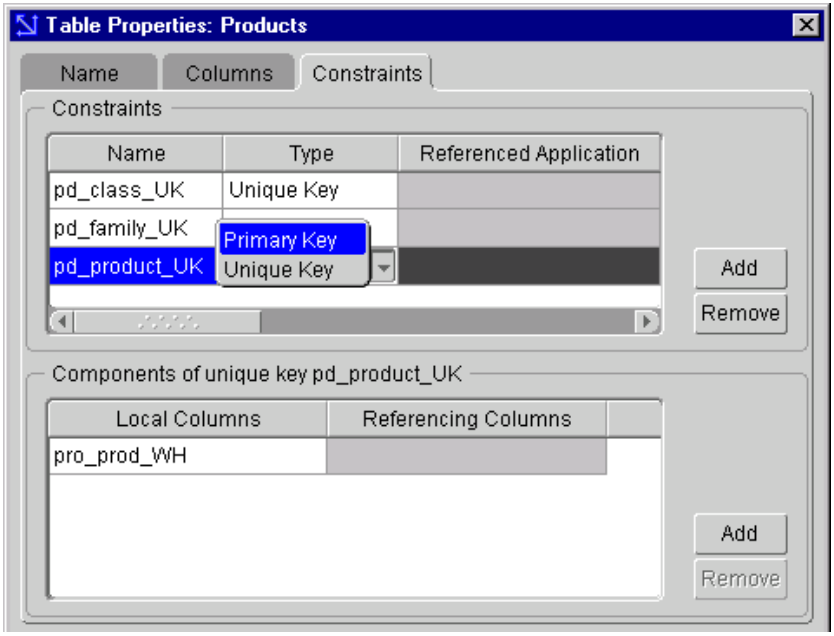


The property sheet for the dimension table contains three tabs: Name, Columns, and Constraints. You can update any active field by editing its contents.

Update a Constraint You can change the UK constraint defined on the *pro_prod_WH* column to a PK constraint in a few steps.

Original constraint name (and type):	pd_Product_UK (Unique Key)
New constraint name (and type):	pd_Product_PK (Primary Key)
Local column:	pro_prod_WH

1. Open the table's property sheet.
2. Select the **Constraints** tab.



3. Click the drop-down arrow under the Type column and select **Primary Key**.

> **Note:** Only the constraint on the base level is meaningful.

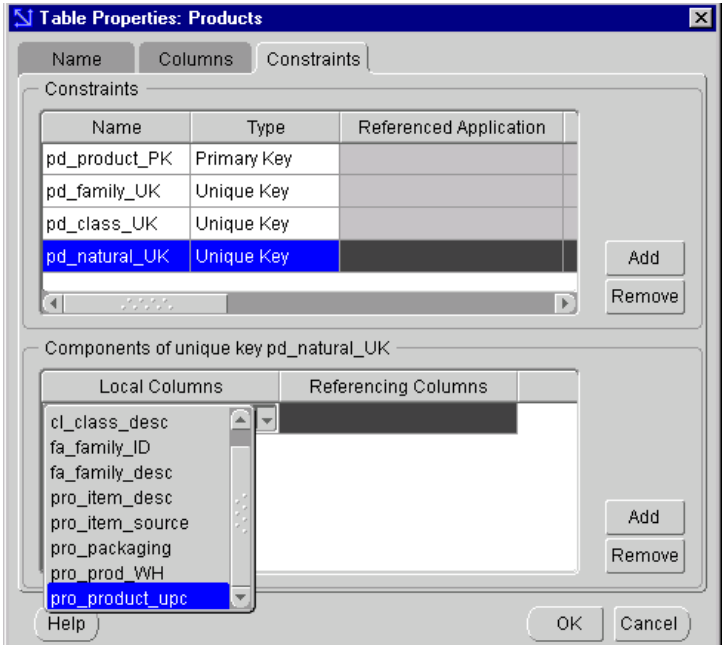
4. Replace the occurrence of 'UK' in the constraint name with 'PK'.

To complete the definition for the Products dimension, you must still create a constraint on the natural key column (*pro_product_upc*).

Add a UK Constraint You can create a UK constraint defined on the dimension's natural key column in a few steps.

Constraint name (and type):	pd_natural_UK (Unique Key)
Local column:	pro_product_upc

1. Open the table's property sheet.
2. Select the **Constraints** tab.



3. Click the upper **Add** button.

   Builder returns and highlights an empty row for the new constraint.
4. Enter the name of the new constraint (*pd_natural_UK*).
5. Click the drop-down arrow under **Type** and select **Unique Key**.
6. Click the lower **Add** button.

   Builder returns and highlights an empty row for the name of a local column.

7. Click the drop-down arrow under **Local Columns** and select *pro_product_upc*.

   Builder adds the constraint to the definition of the Products table.

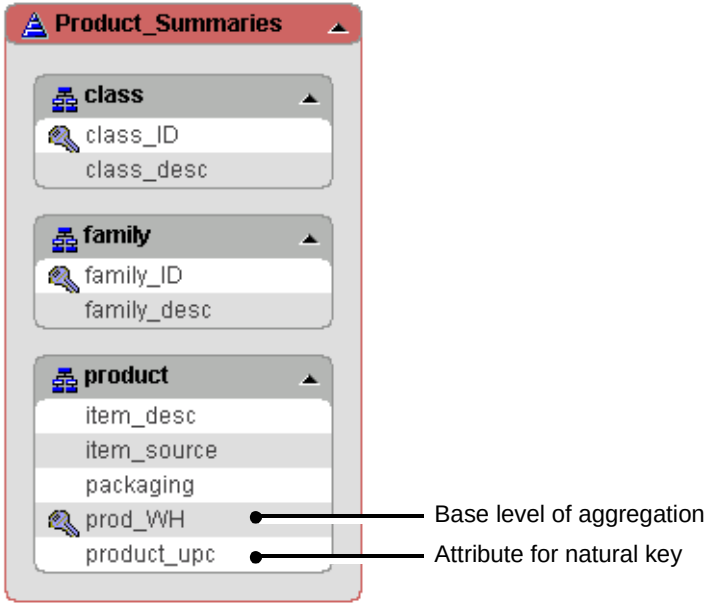
The definition for the Products dimension is now complete and you can move on and create definitions for the three remaining dimensions.

Review the Definition

Before you create definitions for the three remaining definitions, you should verify that the definition for the Products table is correct and complete.

Dimension Object The dimension object should define a single hierarchy named *Product_Summaries* that orders three levels. To display a graph of the dimension object, double-click ***Products*** in navigation tree. Builder returns the Dimension Editor window. Click the triangle for each level to expand the graph.

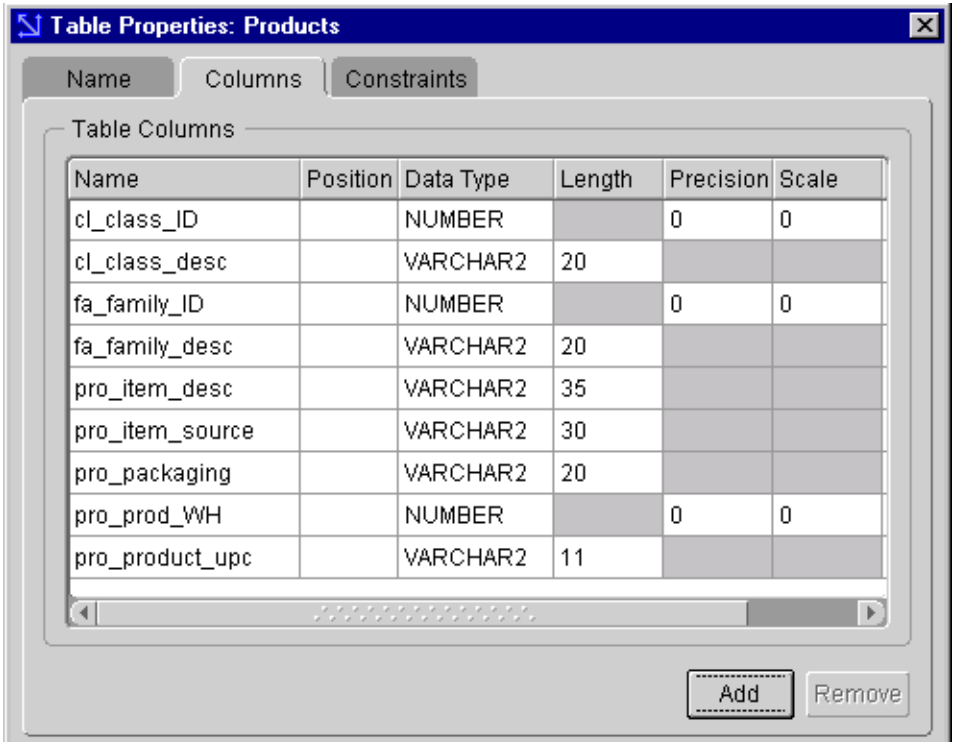
Figure 4–1 Products Dimension Object



You should use the graph to verify that all the attributes have been defined for each level, and the correct attribute was defined for the level key. The graph should show a key next to the *class_ID*, *family_ID*, and *prod_WH* attributes.

The Column Definitions Display the property sheet for the Products table and verify that the data types are correct for each column. To display the property sheet, select **Table Properties** from the Dimension Editor's **Edit** menu. Then, click the Columns tab on the property sheet.

Figure 4–2 Column Definitions for the Products Table

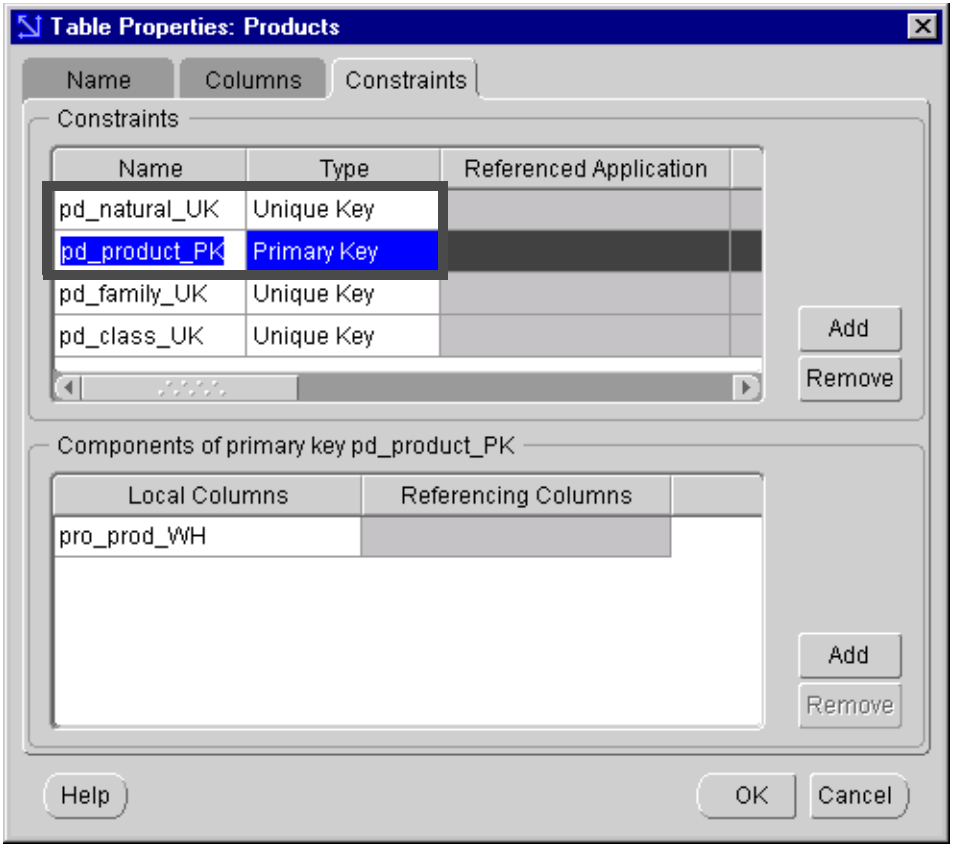


Position: To override the alphabetic order of the column names, enter a value in the Position field. Builder then lists the columns from the lowest to highest position value. When Builder generates the DDL script to create the table, it preserves the order you specify with position numbers.

Note: You can assign position numbers to columns in dimensions, facts, tables, views, and materialized views.

The Constraint Definitions The dimension table must have a primary key constraint defined on its warehouse key column and a unique key constraint defined on its natural key column. The constraints are named *pd_product_PK* and *pd_natural_UK*. Both constraints play an important role during the mapping phase.

Figure 4–3 Constraint Definitions for the Products Table



Note: Builder does not generate actual constraints for the two generated UK constraint names *pd_family_UK* and *pd_class_UK* but uses these names to generate the Create Dimension script.



Builder does not automatically commit your work to the repository. You can commit your work immediately or defer this action until you exit the session. To commit immediately, click the Commit icon on the toolbar. Builder returns a Commit Confirmation panel: click Yes. Alternatively, when you exit a Builder session, Builder always returns an Exit Confirmation panel. Click Yes.

Create Definitions for the Remaining Dimensions

You can use the New Dimension Wizard to create definitions for the three remaining dimensions. The following tables describe these dimensions. The subsequent illustrations in this cookbook assume that definitions for these dimensions exist in the *gccDM* warehouse module.

The Days Dimension

The Days dimension is assigned a 'da' prefix. This dimension also has a primary key constraint defined on its warehouse key column, a unique key constraint (*dy_natural_UK*) defined on its natural key column (*day_desc*), and a single hierarchy (*Calendar*) with a two character prefix (*ch*).

Table 4–2 describes each level of aggregation, the prefix for each level, and the attributes defined on each level. The levels occur in parent to child order: YearL is the parent of QuarterL, and QuarterL the parent of MonthL, and so forth.

Table 4–2 Days Dimension

Level	Prefix	Attribute	Data Type	Description
yearL	yl	year	number	Level identifier
quarterL	ql	quarter	number	Quarter number
monthL	ml	month_number	number	Level identifier (month of the year)
		day_of_month	number	Day of the month
		month_desc	varchar2(9)	Description of month
weekL	wl	week_of_year	number	Level identifier
dayL	dl	day_WH	number	Base level (*warehouse key*)
		day_desc	date	SQL date stamp (*natural key*)
		day_of_year	number	Day of the year

The Customers Dimension

The Customers dimension is assigned a 'cs' prefix. This dimension has a primary key constraint defined on its warehouse key column, a unique key constraint (*cs_natural_UK*) defined on its natural key columns (*account_ID* and *shipto_num*), and a single hierarchy (*Customer_Summaries*) with a two character prefix (*cs*). Table 4–3 describes each level of aggregation, the prefix for each level, and the attributes defined on each level. The levels of aggregation occur in parent to child order.

Table 4–3 Customers Dimension

Level	Prefix	Attribute	Data Type	Description
segment	sg	segment_ID	number	Level identifier
		segment_desc	varchar2(20)	Description of market segment
account	at	account_ID	number	Level identifier (*natural key*)
		account_desc	varchar2(35)	Description of account
customer	cs	cust_WH	number	Base level (*warehouse key*)
		loctype	varchar2(30)	Description of premises
		shipto_num	number	Shipping number (*natural key*)

The Channels Dimension

The Channels dimension is assigned a 'ch' prefix. This dimension has a primary key constraint defined on its warehouse key column, a unique key constraint (*ch_natural_UK*) defined on its natural key column (*channel_selector*), and no hierarchies. Table 4–4 describes the attributes defined on its base level of aggregation.

Table 4–4 Channels Dimension

Level	Prefix	Attribute	Data Type	Description
base	bs	chan_WH	number	Level identifier (*warehouse key*)
		channel_selector	number	Kind of channel (*natural key*)
		channel_desc	varchar2(20)	Description of market segment
		channel_class	varchar2(12)	Description of class

Note: The following examples assume that complete definitions for all the dimensions exist in the *GCC Star* warehouse module.

Fact Tables

This section shows you how to create a definition for the *Sales* fact table using the New Fact Wizard, how to display the fact table and its dimensions, and how to update the table by editing its property sheet.

The New Fact Wizard walks you through a three-step process one page at a time. After working through this example, you can use this same wizard to create any fact table that may be required.

The Sales Fact Table

The Sales fact table tracks the sales of products manufactured by the Global Computing Company. The table includes dollar sales, units sold, product costs, and margin. Values for the margin column are calculated as the arithmetic difference between sales and cost.

Table 4–5 describes the *Sales* table's foreign key reference and measures columns. The *Sales* fact table also has a composite primary key constraint defined on its foreign key reference columns.

Table 4–5 Sales Fact Table

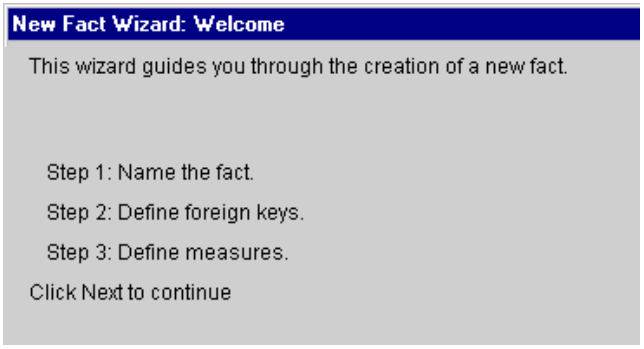
Column	Data Type	Description	Constraint Name
prod_WH	number(38)	FKR to *Products* dimension	pd_product_PK
cust_WH	number(38)	FKR to *Customers* dimension	cs_customer_PK
chan_WH	number(38)	FKR to *Channels* dimension	ch_base_PK
date_WH	number(38)	FKR to *Days* dimension	dy_dayL_PK
dollars	number(36, 2)	Sales in dollars	
units	number(38)	Number of units in the sale	
cost	number(36, 2)	Cost of products sold	
margin	number(30, 8)	Margin as dollars - cost	

Create the Definition

Open the *gccDM* module with the Warehouse Module Editor, and then call the New Fact Wizard from module's navigation tree:

1. Right-click **FACTS > Create Fact**.

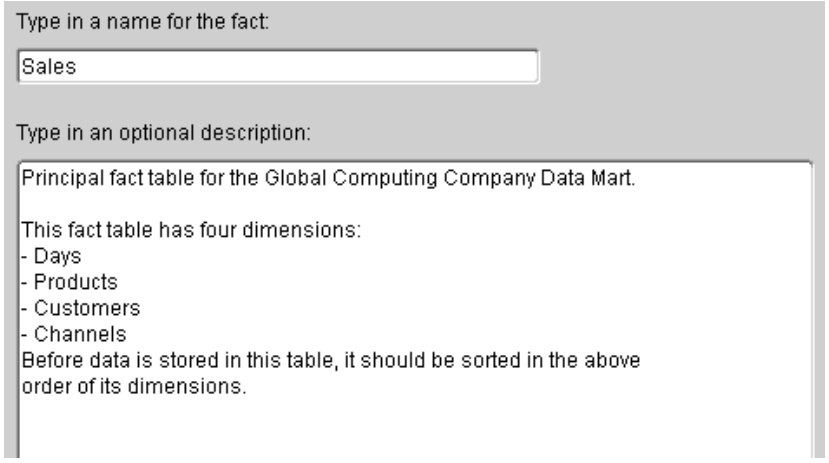
   Builder returns the welcome page for the New Fact Wizard which summarizes each of the three steps required to create the definition:



2. Click **Next**.

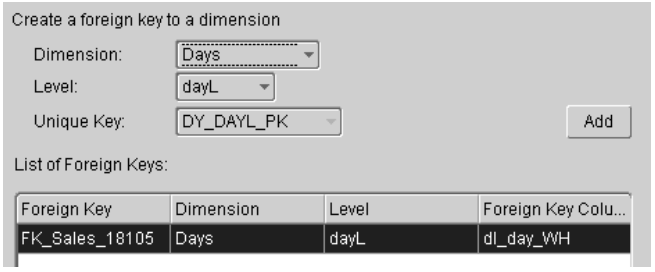
   The wizard returns its Name page which solicits a name and description.

3. Enter the fact table's name (*Sales*) and a concise description.



4. Click **Next**.

The wizard returns the Define Foreign Keys page. Use this page to explicitly define each of the table's foreign key references.



To define a foreign key reference:

1. Click the first **down-arrow** to select the name of a dimension (*Days*).
2. Click the next **down-arrow** to select the *base* or warehouse key Level (*dayL*).

   You must select the dimension's base level of aggregation (finest grain).
3. Click the third **down-arrow** to select the Primary Key column constraint defined on the dimension (*dl_day_WH*).
4. Click **Add**.

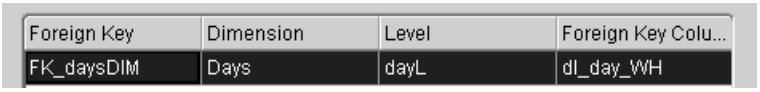
   The wizard generates a name for the foreign key reference constraint and inserts a line in the text box that lists the table's foreign keys.

Note:

- The Fact Wizard displays the name for each generated UK constraint on a dimension's level columns. Only the lowest level UK constraint is an actual physical constraint.
- You can not modify the name or data type of the foreign key reference columns. You can only do this by editing the definition for the referenced table.
- If you add a column to a PK or UK constraint on a dimension, you must also update the fact table foreign key references.

Repeat this four-step procedure to define a foreign key constraint for each dimension referenced by the fact table. *Be sure to choose the lowest level of aggregation for the foreign key reference target.*

Modify the name of the generated foreign key reference constraint to a more meaningful one: select the generated name and enter a new one (*FK_daysDIM*). The other three constraints are similarly named (*FK_prodsDIM*, *FK_custsDIM*, and *FK_chansDIM*). This name must be unique within the Project.



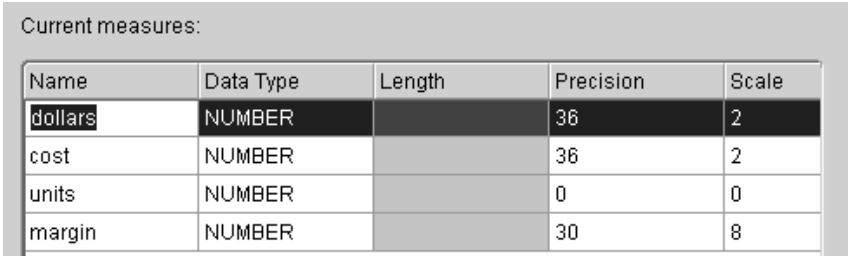
Foreign Key	Dimension	Level	Foreign Key Colu...
FK_daysDIM	Days	dayL	dl_day_WH

5. Check the box at the bottom of the page that says **Create segmented unique key automatically from foreign keys**.

   After the wizard creates and stores the definition, you must change the unique key constraint to a primary key constraint.

6. Click **Next**.

   The wizard returns its Define Measures page. Use this page to define each of the fact table's measures.



Current measures:

Name	Data Type	Length	Precision	Scale
dollars	NUMBER		36	2
cost	NUMBER		36	2
units	NUMBER		0	0
margin	NUMBER		30	8

To add a measure:

1. Click **Add**.
2. Enter a name.
3. Specify its data type.

Continue these steps until you have defined all the measures.

7. Click **Next**.

   The wizard returns its Finish page. Verify the description and make any necessary changes before you click Finish.

8. Click **Finish**.

   The wizard creates a definition for the *Sales* fact table and stores it in the warehouse module (*gccDM*) navigation tree.

Update the Fact Table Definition

A definition for the Sales table now resides in the Builder repository but it is incomplete. The constraint on its foreign key reference columns must be changed from a unique key to a primary key constraint. To make this change, you can update the definition by editing one of its property sheets.

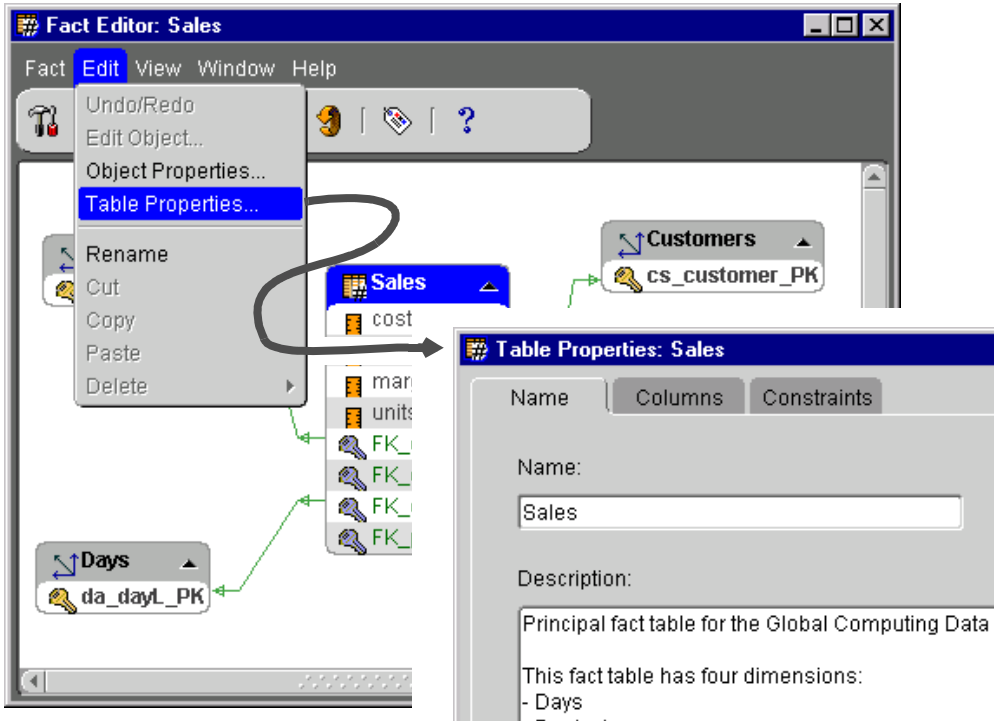
Two Property Sheets per Fact Like a dimension definition, a fact table definition has two property sheets: one for the Fact *Object* and another for the Fact *Table*. The sheet for the object defines its measures and foreign key references; the sheet for the table defines its columns and constraints. Thus, to change the unique key to a primary key constraint, you must edit the property sheet for the fact *table*.

To display the property sheet for the fact table:

1. From the navigation tree for the warehouse module, double-click ***Sales***.

   Builder opens the Fact Editor in a separate window.

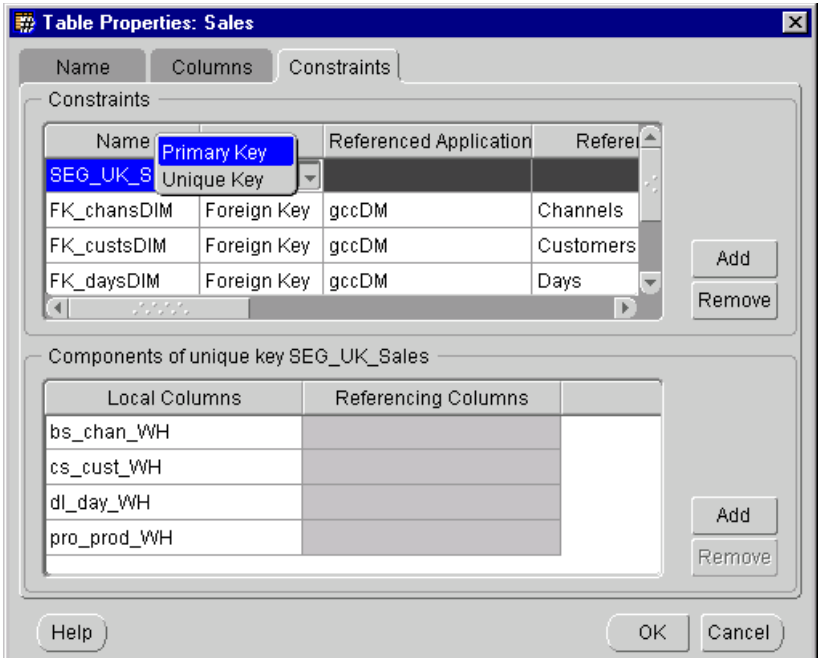
2. Select **Edit** on the toolbar and then select **Table Properties** from the pop-up list.



The property sheet for the fact table contains three tabs: Name, Columns, and Constraints. You can update any active field by editing its contents.

Change a Constraint You can change the UK constraint defined on the foreign key reference columns to a PK constraint in a few steps.

1. Open the table's property sheet and select the **Constraints** tab.



2. Select the constraint name ***SEG_UK_Sales***.
3. Click the **down-arrow** under the Type column and select **Primary Key**.
4. Replace the occurrence of '***UK***' in the constraint name with '***PK***'.
5. Click **OK**.

   Builder updates the definition of the underlying fact table.

The definition for the Sales fact table and the basic star schema for the Global Computing Company Data Mart are now complete. You can implement the schema as it stands, or create definitions for materialized views. The Query Rewrite algorithm can use the materialized views to greatly improve query performance.

Materialized Views

This section shows you how to create a definition for the *Product_Summaries* materialized view using the New Materialized View wizard and how to update the materialized view by editing its property sheet.

The New Materialized View Wizard walks you through a five-step process one page at a time. After working through this example, you can use this same wizard to create the *Customer_Summaries* materialized view.

The Product_Summaries Materialized View

The *Product_Summaries* materialized view summarizes the daily sales of each product manufactured by Global Computing Company. This view together with the Customer_Summaries view supports a wide variety of query rewrites that speed up query execution.

The degree of query rewrite depends on the explicitly defined hierarchies (*Calendar*, *Product_Summaries*, and *Customer_Summaries*), the not null constraints defined on referenced dimension columns, activation of the cost-based optimizer, and the Oracle8*i* configuration parameter QUERY_REWRITE_ENABLED.

Table 4–6 describes the columns and foreign key references defined for the *Product_Summaries* materialized view. This view also has a composite primary key constraint defined on its foreign key reference columns.

Table 4–6 Product_Summaries Materialized View

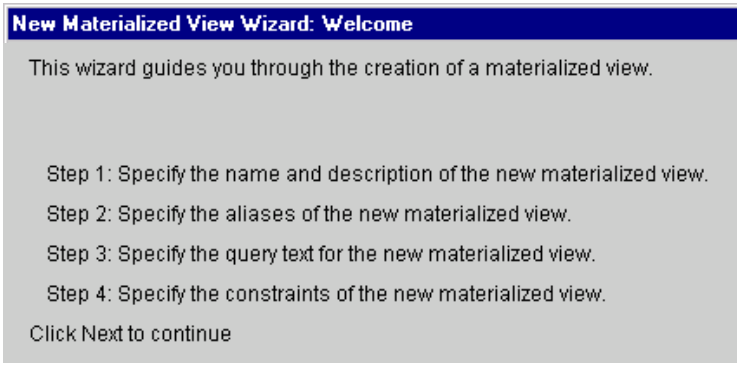
Alias	Data Type	Description	Referenced Key
day_WH	number(38)	FKR column to *Days*	dl_day_PK
product_WH	number(38)	FKR column to *Products*	pd_product_PK
sales	number(36, 2)	Daily subtotal	
cost	number(36, 2)	Daily subtotal	
units	number(38)	Daily subtotal	

Create the Definition

Open the *gccDM* module with the Warehouse Module Editor, and then call the New Materialized View Wizard from module's navigation tree:

1. Right-click **MATERIALIZED VIEWS > Create Materialized View**.

   Builder returns the welcome page for the New Materialized View Wizard which summarizes each of the four steps required to create the definition:



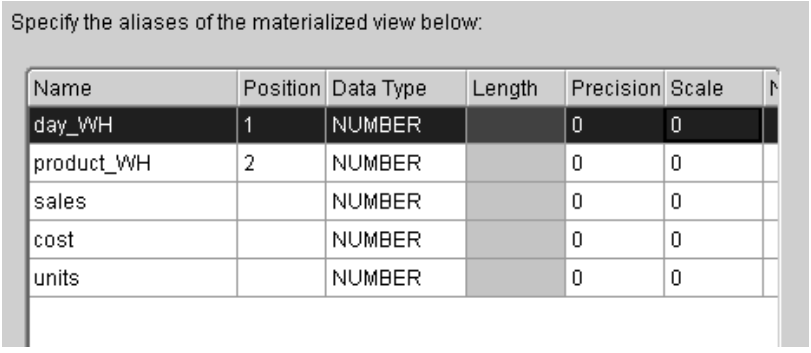
2. Click **Next**.

   The wizard returns its Name page which solicits a name and description.

3. Enter the materialized view's name (*Product_Summaries*) and a concise description.

4. Click **Next**.

   The wizard returns the Columns page. Use this page to explicitly define each of the materialized view's columns (or an alias for each column).



To define a column:

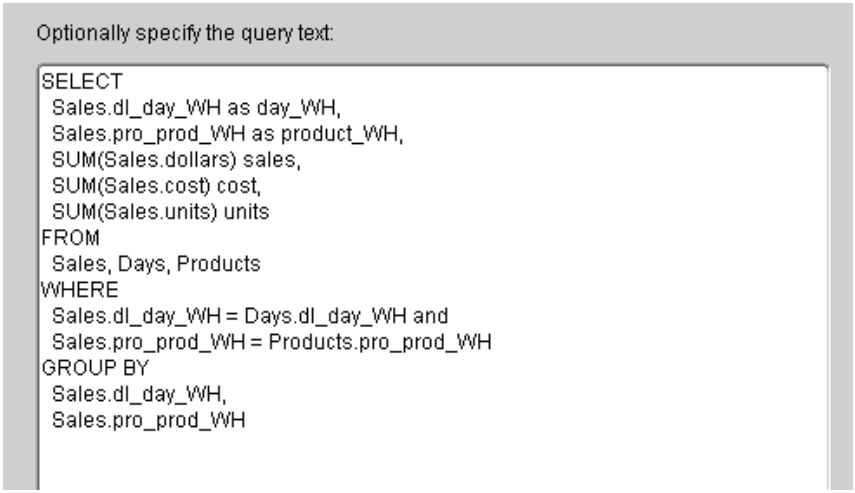
1. Click **Add**.
2. Enter the column's name.
3. Specify the column's data type and length (or precision).

Repeat this procedure for each column.

5. Click **Next**.

The wizard returns its Query Text page. Use this page to enter text for the query that defines the view.

This page is optional because there are two methods to define a materialized view: enter the query text for the view, or create a mapping definition for the view. This examples uses the first method.



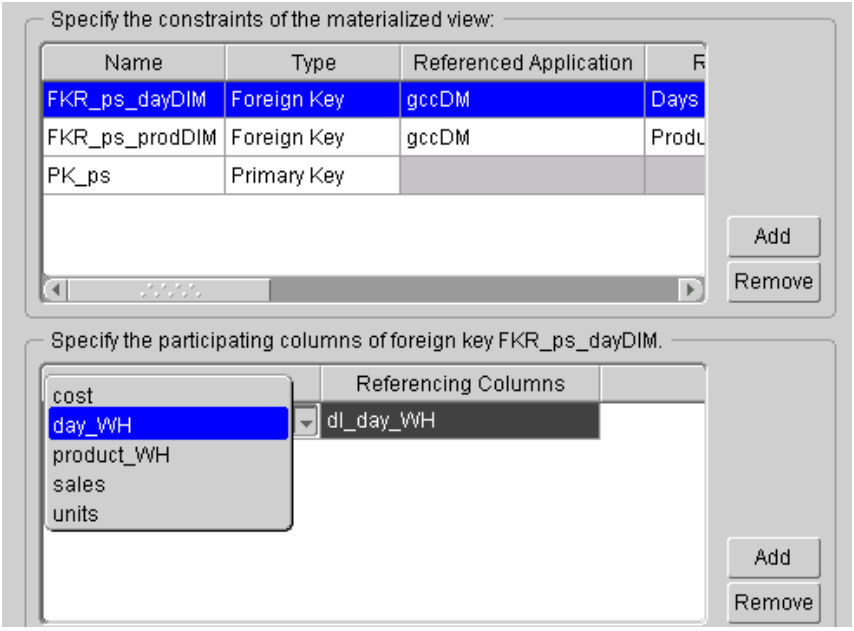
Usage Notes:

- Builder does not generate code for a materialized view if its query text is not included in its Property Sheet nor if it has no columns defined.
- Builder generates a Create Materialized View statement to deploy the view even if its syntax is invalid: Builder does not check the syntax of the select statement used to define a materialized view.

6. Click **Next**.

The wizard returns its Define Constraints page. Use this page to define any constraints defined on the materialized view's columns.

The *Product_Summaries* materialized view has three constraints. It references the *Days* and *Product* dimensions over foreign key reference constraints and includes a primary key constraint on its foreign key reference columns (*day_WH* and *product_WH*).



To define a foreign key constraint:

1. Click **Add** adjacent to the upper box and enter the constraint's name.
2. Select the constraint type: Foreign Key.
3. Select the following:
   * Referenced Application (defined within a Warehouse Module)
   * Referenced table
   * Referenced key
4. Click **Add** adjacent to the lower text box and then select the column local to the materialized view (click the **down-arrow**) that makes the foreign key reference.

Repeat these steps until all of the foreign key constraints have been defined.

To define the PK constraint:

1. Click **Add** adjacent to the upper box, then enter the constraint's name and select its type.
2. Click **Add** adjacent to the lower text box and then select the local columns that participate in the constraint.

7. Click **Next**.

   The wizard returns its Finish page. Verify the description and make any necessary changes before you click Finish.

8. Click **Finish**.

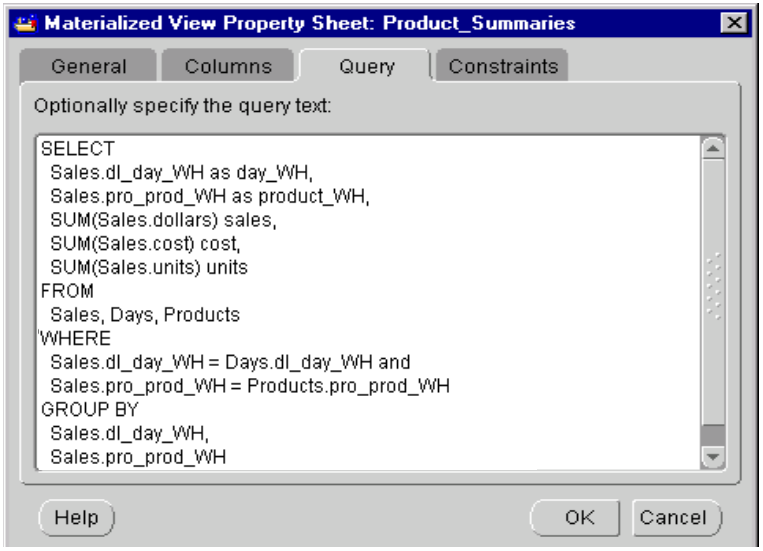
   The wizard creates the definition and stores it in the *gccDM* warehouse module.

The definition for the *Product_Summaries* materialized view is now complete.

Update the Definition

You can update the definition if necessary by editing its property sheet.

To display the property sheet for a materialized view, open the *gccDM* warehouse module, right-click the view's name in the navigation tree, and click **Properties**:



You can update the view by editing any activated fields within the sheets.

The Customer_Summaries Materialized View

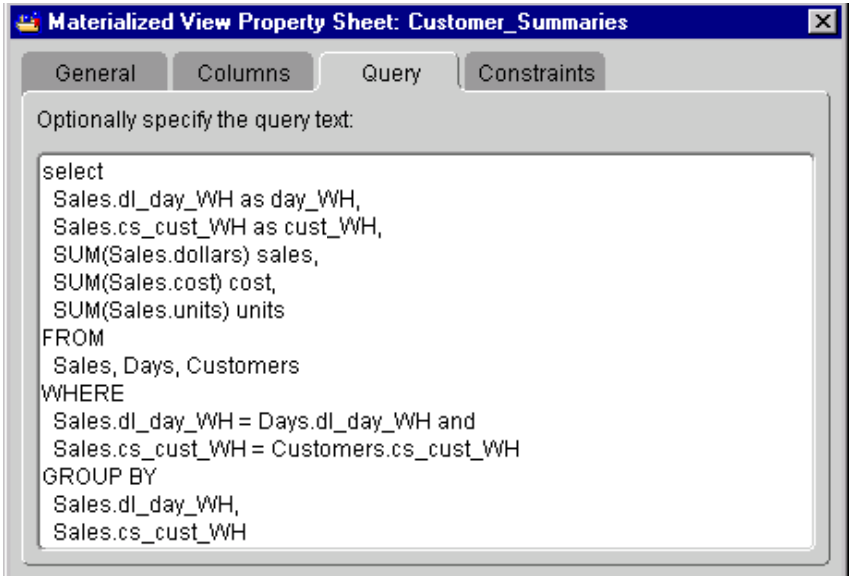
The *Customer_Summaries* materialized view summarizes the daily sales to each of Global Computing Company's customers. You can, as an exercise, create a definition for this view using the New Materialized View Wizard (as described in "Create the Definition" on page 4-25 for the *Product_Summaries* materialized view).

Table 4–7 describes the columns and foreign key references defined for the *Customer_Summaries* materialized view. This materialized view also has a composite primary key constraint defined on its foreign key reference columns.

Table 4–7 Customer_Summaries Materialized View

Alias	Data Type	Description	Referenced Key
day_WH	number(38)	FKR column to *Days*	dl_day_PK
cust_WH	number(38)	FKR column to *Customers*	cs_customer_PK
sales	number(36, 2)	Daily subtotal	
cost	number(36, 2)	Daily subtotal	
units	number(38)	Daily subtotal	

You can enter the following query text for the view or create a mapping definition.



Conventional Views

This section shows you how to create and update definitions for a conventional view. You create a conventional view definition just like you create the a materialized view definition except that the New View Wizard is used instead of the New Materialized View Wizard. The tasks and wizard pages are identical.

The DeskTops View

The *DeskTops* view summarizes the daily sales of each product manufactured by Global Computing Company. The view contains data *only* for products that are within the Desk Top family.

Table 4–8 describes the columns defined for the *DeskTops* view.

Table 4–8 DeskTops View

Alias	Data Type	Description
day_WH	number(38)	Alias for *Days* dimension column dl_day_PK
product_WH	number(38)	Alias for *Products* dimension column pd_product_PK
sales	number(36, 2)	Daily subtotal
cost	number(36, 2)	Daily subtotal
units	number(38)	Daily subtotal

Although this view has exactly the same structure of the *Product_Summaries* materialized view, the views differ significantly:

- **Function**: The *DeskTops* view restricts access to a single family of products; the *Product_Summaries* materialized view makes queries run faster.
- **Visibility**: The *DeskTops* view is visible to a class of users; the *Product_Summaries* materialized view and its operation are transparent to all users.
- **Physical Storage**: The *DeskTops* view occupies no storage space; the *Product_Summaries* materialized view does occupy storage space.

Nevertheless, the procedure to create the views is the same, page by page.

Create the View

Open the *gccDM* module with the Warehouse Module Editor, and then call the New View Wizard from module's navigation tree:

1. Right-click **VIEWS > Create View**.

   Builder returns the welcome page for the New View Wizard which summarizes each of the four steps required to create the definition.

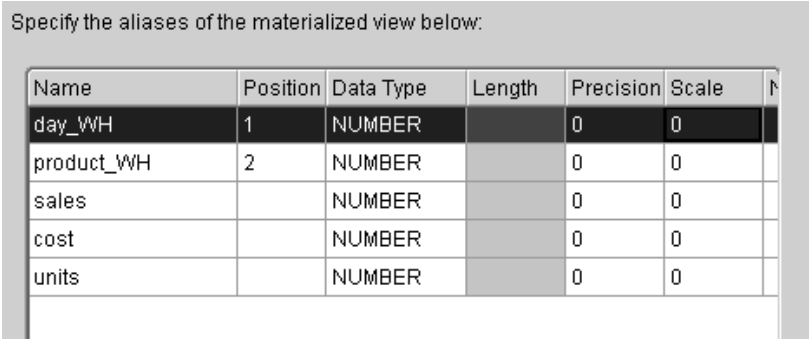
2. Click **Next**.

   The wizard returns its Name page which solicits a name and description for the view.

3. Enter a name (*DeskTops*) and a concise description for the view.

4. Click **Next**.

   The wizard returns its Columns page. Use this page to explicitly define each of the view's columns (or an alias for each column).



   To define a column:

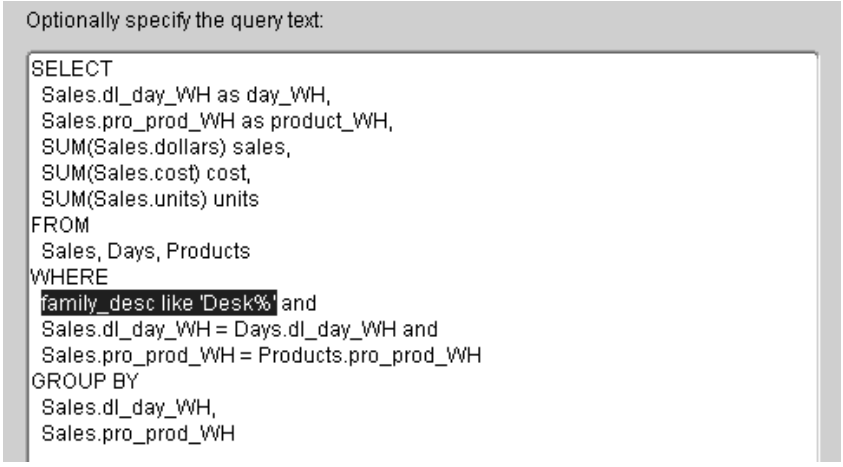
   1. Click **Add**.
   2. Enter the column's name.
   3. Enter the column's data type and length (or precision).

   Repeat this procedure for each column.

5. Click **Next**.

   The wizard returns its Query Text page. Use this page to enter text for the query that defines the view.

This page is optional because there are two methods to define a view: enter the query text for the view or create a mapping definition for the view. This examples uses the first method.



Usage Notes:

- Builder does not generate code for a view if its query text is not included in its Property Sheet nor if it has no columns defined.
- Builder generates a Create View statement to deploy the view even if its syntax is invalid: Builder does not check the syntax of the select statement used to define a view.

6. Click **Next**.

   Builder returns the Finish page. Verify the description and make any necessary changes before you click Finish.

7. Click **Finish**.

   The wizard creates the definition and stores it in the *gccDM* warehouse module.

The definition for the *DeskTops* view is now complete.

Sequence Objects

The case study uses sequence objects to populate the warehouse key columns, and you must create a definition for each of these sequence objects.

You can create definitions for a sequence object implicitly or explicitly depending on how the table is populated. If the table is populated by the SQL*Loader, then the control file for the loader implicitly defines the sequence object. If the table is populated with a PL/SQL package, then you must explicitly create a definition for the sequence object.

The Channels and Products dimensions are populated from flat files by the SQL*Loader, and the Days and Customers dimensions are populated from database tables by PL/SQL packages. Thus, you must define two sequences: one for the Days dimension (*daysWHkey*), and the other for the Customers dimension (*custWHkey*).

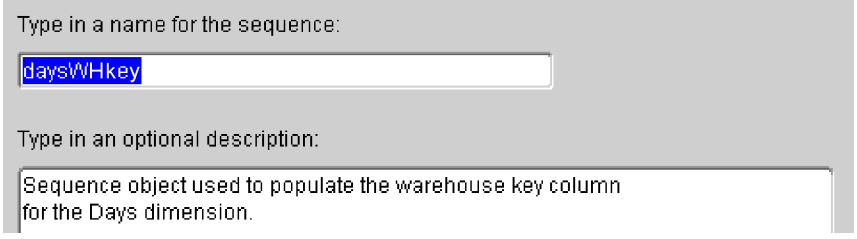
You create a definition for a sequence using the New Sequence Wizard. To create this sequence object, open the *gccDM* module, and then call the New Sequence Wizard from the module's navigation tree:

1. Right-click **SEQUENCES > Create Sequence**.

   Builder returns the welcome page for the New Sequence Wizard which summarizes the single step required to create definitions for a sequence.

2. Click **Next**.

   The wizard returns the Name page.



   Enter the name (*daysWHkey*) and description for the sequence.

3. Click **Next**.

   Builder returns the Finish page. Verify the definition before you click **Finish**.

4. Click **Finish**.

   The wizard creates the definition and stores it in the *gccDM* warehouse module.

You must also create a definition for the *custWHkey* sequence object.

Current State of the Project

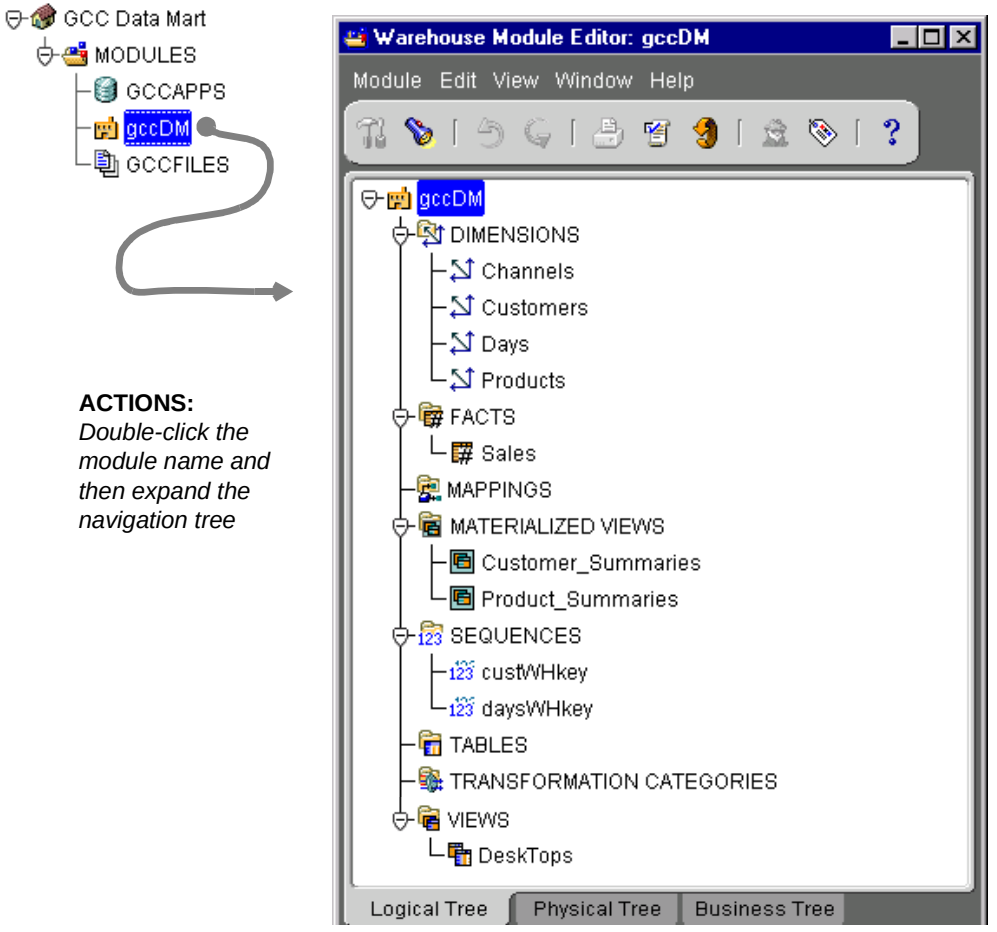
At this stage, the Builder repository now contains a set of definitions for the data mart's star schema. These definitions reside in the *gccDM* warehouse module.

To view the contents of this module, open the module and then

1. Select the module's name
2. Select > **View** > **Expand All** from the toolbar.

Figure 4–4 displays the fully expanded navigation tree for the *gccDM* module.

Figure 4–4 Contents of gccDM Warehouse Module



The Next Step

This chapter showed you how to create and update definitions for the data mart's target schema. The definitions were created and stored in the *gccDM* warehouse module using Builder wizards. Examples also showed you how to update the definitions by editing their property sheets.

Now that a logical description of the target schema exists, you are ready to move on to the next step which is described by the recipe's next step:

1. Create the definitions for the target schema.
2. *Create the definitions for the source data.*
3. Create the definitions for the mapping operations.

The next chapter, "Source Definitions," shows you how to create the definitions for the source data using the Source Module Editor and its associated wizards. You can use these wizards to import definitions of database tables or create formats for flat files. You can also update the definitions by editing their property sheets.

5

Source Definitions

This chapter shows you how to create definitions for the case study's data sources using Builder wizards. These sources were summarized in Chapter 2.

You must create a definition for each table and each file that serves as a source of data. A series of examples show you how to import definitions into the *GCCAPPS* source module from the order entry database system using the Import Metadata Wizard. Another series of examples show you how to create formats that describe these files in the *GCCFILES* source module using the same wizard. You also learn how to update and display the definitions.

This chapter covers the following topics:

- The Basic Steps
- The Database Sources
- The Flat File Sources
- Current Status of the Project

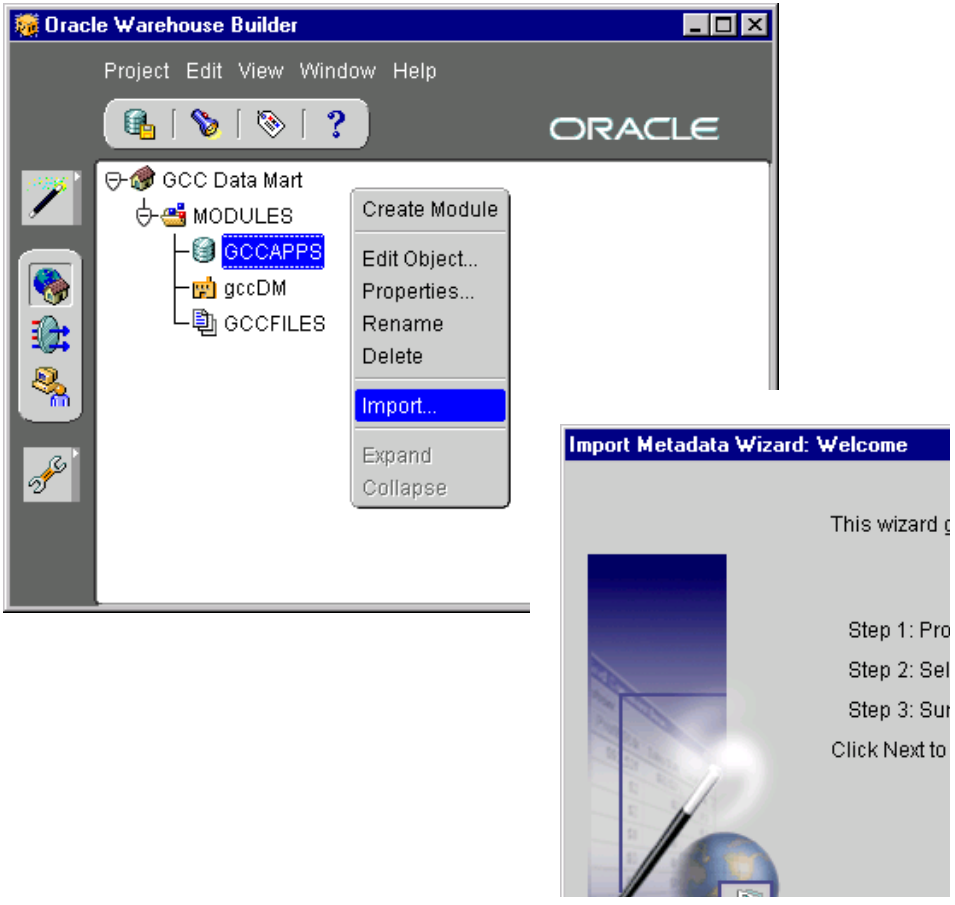
The Basic Steps

This chapter shows you how to create definitions in a source module for the case study's data sources using the Metadata Import Wizard and the Flat File Sample Wizard. All of the examples begin with the Metadata Import Wizard.

To call the Metadata Import Wizard

1. Fully expand the navigation tree for the *GCC Data Mart* Project.
2. Right-click the name of a source module (*GCCAPPS*) and select **IMPORT** from the pop-up list.

   Builder calls the wizard Metadata Import Wizard.
3. Provide the information solicited by the wizard, one page at a time.



The Database Sources

This section shows you how to import definitions for all the tables in the order entry database system using the Metadata Import Wizard. This wizard walks you through a three-page process one page at a time. After creating the definitions, you can update and display them using the Source Module Editor.

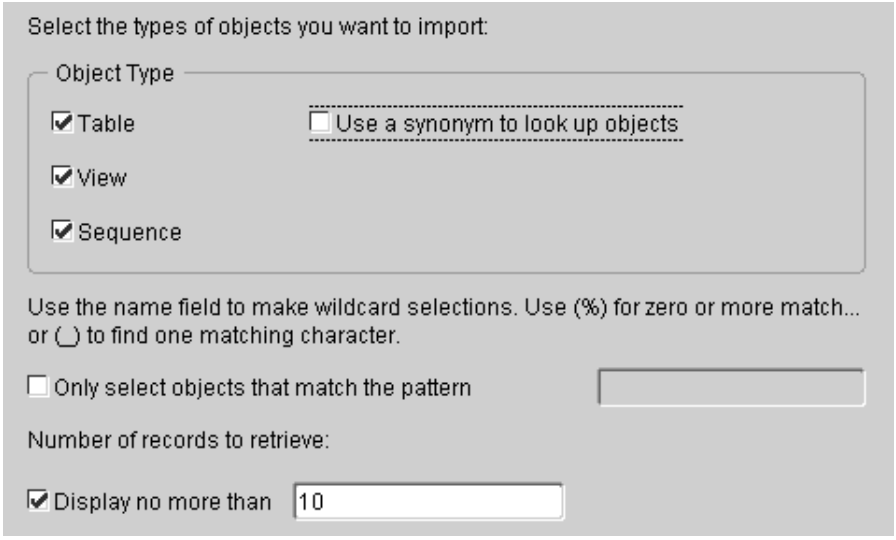
Import the Definitions

To import definitions from a database application:

1. Fully expand the navigation tree for the *GCC Data Mart* Project.
2. Right-click *GCCAPPS* > **Import** …

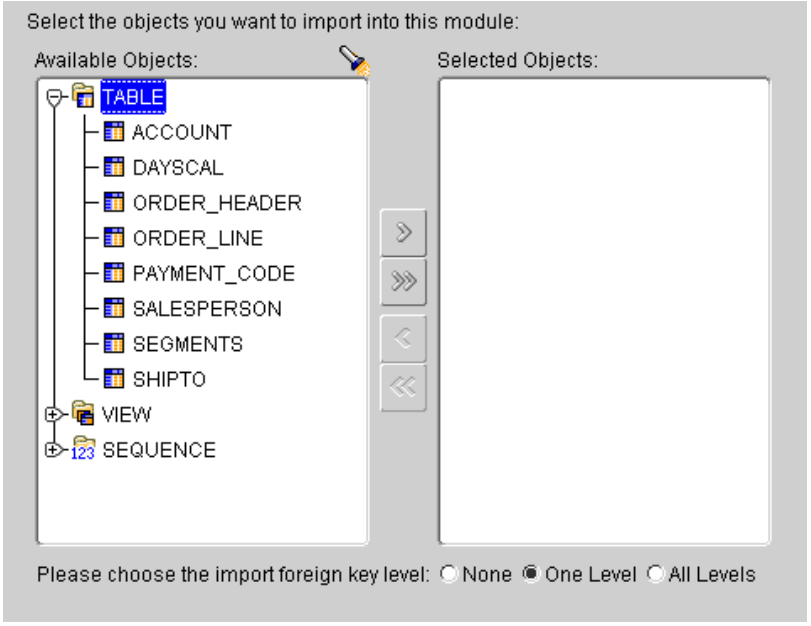
   Builder returns the welcome page for the Import Metadata Wizard which summarizes the steps required to import a set of definitions.
3. Click **Next**.

   The wizard returns the Filter Information page. Use this page to limit the search through the application's data dictionary. You can limit a search using check boxes for target objects, a search name, or a max value.



4. Click **Next**.

Builder connects with the application over the database link defined in the source module, reads its data dictionary, retrieves the list of names that meet the search condition, and then displays their names in the Object Selection page.



To import definitions, expand the navigation tree and then use the arrow keys to move the names from the Available to the Selected box.

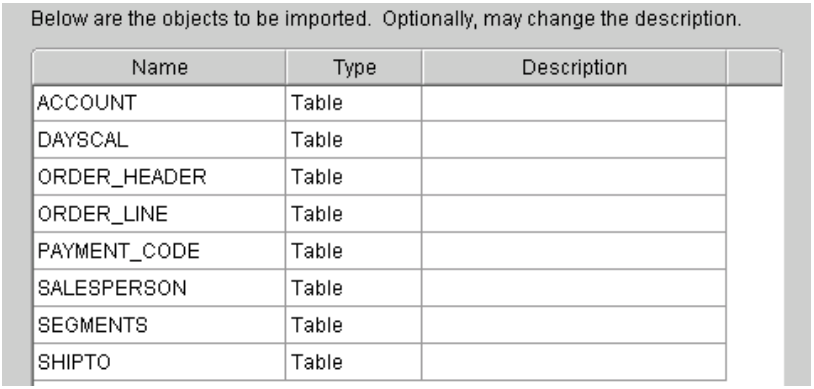
5. Select the **All Levels** button

   This button directs the wizard to import the definition of selected tables and any tables they reference directly or indirectly. When a table *X*, references another table *Y*, and table *Y* references table *Z*, *X* is said to indirectly reference *Z*.

6. Move the names of all eight tables in the Available Objects box to the Selected Objects box using the arrow keys.

7. Click **Next**.

   The wizard returns the Summary and Import page. This page summarizes your selections. Verify the contents of this page and add descriptions for each of the objects before you click Finish.



8. Click **Finish**.

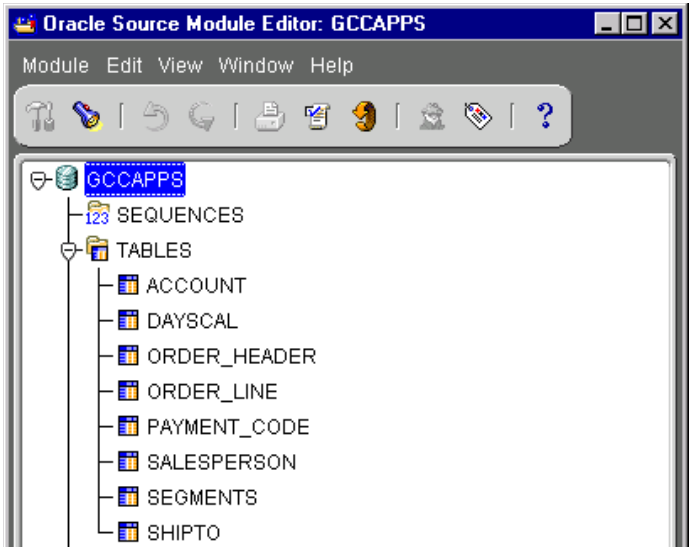
   Builder stores definitions for these tables in the *GCCAPPS* module.

All of the database source tables have now been defined.

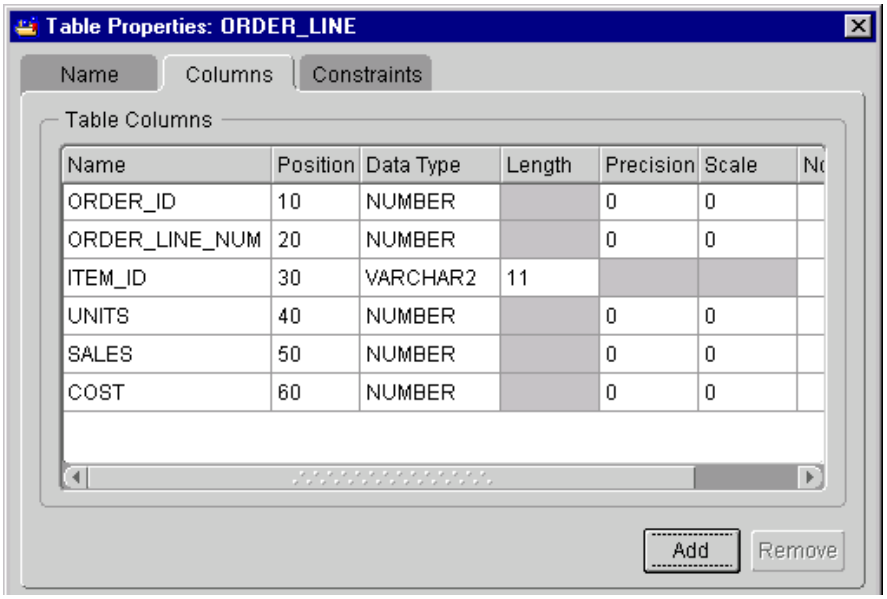
Update the Source Definitions

You can view and update the individual source definitions using the Source Module Editor. These updates apply only to the definitions stored in the Builder repository.

To call this editor, double-click the name of the source module (*GCCAPPS*):



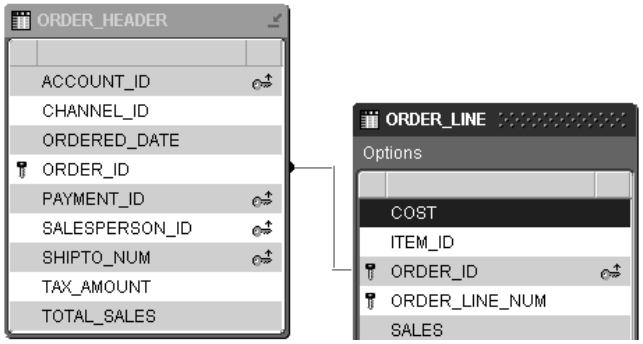
Update a definition You can update the definition of a source table by editing entries in its property sheet. To display a table's property sheet, right-click its name in the navigation tree and then select Properties from the pop-up list.



To update an entry, select the entry and enter the new information. Some entries have down-arrow lists that limit the range of selections. For example, when you change the data type of a column, you must choose an entry from a down-arrow list. You can also add new entries or remove existing ones.

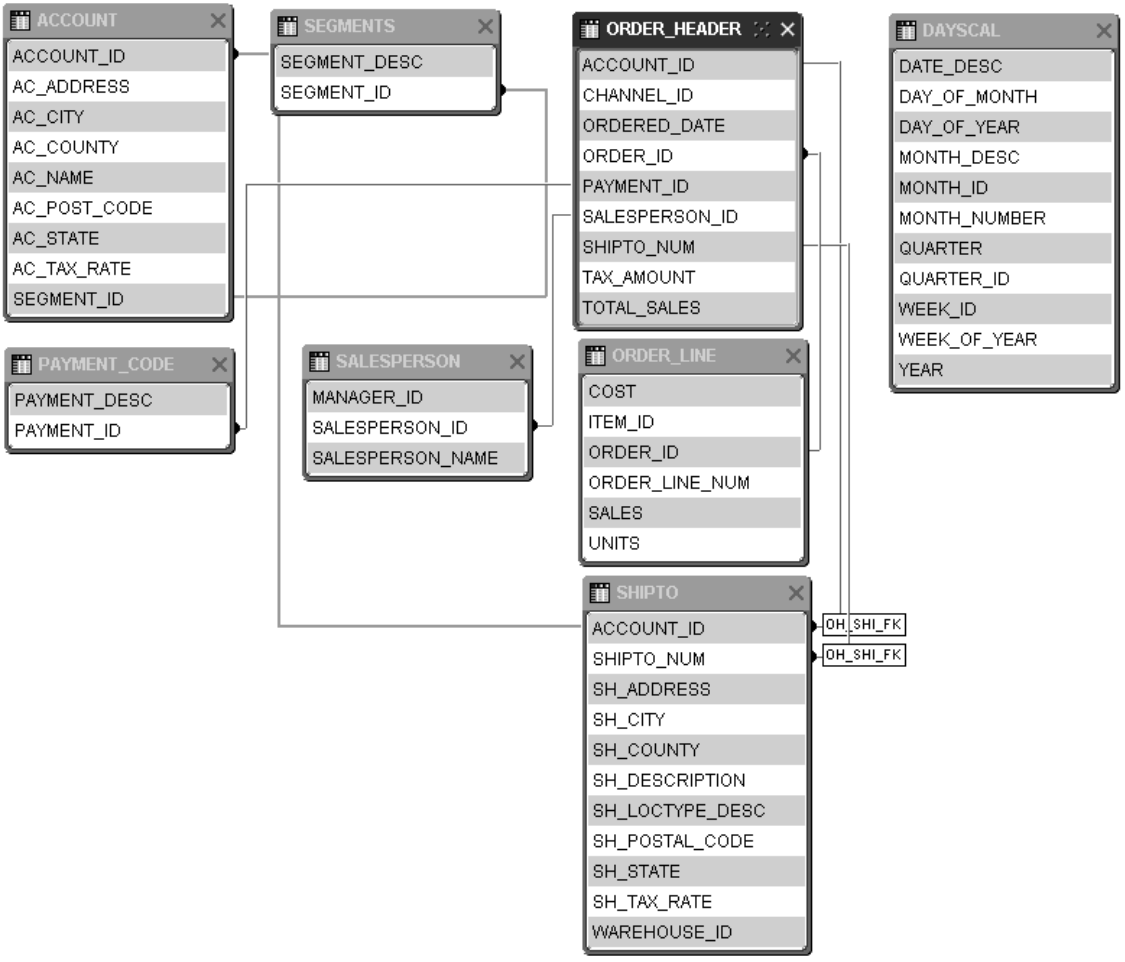
Order the Columns The column names within a table's property sheet are ordered alphabetically by default. You can change this default ordering by assigning a position number to each column or to selected columns in the property sheet. To see the new ordering, you must close and reopen the editor. The editor then orders the columns with position numbers first (increasing value) and the columns without position numbers second (alphabetic order).

Diagram a Source Definition To display a diagram for a definition, select its name in the navigation tree and then select **Edit** from the pop-up list.



Display All the Definitions To display a diagram for all the definitions within a source module, select **View** from the Source Module Editor's toolbar, and then select **Schema** from the drop-down list.

Figure 5–1 GCC Database Application



The diagrams and property sheets for the database data sources are useful during the development of a data mart project because they promote communication and tend to minimize errors.

The Flat File Sources

This section shows you how to create format definitions for the two flat file sources and store them in the *GCCFILES* source module using the Metadata Import Wizard. These examples also show you how to sample the flat files so that you can create the format definitions faster.

There are two examples. The first example shows you how to create a format definition for a fixed-format file (*products.fix*) and the other shows you how to create a format definition for a character separated file (*channels.csv*). Both examples begin with the Metadata Import Wizard.

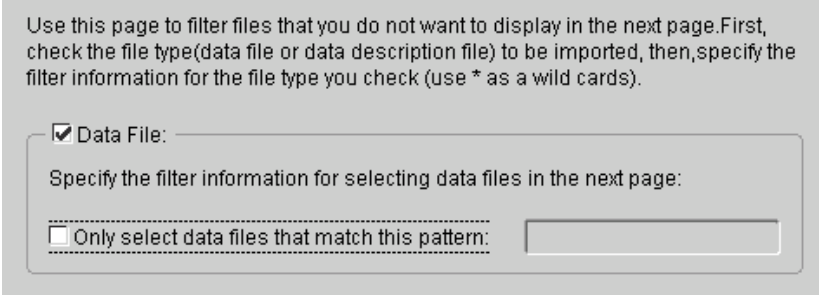
Create Format Definitions

To create a format definition for the two flat files:

1. Fully expand the navigation tree for the *GCC Data Mart* Project.
2. Right-click *GCCFILES* > **Import** …

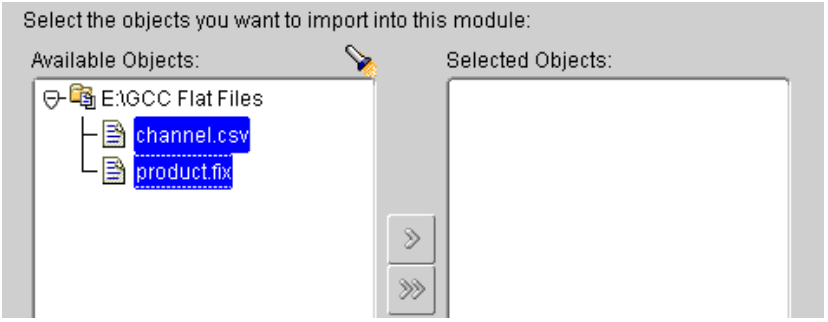
   Builder returns the welcome page for the Import Metadata Wizard which summarizes the steps required to import a set of definitions.
3. Click **Next**.

   The wizard returns the Filter Information page. Use this page to limit the search through the application's directory.



4. Click **Next**.

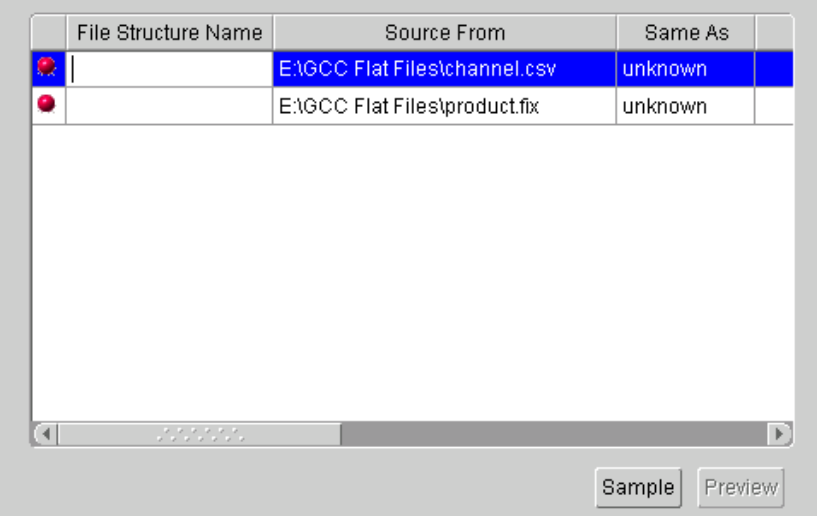
   The wizard returns the Object Selection page. Use this page to select the files that serve as data sources by moving file names from the Available Objects pane to the Selected Objects pane.



5. Move the files named *channel.csv* and *product.fix* to the Selected Objects pane using the arrow buttons.

6. Click **Next**.

   The wizard returns the Summary and Import page.



The left-most column of this page contains a red or green status ball:

Green indicates that Builder already has a definition of the file's format.

Red indicates that Builder does not have a definition for the file's format and you must create one using the Flat File Sample Wizard.

In the present case, the ball is red for both cases and thus you must create a format definition for each file.

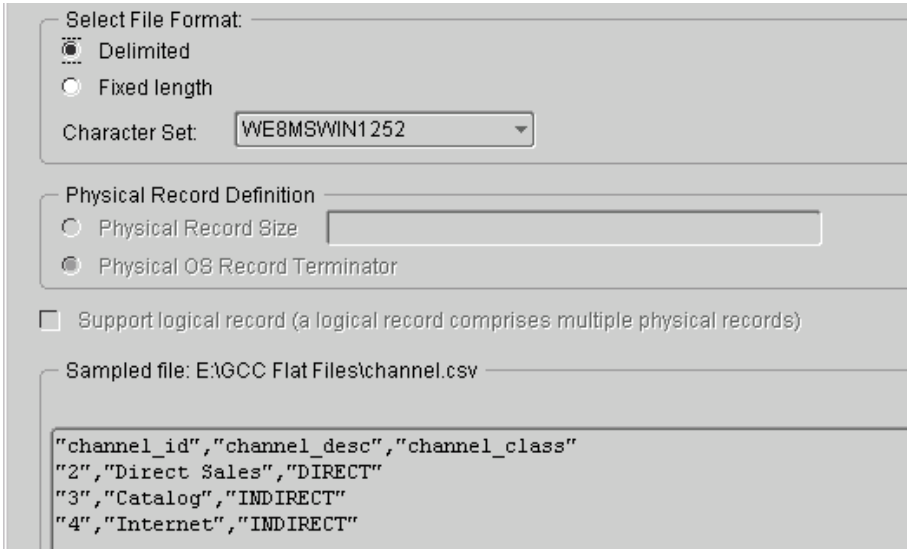
To create a format definition for the channel.csv file

1. Select the file's name, and then click **Sample** at the bottom of the Summary and Import page.

   The wizard returns the welcome page for the Flat File Sample Wizard which summarizes the steps.

2. Click **Next**.

   The wizard opens the file, reads a sample of data, and returns the File Format page. This page displays a sample of data in with a few initial values set for the file's global properties.



   The global properties include:

   * *Delimited* or Fixed Length
   * Character set (*WE8MSWIN1252)*

     This value reflects the active character set for the Windows NT system that hosts Builder.

   * Physical Record size (Applicable only for a fixed length file).
   * Multiple physical records (Applicable only for a fixed length file).

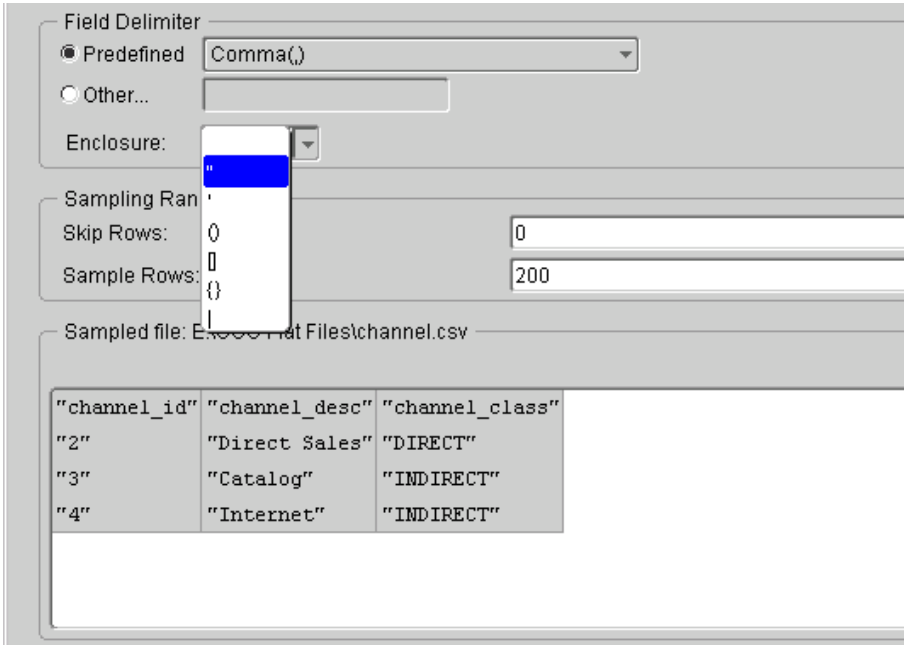
3. Look at the sample, verify that the global parameters are correct, and reset them if necessary. In the present case, they are all correct.

Note: If Builder's character set differs from the source's, then the sample might not be readable. In that case, select the native character set of the source from the down-arrow list and Builder then translates the sample accordingly. For complete information on NLS character sets, see the *Oracle8i National Language Support Guide Release 2 (8.1.6).*

4. Look at the data sample, and determine the delimiter character (*comma*), the presence of a header record (*yes*), and the presence of quote characters which are not part of the actual data (*double quotes*).

5. Click **Next**.

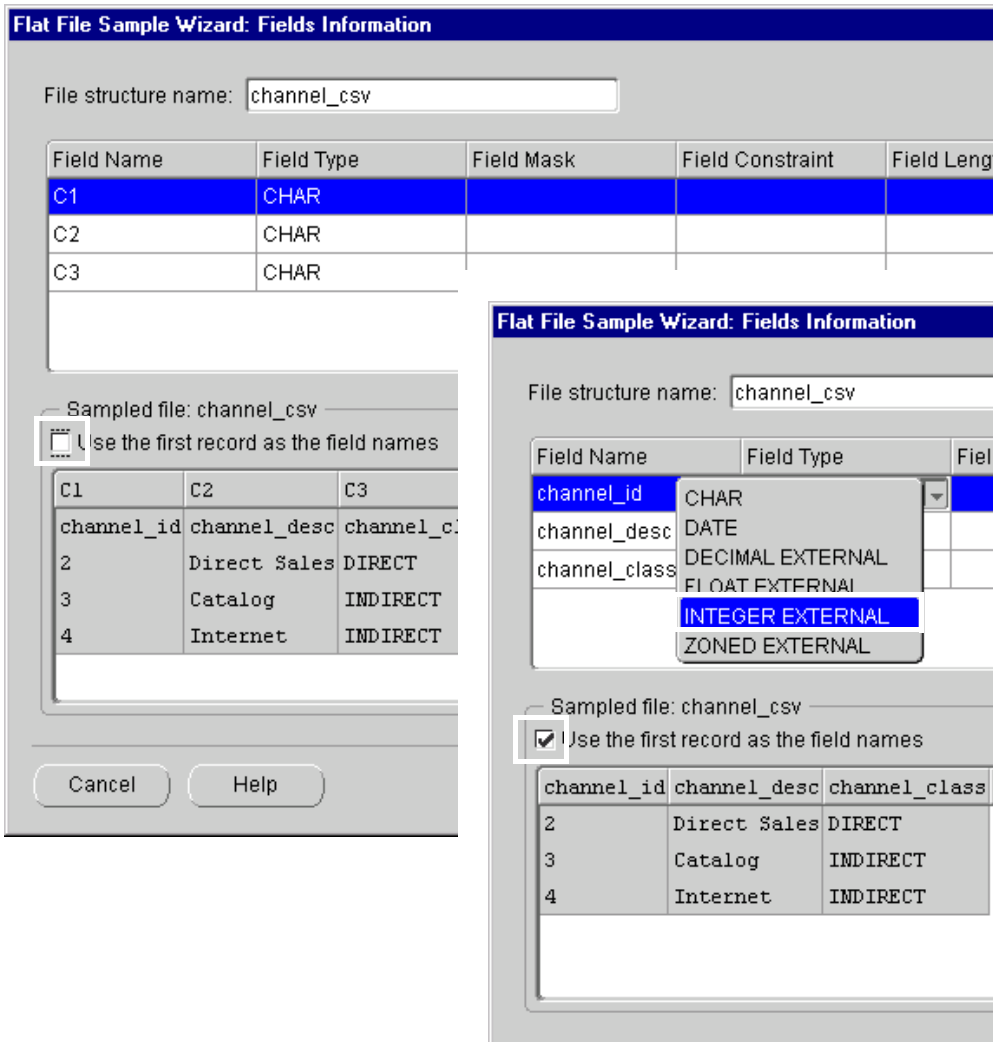
   The wizard returns the Field Breaking page. This page solicits specific information about the delimiter and enclosure characters. You can also sample a different range of records.



6. Select the double quote character from the down-arrow list and Click **Next**.

   The wizard returns the Fields Information page. Use this page to complete the description: check the "Use first record as the field names" box and

select the correct datatype for each column. The datatype of *channel_id* is INTEGER EXTERNAL and the remaining columns are CHAR.

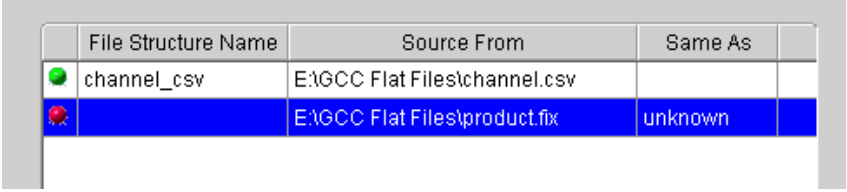


7. Click **Next**.

   The wizard returns the Summary page. Verify the format definition is correct and make any necessary corrections before you click Finish.

8. Click **Finish**.

The Flat File Sample Wizard stores the format definition and then returns to the Summary and Object page of the Import Object Wizard.



The status ball for the *channel.csv* file is now green and Builder assigned the name "channel_csv" to the format definition for this file. You can reuse this definition on other files that have the exact same format.

You must still create a format definition for the product.fix file. The general process is the same, sample the file and create the definition, but the specific steps differ considerably. For that reason, this example continues.

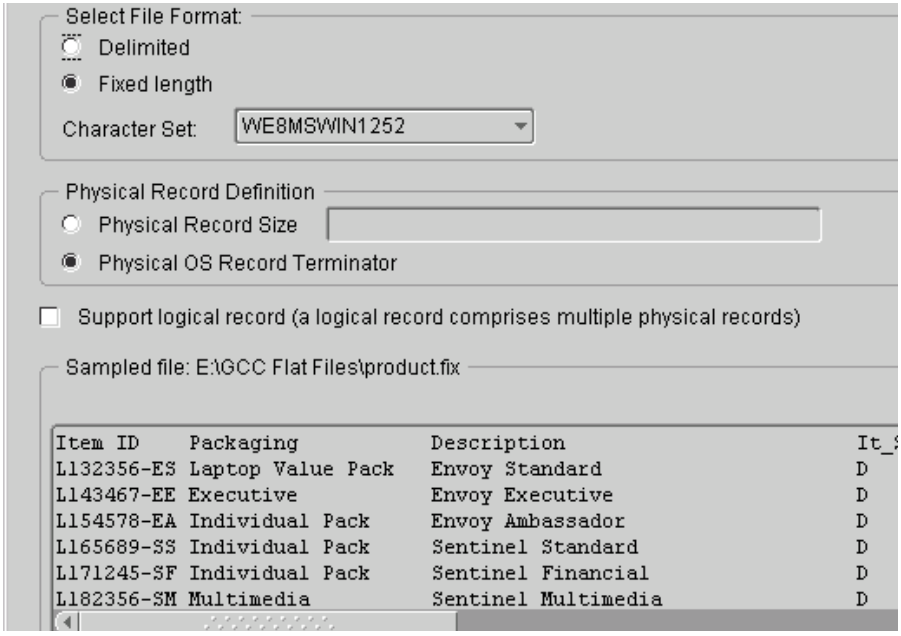
To create a format definition for the product.fix file

1. Select the file's name, and then click **Sample**.

   The wizard returns the welcome page for the Flat File Sample Wizard.

2. Click **Next**.

   The wizard returns the File Format page.



This page displays a sample of data in a template with a few initial values set for the file's global properties. The global properties include:

* Delimited or *Fixed Length*
* Character set (*WE8MSWIN1252)*
* Physical Record size (*Determined by an OS Record Terminator*)
* Multiple physical records (*NO*)

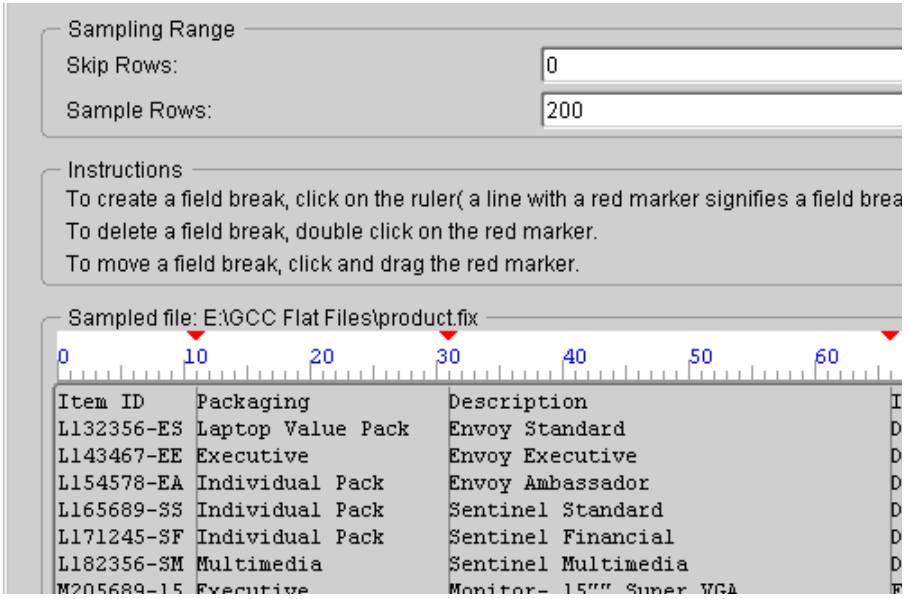
3. Look at the sample, verify that the global parameters are correct, and reset them if necessary. In the present case, they are all correct.

Note: If Builder's character set differs from the source's, the sample might not be readable. In that case, select the native character set of the source from the down-arrow list and Builder then translates the sample.

---

4. Look at the sample, verify that the global parameters are correct, and reset them if necessary. In the present case, they are all correct.

5. The wizard returns the Field Breaking page. Use this page to specify the field boundaries for a logical record.



To define a field:

Locate where the field ends in the sample and click that position on the ruler. The wizard returns a red tick mark on top of the ruler and marks the boundary with a red line.

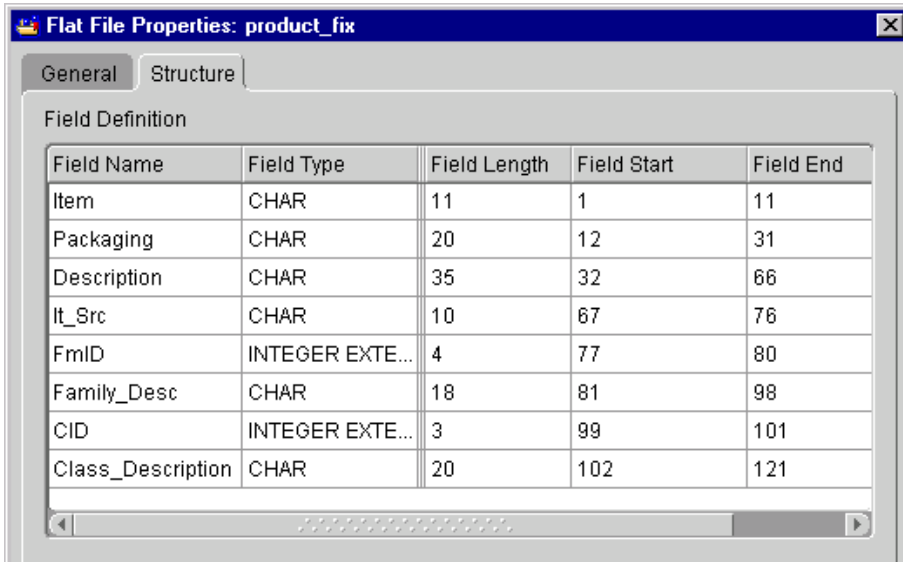
To correct a mistake:

If you make a mistake, double-click the tick mark and the wizard erases the boundary marker.

To navigate the file:

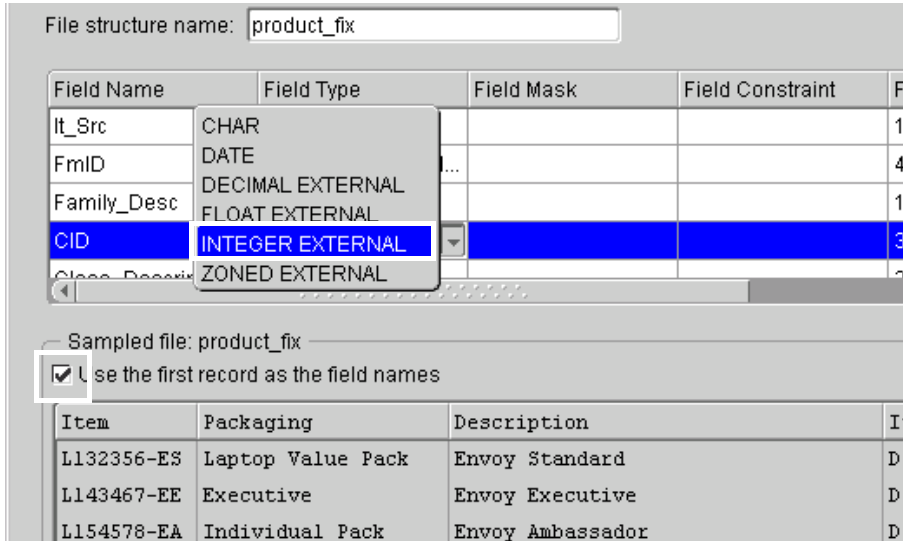
Enter a value in the Skip Rows or Sample Rows text boxes. For example, to start the sample on the tenth row, enter ten in the Skip Rows text box.

Figure 5–2 Field Descriptions for Product File



6. Click **Next**.

   The wizard returns the Fields Information page. This page describes each field of the logical record just as you marked it off in the previous step.



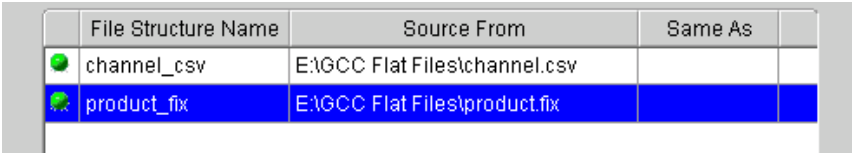
7. Use this page to complete the format definition: check the "Use first record as the field names" box and select the correct datatype for each column. The *FmID* and *CID* columns are INTEGER EXTERNAL, and the remaining columns are CHAR.

8. Click **Next**.

   The wizard returns the Summary page. Verify the format definition is correct and make any necessary corrections before you click Finish.

9. Click **Finish**.

   The Flat File Sample Wizard stores the format definition and then returns to the Summary and Object page of the Import Object Wizard.



   The status ball for the *product.fix* file is now green and Builder assigned the name "product_fix" to the format definition for this file. You can reuse this definition on other files that have the exact same format.

Builder now has format definitions for both flat files. To store these formats in the *GCCFILES* source module, you must exit the Import Metadata Wizard.

7. Click **Finish**.

   Builder stores the definitions in the *GCCFILES* source module.

Update the Format Definitions

Definition for the flat file formats now reside in the *GCCFILES* source module. You can update these format definitions by editing their property sheets.

To display the property sheet for the *channel_csv* format definition:

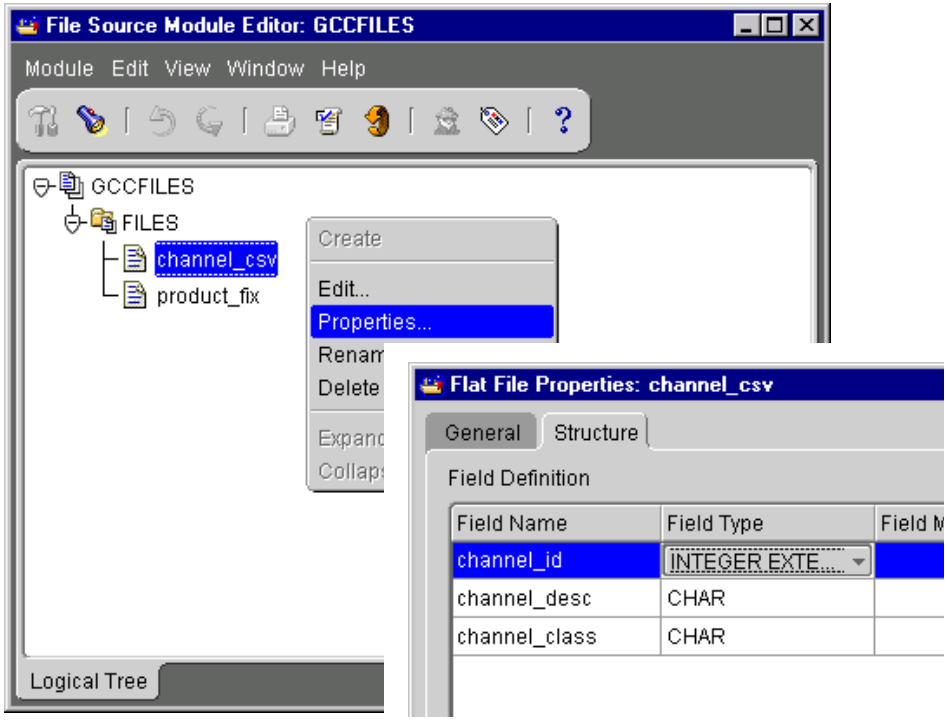
1. Double-click *GCCFILES* in the Project navigation tree.

   Builder opens the module with the Source Module Editor in a separate window.

2. Right-click ***channel_csv*** and select **Properties** from the pop-up list.

   Builder returns the property sheet for the format definition.

3. Select the **General** or **Structure** tab and then edit the active entries.

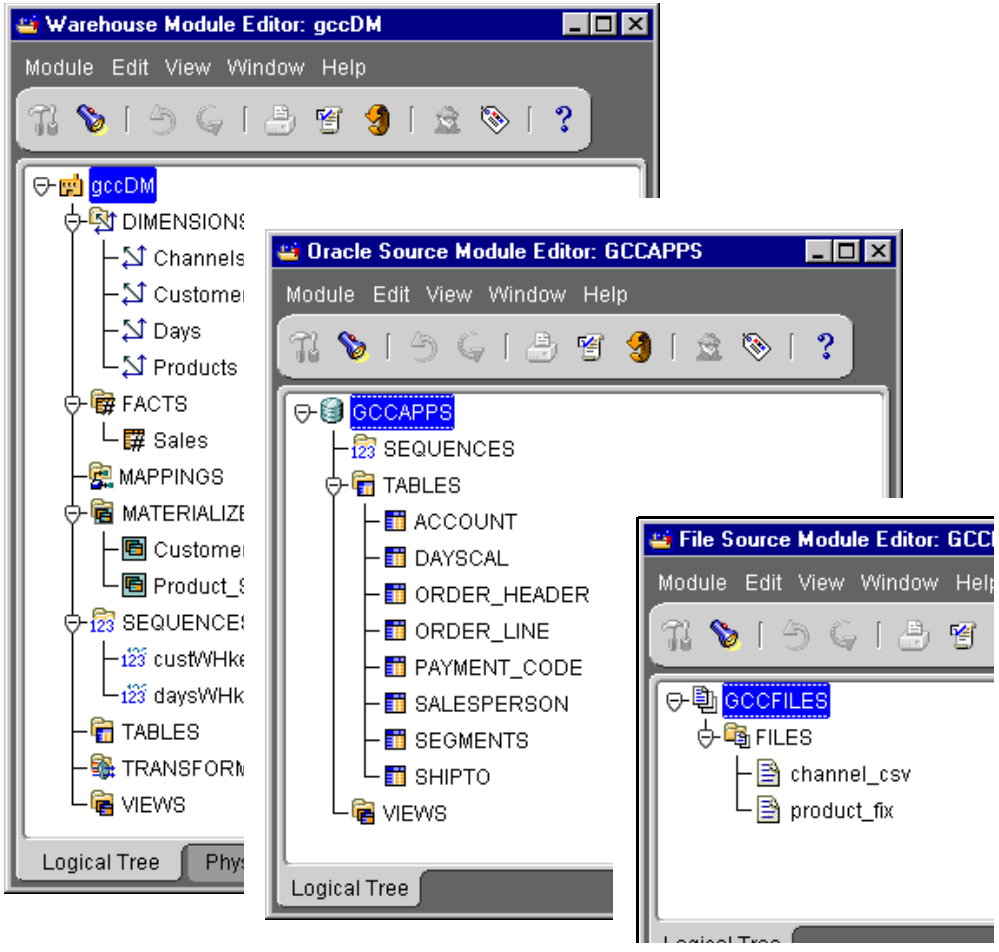


Current Status of the Project

At this stage, the Builder repository contains definitions for the data mart's target schema and all its data sources. The definitions for the target schema reside in the *gccDM* warehouse module, and those for the data sources in the *GCCAPPS* and the *GCCFILES* source modules.

To view the contents of these modules, double-click the module's name and fully expand its navigation tree. Figure 5–3 displays the fully expanded navigation trees for each of these modules.

Figure 5–3 Contents of Modules



The Next Step

This chapter showed you how to create and update definitions for the data sources that will be used to populate the data mart. The definitions were created and stored in the *GCCAPPS* and *GCCFILES* source modules using the Import Metadata wizard. The examples also showed you how to update these definitions by editing their property sheets.

Now that a logical description of the target schema and its data sources exist, you are ready to move on to the next step as described by the recipe's next step:

1. Create the definitions for the target schema.
2. Create the definitions for the source data.
3. *Create the definitions for the mapping operations.*

The next chapter, "The Mappings," shows you how to create definitions for the operations that extract and transform data from the sources described by *GCCAPPS* and *GCCFILES*, and then store the transformed data in the target schema described by *gccDM*.

You create these transformation definitions using another set of Builder wizards, and then store them in the *gccDM* warehouse module. You can also update these definitions by editing their property sheets.

6

The Mappings

This chapter shows you how to create and update definitions for operations that extract data from a source, transform that data as necessary, and then load the data into the target schema. These definitions are called "mappings," and each operation within a mapping is called a "transformation." The mappings are stored in the warehouse module that defines the target schema.

The mappings for the case study use a single staging table to manage warehouse keys. This table resides in the target schema. The examples in this chapter show you how to create a definition for this table quickly by using the Create Like Wizard.

A series of examples show you how to create mapping definitions for the dimensions, the stageONE table, and the Sales table. The mapping definition for the Sales table shows you how to create a definition for transformation operations.

This chapter covers the following topics:

- The Basic Steps
- Mappings for Dimensions
- Mapping for the Staging Table
- Mapping for the Sales Table
- Current State of the Project
- The Next Step

The Basic Steps

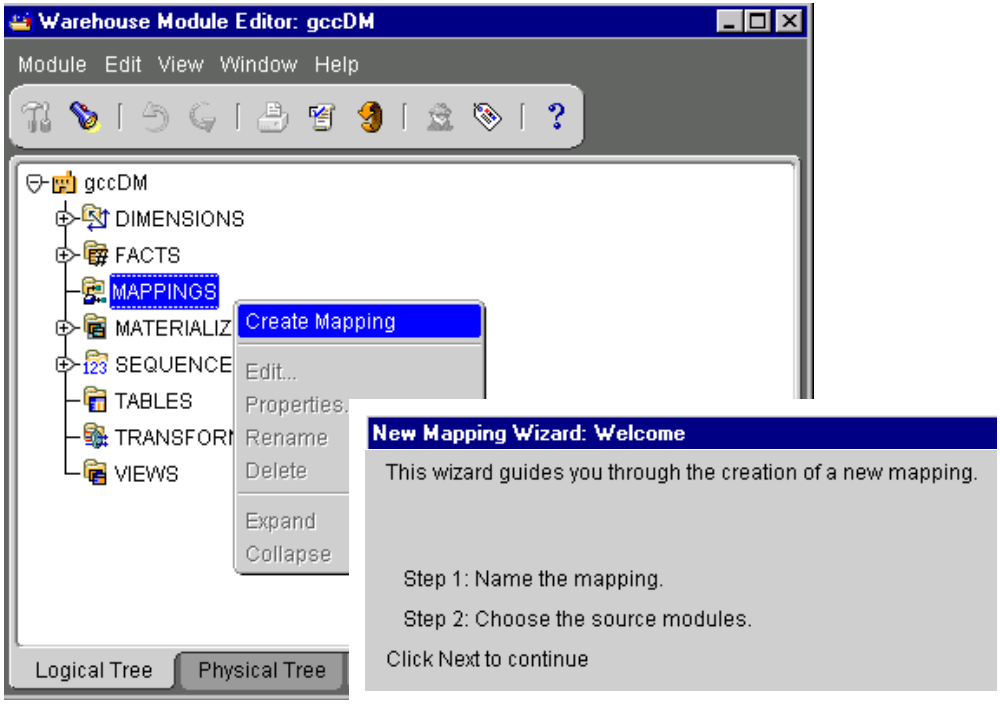
You must create a mapping definition for each table in the target schema and one for the intermediate staging table. A mapping for a target table formally describes how data is extracted from a source, how the data is transformed, and how the data is loaded into the target table. Mapping definitions reside in the warehouse module that defines the target schema.

Two Step Process

You create a mapping definition for a specific target table in two general steps.

Step One: Create a *high-level* mapping for a target table using the New Mapping Wizard. Use this wizard to name and describe the mapping. You can also specify a data source using this wizard.

To call the New Mapping Wizard, open the warehouse module that defines the target schema, expand its navigation tree, right-click **MAPPINGS**, and then select **Create Mapping** from the pop-up list.



Step Two: Create a set of *detail* mappings that specify the source for individual target columns using the Detail Mapping Editor. You can also use this editor to specify any secondary sources. The following examples show you how to open a detail-level mapping using this editor.

The Case Study Mappings

You must create a mapping definition for each target table in the *GCC Data Mart* schema including one for the intermediate staging table. Thus, the case study data mart requires six mappings.

Name	Description	Operation
map_channels	Flat file to the Channels dimension	SQL*Loader
map_days	Database table to Days dimension	PL/SQL package
map_products	Flat file to Products dimension	SQL*Loader
map_customers	Database table to Customers dimension	PL/SQL package
map_stageONE	Database table to staging table	PL/SQL package
map_ Sales	Staging table to fact table	PL/SQL package

Builder uses these definitions to generate scripts that extract, transform, and load the data. Builder generates two different kinds of scripts from these definitions depending on whether the source is a flat file or a set of database tables:

- When the source is a flat file, Builder generates a set of scripts that use the SQL*Loader to perform the extraction. These scripts include a loader control file and three scripts that implement loader job tasks.
- When the source is a set of database tables, Builder generates a set of scripts that use a PL/SQL package to perform the extraction. These scripts include a set that creates the PL/SQL package and another set to implement load job tasks.

The following examples show you how to create definitions for these mappings using the Create New Mapping Wizard and how to update the mappings using the Mapping Editor, the Detail Mapping Editor, and the property sheets for high-level and detail mappings.

Mappings for Dimensions

This section shows you how to create mappings for the Customers and Products dimensions. The Customers dimension pulls its data from database tables and the Products dimension from a flat file (products.fix). Both of these examples show you how to describe a detail mapping that generates warehouse keys using a sequence object. After you work through these examples, you can create definitions for the Days and Channels dimensions as an exercise.

The Customers Mapping

The Customer dimension pulls its data from four sources: the primary source is the ShipTo table; the secondary sources are the Accounts and Segments table plus the custWHkey sequence object.

Create the Definition

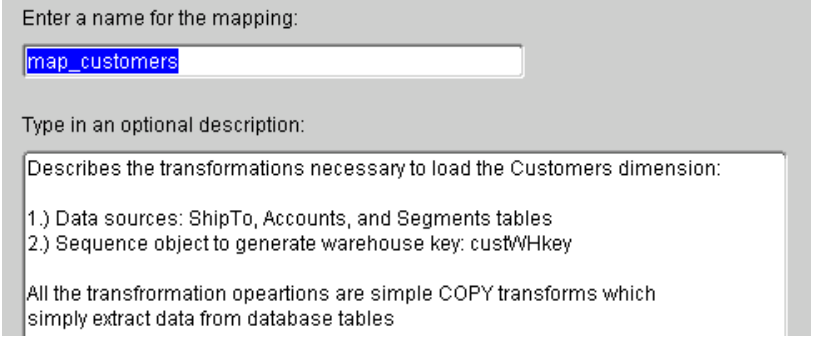
Open the *gccDM* warehouse module and fully expand its navigation tree. You can then create the high-level mapping in a few steps.

1. Right-click **MAPPING > Create Mapping**

   Builder returns the welcome page for the New Mapping wizard.

2. Click **Next**.

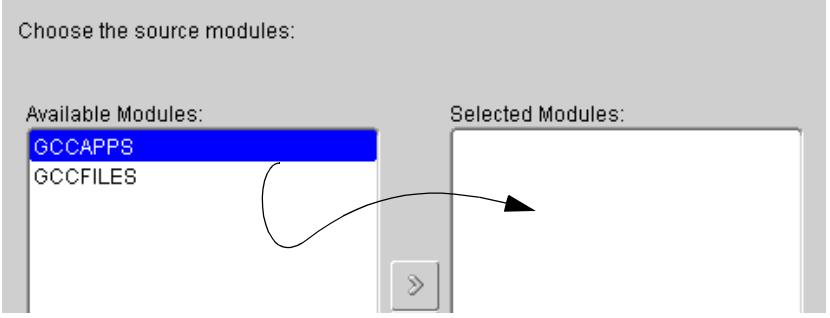
   The wizard returns its Name page.



3. Enter a name and description for the mapping (*map_customers*).
4. Click **Next**.

   The wizard returns its Source Module page. The left pane contains a list of all modules defined within the Builder Project (GCC Data Mart).

5. Move *GCCAPPS* from the Available Modules to the Selected Modules.



6. Click **Next**.

   The wizard returns its Finish page. Verify the description and make any necessary corrections before you click Finish.

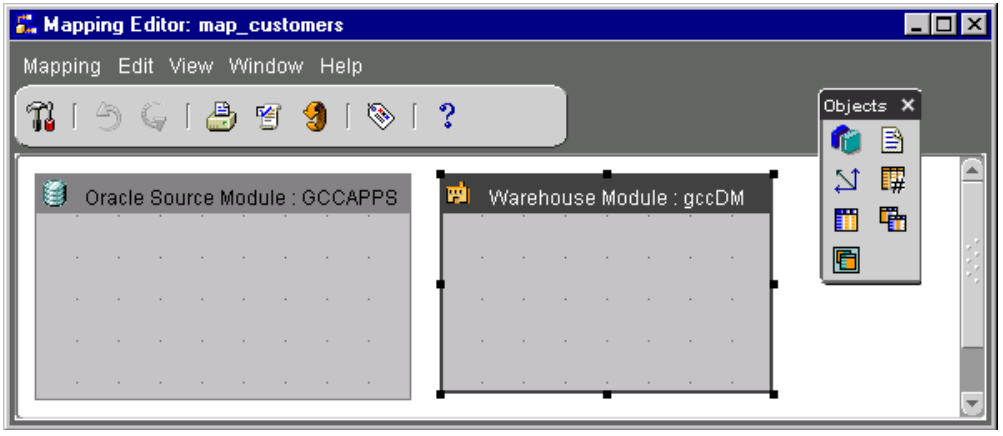
7. Click **Finish**.

   Builder stores the high-level mapping in the *gccDM* warehouse module.

Specify a Primary Source

Use the Mapping Editor to specify the primary source as the *ShipTo* table and the target table as the Customer dimension.

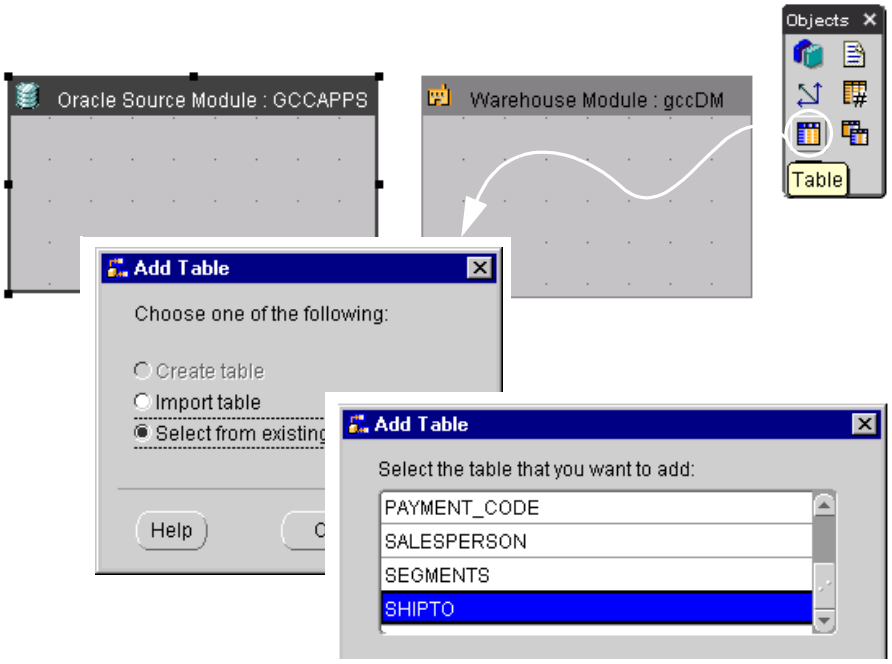
To call the Mapping Editor editor, open the *gccDM* warehouse module, expand the MAPPINGS branch of its navigation tree, and then double-click *map_customers*. Builder returns the Mapping Editor window.



To specify the primary source and target dimension

1. Drag the Table icon from the tool palette to the *GCCAPPS* rectangle.

   The Mapping Editor returns a pop-up list. You can import a new definition or select an existing one.



2. Click the button for “Select from existing tables” and then click **OK**.

   The editor returns a list of table names defined within *GCCAPPS*.

3. Select ***ShipTo*** from the list and click **OK**.

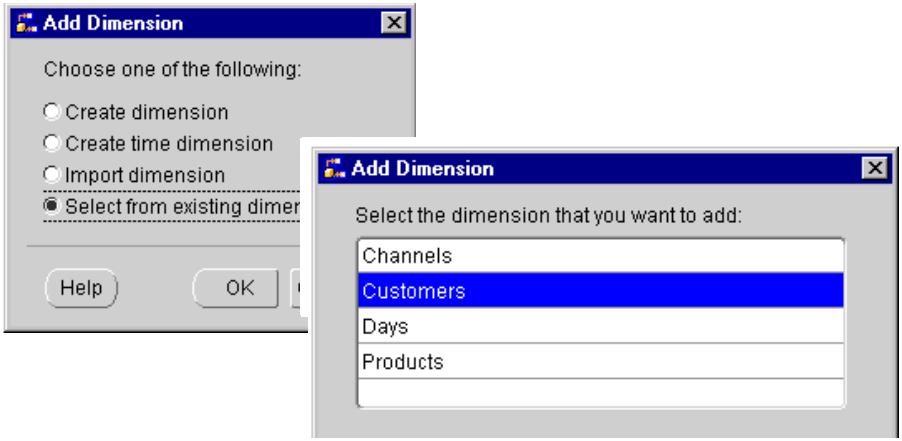
   The editor places an icon on the ***GCCAPPS*** rectangle to represent the table.

4. Drop the Dimension icon from the tool palette on to the ***gcc****DM* rectangle.

   The editor returns a list of actions.

5. Check **Select from existing dimensions** and then click **OK.**

   The editor returns a list of dimensions defined within the ***gcc****DM* module.



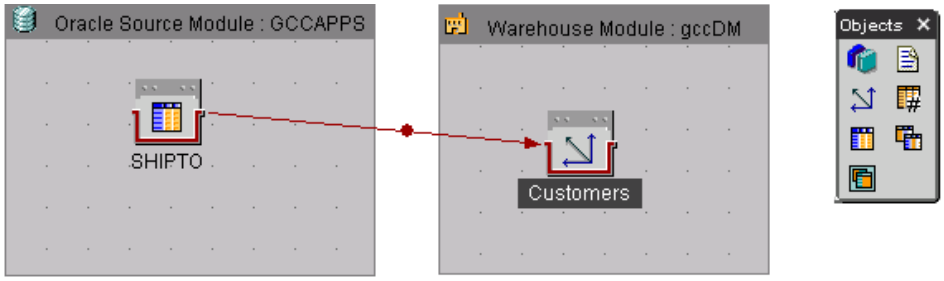
6. Select ***Customers*** from the list and click **OK.**

   The editor represents the dimension by placing an icon on the ***gcc****DM* rectangle.

7. Click on the ***SHIPTO*** icon and hold, then draw a line to the ***gcc****DM* rectangle.

   The editor represents the mapping with a line that connects the data source to the target table.

Figure 6–1 High-Level Mapping for the Customers Dimension

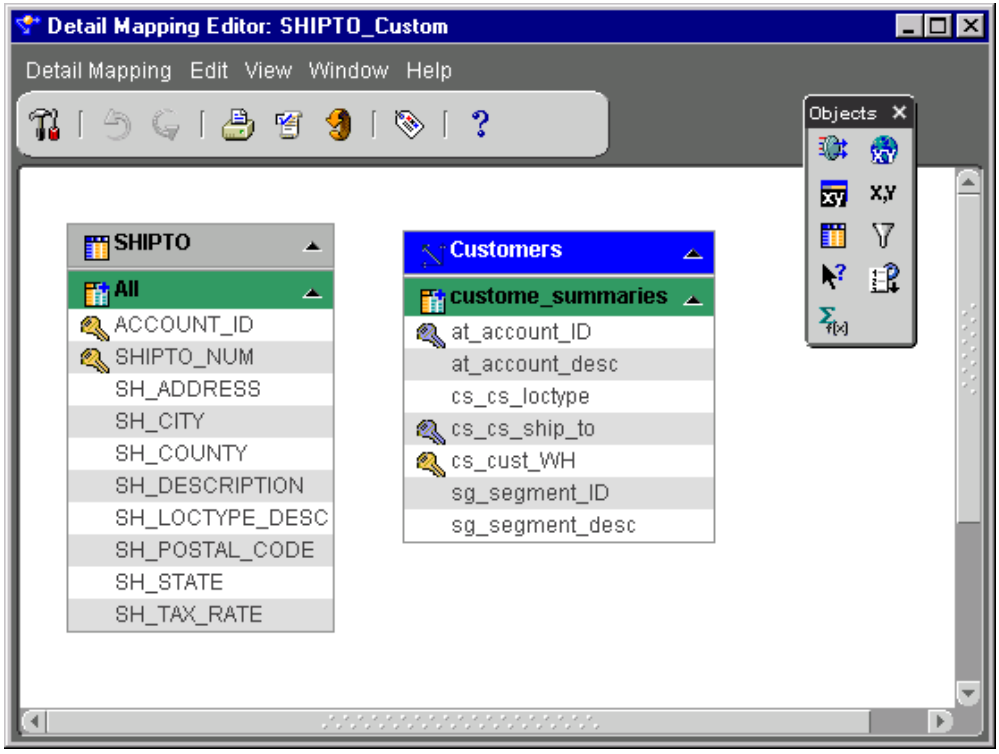


The high-level mapping now defines a primary source for the target table. To complete the mapping, you must create the detail mappings which define all the transforms that extract, transform, and load the data.

Describe the Extract Transformations

Use the Detailed Mapping Editor to create detail mappings that describe the individual transformation operations whether they are simple extraction operations (COPY transform) or more complicated, custom defined transformations.

Detailed Mapping Editor To call the Detailed Mapping Editor, open the high-level mapping with the Mapping Editor. Then, double-click the ball on the line representing the high-level mapping (see Figure 6–1). Builder returns the Detail Mapping Editor in a separate window.



The Detail Mapping Editor displays the primary source, the target dimension, and the editor's tool palette.

Secondary Sources The Customer dimension pulls data from the ShipTo table and three secondary sources:

- The ***Segment_ID*** from the ***Account*** table
- The ***Segment_Desc*** from the ***Segment*** table
- The ***cust_WH*** from the ***custWHkey*** sequence.

The ShipTo table directly references the Account table, and the Account table references the Segment table. A definition for the sequence object resides in the warehouse module.

Before you can complete the detail mapping, you must include these secondary sources in the Detail Mapping Editor's window.

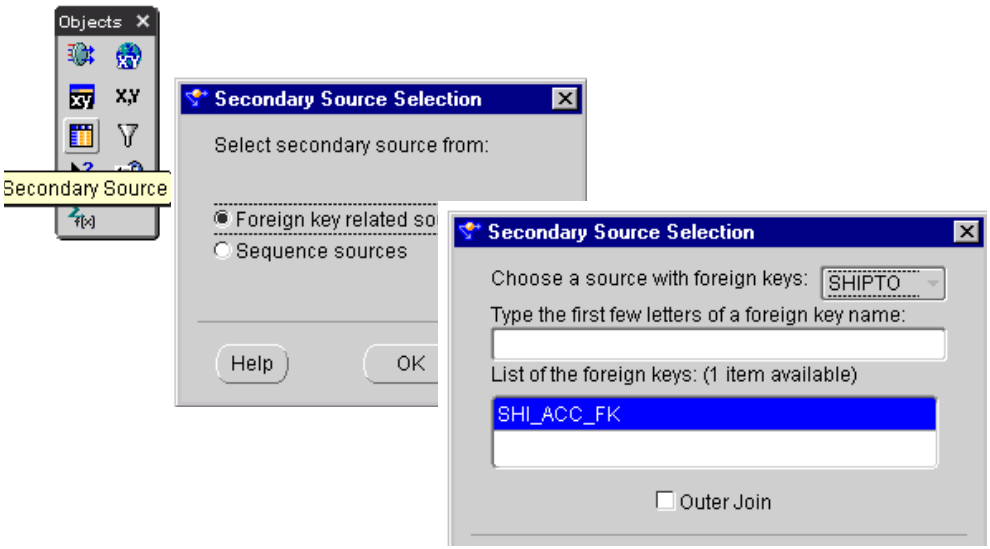
To include secondary sources

1. Drop the Secondary Source icon anywhere on the editor's white space.

   The editor returns a pop-up Secondary Source Selection list.

2. Check the button **Foreign key related objects** and click **OK**.

   The editor returns another selection panel which lists the names of objects referenced by a specified source. The default source is the primary source table displayed in the Detailed Mapping Editor's window (***ShipTo***).



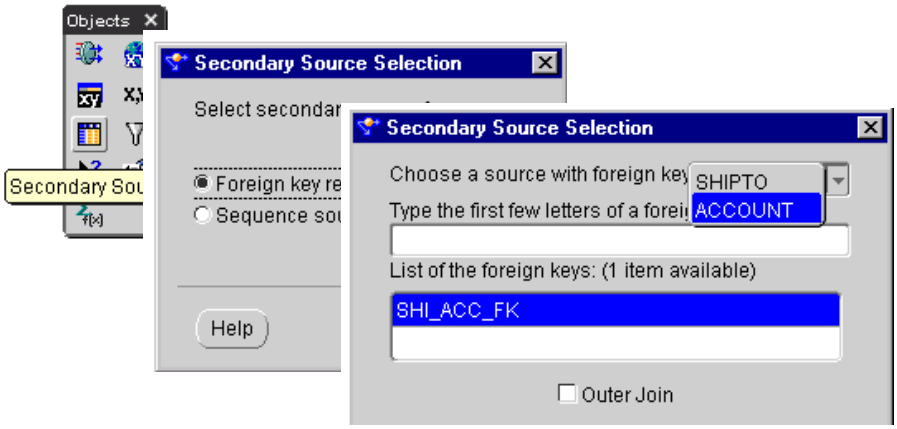
3. Click **OK**.

The editor includes a diagram of the Account table in its window and draws a line between the ShipTo and the Account tables to represent the relationship.

To include the Segment table, you can repeat these steps with one exception: in the third step choose Account as the source with the foreign key reference keys. Steps four through seven include the Segment table.

4. Drop the Secondary Source icon on the editor's white space.
5. Check the button **Foreign key related objects** and click **OK**.
6. Select ***Account*** as the table with the foreign key references.

   When you make this choice, the editor includes a foreign key reference to the ***Segment*** table in the selection list (***ACC_SEG_FK***).



7. Select ***ACC_SEG_FK*** and click **OK**.

   The editor includes a diagram of the Segment table in its window and draws a line between the Account and Segment tables to represent the relationship.

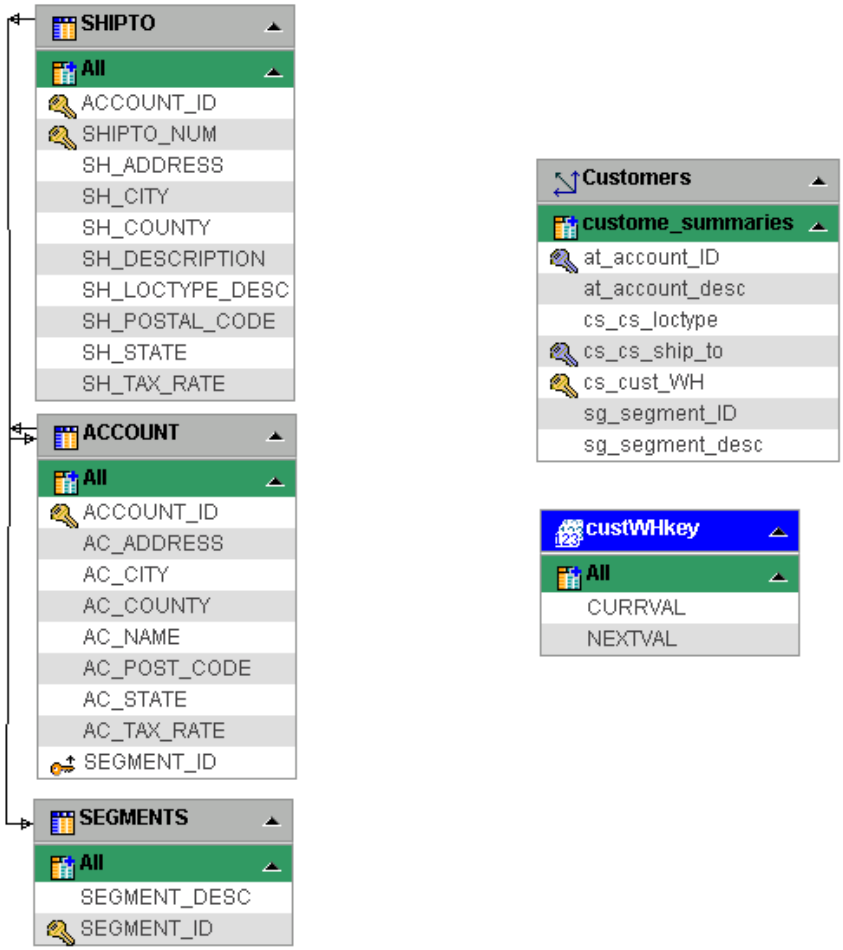
   To include the sequence object (*custWHkey*) that generates warehouse key values, you can repeat the same steps except you check the button for Sequence sources in the second step. Steps eight through ten include the sequence object in the editor's window.

8. Drop the Secondary Source icon on the editor's white space.
9. Check the button **Sequence sources** and click **OK**.
10. Select ***gcc****DM* as the warehouse module and select ***custWHkey***.

    The editor includes a diagram of the custWHkey sequence object in its window.

The Detailed Mapping Editor now displays all the sources required to load data into the Customers dimension table.

Figure 6–2 Primary and Secondary Sources for Customers Dimension

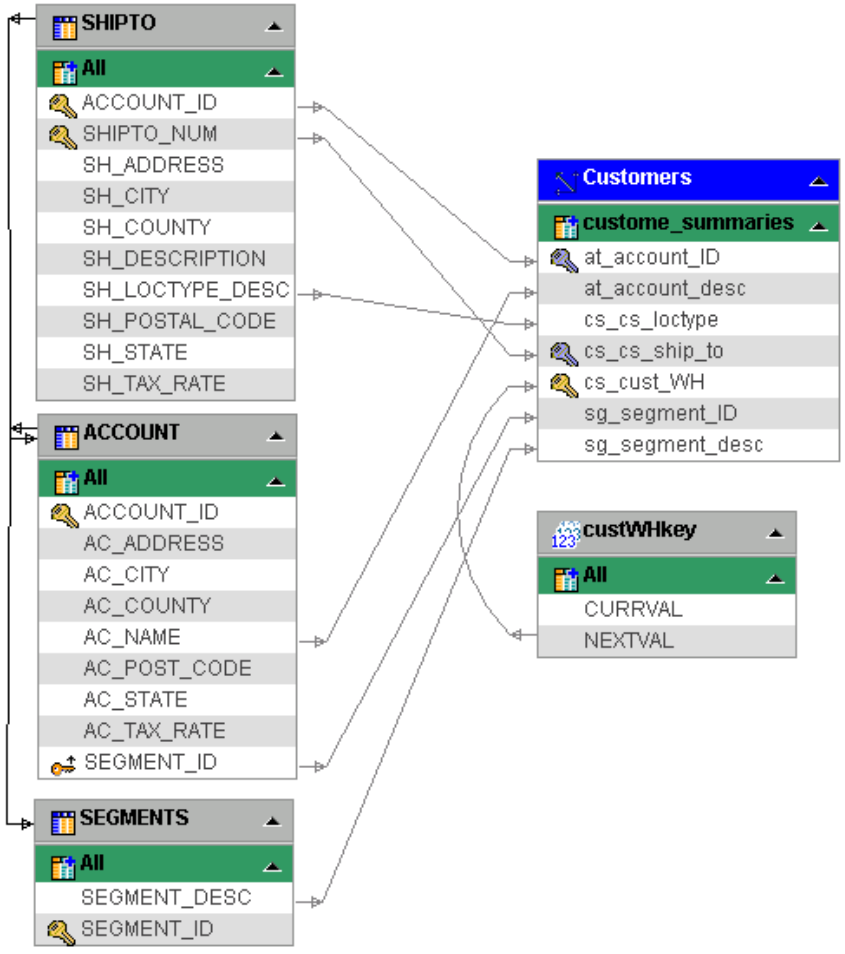


The Extract Transforms You can now create the detail mappings which describe the data source and necessary transformation operations for individual target columns. The Customers dimension is populated directly from the source tables, thus the detail mappings are all simple COPY transforms.

To specify the detail extract transforms

Drop a source column name on a target column name. The Detailed Mapping Editor represents the transform with a solid line.

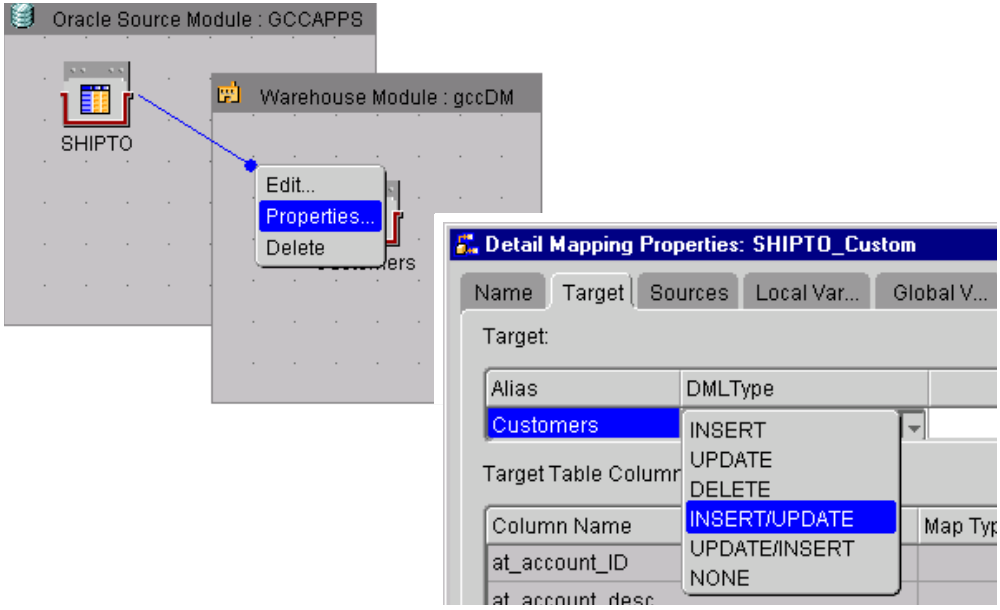
Figure 6–3 Completed Mapping for the Customers Dimension



Update the Definition

This section shows you how to change the COPY transforms for a detailed mapping. Builder implements a simple COPY transform as an INSERT INTO SELECT statement. This operation suffices for an initial load of the data mart, but may not suffice for periodic updates. The example in this section shows you how to change this operation to a different one that may be more suitable for periodic updates.

Property Sheet You can update a detailed mapping by editing its property sheet. To display the property sheet for a detailed mapping, open its high-level mapping, right-click the ball on the line representing the mapping, and then select **Properties** from the pop-up list.



Modify COPY Transforms To modify the default COPY transforms, select **Target** and then select an entry from the **DML Type** down-arrow list. The INSERT/UPDATE and UPDATE/INSERT call transformations that combine insert and update operations. The INSERT/UPDATE operation attempts to insert a source row first. If the row already exists in the target table, the operation updates the row. The other operation, often referred to as an "UPSERT" operation, reverses the sequence.

Chapter 13, "Manage the Data Mart" shows you how to use these specialized operations when you need to update dimensions. For the present, however, all the mappings assume the default case (INSERT).

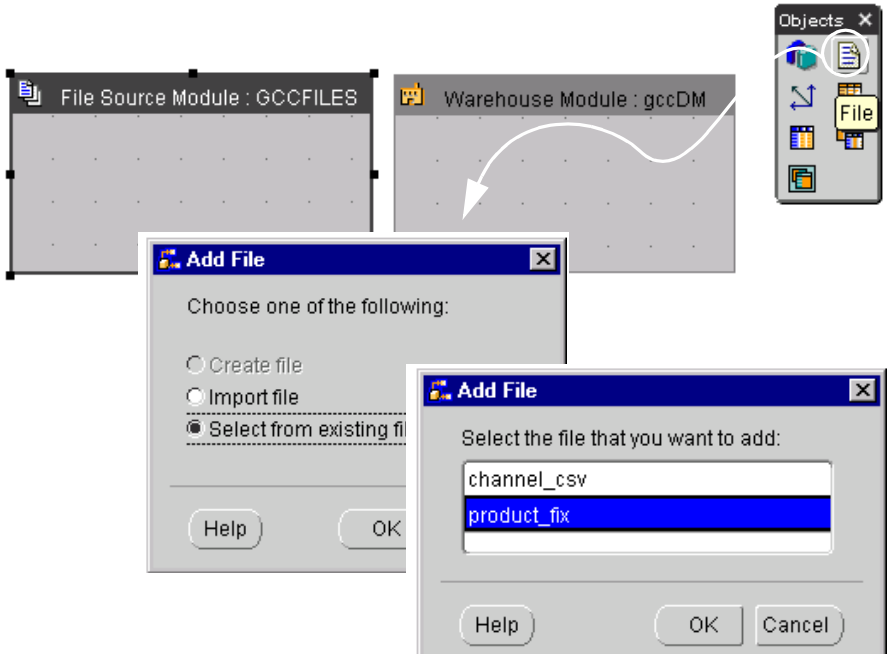
The Products Mapping

The Products dimension pulls its data from a single flat file and uses a sequence object to generate values for its warehouse key column. To create a definition for this mapping, you can with a few exceptions follow the same general steps used to create the Customers mapping.

Create the Definition

Open the *gccDM* warehouse module, fully expand its navigation tree, and then use the New Mapping Wizard to create a mapping named *map_products*. You can then create the high-level mapping in a few steps. Figure 6–4 summarizes the steps which create the mapping and designates *GCCFILES* as its source module.

Figure 6–4 Create the High-Level Mapping for Products



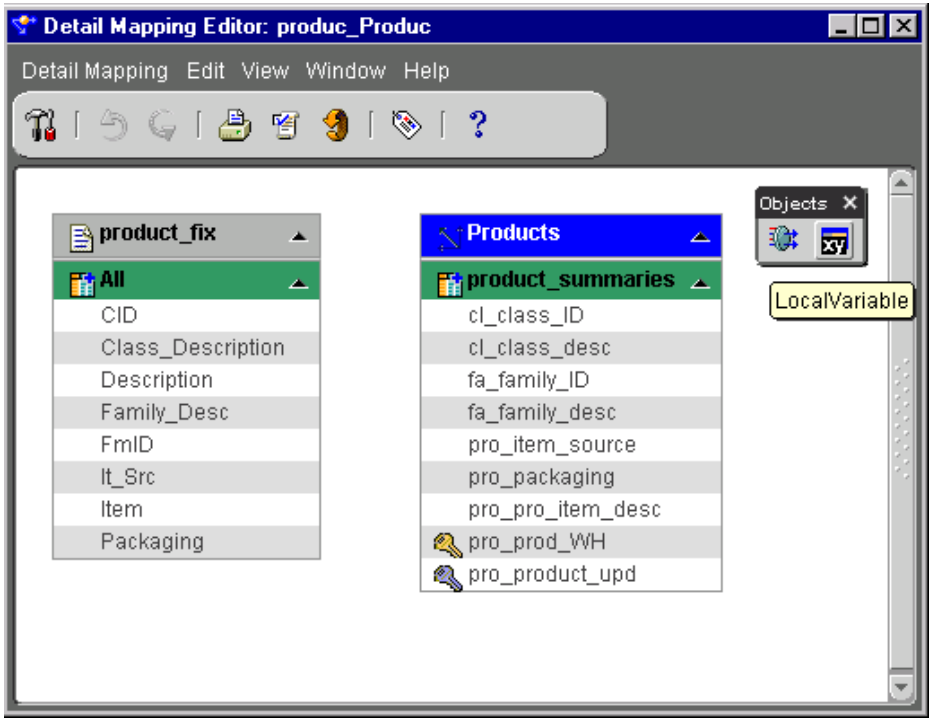
You now open this high-level mapping with the Detail Mapping Editor and describe the detail mappings.

Describe the Extract Transformations

Use the Detailed Mapping Editor to create mappings that describe the individual extract operations (COPY transforms).

Except for the warehouse key column, you define the individual transforms exactly as in the previous case. For the warehouse key column, however, you must define the sequence object as a local variable named 'SEQ'. This is because the mapping is implemented as a set of scripts that call the SQL*Loader to extract data from a file. You define this local variable using the Detail Mapping Editor.

Detailed Mapping Editor To call the Detailed Mapping Editor, open a high-level mapping with the Mapping Editor. Then, double-click the ball on the line representing the high-level mapping (see Figure 6–3). Builder returns the Detail Mapping Editor in a separate window.



The Detail Mapping Editor displays the primary source, the target dimension, and the editor's tool palette. You can now describe the individual mappings by selecting a source field and dropping it on a target column.

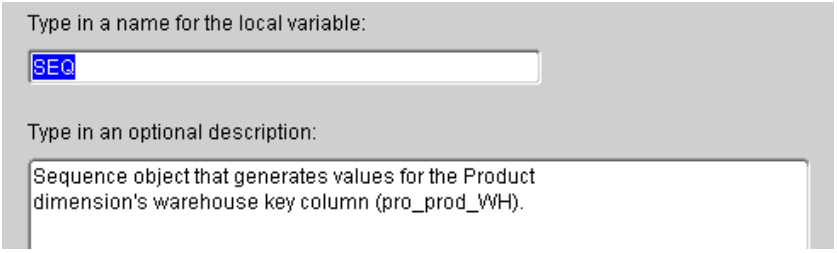
Create a local variable To complete the mapping, you must define a local variable named 'SEQ' and map it to the pro_prod_WH column. You can complete this quickly using the Detailed Mapping Editor's tool palette.

1. Select the Local Variable icon and drop it on the editor's white space.

   The editor returns the welcome page for the Local Variable Wizard.

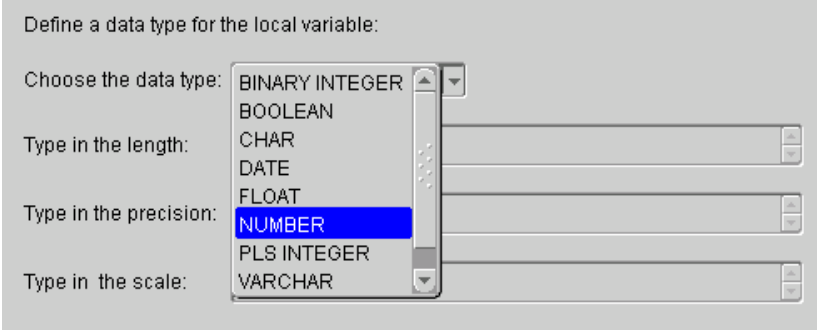
2. Click **Next**.

   The wizard returns its Name page. Use this page to name the variable ***SEQ***. The variable must be named ***SEQ*** otherwise Builder generates a control file for the SQL*Loader that is malformed.



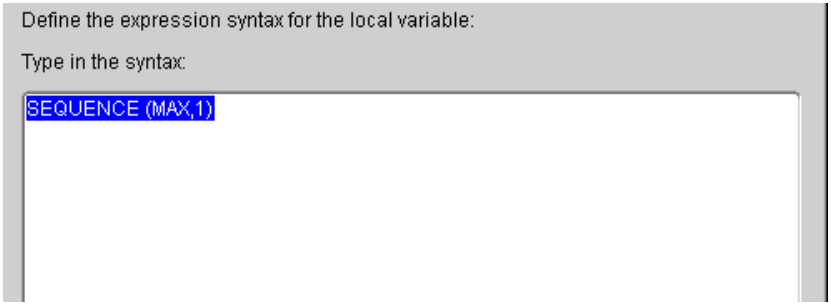
3. Enter the name ***SEQ***, a description, and then click **Next**.

   The wizard returns the Data Type page.



4. Select **Number** from the drop-down list and then click **Next**.

   The wizard returns its Syntax page. Use this page to describe the syntax for the sequence object.

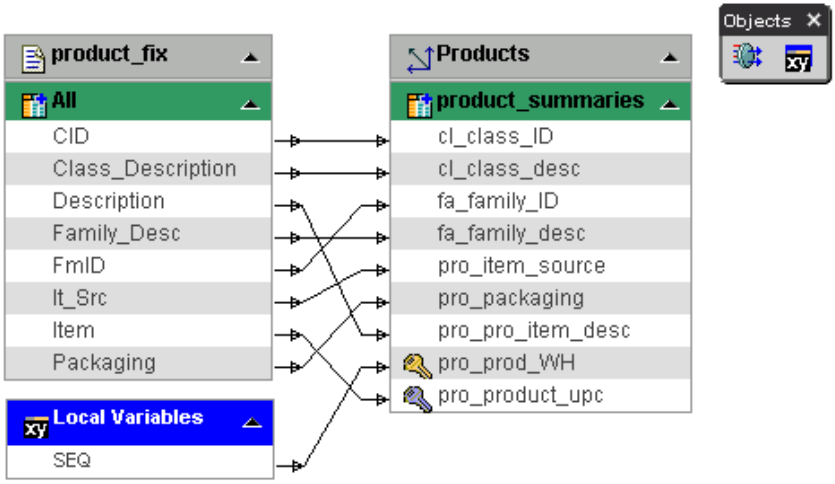


5. Enter the expression `sequence(max,1)` to define the sequence object and then click **Next**.

   The wizard returns its Finish page. Verify the description and make any necessary corrections before you click Finish.

6. Click **Finish**.

   The wizard places a rectangle on the Detail Mapping Editor's workspace to represent the local variable. Use this variable to complete the detailed mapping.





Builder does not automatically commit your work to the repository. You can commit your work immediately or defer this action until you exit the session. To commit immediately, click the Commit icon on the toolbar. Builder returns a Commit Confirmation panel: click **Yes**. You can also defer this action until you exit the session. On exit, Builder always returns an Exit Confirmation panel. Click **Yes**.

Mapping Definitions for the Other Dimensions

You can create a mapping definition for the Days and Channels dimensions using the same procedures described above. The Days dimension pulls its data from the Dayscal table (follow the instructions used for Customers) and the Channels dimension pulls its data from a file (follow the instructions used for Products).

See "The Days Dimension" on page 4-16 and "The Channels Dimension" on page 4-17 for more information.

Mapping for the Staging Table

This section shows you how to create a mapping definition for the stageONE table. The example follows the same general procedure that was used in the previous sections with a few variations. The role of this table is explained on page 2-21.

Of course, before you can create a definition for this mapping, you must create a definition for the table itself. This section shows you how to clone a definition for the table from its primary source using the Create Like Wizard.

The Basic Sequence

You can create a mapping definition for the staging table in four steps. Step three of the procedure creates a definition for the table itself. Except for the addition of the second step, the procedure is the same as that used in the previous examples:

1. Create a high-level mapping for the stageONE table.

   The principal source for the staging table is the Order_Line table because it contains all the fact measures. Thus, when you create the high-level mapping, specify *GCCAPPS* as its source module, and name the mapping *map_stageONE*.

2. Clone a definition for the staging table from the Order_Line table.

   Use the Create Like Wizard to clone the definition for the staging table. You can call this wizard from the Mapping Editor's window.

3. Describe the individual detail mapping.

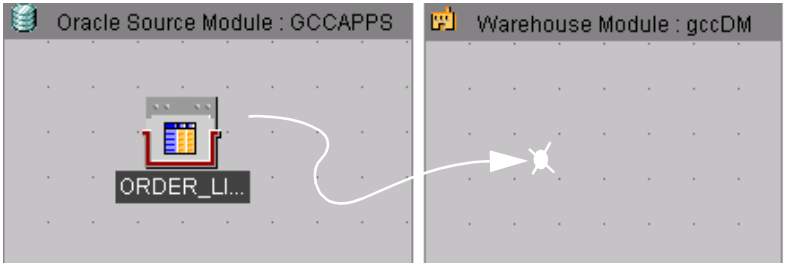
   Use the Detailed Mapping Editor to complete this final step.

The following example assumes that the high-level mapping *map_stageONE* has already been created. To create this mapping, you can follow the same steps used to create a high-level mapping for the Customers dimension (see "The Customers Mapping" on page 6-4).

Clone the Table Definition

Use the Mapping Editor to clone the definition of the staging table.

To call the Mapping Editor, open the high-level mapping for the staging table by double-clicking its name. Builder opens the Mapping Editor in a separate window. The source module includes the Order_Line table as the primary source for the staging table.



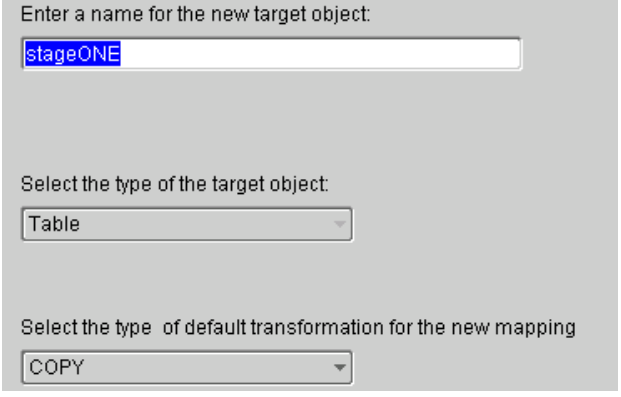
To clone the definition

1. Select ***ORDER_LINE***, hold down the shift key, drag the cursor and drop it on the rectangle that represents the ***gcc****DM* warehouse module.

   Builder returns the welcome page of the Create-Like wizard.

2. Click **Next**.

   The wizard returns its Name page.



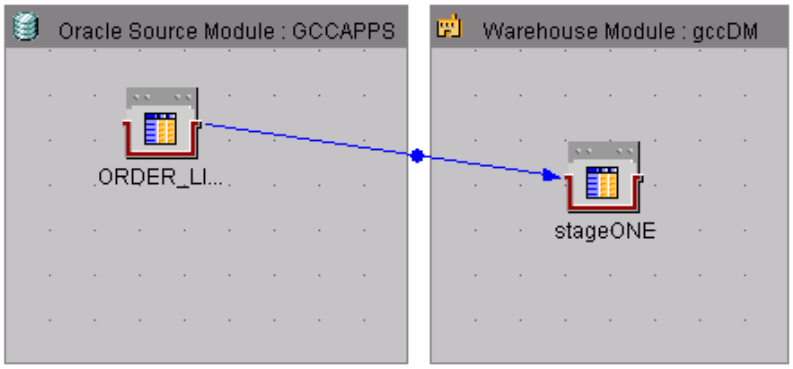
This page solicits a name for the table and a default transform type. Select **COPY** transform from the down-arrow list. When you make this selection, the wizard automatically creates detail mappings between the source and target columns.

3. Enter the name for the staging table (***stageONE***), select **COPY** from the down-arrow list, and then click **Next**.

   The wizard returns its Finish page. Verify the description and make any necessary corrections before you click Finish.

4. Click **Finish**.

   The wizard creates a partial definition for the table, stores it in the warehouse module, and represents the high-level mapping in the Mapping Editor's window with a line.



You can use the Detail Mapping Editor to complete the definition for the table and the detail mapping in two steps.

Complete the Table Definition

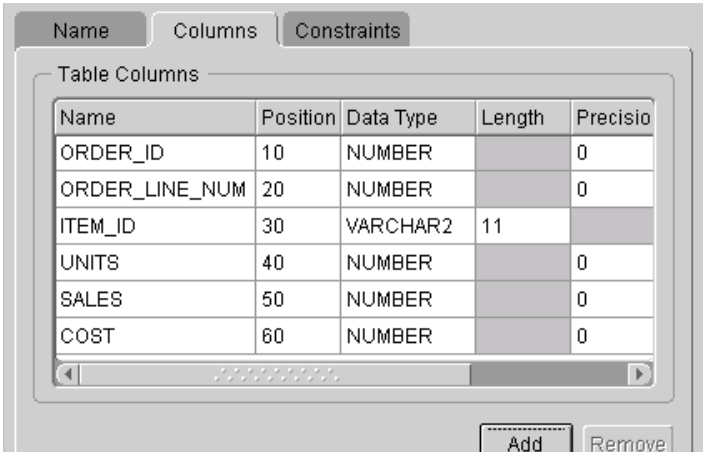
The cloned definition contains only measures and a single natural key column (***item_ID***). The definition also includes a few extraneous columns. To complete the definition you must:

- Add columns for each dimension's natural key so that the staging table can be joined to the dimensions.
- Add Foreign Key constraints to the natural key columns to support the join.
- Remove the extraneous columns (***order_line_num*** and ***order_id***) because there is no need to move this data into the staging table.

The following example shows you how to add the natural key columns and define the necessary constraints.

Add or Remove Columns You can use the table's property sheet to add, modify, or remove columns from the definition.

To display the table's property sheet, expand the navigation tree for the ***gcc****DM* warehouse module, right-click the name of the staging table, and then select **Properties** from the pop-up list.



1. Select **Columns**.

   Builder displays the Columns sheet.

2. Add the natural key columns to the stageONE table:

   - ***account_ID*** (data type NUMBER)
   - ***shipto_num*** (data type NUMBER)
   - ***channel_ID*** (data type NUMBER)
   - ***ordered_date*** (data type DATE)

   Use the **Add** button.

3. Remove the extraneous columns:

   - ***order_ID***
   - ***order_line_num***

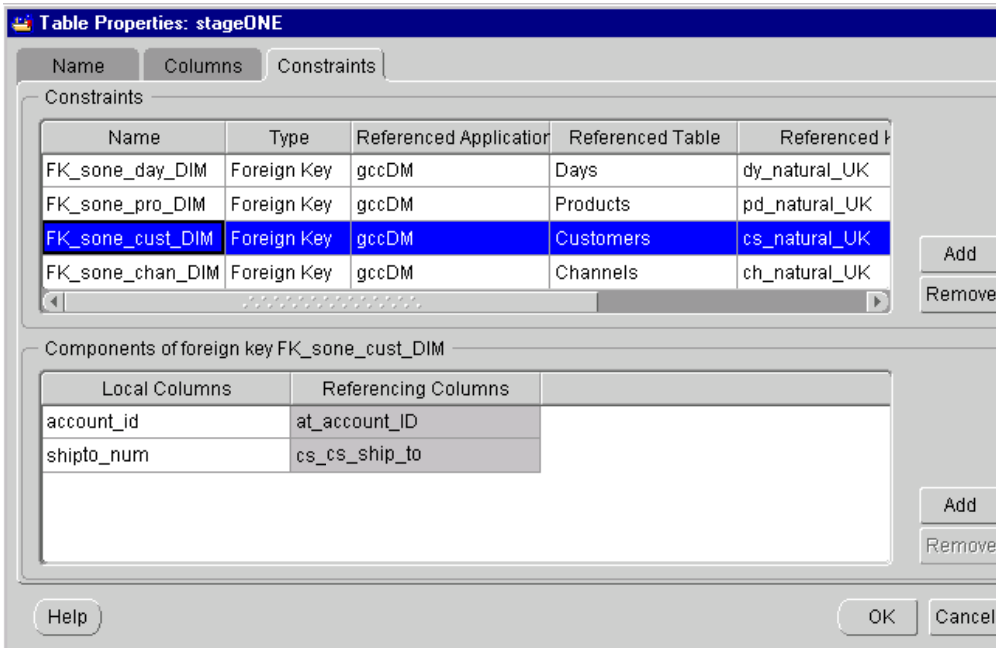
   Use the **Remove** button.

4. Click **OK**.

Add Constraints You can now add the necessary constraints to the stageONE table. A foreign reference key constraint (FK) must be created for each natural key. Use the table's property sheet to create these constraints.

1. Click **Constraints**.

   Builder displays the Constraints page.



2. Click **Add** in the upper text box.

   Builder inserts a blank line in the upper text box. Enter the following:

   - Name for the constraint — Form of examples: FK_sone_*table_name*_DIM
   - Constraint Type — Select **Foreign Key** from the down-arrow list
   - Referenced Application — Select ***gcc****DM* from the down-arrow list
   - Referenced Table — Select the referenced dimension from the list
   - Referenced Key — Select the dimension's UK constraint

3. Click **Add** in the lower text box.

   Builder inserts a blank line in the lower text box.

4. Enter the name of each local column that references the dimension.

   Table 6–1, "Foreign Key Reference Columns" summarizes the information required to create the foreign key reference constraints.

5. Click **OK**.

   Builder updates the definition for the stageONE table.

Repeat this procedure for each dimension referenced by the stageONE table.

Table 6–1 Foreign Key Reference Columns

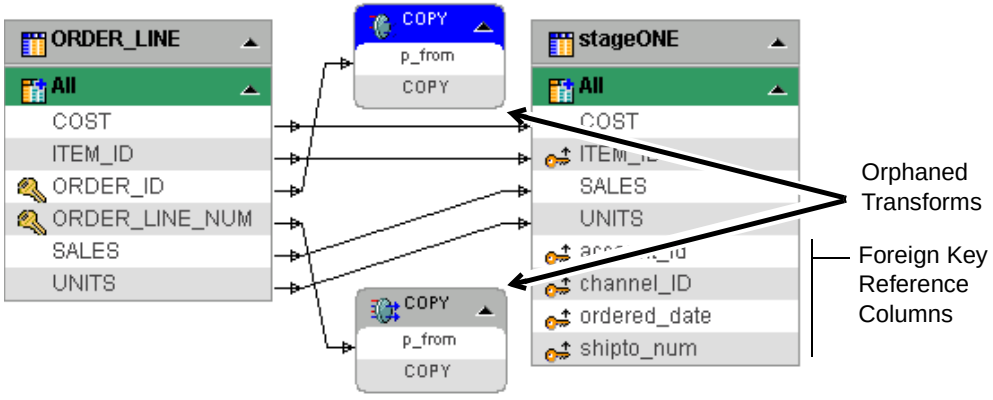
Dimension	Local Columns	Referencing Columns
Days	ordered_date	dl_day_desc
Products	item_ID	pro_product_upc
Customers	account_ID shipto_num	at_account_ID cs_shipto_num
Channels	channel_ID	bs_channel_selector

Complete the Mapping Definition

Use the Detail Mapping Editor to complete the partial mapping definition.

1. Open the detail mapping for the staging table.

   The diagram for the staging table now includes the new columns and constraints defined in the previous section. The editor also displays orphan copy transforms for the columns that were removed from the staging table.



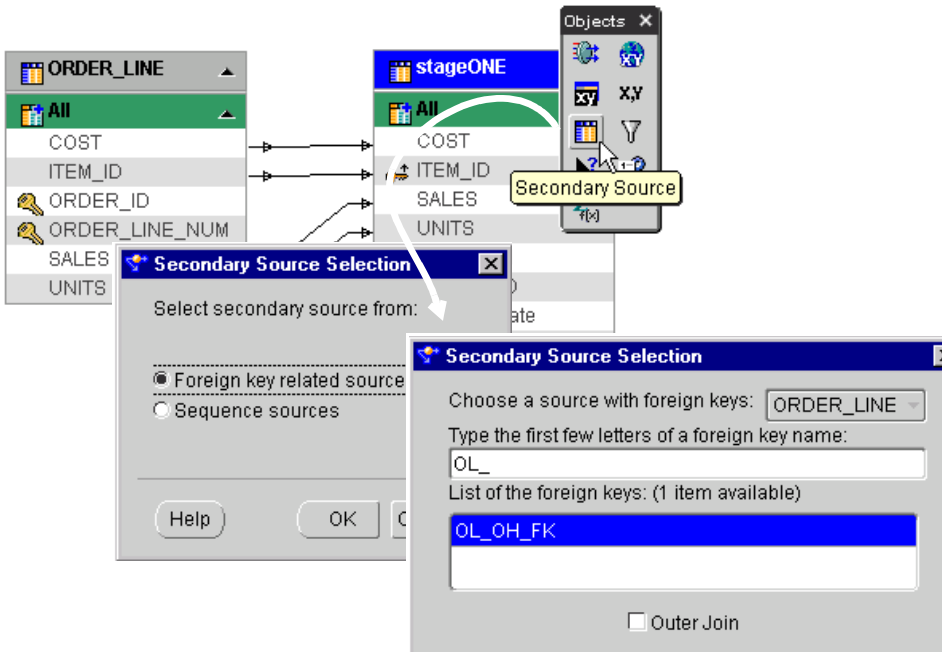
To complete the mapping, you must remove the orphaned copy transforms and create detail mappings for the foreign key reference columns.

2. To remove a transformation, right-click its name and then select **Remove**.

   To create the detail mappings for the foreign key reference columns, you need to include the ***ORDER_HEADER*** table as a secondary source. This table contains the necessary columns and is also referenced by the ***ORDER_LINE*** table.

3. To add the ***ORDER_HEADER*** table as a secondary source, drag the Secondary Source icon from the tool palette and drop it on the editor's workspace.

   The editor returns the Secondary Source Selection list.

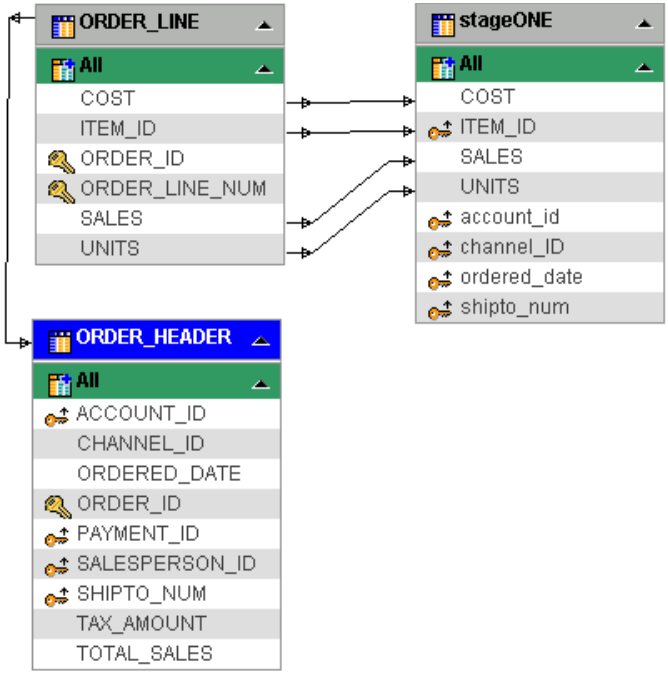


4. Select the Foreign key related sources button.

   The editor returns a list of all foreign key reference constraints defined on the ***ORDER_LINE*** table.

5. Select the constraint that references the ***ORDER_HEADER*** table (***OL_OH_FK***) and then click **OK**.

   The editor now includes a diagram of the ***ORDER_HEADER*** table in its window.



To complete the mapping, create detail mappings for each of the unmapped target column: left-click the name of a source and drag it to a target name.

Add a Filter with Parameters This section shows you how to add a date filter to the mapping using two parameters (*start_date*, *end_date*). After you deploy a physical instance of the data mart, you can then change the parameter values to a range of dates that include only dates defined by the range.

These parameters are used during the initial load to restrict the input data to the years 1995 through 1998. Subsequent jobs that refresh the job on a monthly basis use the parameters to restrict the data extracted to a single month.

When you define the filter, you create the two parameters first and then reference the filters from the filter. This takes only a few steps.

To create the parameters:

1. Select **Detail Mapping > Add > Parameter** from the toolbar.

   The Detail Mapping Editor returns the welcome of the New Parameter wizard.

2. Click **Next**.

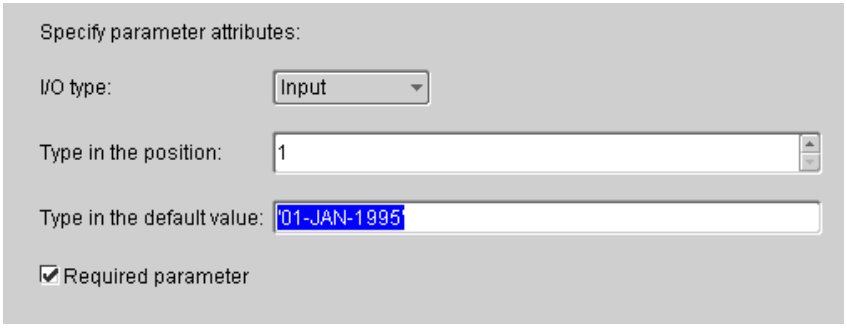
The editor returns the Name page.

3. Enter a name (*start_date*) for the parameter and a brief description of its usage and datatype, and then click **Next**.

   The editor returns the Datatype page.

4. Select the **DATE** datatype from the down-arrow list and then click **Next**.

   The editor returns the Attribute page which solicits a type, a position, a default value, and a declaration of whether the parameter is required.



5. Enter the type as *Input* (input to the mapping), position number (*1*), a default value of '*01-JAN-1995*', check the **Required parameter** box, and then click **Next**.

   The editor returns the Finish page which summarizes your description. Verify the description and make any necessary corrections before you click Finish.

6. Click **Finish**.

   The editor includes the parameter in the detailed mapping.

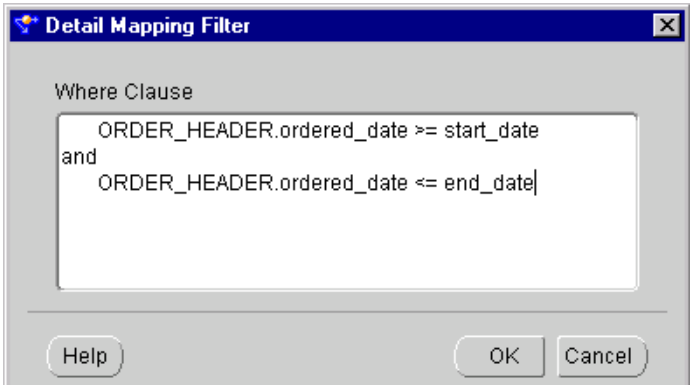
7. Create another definition for a required parameter (*end_date*) that determines the upper limit of the date range. Enter the type as *Input* (input to the mapping), position number (*2*), and a default value of '*31-DEC-1998*'.

   The detailed mapping now includes two parameters. You can now continue and define a filter that uses the two parameters.

8. Select **Detail Mapping > Add > Filter** from the toolbar.

   The detailed mapping editor draws a rectangle around the primary data source (*order_line*) and returns a panel with the title **Detailed Mapping Filter**. Enter a search conditions for the query that extracts data from the source tables in the panel. Do not include the keyword WHERE nor terminate the statement with a semicolon (;).

The example below references the parameters within the search condition.



When Builder generates the PL/SQLQ routine to extract data for the staging table, it will include the above search condition within the routine's WHERE clause. Moreover, you can modify the date range before your run the job that extracts the data.

9. Click **OK**.

Note: When you define a search condition for a filter:

- Fully qualify each column name with the name of its owner. The owner name must correspond with how it is represented within the detailed mapping.
- Do not include the WHERE key word within the condition.



The detailed mapping for the stageONE table is now complete. Close the Mapping Editors and commit your work now.

Two Steps Out of Three

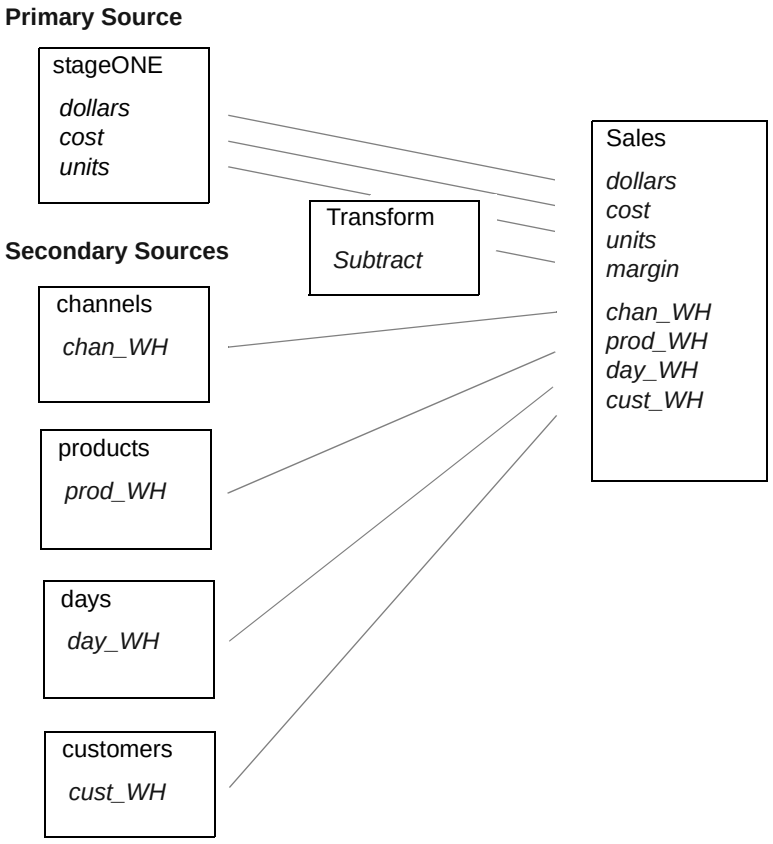
Chapter 2 described a three step load strategy for the case study data mart. Step one loads the dimensions and step two loads the staging table. The dimensions now contain values for the warehouse keys and the staging table contains the measures. Step three extracts the warehouse key values from the dimensions, the measures from the staging table, and loads this data into the Sales table.

Mapping for the Sales Table

This section shows you how to create a mapping definition for the *Sales* table using the general procedures illustrated in the previous sections with a few additions.

The *Sales* table must be loaded with measures and foreign key reference values. The mapping for this table extracts the base measures from the ***stageONE*** table and uses a custom transform (*Subtract*) to calculate values for the Margin column. The mapping extracts the foreign key reference values (warehouse keys) from the dimensions. The mapping includes the dimensions as secondary sources by joining the ***stageONE*** table to the dimensions over the natural key columns.

Figure 6–5 Mapping for Sales Fact Table



Note that the *Sales* table is like the staging table except that it contains warehouse key columns in positions where the staging table contains natural key columns.

Before you can complete this mapping, you must make a minor detour and create a definition for the ***Subtract*** transformation. The definition for this transformation is stored in a transformation category (Arithmetic) which resides in the *gccDM* warehouse module. At the same time, you also create a post-processing transformation called *Truncate_Table*. The mapping calls this transformation using a post-map trigger.

Five Basic Steps

You can create a mapping definition for the *Sales* table in five steps.

1. Create definitions for any custom transformations referenced by the mapping (***Subtract*** and ***Truncate_Table***).
2. Create a high-level mapping for the *Sales* table.

   This high-level mapping does not specify a source module because its data is extracted from a staging table whose definition is stored in the *gccDM* warehouse module, not a source module. Thus, when you create the *map_Sales* mapping you do not specify a source module.
3. Specify the primary source for the *Sales* table.

   The measures for the *Sales* table all occur in the ***stageONE*** table.
4. Define the copy transforms.

   Define a set of copy transforms to extract the measures from the ***stageONE*** table and warehouse key values from the dimensions using the Detail Mapping Editor. After you open the mapping with this editor, you include each dimension within the mapping as a secondary source. Afterwards, you can extract the warehouse keys and copy them to the *Sales* table.
5. Complete the mapping definition.

   Include the custom transforms *Subtract* and *Truncate_Table* in the mapping.

The following examples walk you through these steps.

Create Definitions for the Custom Transforms

This section shows you how to create the custom transformations *Subtract* and *Truncate_Table* both of which are referenced by *map_Sales*. Custom transformations reside in Transformation Categories (libraries), and you must create these categories before you create the transformation.

The following examples create two transformation libraries within the *gccDM* warehouse module: *Arithmetic* and *PostProcess*. Multiple transformations can reside in each library. You create the category with the New Transform Category Wizard, and you create transformations within a library using the New Transform Wizard.

Create the Transformation Categories

You can quickly create a Transformation Category using the New Transform Category Wizard. To call this wizard, open the *gccDM* warehouse module, fully expand its navigation tree:

1. Right-click **Transformation Categories > Create Transformation Category**

   Builder returns the welcome page for the New Transform Category Wizard.

2. Click **Next**.

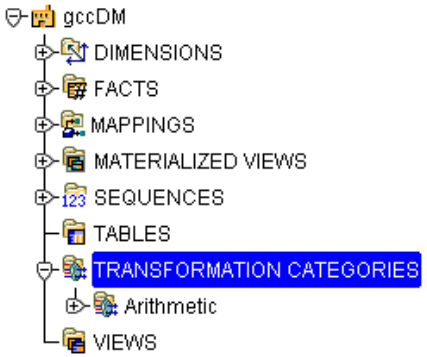
   The wizard returns its Name page.

3. Enter a name ***Arithmetic*** and description for the category.

   A category name can have from 2 to 40 alphanumeric characters but no spaces.

4. Click **Finish**.

   The wizard inserts the category's name the module's navigation tree.



You can also create the *PostProcess* category at this time using the same steps.

The Subtract Transform

The *Subtract* transform calculates the difference between two columns and is stored as an *Arithmetic* transformation. The example described on page 6-37 shows you how to include this transform in a detailed mapping.

To create the Subtract transform

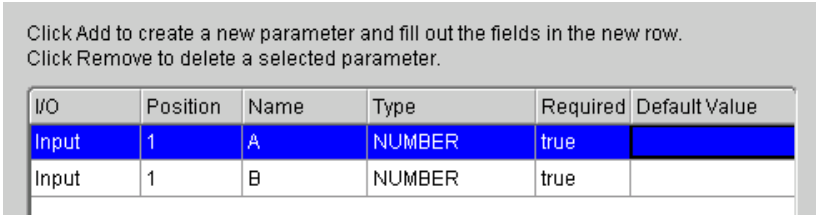
1. Open the ***gcc****DM* warehouse module and expand the ***Arithmetic*** branch of the navigation tree.
2. Right-click **Transformations > Create Transformation**.

   Builder returns the welcome page for the New Transform wizard.
3. Click **Next**.

   The wizard returns its name page which solicits the following information:

   1. Name of the transform (***Subtract***)
   2. Kind of transform: procedural (doesn't return a result) or not (***Not***)
   3. Data type of the transform's result (***Number***)
   4. Description of the transform
4. Enter a name and description, select a return data type, and click **Next**.

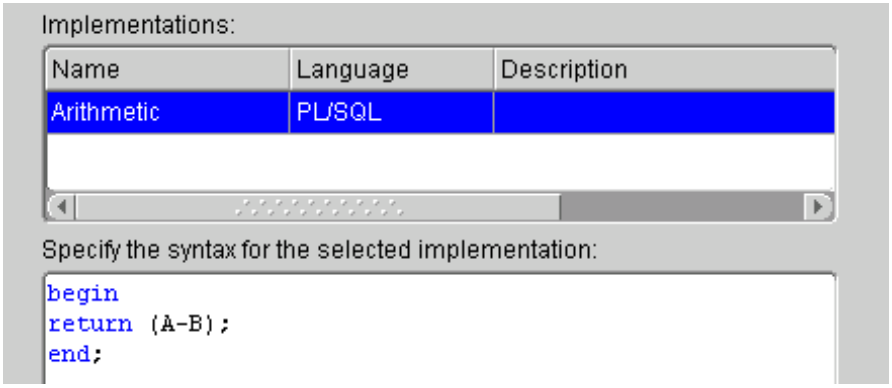
   The wizard returns its Parameters page.



   Use this page to describe each of the transform's parameters:

   - Parameter Type (**Input**, Output, or Input/Output)
   - Position within the function call (***1*** and ***2***)
   - Name (***A*** and ***B***)
   - Data Type (***Number***)
   - Required (**True**)
5. Enter values for each parameter and click **Next**.

The wizard returns its Implementations page.



Use the upper text box to name the transform and the lower box to specify its syntax. The default implementation language is PL/SQL.

6. Click **Next**.

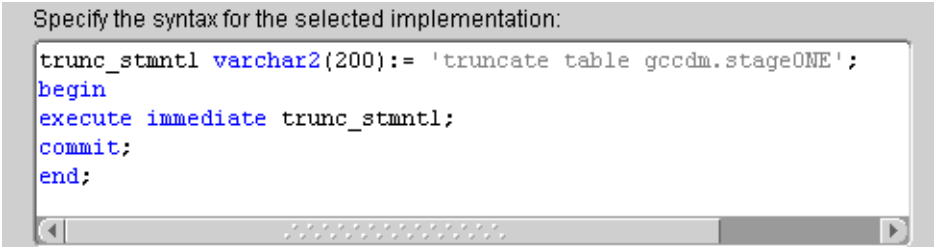
   The wizard returns its Finish page. Verify the description and make any necessary corrections before you click Finish.

7. Click **Finish**.

   The wizard creates a definition for the ***Subtract*** transform and stores it in the ***Arithmetic*** category of the ***gcc****DM*.

The Truncate_Table Transform

The mapping for the Sales table uses a post-map trigger to remove rows from the *stageONE* table after a successful load operation. This transformation is stored in the *PostProcess* library and it is a procedure (does not return a result). To create this transformation follow the previous steps except in Step 1 you must check the Procedure box to declare that the transformation is a procedure. Then you can enter the following as the implementation syntax (fully qualify the column name):



Create the Mapping

Now that definitions for the custom transforms referenced by the mapping exist, you can create the mapping definition for the *Sales* table. This process begins with the creation of a high-level mapping for the *Sales* table (step two of the five basic steps) and finishes by including the custom transforms within the mapping (step five of the five basic steps). This section walks you through the last four steps.

Create Definition for the High-Level Mapping

Open the *gccDM* warehouse module and fully expand its navigation tree. You can then create the high-level mapping in a few steps.

1. Select **MAPPING > Create Mapping**

   Builder returns the welcome page for the New Mapping wizard.

2. Click **Next**.

   The wizard returns its Name page.

3. Enter a name and description for the mapping (*map_Sales*).

4. Click **Next**.

   The wizard returns its Source Module page. The mapping for the Sales table pulls its data from a staging table (*stageONE*) which resides within the target schema's warehouse module. Thus, you do not need to name a source module. *Skip this page.*

5. Click **Next**.

   The wizard returns its Finish page. Verify the description and make any necessary corrections before you click Finish.

6. Click **Finish**.

   Builder stores the high-level mapping in the *gccDM* warehouse module.

Now that the high-level mapping has been defined, you can specify the primary source and target tables. This process is similar to the previous examples.

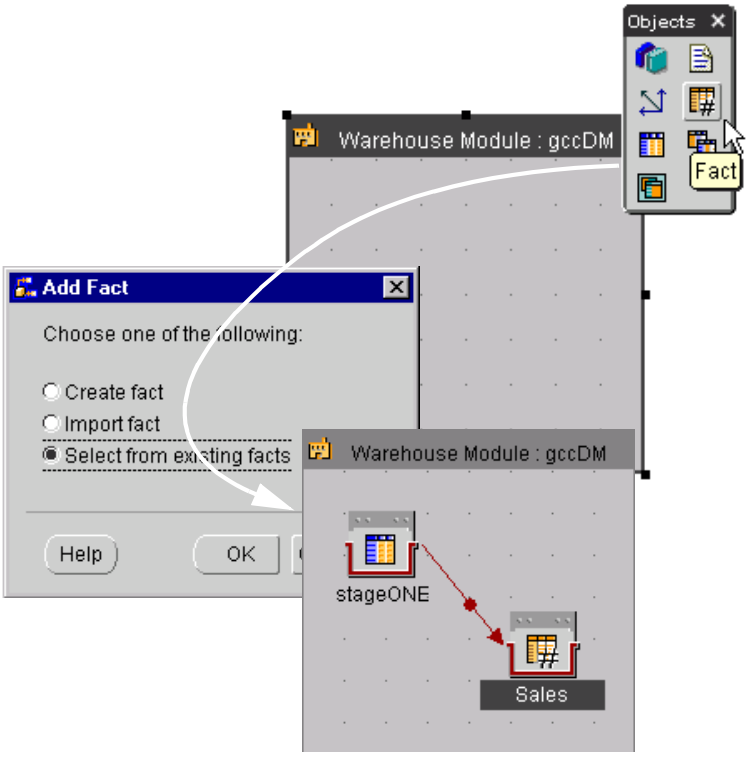
Specify the Primary Source and the Target

The Sales table is loaded with data extracted from the stageONE and dimension tables, and the definitions for these tables reside in the *gccDM* warehouse module. The mapping definition does not depend on a source module.

1. Open the ***map_Sales*** mapping with the Mapping Editor.

   Builder returns the Mapping Editor window which contains a single rectangle.

2. Use the **Table** icon from the tool palette to include ***stageONE*** in the ***gcc****DM* warehouse module, and then use the **Fact** icon to include ***Sales*** in the same warehouse module.



3. Click on the ***stageONE*** icon and hold, then draw a line to the ***Sales*** icon to create the high-level mapping.

   The editor draws a line to represent the high-level mapping between the fact table and its primary source. You can now define the detailed mapping.

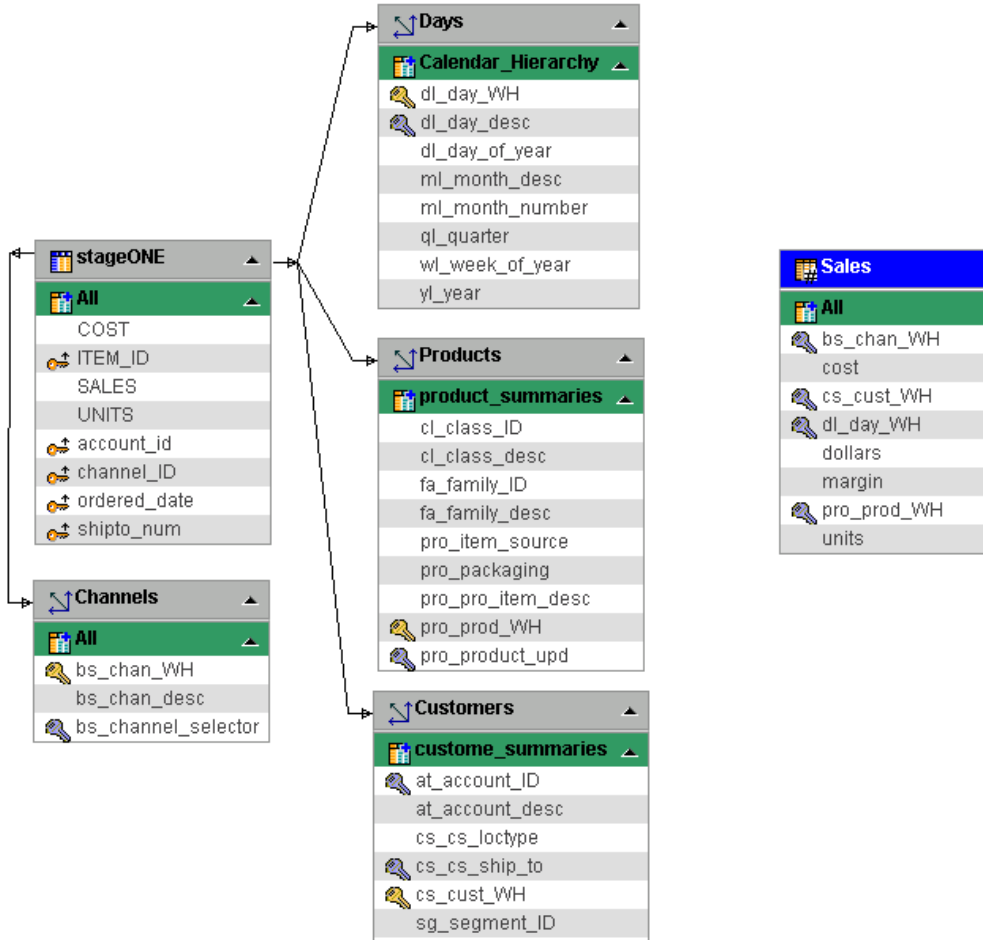
Define the Copy Transforms

To complete the mapping, you must define a set of transforms that copy data from the stageONE table and the dimensions to the Sales table. You must also define a transform that calculates values for the Margin column.

1. Double-click the ball on the line that represents the high-level mapping.

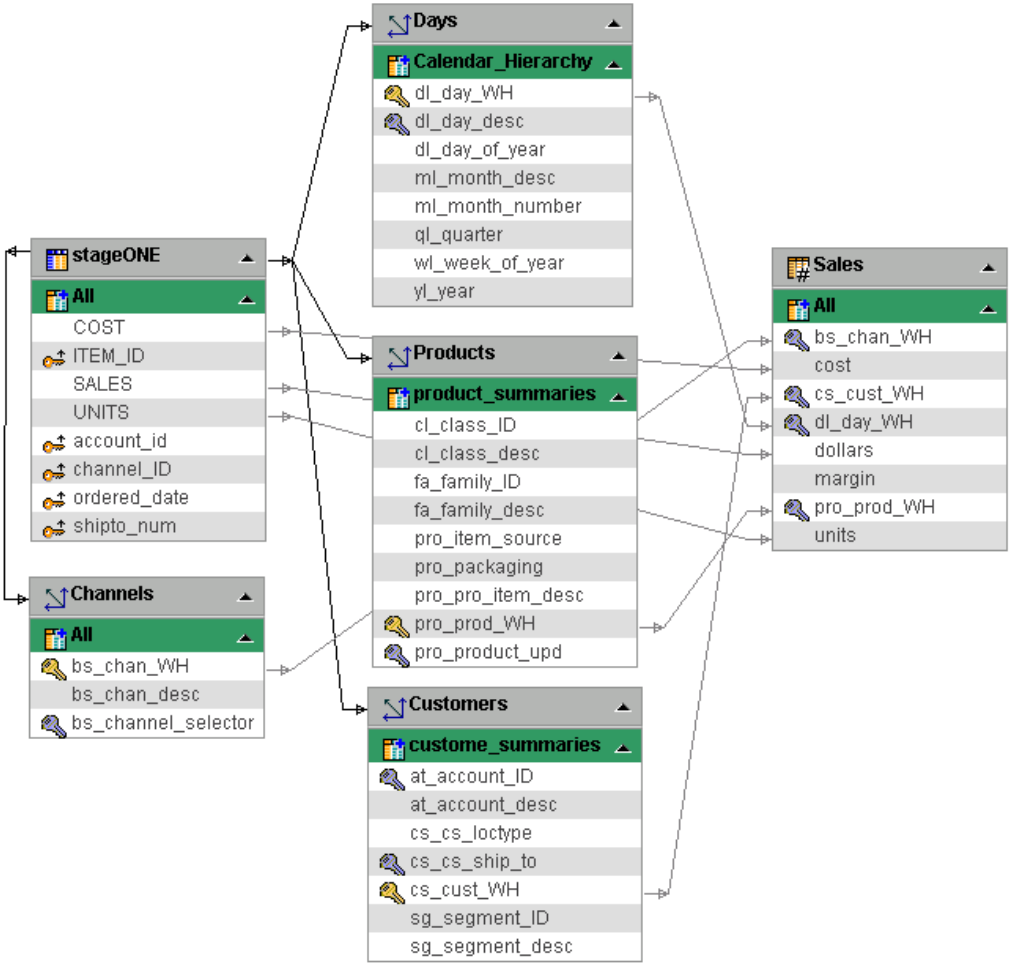
   Builder returns the Detailed Mapping Editor window which displays the ***stageONE*** and *Sales* tables.

2. Use the Secondary Source icon to include each dimension in the mapping as a foreign key related source. You must include the dimensions in the mapping.



The editor's window now displays the stageONE table, each dimension table, and the Sales table, and you can now map measures and warehouse keys from the sources to the Sales table. Figure 6–6 illustrates the final mapping.

Figure 6–6 Copy Transforms for the Sales Mapping



All the copy transforms that extract data from the sources have been defined. To complete the mapping, you must add a transform to calculate values for Margin.

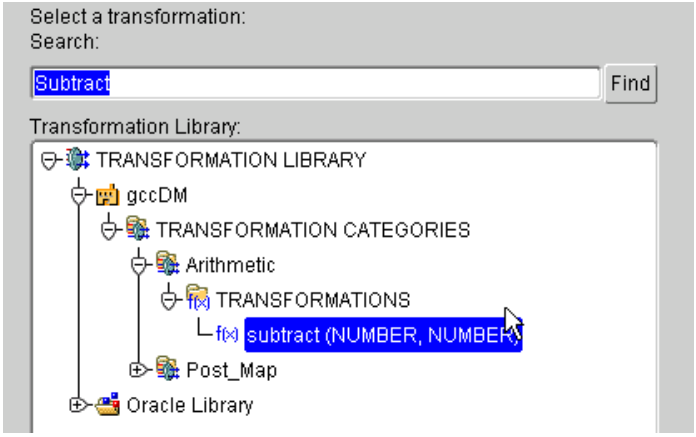
Include the Subtract Transform

To complete the mapping, you must include the Subtract transform to calculate values for the Margin column. This example assumes that a definition for a custom Subtract transform has already been created and resides in the *gccDM* warehouse module. The next section shows you how to create a definition for this transform.

1. Open the ***map_Sales*** mapping with the Detailed Mapping Editor.
2. Drop the **Transform** icon from the tool palette on the editor's workspace.

   The editor returns the New Transform Usage Wizard's welcome page.
3. Click **Next**.

   The editor returns the wizard's Transform Selection page. This page contains a search text box and a directory tree for all transforms stored in the repository.



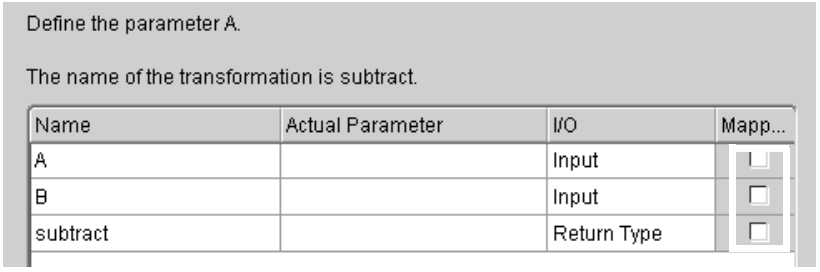
4. Enter 'Subtract' in the search box and click **Find**.

   The wizard highlights the transformation.
5. Click **Next**.

   The wizard returns the first of several pages used to specify parameter values for the selected transform. The ***Subtract*** transform has three parameters:

   - The *A* input parameter (***stageONE.dollars***).
   - The *B* input parameter (***stageONE.cost***).
   - The return parameter (*Sales.margin*).

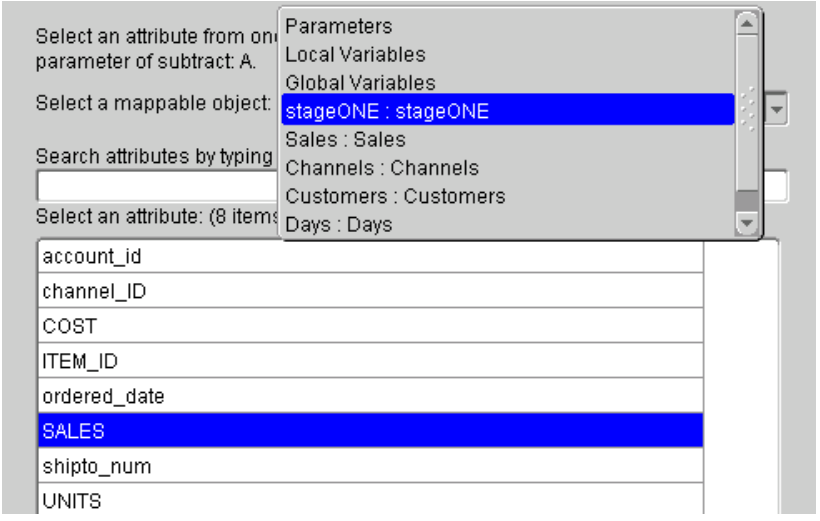
   You must specify a value for each parameter using the wizard pages.



This page describes each parameter that must be mapped.

6. Click **Next**.

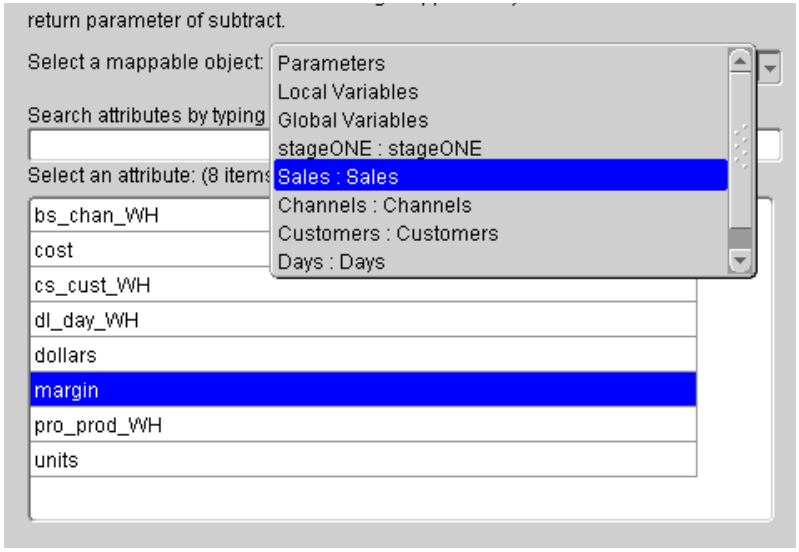
   The wizard returns the next page which solicits information for the first parameter (***A***).



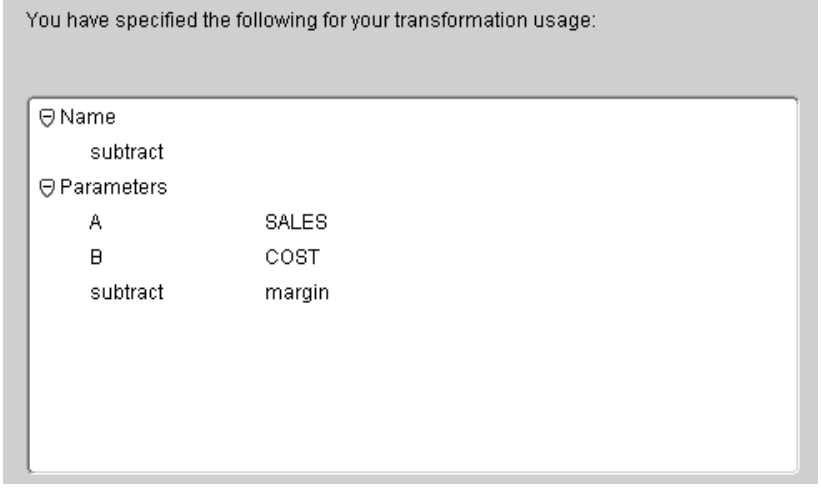
7. Select ***stageONE:stageONE*** as the Mappable Object, select ***SALES*** as the parameter's value, and then click **Next**.

   The wizard returns the next page which confirms the mapping of the parameter by placing a check in its Mapped column.

   Continue through these pages until values have been specified for each of the transform's parameters. Note that values for input parameters are copied from the ***stageONE*** table but the value for the output parameter is copied from the ***Sales*** table.



After values have been specified for each parameter, the wizard returns its Finish page. This page summarizes the values you selected. Verify the description and make any necessary corrections before you click Finish.



8. Click **Finish**.

The wizard includes the ***Subtract*** transform in the detailed mapping and places a diagram for it on the Detailed Dimension Editor's window.

Include the Post Processing Cleanup Transform

After the initial load of the Sales table, the data in the stageONE is no longer needed. Moreover, this data must be deleted before you can use the staging table to update the Sales table with next month's data.

You could write a simple TRUNCATE statement to remove data from the staging table after you verified that the Sales table load was successful. You can also automate this task by defining a post-map trigger within the mapping for the staging table that executes the statement automatically.

Specify the Transformation This requires only a few steps. First, you must create a definition for a transformation that deletes the rows from the table. Second, you define a trigger within the mapping to execute the transformation function. This section shows you how to define the trigger for the *truncate_table* transformation that you defined previously (see "The Truncate_Table Transform" on page 6-32).

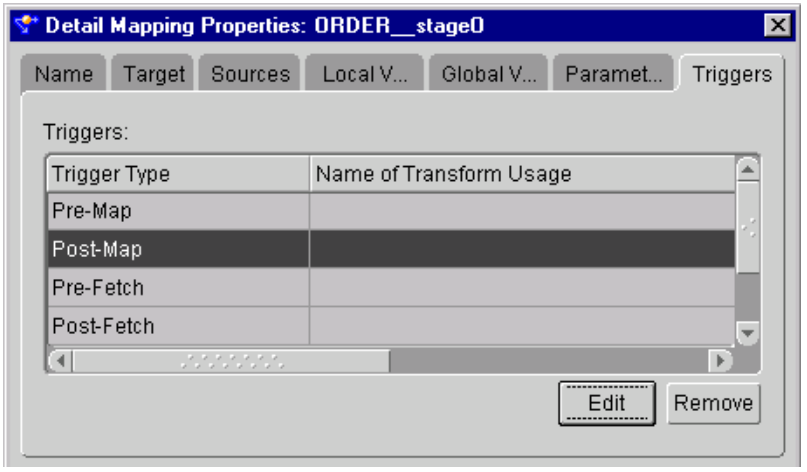
To define a post-map trigger that executes this transformation:

1. Right-click the editor's white space and select Detail Mapping Properties from the pop-up list.

   The editor returns the property sheets for the detail mapping.

2. Select the **Triggers** tab.

   The editor returns the Trigger sheet.



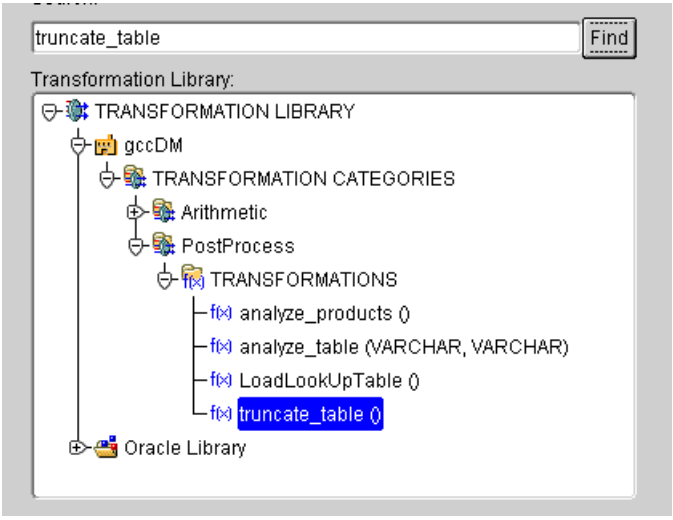
3. Select **Post-Map > Edit**.

The editor returns the welcome page for the New Transformation Usage wizard. This page list the number of steps.

4. Click **Next**.

   The editor returns the Transform Selection page. Use this page to specify the transformation that will be executed by the post-map trigger.

5. Enter the name of the transformation (*truncate_table*) and click **Find**.



   The wizard expands the Transformation Library's navigation tree as necessary and highlights the transformation.

6. Click **Next**.

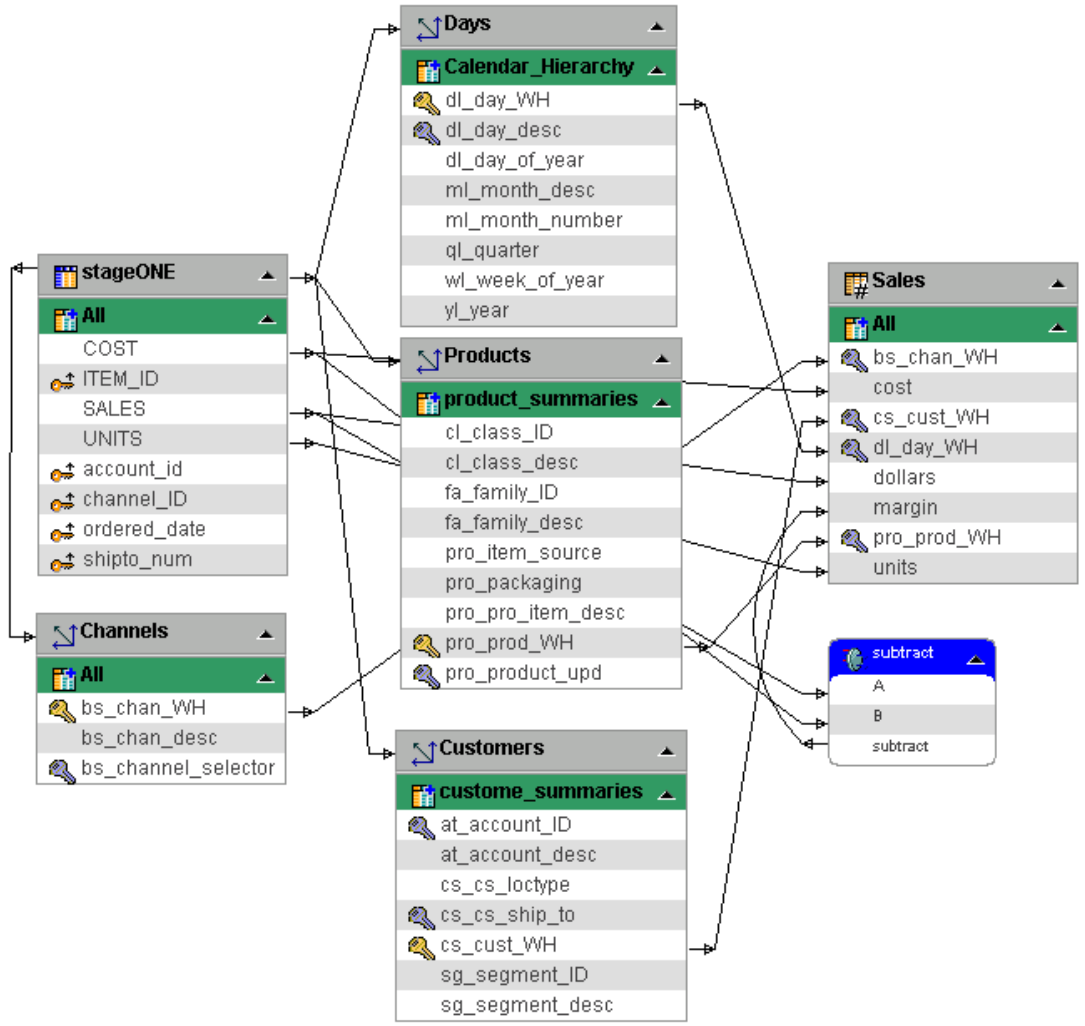
   The wizard returns the Finish page that reports the selection. Verify the selection before you click Finish.

7. Click **Finish**.

   The wizard includes the post-map trigger in the mapping. You can only verify this by displaying the property sheet for the detailed mapping, and then look under the Triggers sheet.

The mapping for the Sales table is now complete.

Figure 6–7 Final Mapping for the Sales Fact Table

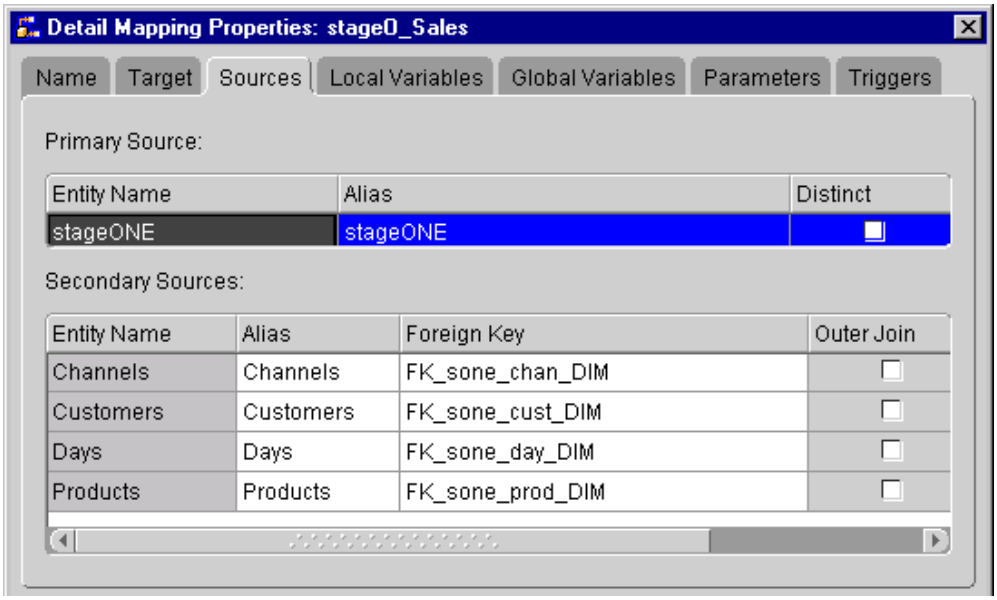


Update a Mapping

You can update a mapping by editing entries stored in its property sheet or by using the Detail Mapping Editor.

Detail Mapping Property Sheet

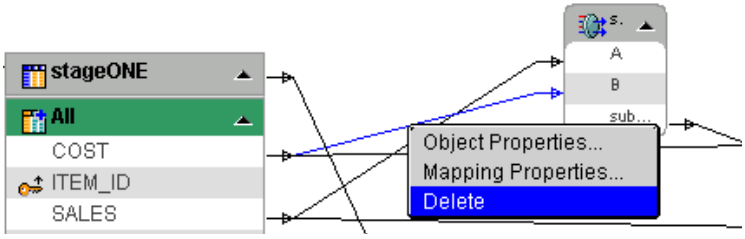
To call the property sheet for a detail mapping, open the mapping with the Detail Mapping Editor, and then click the Properties icon on the editor's toolbar.



Detail Mapping Editor

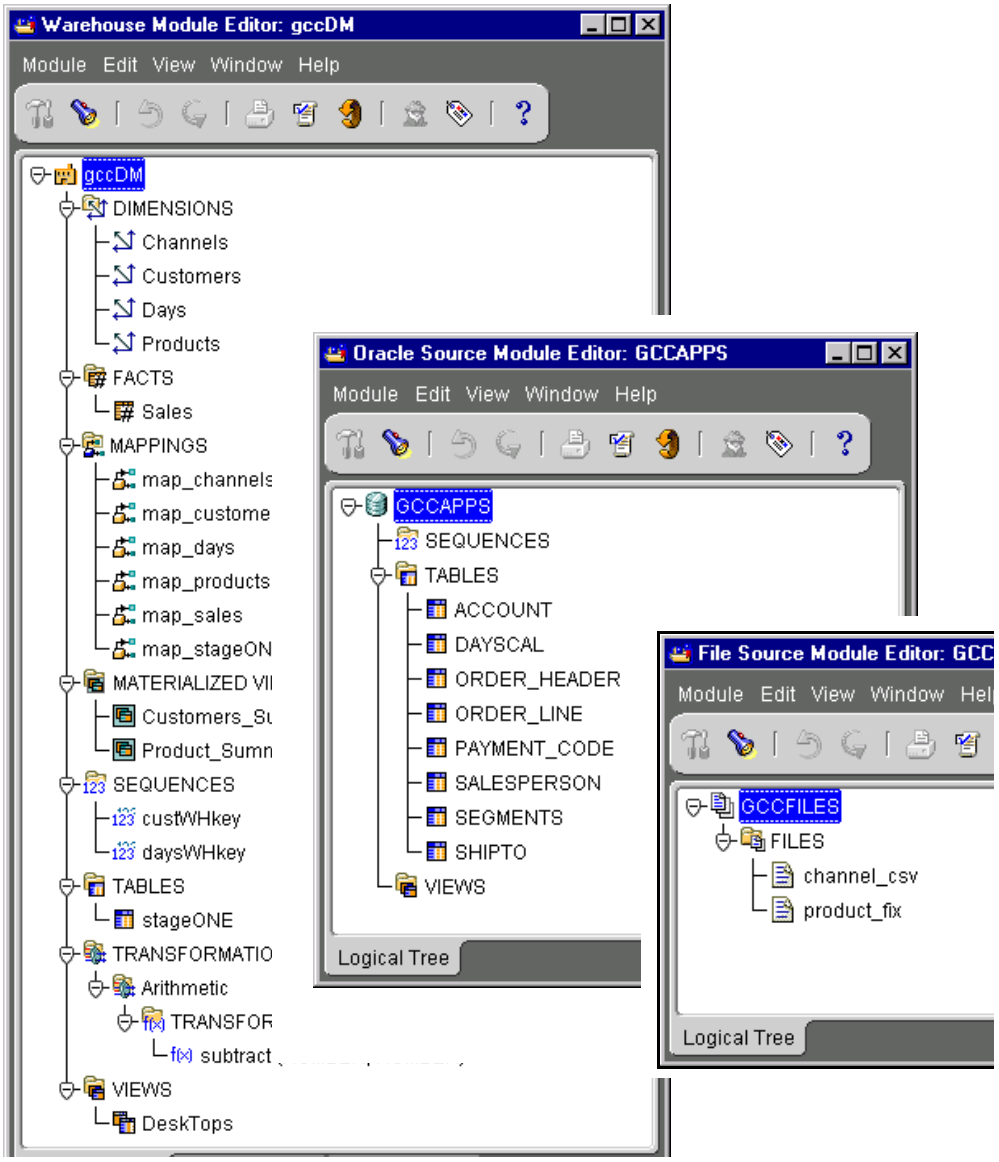
You can edit and remove detail mappings using the Detail Mapping Editor.

To remove or edit a detail mapping, right-click the line that represents the mapping and select an entry from the pop-up list:



Current State of the Project

At this stage, the Builder repository now contains a set of definitions for all the data sources (*GCCAPPS* and *GCCFILES* source modules), for the target schema (*gccDM* warehouse module), and now for all the mappings (*gccDM* warehouse module).



The Next Step

This chapter showed you how to create mappings for the operations that extract and transform data from the data sources described by *GCCAPPS* and *GCCFILES*. These definitions are stored in the warehouse module *gccDM*. The examples also showed you how to update these definitions by editing their property sheets.

Now that the warehouse module contains a logical model of the target schema, its data sources, and its mapping operations, you are ready to move on to the recipe's second instruction.

INSTRUCTIONS

1. Create definitions for the star schema, the individual data sources, and the extract, transform and load routines. These definitions constitute a logical model of the data mart.
2. *Configure, validate, and generate scripts from these definitions and then use the generated scripts to deploy the data mart.*
3. Populate the data mart and publish its contents.

Part III of the Cookbook, "Deploy the Data Mart," shows you how to configure and deploy a physical instance of the case study data mart.

Part III

Deploy the Data Mart

This part of the cookbook follows the recipe's second instruction and shows you how to deploy a physical instance of the data mart from the logical model defined by the *gccDM* warehouse module.

The second instruction directs you to configure a set of definitions that describe a physical instance of the data mart, validate those definitions, generate a set of scripts to create the instance, and then use the generated scripts to deploy a physical instance of the model.

Before you configure a physical instance, however, you need to create a schema object for the data mart using the Builder Runtime Assistant and set up the operating environment for the data mart.

This part includes three chapters:

- Chapter 7, "The Physical Environment"
- Chapter 8, "Configure a Physical Instance"
- Chapter 9, "Generate and Deploy"

7

The Physical Environment

This chapter describes the physical environment that supports the data mart and shows you how to create a schema object that includes all the necessary runtime libraries for the target instance using Warehouse Builder Runtime Assistant.

The data mart runs on a single Oracle8*i* database instance, and this instance also includes all the components required to manage the data mart. This chapter describes the instance and shows you how to set up a batch user on Windows NT so that Oracle Enterprise Manager can schedule job scripts generated by Builder. These jobs initially populate and periodically refresh the data mart.

This chapter covers the following topics:

- The Basic Steps
- The Basic Operating Environment
- Batch User Name for OEM
- Current State of the Project
- The Next Step

The Basic Steps

Before you can configure a physical instance of the data mart, you need to collect specific information about the operating environment and complete a few tasks:

1. Collect specific information about the Oracle8*i* database instance and the Windows NT system which together host the data mart.
2. Create a batch username (OEM) on the Windows NT system. Oracle Enterprise Manager schedules jobs that initially populate and periodically refresh the data mart using this username.
3. Create a database username (gccDM) that owns the data mart's schema object.
4. Initialize the schema object with a set of runtime audit and logging tables using Builder Runtime Assistant. These tables are mandatory because the generated routines that populate and refresh the data mart store audit and logging information in the tables.

The first section of this chapter describes the overall physical environment, the second shows you how to create the batch user name, and the concluding section shows you how to create and initialize a schema object for the data mart.

The Basic Operating Environment

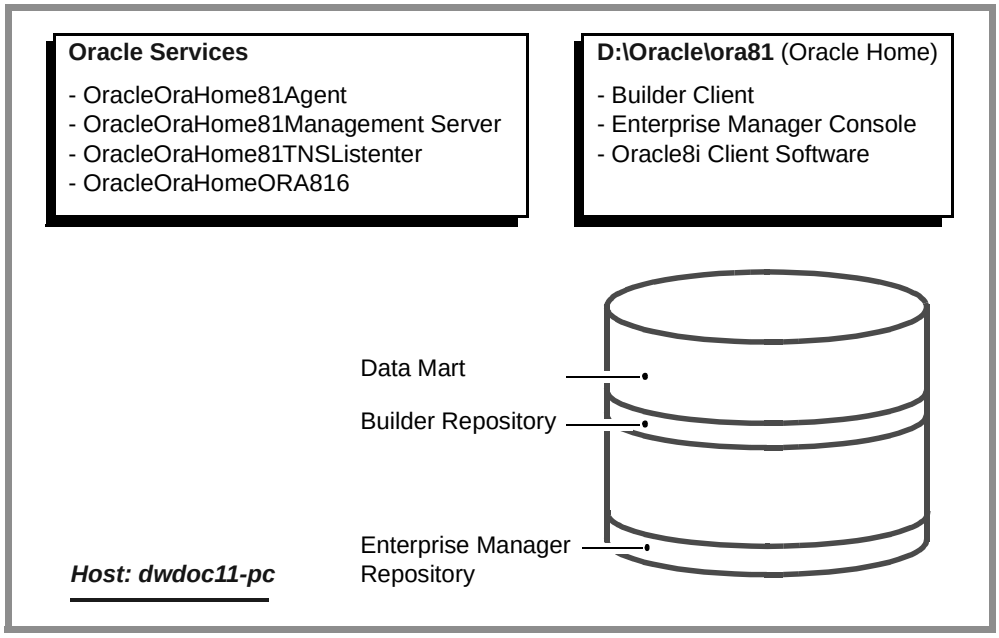
The entire case study data mart and all the Oracle components required to run the data mart reside on a single Oracle8*i* database instance. This includes the target schema, Oracle Warehouse Builder, and the Oracle support components.

The following discussion assumes that this environment has been established. Appendix A, "The Case Study Environment" summarizes in several steps how to install and configure the environment. This environment represents probably the most basic implementation of a data mart that is feasible and it serves to simplify how all the necessary components are configured.

Once you understand this basic environment, you will have sufficient background to implement a distributed environment. For example, you will be able to configure an environment where the target schema resides on one Oracle8*i* database instance, the Builder repository on another, and the Oracle Enterprise Manager repository and Oracle Management Server on a third instance.

Note: Oracle Discoverer and Reports are installed in a separate Oracle home directory.

Figure 7–1 Overview of the Operating Environment



The Components

The components required to develop, deploy, and manage the data mart include Oracle Enterprise Manager, its repository, and an Oracle Intelligent Agent.

Table 7–1 *The Oracle8i Instance and its Components*

Component	Description
Oracle8*i* Database Instance	Database host for the data mart
Home: D:\Oracle\ora81	Home directory for the database instance
SID: ORA816	SID for the database instance
Oracle Warehouse Builder	Builder client and three utilities
Builder client	Client used to define and create scripts to implement data mart
Repository Assistant	Utility that creates Builder repositories
Runtime Audit Viewer	Utility that reads audit records
Runtime Assistant	Utility that creates target schema and runtime audit tables

Table 7–1 *The Oracle8i Instance and its Components*

Oracle Enterprise Manager	Database Administration Tools
OEM Console	Oracle Enterprise Manager client
OMS	Oracle Management Server manages the OEM repository
Oracle Intelligent Agent	Executes batch jobs for Oracle Management Server
Oracle TNS Listener	Network database listener
dwdoc11-pc	Computer Name (Node)

The Schema Objects

The Oracle8*i* database instance contains three schema objects: the target schema, the Builder repository, and the Oracle Enterprise Manager repository.

Table 7–2 *The ORA816 Schema Objects*

Owner Name	Description
gccDM	Target schema for the data mart
Target Tables	Dimensions, facts, stageONE table, and materialized views
PL*SQL Packages	Stored procedures generated and deployed by Builder that extract and transform data from database sources (GCCAPPS).
Runtime Audit Tables	Tables created by the Runtime Audit Viewer. The Builder generated jobs write log entries to these tables during a load operation. You can display their contents using the Runtime Audit Viewer or SQL*Plus.
gccrep	Builder repository which contains definitions for the case study. These include definitions for all data sources, all target tables, and all the mapping definitions that describe how to extract, transform, and load the data mart. Currently, these definitions describe a logical model of the data mart which can be configured to describe a physical instance. The next chapter shows you how to configure a basic physical instance of the model.
OEM	Oracle Enterprise Manager repository contains generated Tool Command Language (TCL) scripts in its Job Library. Enterprise Manager uses these scripts to schedule and run jobs that initially populate and periodically refresh the data mart. TCL is a language product developed by Scriptics.

Batch User Name for OEM

Builder generates a set of scripts which you deploy to a file system and then register with Oracle Enterprise Manager. When you register the scripts, Builder stores them as entries in the Enterprise Manager Job Library. You can schedule these jobs to initially populate and periodically refresh the data mart using Enterprise Manager.

After a job has been scheduled, the scheduling and execution of the job are managed by the Oracle Management Server. When a job is ready to run, the Management Server logs onto the Windows NT system as a batch user, logs onto the target schema, and then calls an Oracle Intelligent Agent to run the job.

For the scheduling process to work, you must create a batch username on the Windows NT system and then record that username and password pair as Preferred Credentials for Enterprise Manager. You must also record a username and password pair that allows the Management Sever to log on to the target schema using Preferred Credentials. This section shows you how to complete these tasks.

Create Windows NT Batch User

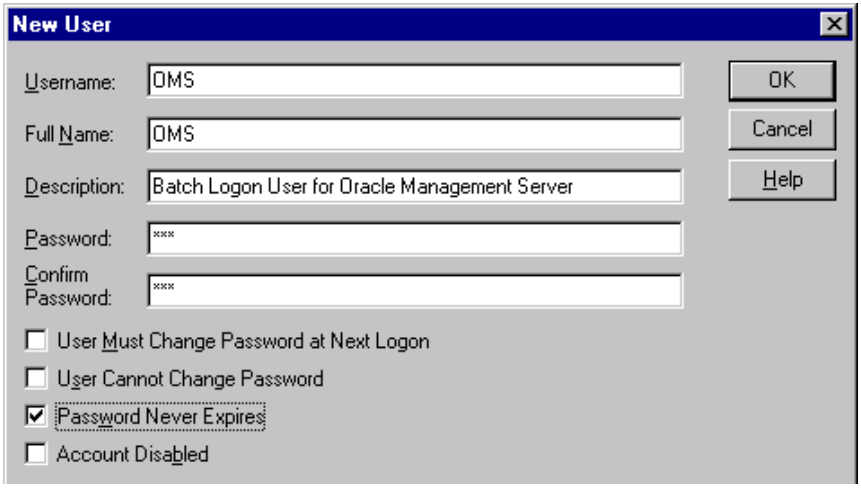
To create the Windows NT batch user on the target system host:

1. **Start > Administrative Tools > User Manager**

   Windows NT returns the User Manager window.

2. Select **User > New User**

3. Windows returns a window which solicits information about the new user.



4. Enter the following values:
   - User Name: *OMS*
   - Full Name: *OMS*
   - Description: *Batch logon for Oracle Management Server*
   - Password
   - Password never expires
5. Click **OK**.

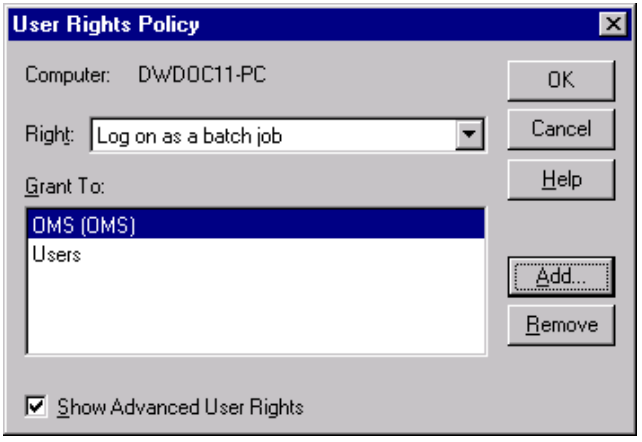
   Windows inserts the new user's name into the list of user names.

   Note:

   Windows NT can be sensitive to upper and lower case letters. Thus, when you record this user name as a Preferred Credential for Oracle Enterprise Manager, enter the characters in the same case as the NT logon.

6. Select the new user name, and then select **Policies > User Rights**...

   Windows returns the User Rights panel.



7. Check the box to display the advanced rights, and then select the entry that permits logging on as a batch job.
8. Click **Add**.

Windows returns another panel (Add Users and Groups).

9. Select the name of the just created *OMS* user and click **Add**. You may need to select the domain for that user first.

Configure Preferred Credentials

You must register two sets of usernames and passwords as Preferred Credentials for Enterprise Manager so that the Oracle Management Server can logon as a batch run to the Windows host and the target schema as a database user.

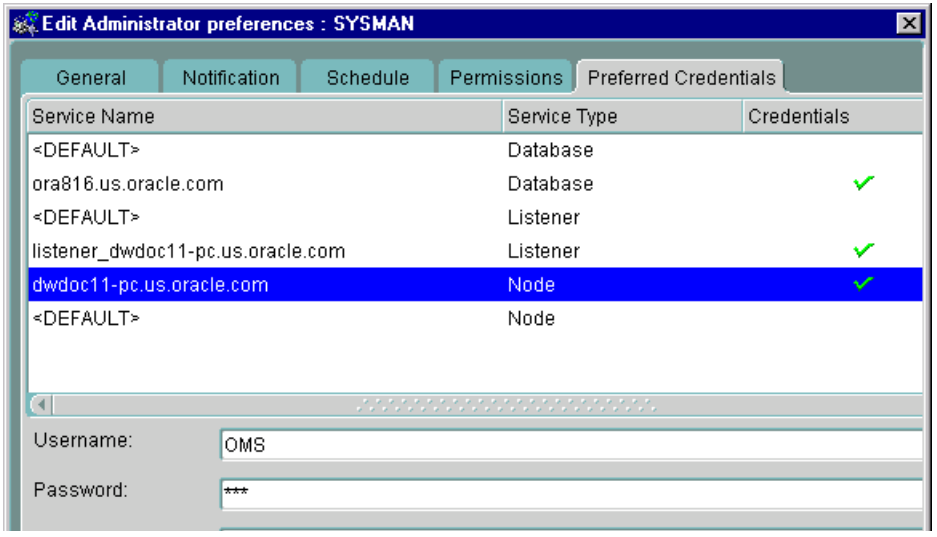
To record these credentials, start the Enterprise Manager Console:

1. Select **System > Preferences**

   Enterprise Manager returns its Edit Administrator Preferences window.

2. Click **Preferred Credentials**

   Enterprise Manager returns a panel that lists Service Names and Types.



3. Select the Service Name that hosts the target schema (Service Type is Node): *dwdoc11-pc.us.oracle.com*.

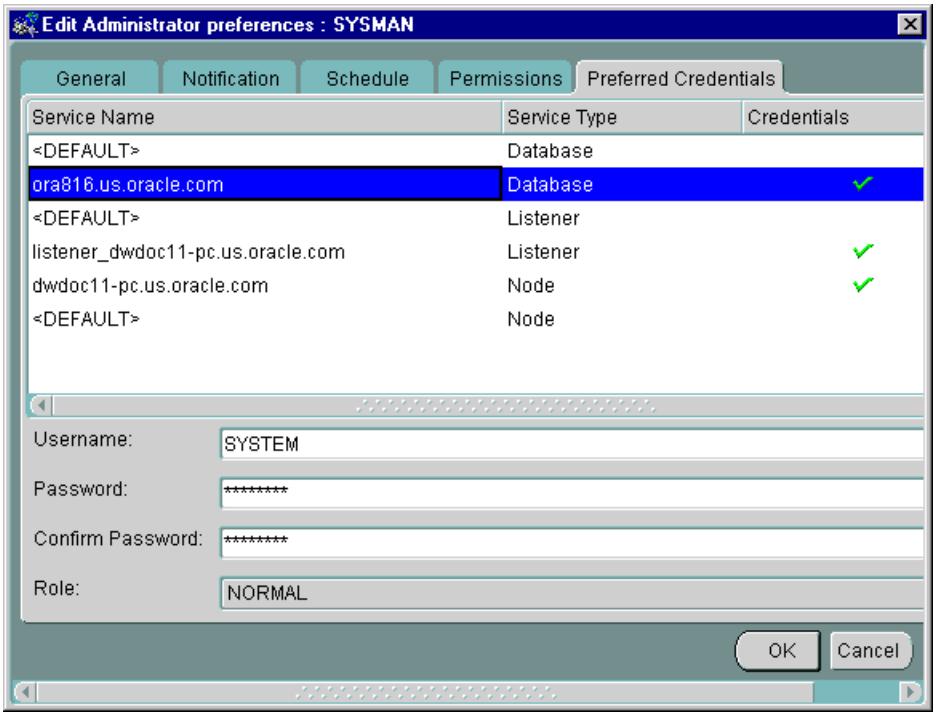
   Enterprise Manager returns a User Name and Password panel.

4. Enter a user name and password that provides access to the host: *OMS*.

   This is the user name that was just created for the Windows NT host.

5. Select the Service Name of the target schema (Service Type is Database): *ora816.us.oracle.com*.

   This is the service name Enterprise Manager assigned to the database.



6. Enter a user name and password that provides access to that database.

The Schema and Runtime Tables

You create the target schema object for the data mart using the Builder Runtime Assistant. This utility creates a database username (gccDM) with the correct authorizations and privileges and then initializes the resulting schema object with Builder runtime audit and logging tables.

Builder's Runtime Support component uses these tables to log audit and error information captured at runtime from jobs that load and refresh the data mart. You can analyze this information using the Builder Runtime Audit Viewer. Chapter 10, "Populate the Data Mart," shows you how to access these tables using the Viewer.

Tablespace Allocation

To create a schema object, you actually create a database username for the owner of the object and assign permanent and temporary tablespaces to the username. When that username creates persistent database objects, Oracle8*i* creates the object in the assigned permanent tablespace. The following example assumes that two tablespaces have been created (100 MB is more than sufficient for these tablespaces):

- GCC_MAIN
- GCC_TEMP

See "Create a Tablespace for the Repository" on page A-11 for an example that shows how to create a tablespace using DBA Studio.

Create the Schema Object

To create and initialize the target schema with the runtime libraries, call the Oracle Warehouse Builder Runtime Assistant.

To create and initialize the target schema

1. Select **Start > Oracle > Oracle Warehouse Builder Runtime Assistant**.

   The assistant returns the Welcome page for its wizard.

2. Click **Next**.

   The wizard returns its Chose Operation page which prompts you:

   - Install a new Warehouse Builder Runtime Library.
   - Drop an existing Warehouse Builder Runtime object.

3. Select the button for **Install a New Runtime** and then click **Next**.

The wizard returns the Server Information page. This page solicits information required to establish a connection with the data mart's Oracle8*i* host.

- Hostname (***dwdoc11-pc***)
- Port (***1521***)
- SID (***ora816***)

4. Enter the connection information and then click **Next**.

   The wizard returns the New or Existing Schema page. This page prompts you for another choice:

   - Create and install into new user Schema.
   - Install into an existing user Schema.

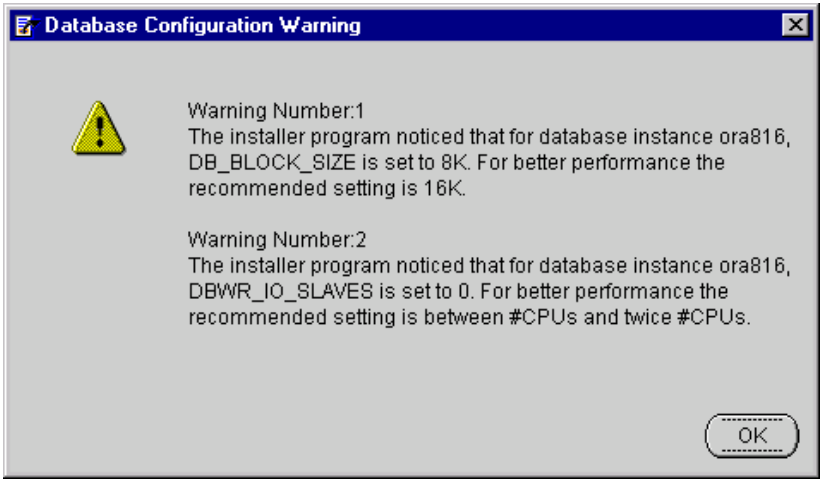
   If you install into a new user schema, then the wizard creates the new user schema *and* a Runtime Library within that schema.

5. Choose **Create and install into new user Schema** and then click **Next**.

   The wizard returns the Username and Password page. This page prompts you for a Username and Password authorized to perform DBA operations. This level of authorization is required to create database users.

6. Enter a Username and Password (***system/manager***) and then click **Next**.

   If you use the operating environment described in Appendix A, "The Case Study Environment," then the wizard returns a warning:



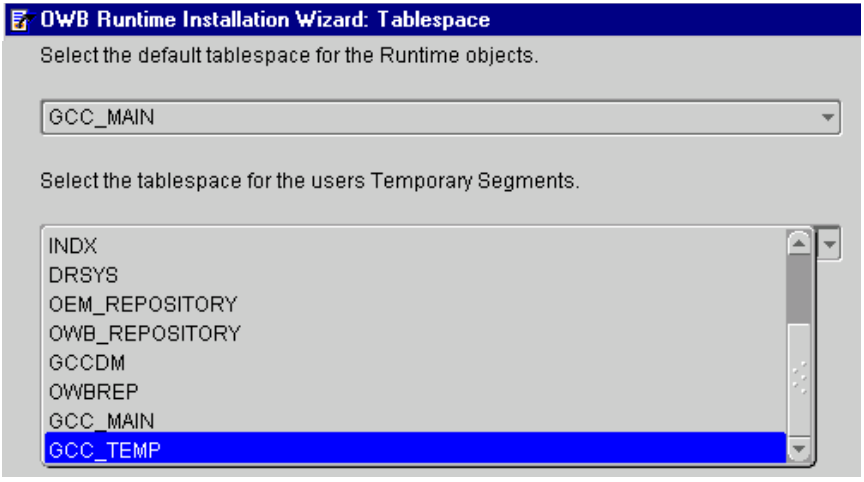
The wizard returns the warning because the block size parameter for the Oracle8*i* database instance is set to 8192:

```
db_block_size = 8192
```

Large data marts and data warehouses operate more efficiently with larger blocksizes, 16,384 for example. While this is an extremely important parameter for an actual data mart, you can work through the case study examples with no problem using the smaller block size.

7. Click **OK** on the warning panel.

   The wizard returns the Tablespace page. This page prompts you for the names of a default and a temporary tablespace.

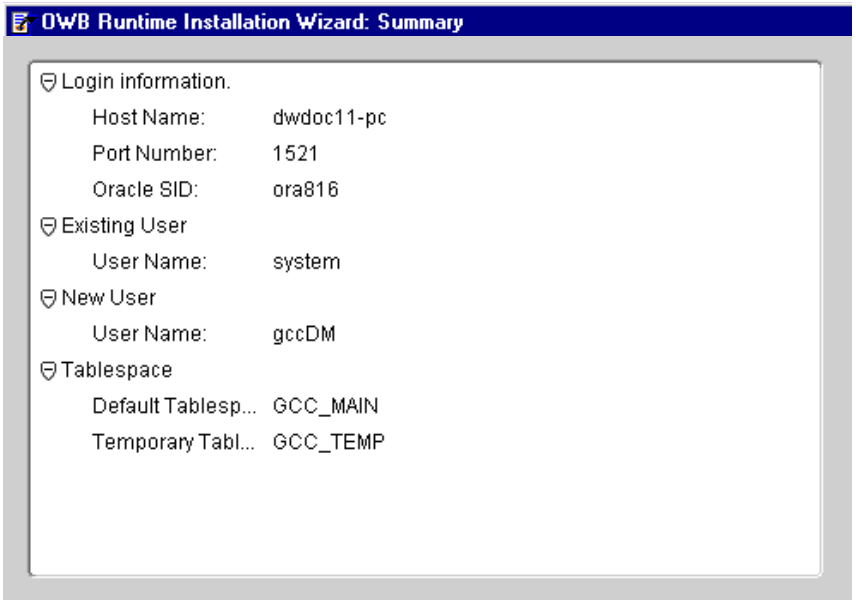


8. Use the down-arrows to select the tablespace names and then click **Next**.

   The wizard returns the Username and Password page. This page prompts you for a username and password for the owner for the target schema.

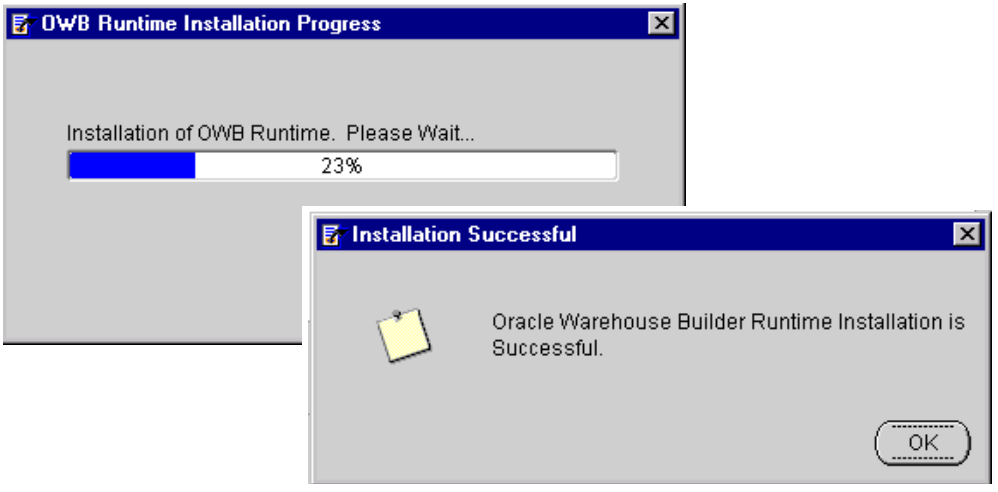
9. Enter a Username and Password (***gcc****DM****/gcc200***) for the owner of the new schema object and then click **Next**.

   The wizard returns its summary page. Verify the description and make any necessary corrections before you click Finish.



10. Click **Finish**.

    The wizard returns its Progress Panel while it creates the Runtime Library. After the operation completes, the wizard returns a completion status panel.



You can now log on to the schema with SQL*Plus and list the contents of the Runtime Library. The runtime library requires roughly 60 MB of disk space and consists of tables, PL/SQL packages, sequences, database triggers, and synonym. For additional information regarding these database objects, refer to the *Oracle Warehouse Builder User's Guide.*

Current State of the Project

At this stage, the project has includes:

- An Oracle8*i* database instance (ORA816) that runs on a Windows NT host named dwdoc11-pc. The Oracle Enterprise Manager Console, the Oracle Management Server, and an Oracle Enterprise Manager repository are installed on this database instance.
- A Builder repository on the ORA816 database that contains a set of definitions that completely describe a logical data mart for the case study.
- A batch username on the data mart's Windows NT host. This username/password pair and a database pair have been registered as Preferred Credentials with Oracle Enterprise Manager.
- A schema object (gccDM) on the ORA816 database which has been initialized with the Builder runtime library.

The project team also has a complete description of the physical environment for the case study data mart.

The Next Step

This chapter described the data mart's physical operating environment and showed you how to complete several preliminary tasks.

Now that you have a description of the data mart's physical environment and the preliminaries have been completed, you are ready to configure a physical instance of the data mart from the logical model described in the *gccDM* warehouse module. The next chapter, "Configure a Physical Instance," shows you how. The configuration process depends on most of the details spelled out in this chapter.

8

Configure a Physical Instance

This chapter shows you how to configure a *physical* instance of a data mart from the definitions stored in the *gccDM* warehouse module. These definitions currently describe a logical model for a data mart which can be physically realized in several ways. The examples in this chapter show you how to configure a physical instance for the operating environment described in the previous chapter. The chapter also shows you how to validate a set of definitions stored in a warehouse module.

To configure a physical instance of a logical model, you assign specific values to selected configuration parameters. These parameters reside in the same warehouse module as the definitions that describe the logical model. You also define any additional indexes which are necessary to support the expected query workload.

This chapter covers the following topics:

- The Basic Steps
- Configure the Instance
- Create Additional Indexes
- Validate the Definitions
- Current State of the Project
- The Next Step

The Basic Steps

The definitions stored in the *gccDM* warehouse module define a logical model which can be physically realized in a number of ways. To describe a physical instance of this model, you assign specific values to selected configuration parameters. During this process, you also define and configure any additional indexes that are required to support the anticipated query workload.

From this set of configured definitions, Builder generates a set of scripts which are used to create and populate the physical instance. Before you can generate scripts for the case study instance, you must first configure eight parameters which describe the actual operating environment. The remaining configuration parameters, which represent the majority, are allowed to assume default values.

Builder uses the eight parameter values to generate and store a set of job scripts in the Oracle Enterprise Manager's job library. These jobs can be used to initially load or periodically refresh the data mart, and you can schedule the jobs to run automatically or on demand using the Enterprise Manager Console.

Three Steps

The example instance is easy to configure because most of the configuration parameters are allowed to assume their defaults:

1. Assign specific values to eight configuration parameters which describe the actual operating environment.
2. Optionally change the configuration of the database links.
3. Define and configure indexes necessary to support the anticipated workload.

   Developers routinely define a set of bitmap indexes on the foreign key reference columns of each fact table so that Oracle8*i* can take advantage of the star transformation algorithm during query processing. When the anticipated workload is well-understood, additional indexes are usually defined.

After you configure the physical instance, you can validate the definitions stored in the warehouse module and then use them to create and populate the data mart.

Access Configuration Parameters

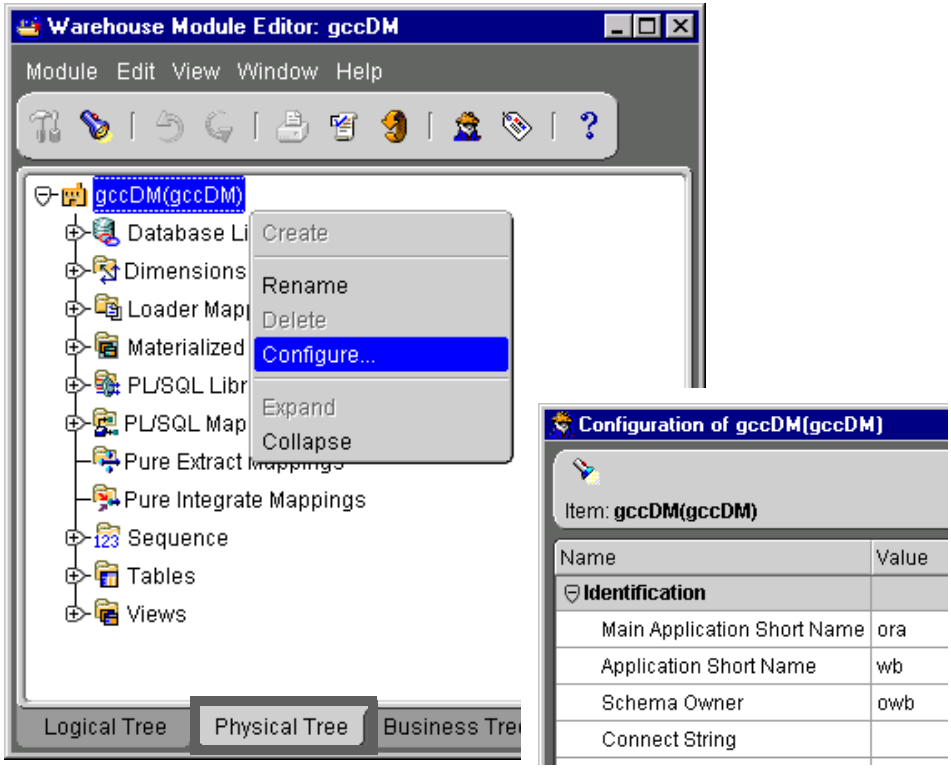
The configuration parameters and their default values reside in the *gccDM* warehouse module. To access these configuration parameters, you must open the warehouse module, switch to its *physical navigation tree*, and open the Configuration

Panel for the warehouse module. You assign a new value to a parameter by editing its default value.

To display the warehouse module's configuration panel:

1. Double-click the warehouse module's name in the Project navigation tree.

   Builder opens the warehouse module in a separate window.



2. Click **Physical Tree** at the bottom of the editor's window.

   The editor displays the physical navigation tree.

3. Right-click ***gcc****DM* > **Configure**.

   The editor displays the configuration panel for the warehouse module.

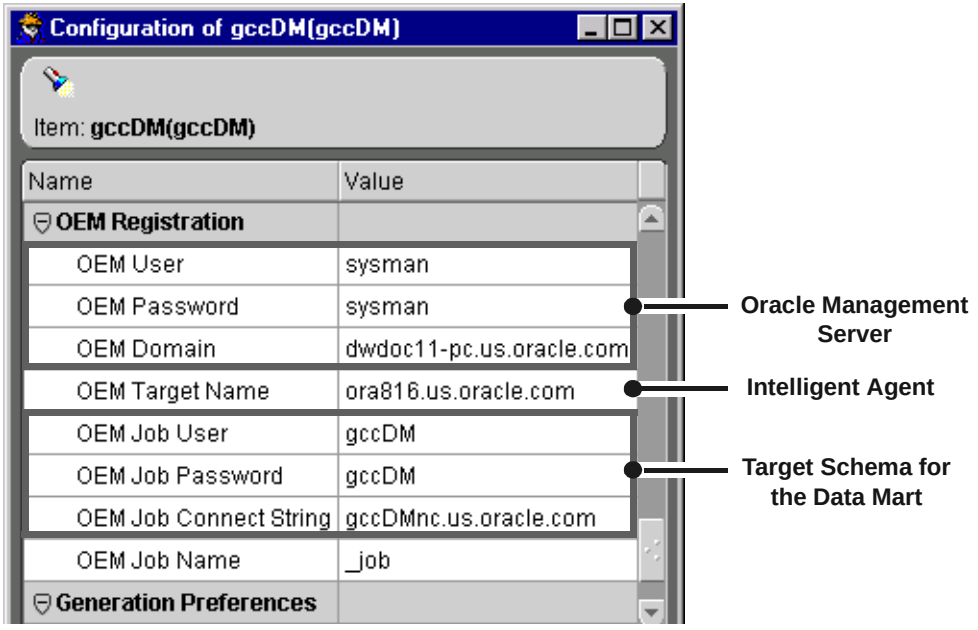
The configuration panel for a warehouse module is divided into seven sections, and the parameters you need to configure reside in the OEM Registration and Generation Target Directories sections.

Configure the Instance

To configure the example instance, you must assign specific values to eight configuration parameters. Builder uses these values to generate and store a set of jobs in the Enterprise Manager job library which are *designed* to be scheduled and run by Oracle Enterprise Manager. These parameters reside in two sections of the warehouse module's configuration panel: the OEM Registration and the Generation Target Directories sections.

OEM Registration

You *must* supply the following information about the operating environment in the OEM Registration section:



Oracle Management Server

To store the generated scripts in the Job Library, Builder must log on to the Oracle Management Server. This server manages the job library. The values in the above example reflect the physical environment described in the previous chapter.

To determine the actual values required for your site, use the same information that you would use to log on to an OMS with the Enterprise Manager console.



The above illustration shows that two Oracle Management Servers are visible. The data mart resides on dwdoc11-pc.us.oracle.com and that is the value selected.

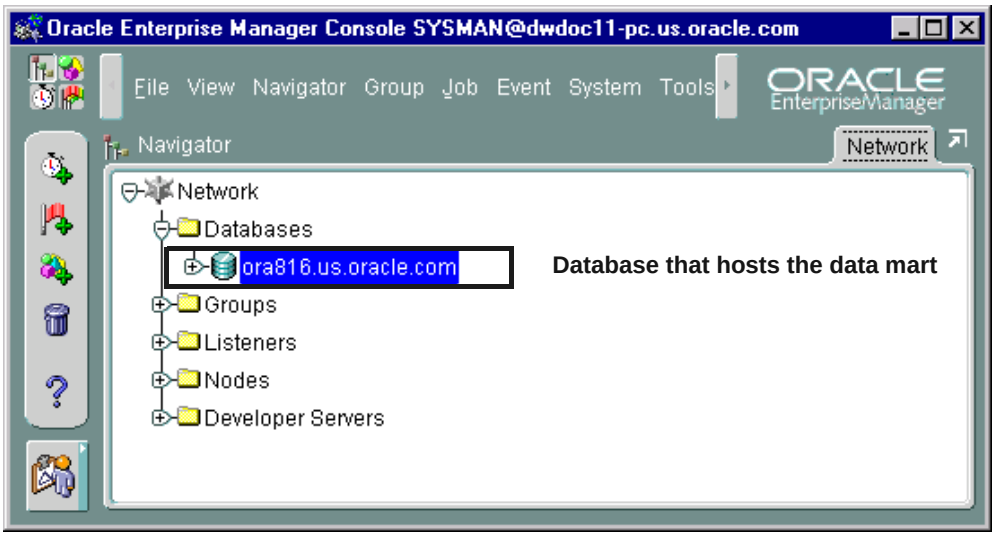
Note:

- Enter the name of the Oracle Management Server *exactly* as shown on the log on panel; otherwise, Builder will not be able to connect and store the scripts in the Job Library.
- The domain (us.oracle.com above) varies considerably depending on how your site configures Net8.

Intelligent Agent

When you schedule a job with Enterprise Manager, the job is actually managed by an Oracle8*i* Intelligent Agent. Assign the name of that Agent to the OEM Target Name parameter. For the case study, this name is the same as the name of the database instance that hosts the target schema (ora816.us.oracle.com).

To determine the actual value required for your site, log on to the OMS with the Enterprise Manager Console and locate the name of the database instance that hosts the data mart. Specify the name of that instance exactly as it occurs in the Enterprise Manager Console window (see example on the following page).



Note:

- Enter the name of the database instance *exactly* as it appears in the Enterprise Manager Navigator pane; otherwise, the OMS will not be able to connect with the Oracle8*i* Agent at runtime.
- The domain (us.oracle.com above) varies considerably depending on how your site configures Net8.

Target Schema

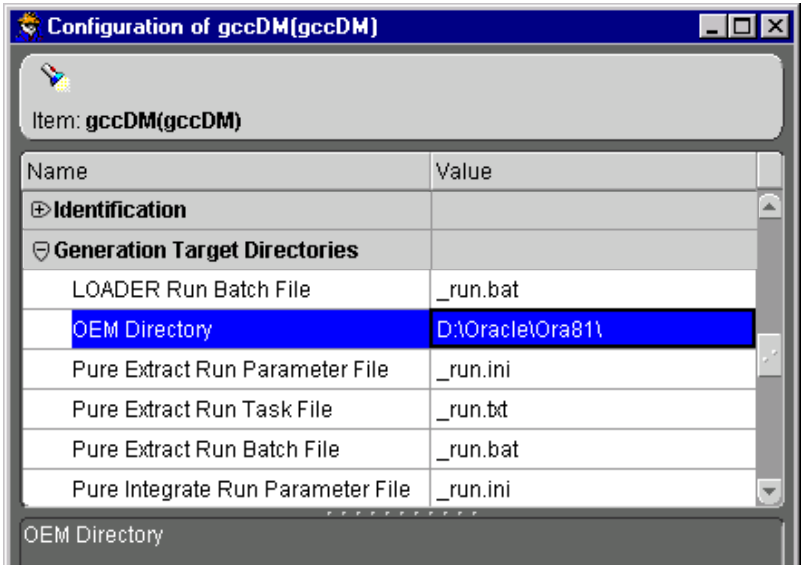
The generated job scripts include connection information which the Oracle8*i* Agent uses to log on to the target schema at runtime. You provide this information by configuring three parameters:

Parameter	Value	Description
OEM Job User	gccDM	Owner of the target schema
OEM Job Password	gccpword	Password for owner username
OEM Job Connect String	gccDMnc	SQL*Net connect string for the target schema. This connect string resides in the host database tnsnames.ora file. You or your database administrator can create an entry using the Net8 Assistant.

Generation Target Directories

Builder uses several routines to generate and store the job scripts. These routines reside in the Oracle Enterprise Manager's client library. You must specify the location of this library using the OEM Directory parameter.

The OEM Directory parameter is located in the Generation Target Directories section.



The value for this parameter is the name of the Oracle8*i* home for the Oracle Enterprise Manager client libraries (default value is c:\orant\).

The example operating environment stores these libraries and the database instance that hosts the target schema in the same Oracle8*i* home (D:\Oracle\Ora81\).

Note: This value is highly dependent on the installation of the Oracle8*i* database instance and the Oracle Enterprise Manager's client.

Configure the Database Links

Each time you create a mapping that pulls data from a database source, Builder verifies that a definition for the required database link exists. If the definition does not exist, Builder creates one from the connection information stored in mapping's source module. Thus, the default parameter values for a database link correspond with the connection information in a source module.

When you deploy a physical instance, you must verify that the database links generated by Builder point to the correct sources. For example, if a test application system was used to develop the logical warehouse but the physical instances use real data stored in production systems, then you must reconfigure the definitions to reflect this change.

Builder stores definitions for database links in the warehouse module that pulls the data but these links are visible only from the physical navigation tree.

To display a database link's configuration panel:

1. Double-click the warehouse module's name in the Project navigation tree.
2. Click **Physical Tree** at the bottom of the editor's window.

   The editor displays the physical navigation tree.
3. Expand the **Database Links**.
4. Right-click a database link and select **Configure**.

   The editor displays the configuration panel for the database link.

You can specify the host machine using a SQL*Net connect string or provide the host name, port, and the ID of the database instance.

Create Additional Indexes

Unlike the OLTP environment, indexes are common and plentiful in a decision support environment where query performance is paramount. Developers create indexes on dimension and fact tables primarily to boost query processing, and they often create indexes on materialized views for the same reason.

Nevertheless, developers do not create indexes for every column on every table. Indexes occupy space and can be detrimental to load and refresh operations. Usually, a set of indexes is initially defined for the anticipated query workload. After the data mart goes into operation, actual query performance is evaluated over time and the set of indexes adjusted accordingly.

Case Study Indexes

The query profile for the case study data mart indicates that users often constrain a few dimensions loosely and retrieve a relatively large number of rows from the fact table. Oracle8*i* can boost the performance of these queries if you define bitmap indexes on each foreign key reference column of the Sales fact table. Thus, the case study instance includes a bitmap index on each of the fact table's foreign key reference columns.

Bitmap indexes are more suitable on fact table columns than btree indexes because they occupy far less space and because Oracle8*i* can calculate their combined effect more efficiently. See the *Oracle8i Data Warehousing Guide Release 2 (8.1.6)* and *Oracle8i Designing and Tuning for Performance Release 2 (8.1.6)* guides for an in-depth comparison and discussion of indexing strategies.

The composite btree index that is already defined on the Sales fact table covers situations where a user query tightly constrains all the dimensions and returns relatively few rows from the fact table.

The case study does not initially create additional indexes on the dimension tables. When dimension tables are large and queries frequently attack specific columns, additional indexes on these columns can improve query performance considerably.

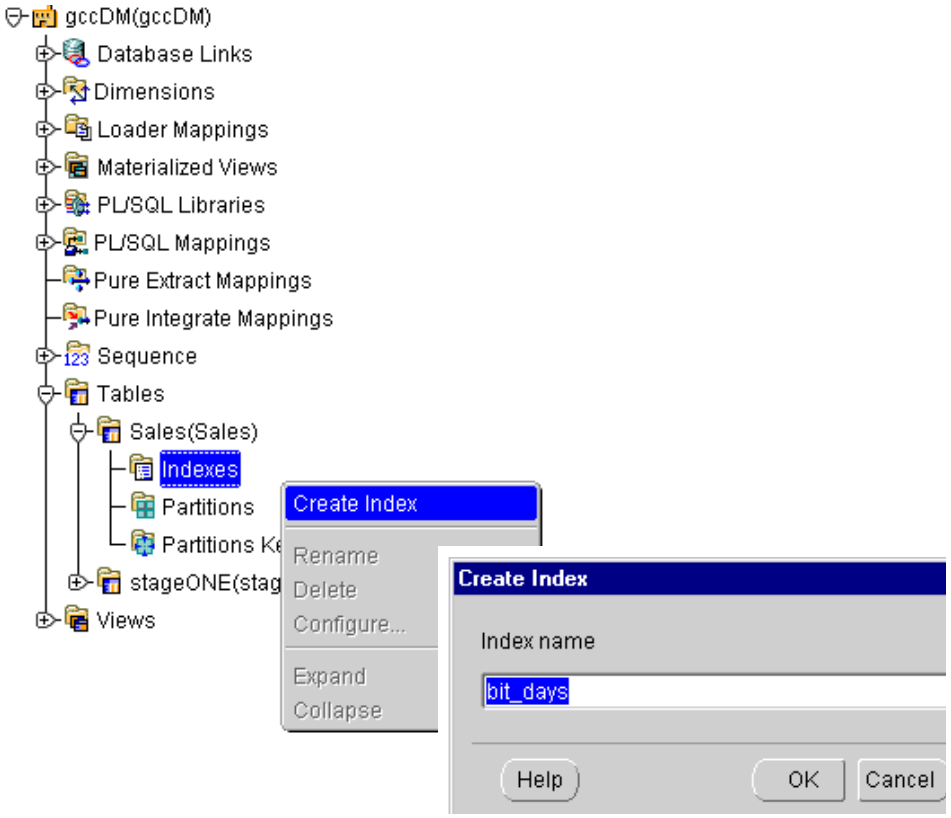
Define Bitmap Indexes

This example shows you how to create and configure a definition for a bitmap index on the *Sales* table's *dl_day_WH* column. To take full advantage of the Star Transformation algorithm, you need to define a bitmap index on each of the Sales table's foreign key reference columns.

To create a definition for an index:

1. Expand the physical navigation tree for the *gccDM* warehouse module.
2. Select **Indexes > Create Index**.

   Builder solicits a name for the index.



3. Enter a name (*bit_days*) and click **OK**.

   Builder insets the name of the index in the navigation tree.

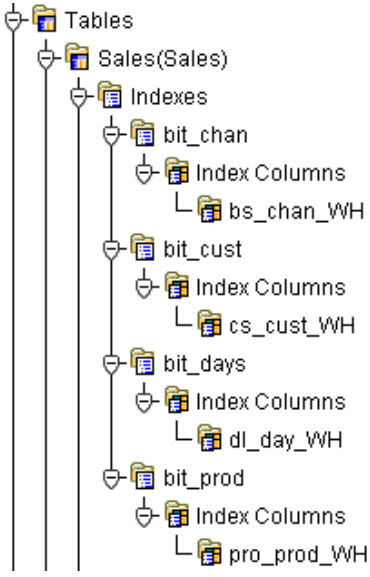
4. Expand the navigation tree, and select **Index Columns > Create Index Column.**

   Builder solicits a column name for the index. You must enter the name of a column *defined by the selected table*.

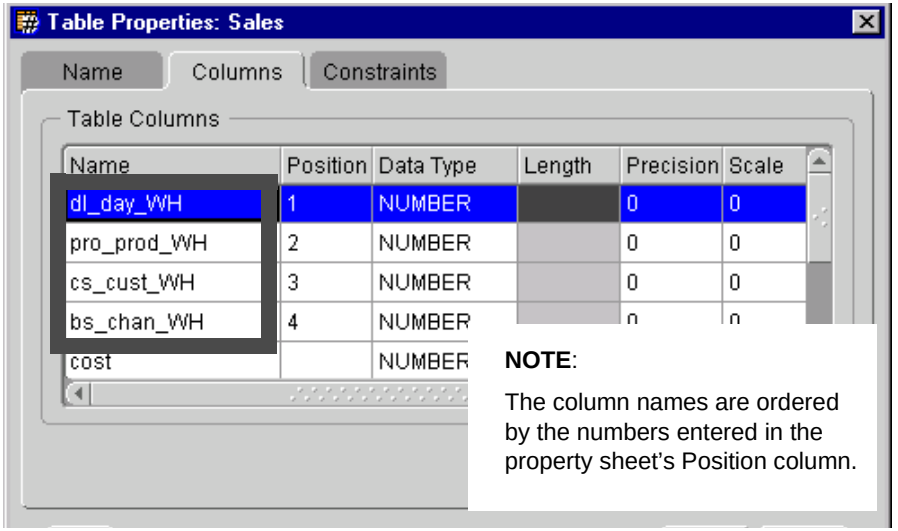
5. Enter *dl_day_WH* and click **OK**.

   Builder inserts the name of the column under the Index Columns branch.

Repeat these steps to define an index on each of the Sales table's foreign key reference columns. After you complete these tasks, you navigation tree should contain the following entries.



To determine the correct name for each index column, open the Sales table's property sheet (switch to the Warehouse Module Editor's logical navigation tree, double-click ***Sales*** to open the Fact Editor, select **Edit** on the toolbar, and select **Table Properties** from the pop-up list).



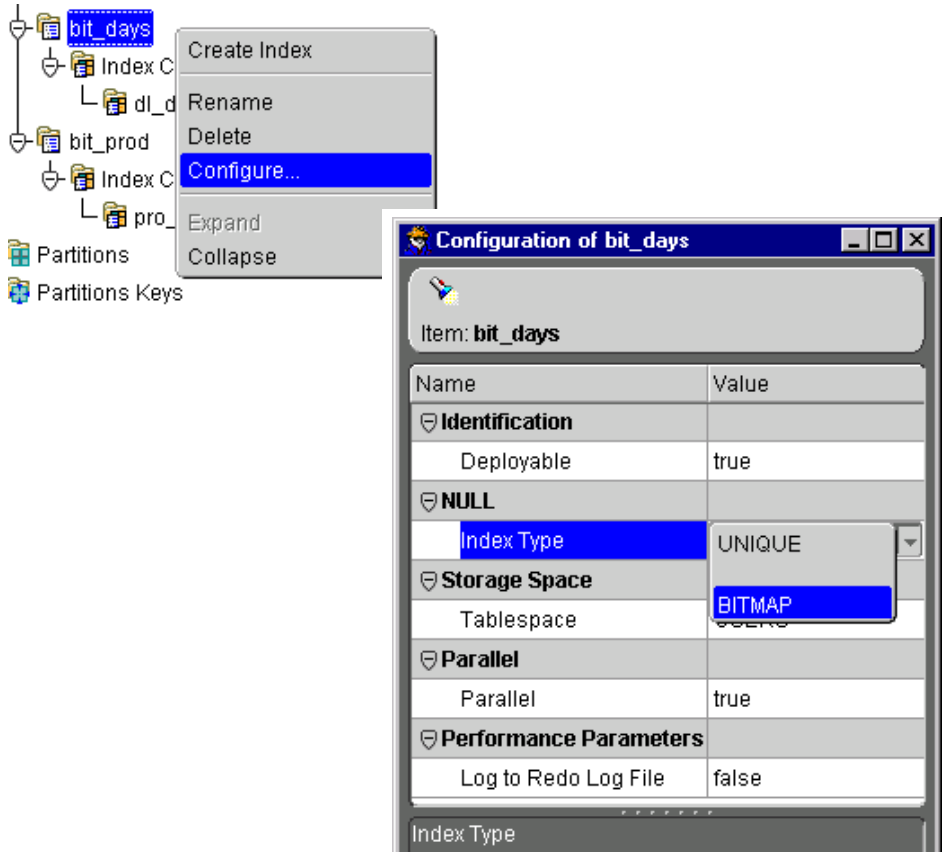
Configure Bitmap Indexes

The indexes defined on the Sales table's foreign key reference columns can be configured as a unique or a bitmap index. The default value for the Index Type parameter is unique.

To configure an index as a bitmap index, display its configuration panel and then assign *bitmap* to the Index Type parameter. The following example shows you how to configure the definition for the *bit_days* index.

1. Expand the physical navigation tree for the *gccDM* warehouse module.
2. Select *bit_days* > **Configure** …

Builder returns the configuration panel for the selected index.



3. Select **BITMAP** from the down-arrow list.

   At this time you can also assign values to the other parameters that determine the index's physical characteristics. The case study instance assigns the default values to these parameters.

   - Deployable: *true*
   - Tablespace: *USERS*
   - Parallel: *true*
   - Log redo: *false*

Builder uses these values to generate the DDL script that creates the index provided that the Deployable parameter is assigned the value *true* (which is the default value).

4. Close the Configuration Panel.

The definition for a *bit_days* index has now been configured. To complete the configuration process, you need to configure the definitions for the remaining three indexes: *bit_prod*, *bit_cust*, and *bit_chan*.

Note:

Each column that participates in an index also has a single configuration parameter: Deployable. The default value for this parameter is *true*.



Builder does not automatically commit your work to the repository. You can commit your work immediately or defer this action until you exit the session. To commit immediately, click the Commit icon on the toolbar. Builder returns a Commit Confirmation panel: click **Yes**. You can also defer this action until you exit the session. On exit, Builder always returns an Exit Confirmation panel. Click **Yes**.

Validate the Definitions

Now that the definitions in the *gccDM* warehouse module describe a physical instance of the data mart, you are nearly ready to generate the scripts that create the target schema's objects and populate the data mart. Before you generate the scripts, you should validate all of the definitions.

Validation Results

You need to validate your definitions before you generate scripts for several reasons. Common errors include:

- Malformed definitions

  As a flexible design tool, Builder allows you to create definitions that can be malformed. For example, you could add a foreign key constraint to a fact table that does not as of yet reference a column in a dimension table. Builder stores this definition in the warehouse module but the definition will fail the validation process until the reference is satisfied.

- Inconsistent configuration values

  You could configure an instance whose fact tables reference a dimension that is not marked to be deployed. Builder stores the configuration values in the configuration panels but the validation process returns the error message *Dangling foreign key.*

- Data type mismatches

  Builder allows you to create definitions that allow data type mismatches but the validation process traps these cases and returns an error message.

- Metadata Import Utility

  The Import Utility can create definitions in the repository that are malformed for a number of reasons.

The *Oracle Warehouse Builder User's Guide* describes the process and the validation rules for constraint definitions in much greater detail. Table 8–1, "Validation Rules for Constraint Definitions" summarizes the rules that govern constraint definitions.

Table 8–1 Validation Rules for Constraint Definitions

Constraint Type	Rules
Custom FK	**1.** FK needs to point to a UK. **2.** Number of FK attributes need to match its UK. **3.** Data types of FK attributes need to match its UK. **4.** FK attribute list cannot be empty. **5.** None of the FK components can be null. **6.** In a mapping, all secondary sources must have at least one foreign key to the primary source.
Custom UK	**1.** UK attribute list cannot be empty. **2.** None of the FK components can be null. **3.** UK attribute list cannot contain duplicate columns.
Dimension: Generated UK	**1.** The generated attribute of level UK should not be removed. **2.** The component of a level UK cannot be altered once there is a fact pointing to this UK. **3.** A level attribute cannot be deleted if it is part of a level UK and there is a fact pointing to this UK. **4.** The level unique key cannot be deleted. **5.** A generated unique key of a dimension must contain columns from the same level or custom column.
Fact Table: Generated UK	**1.** The generated fact FK cannot point to another unique key once it is created. **2.** The component of a generated fact foreign key cannot be altered once it is created.
Fact Segmented UK	**1.** Returns exception dialog when these segmented unique keys are deleted from the table property sheet.
Dimension Generated Level ID	**1.** Returns an exception dialog when these columns are deleted.
Dimension additional level columns	**1.** Returns an exception dialog when these columns are deleted.
Fact Column Generated from fact / dimension relationship	**1.** Returns an exception dialog when these columns are deleted.

Table 8–1 Validation Rules for Constraint Definitions

Constraint Type	Rules
Mapped Foreign Keys (Foreign keys used in mapping to identify between primary source and secondary source) (Excluding generated FKs)	**1.** If foreign key is deleted or foreign key points to another table, then the foreign key usage be deleted. The user has an entry in the mapping property sheet to re-create the foreign key usage.
Mapped Unique Keys (Unique keys used in mapping to identify between primary source and secondary source) (Excluding generated FKs)	**1.** If the unique key is deleted, then the foreign key usage should be deleted. The user has an entry in the mapping property sheet to re-create the foreign key usage.

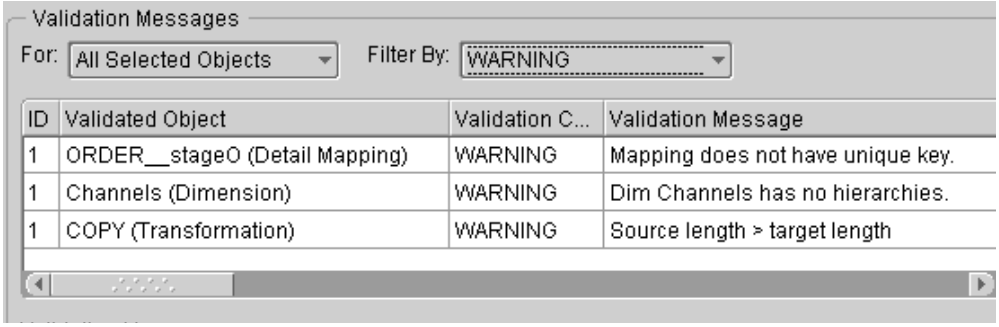
Valid and Invalid Definitions

During the validation process, Builder assigns a Warning or Error code to each malformed expression within an individual definition. A data type mismatch rates an Error code but a receiving field whose length is less than its source field rates a Warning code.

Valid Definition

A definition is *Valid* if the validation process returns no Error codes. A valid definition can include one or more Warning codes.

For example, the definitions for the following objects are all marked as valid even though they contain warning codes.

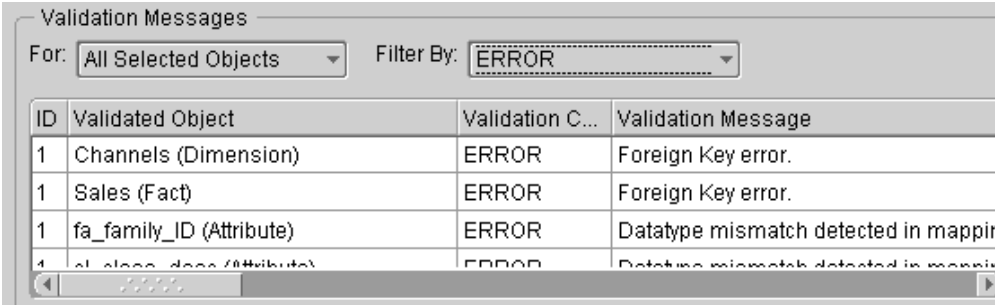


For each warning code, you can choose whether to correct the definition or leave it as it is. A warning code does not prevent you from generating and deploying the definition.

Invalid Definition

A definition is *Invalid* if the validation process returns at least one Error code. The definition can also include one or more Warning codes.

The definitions for the following objects are all invalid because their definitions contain at least one error code.



Validate a Set of Definitions

You can validate a single definition, a set of definitions, or all the definitions stored in a warehouse module.

To validate a set of definitions, open the warehouse module's physical tree, select a set of definitions, and then click the Validate button located on the toolbar. You can select single or multiple objects:

- A single definition

  Select the definition's name from the navigation tree.

- A set of definitions

  Select the names of the definitions from the navigation tree by holding down **Ctrl** while you select each name. You can select several definitions or a complete branch of the physical tree. For example, to validate all dimensions:

  Select **DIMENSIONS > Validate**.

- All definitions

  Select the warehouse module's name.

Builder validates the selected definitions and returns the results in a Validation window (See Figure 8–1 on page 8-20).

The results of a validation are reported in a Validation Results window, a window with three panes: Selected Objects, Validation Messages, and Validation Message.

Selected Objects lists summary information for each object selected for validation. The summary information includes:

- Names of the objects selected for the validation
- Date and time of the last validation
- Status: Valid or Invalid
- Date and time when the object was last updated.

Validation Messages lists individual violations for each object:

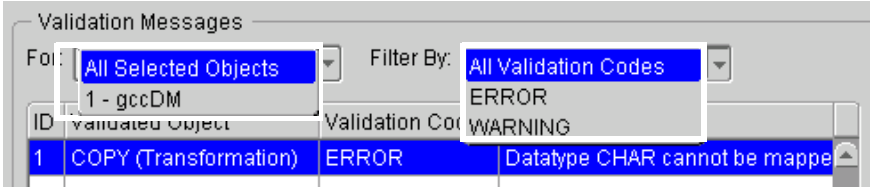
- Names of objects that violate a validation rule
- Severity of the violation (Warning or Error)
- Validation message that describes the nature of the violation
- Suggestion for corrective action
- Validation Message for a selected violation

Select a specific violation and click Details, and Builder returns additional details about the violation.

Validation Message lists individual violations for each object.

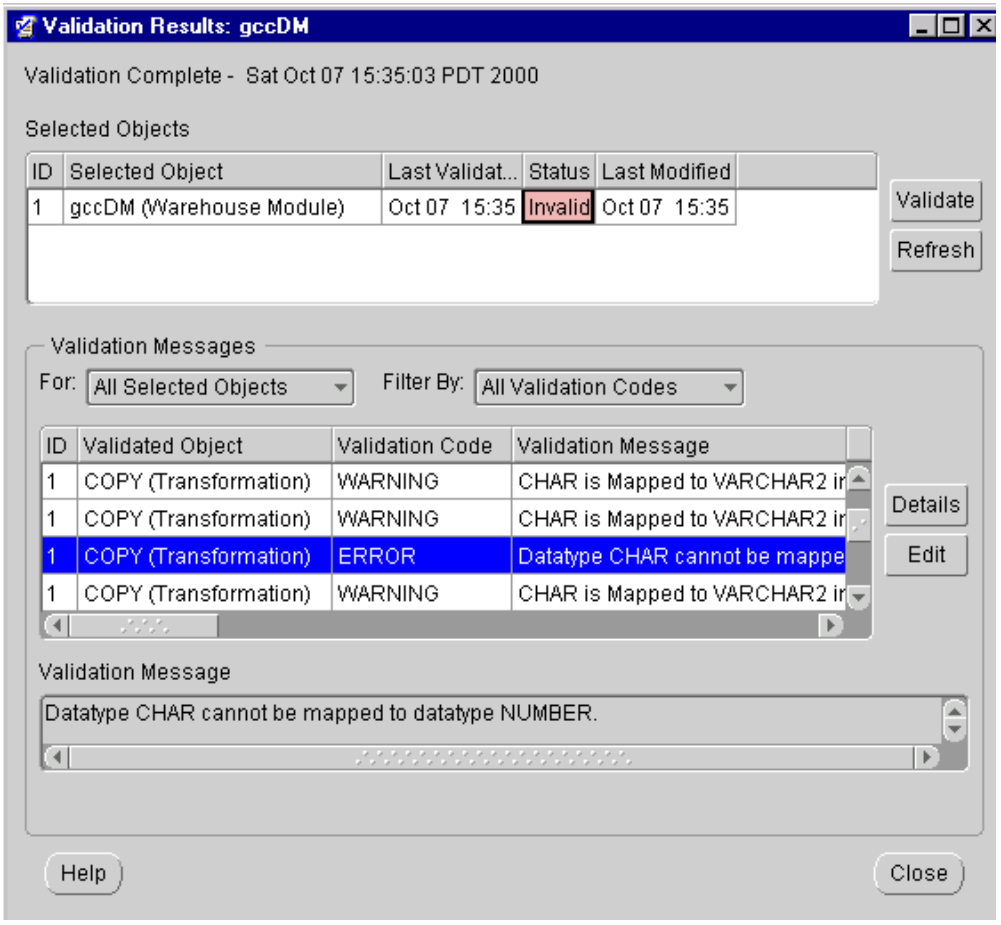
To activate this pane, select an individual violation from the middle pane.

Use the down-arrows to limit the messages to a specific object class or to messages of a specific severity.

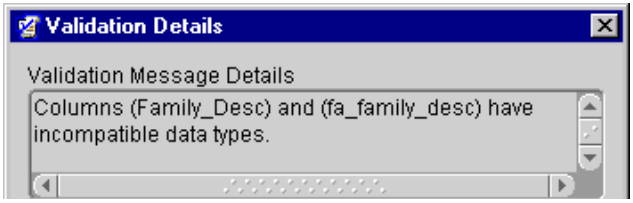


Note: Builder returns a Validation Results window when you validate a definition. To view these results at a later time you can select **View > Validation Messages**. Builder returns the previous results in a window labeled Validation Status.

Figure 8–1 Validation Window



For additional information on a specific violation and a corrective action, click the Edit button on the Validation Status window.



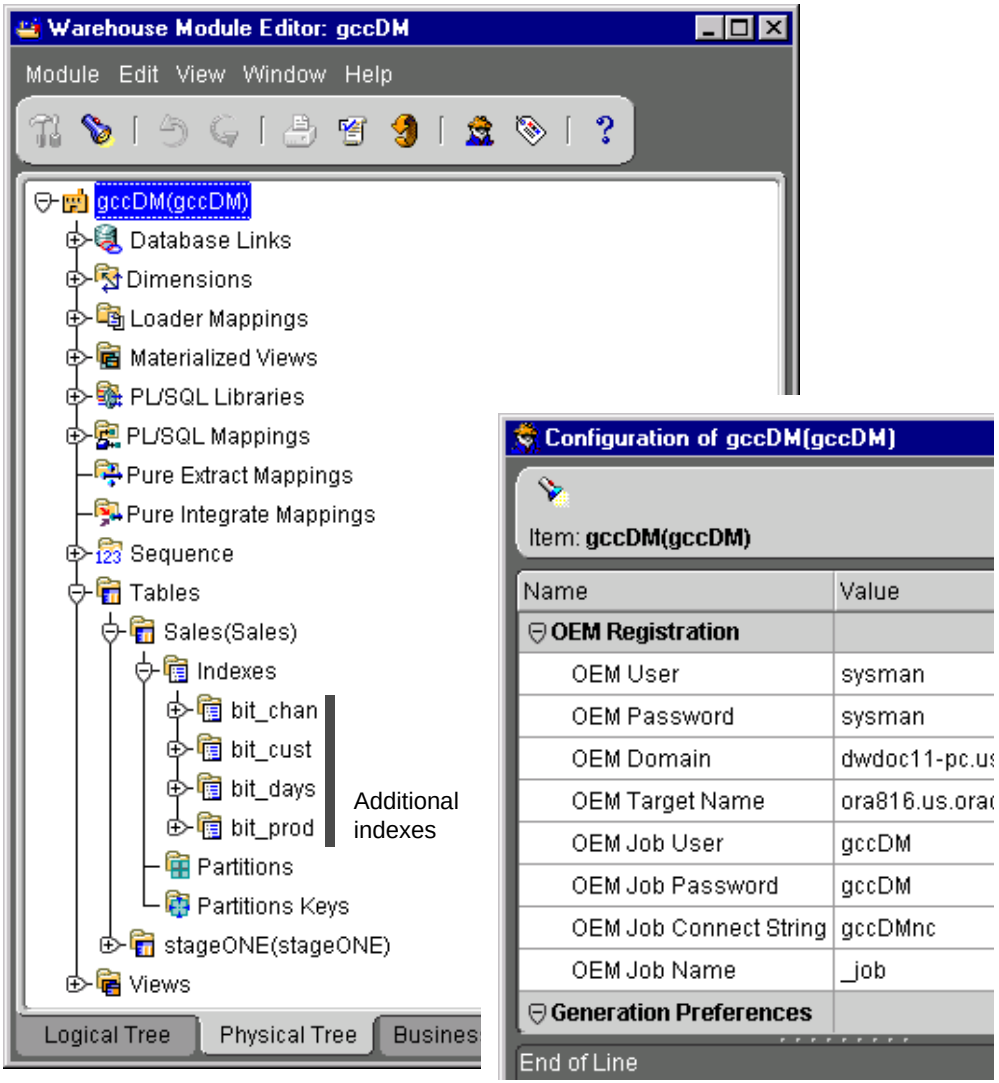
Builder returns the information in a Validation Details panel.

After you run the validation process, resolve all error messages whether they are Error or Warning. Do not generate the scripts until all the messages have been resolved and the problems fixed.

Once you have a validated a set of definitions for a configured instance, you can generate the scripts required to create and populate that instance. During this phase, Builder generates scripts for the selected warehouse module and stores the scripts in the Builder Project. The next chapter shows you how to generate the scripts and deploy the target schema's objects.

Current State of the Project

At this stage, the Builder repository contains a set of definitions which have been configured for the operating environment described in the previous chapter. The configuration parameters for this instance are visible in the physical navigation tree for the *gccDM* warehouse module.



The Next Step

This chapter showed you how to configure a physical instance of the logical model defined by the *gccDM* warehouse module. The example instance was configured to run in the operating environment described in the previous chapter. During the configuration process, four additional bitmap indexes were defined on the Sales table's foreign key reference columns so that Oracle8*i* can boost query performance using the star transformation algorithm.

This chapter also showed you how to validate the set of definitions stored in the *gccDM* warehouse module and how to repair malformed definitions before you generate the set of scripts that create target schema's objects and load the data mart's tables.

Now that a physical instance has been configured and has passed validation, you are ready to generate scripts and use these scripts to deploy the target schema's objects. The next chapter, "Generate and Deploy," shows you how. You complete this task from the physical tree of the warehouse module.

9

Generate and Deploy

This chapter shows you how to generate a set of scripts to create and load the physical instance of the Global Computing Company Data Mart. Builder generates these scripts from the set of configured definitions that reside in the *gccDM* warehouse module. The chapter also shows you how to deploy the generated scripts to a local directory, register the scripts with Oracle Enterprise Manager, and then create all the target schema objects.

This chapter covers the following topics:

- The Basic Steps
- Generate the Scripts
- Deploy the Scripts
- Deploy the Target Objects
- Register the Jobs
- Current State of the Project
- The Next Step

The Basic Steps

The definitions stored in the *gccDM* warehouse module now describe a physical instance of the logical model. You can now generate the scripts required to create and populate that instance.

The generate and deploy phase consists of four major steps:

1. Generate the scripts from the configured definitions
2. Deploy the scripts to the Top Directory
3. Create the target schema's objects using selected scripts
4. Register the scripts with Oracle Enterprise Manager

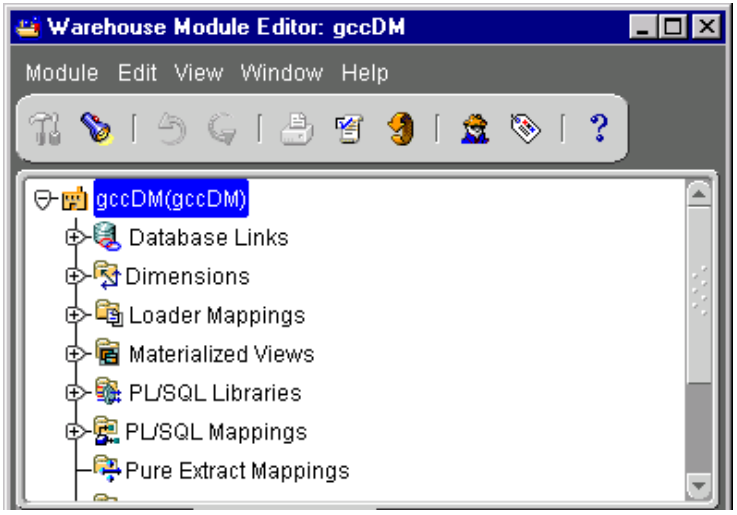
After you create the target schema's objects, you can then publish the contents of the data mart. Part IV of this cookbook covers this step.

Generate the Scripts

This section first shows you how to generate the scripts and then summarizes the actual scripts generated for each object in the warehouse module.

Script Generation

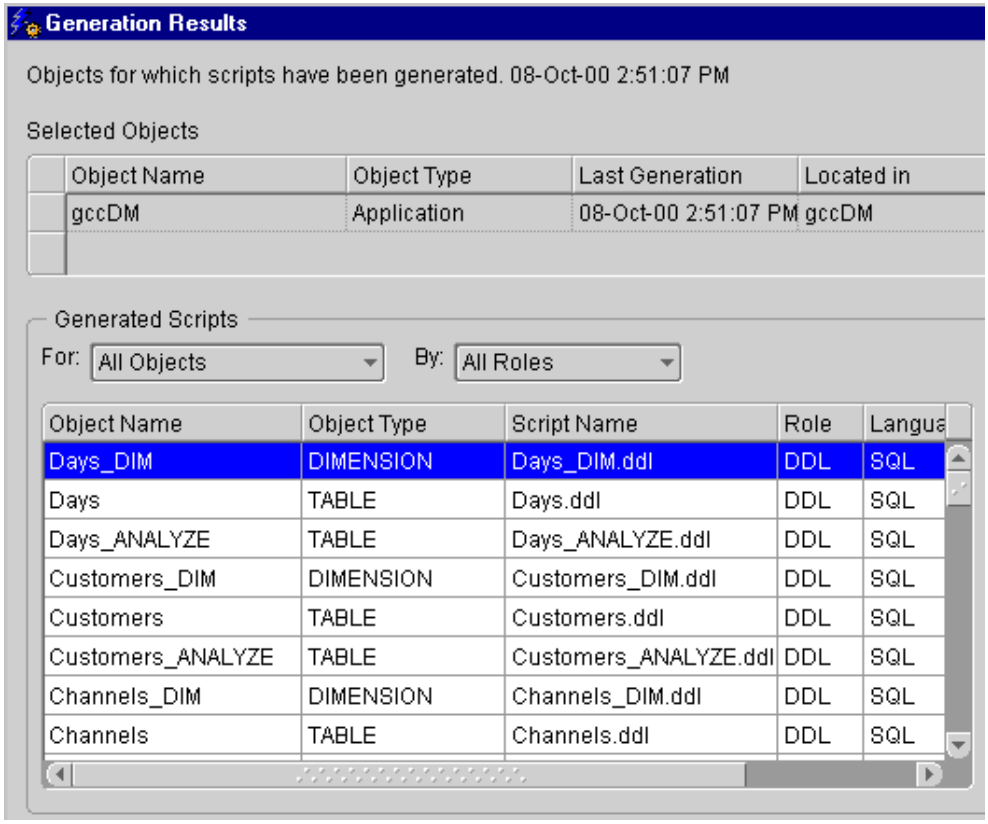
To generate the scripts, open the *gccDM* warehouse module (double-click its name), switch to the physical navigation tree, and select the module's name:



From the Warehouse Module Editor's toolbar:

Select **Module > Generate** …

The editor generates all of the scripts and returns the Generated Scripts window.



Builder generates several kinds of scripts:

- DDL scripts to create database objects
- PL/SQL scripts to create packages that extract, transform and transport source data from databases
- SQL*Loader control files to extract and transport data from file sources
- TCL scripts to schedule and manage jobs (registered with Enterprise Manager)

The following discussion summarizes the actual scripts generated for each object in the warehouse module.

The Generated Scripts

Builder generates a variety of scripts that are used to implement, populate, and run a configured instance of the data mart. The following discussion summarizes the scripts generated for each kind of warehouse object. These summaries correspond with the branches of a warehouse module's physical navigation tree.

Database Links

A data mart pulls data from its database sources using database links. The mechanism is straightforward:

- Builder generates a script for a database link to the source database.
- The database link is deployed *in the target schema*.
- A synonym for each source table is generated.
- The synonyms are deployed *in the target schema*.
- The PL/SQL scripts that pull data from a source database reference the source tables using the deployed synonyms.

For example, Builder generates the following script to create a database link to the *GCCAPPS* source.

```
ALTER SESSION ENABLE PARALLEL DDL;
CREATE DATABASE LINK GCCAPPS
 CONNECT TO gccapps IDENTIFIED BY manager USING 'gccnc';
```

This database link is deployed to the physical instance by executing the script.

Whenever a detail mapping references a table within that database source, Builder generates a script to create a synonym for the table that includes the database link:

```
CREATE SYNONYM SHIPTO FOR GCCAPPS.SHIPTO@GCCAPPS;
-- Installed Module: GCCAPPS

CREATE SYNONYM ACCOUNT FOR GCCAPPS.ACCOUNT@GCCAPPS;
-- Installed Module: GCCAPPS

CREATE SYNONYM SEGMENTS FOR GCCAPPS.SEGMENTS@GCCAPPS;
-- Installed Module: GCCAPPSCREATE SYNONYM SHIPTO FOR SHIPTO@GCCAPPS;
```

In this case, Builder generated three scripts: one for the primary source (*SHIPTO*) and one for each secondary (referenced) source (*ACCOUNT* and *SEGMENTS*).

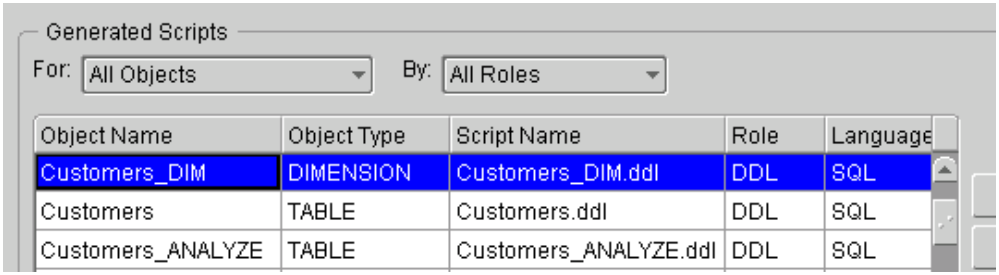
After the database link and the synonyms are deployed to the physical instance, PL/SQL packages can be deployed to the instance and then used to pull data from the source database, transform the data as required, and then load it.

Dimensions

Builder can generate multiple ddl scripts for each dimension depending on whether indexes are configured. The scripts can include a ddl script for:

- The underlying table and its configured partitions
- The hierarchies and levels defined on the table
- The indexes and index partitions configured for the table

For example, Builder generates the following scripts for the Customers dimension:



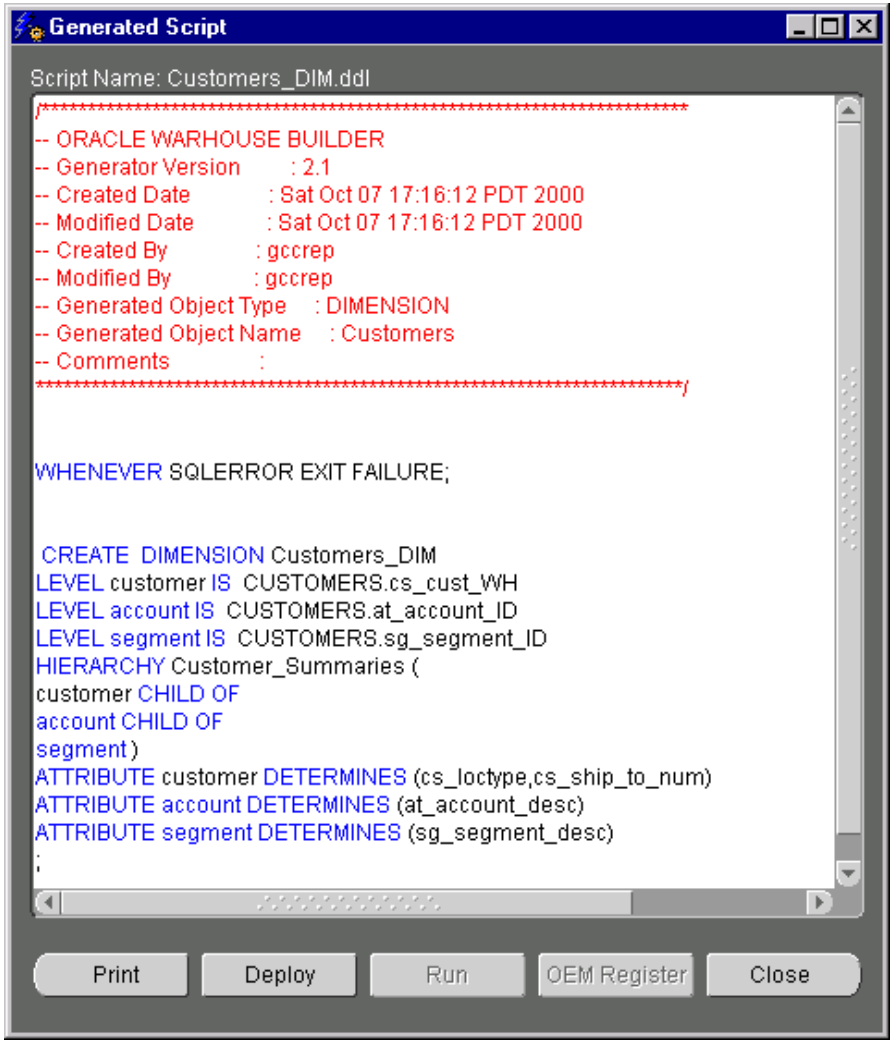
The Generated Results window includes scripts that create the dimension object (levels and hierarchies defined on those levels) and the dimension table. The third script analyzes the indexes defined on the table.

To view a generated script, select the object's name and click the **View** button. The module editor returns the generated code in another panel. See Figure 9–1, "Generated Script" on page 9-6.

The text displayed in the Generated Script panel is read-only: Builder does not allow you to edit the generated scripts. The policy is that Builder manages all changes. Thus, to edit the script, you must open the property sheet for the dimension, edit the values in the property sheet, make any necessary configuration changes, and then re-generate the DDL script.

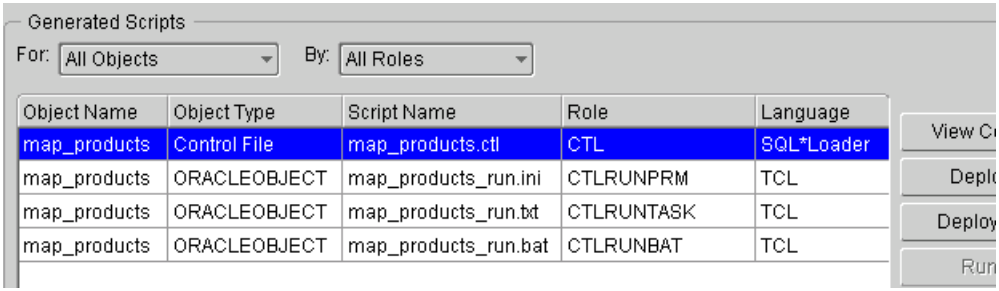
This deliberate normalization eliminates dependency errors that result from isolated changes. For example, if the name of a dimension's primary key is changed, then every occurrence of that name in foreign key reference constraints must also be changed. This latter task, when carried out manually, is tedious and error prone.

Figure 9–1 Generated Script



Loader Mappings

When you map a file to database table, Builder generates a control file for the SQL*Loader and three TCL scripts you can register with Oracle Enterprise Manager. See Chapter 10, "Populate the Data Mart" for additional information on using the TCL scripts to schedule jobs using Enterprise Manager.



You can deploy the control file and the TCL scripts to the Top Directory for the warehouse module and register the TCL scripts with Oracle Enterprise Manager.

Materialized Views

Builder can generate multiple ddl scripts for each materialized view depending on whether indexes are configured for the view. The scripts can include ddl scripts for:

- The materialized view and its configured partitions.
- The foreign key reference constraints.
- The indexes and index partitions configured for the table.
- The partitions configured for the table.

After generating the ddl, Builder returns the Generated Results window. You can view the generated code and deploy the materialized view from this window.

Notes: When you generate scripts for a view

- Builder does not generate code for the view if a query script in its definition has an error. Builder does not check the query script for errors when you define the view.
- If a view references dimensions, then you must deploy the dimensions before you deploy the view.

PL/SQL Libraries

Builder generates a single ddl script for each library defined by the warehouse module. The script is for a package and its body.

PL/SQL Mappings

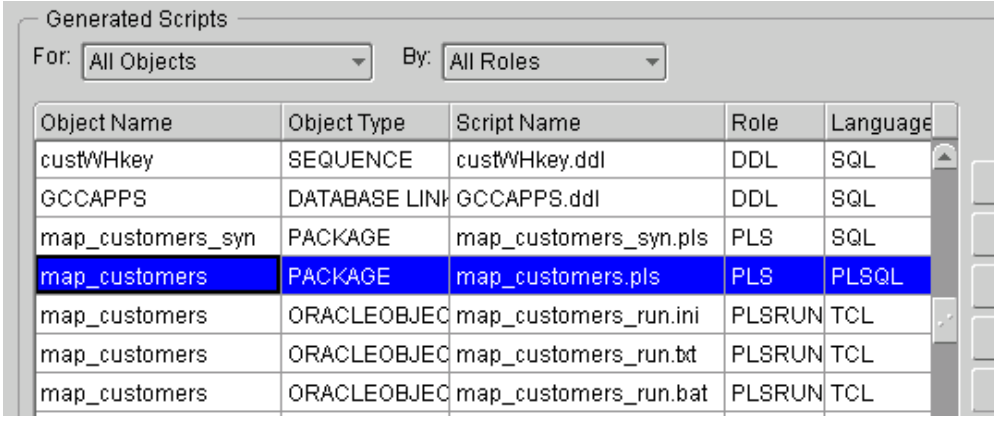
Builder generates multiple scripts to implement each PL/SQL mapping:

PLS scripts create the PL/SQL packages that pull data from database sources, perform transformation operations, and load data into the physical instance of the data warehouse.

DDL scripts create synonyms for each table that serves as a source. Builder generates a synonym for the primary source table and one for each secondary source table. These create statements are all included in a single DDL script.

TCL scripts that are registered with Oracle Enterprise Manager and Oracle Workflow to support job scheduling as well as capture and log audit and error information.

The Generated Scripts window below illustrates the set of scripts generated to implement the mapping for the *Customers* dimension.



Sequences

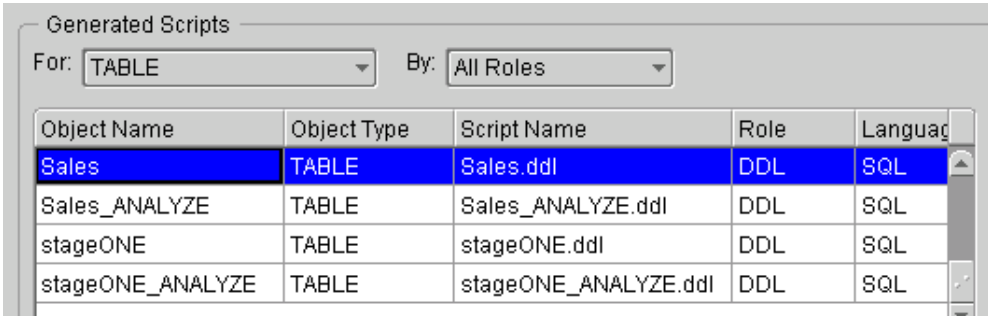
Builder generates a single DDL script for each sequence defined by the warehouse module. Each script creates a sequence object in the physical instance.

Tables

Builder generates multiple DDL scripts for each table which include DDL scripts for:

- The table and its configured partitions
- Constraints on the table
- The indexes and index partitions configured for the table
- The partitions configured for the table.

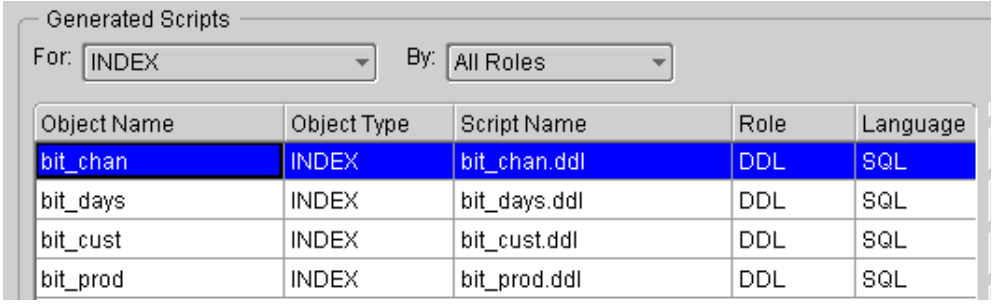
After generating the DDL, Builder returns the Generated Results window. You can view the generated code and deploy the dimensions from this window.



Note: Fact tables always reference dimensions over foreign key reference columns. You must deploy the referenced dimensions to the physical instance before you deploy the fact table.

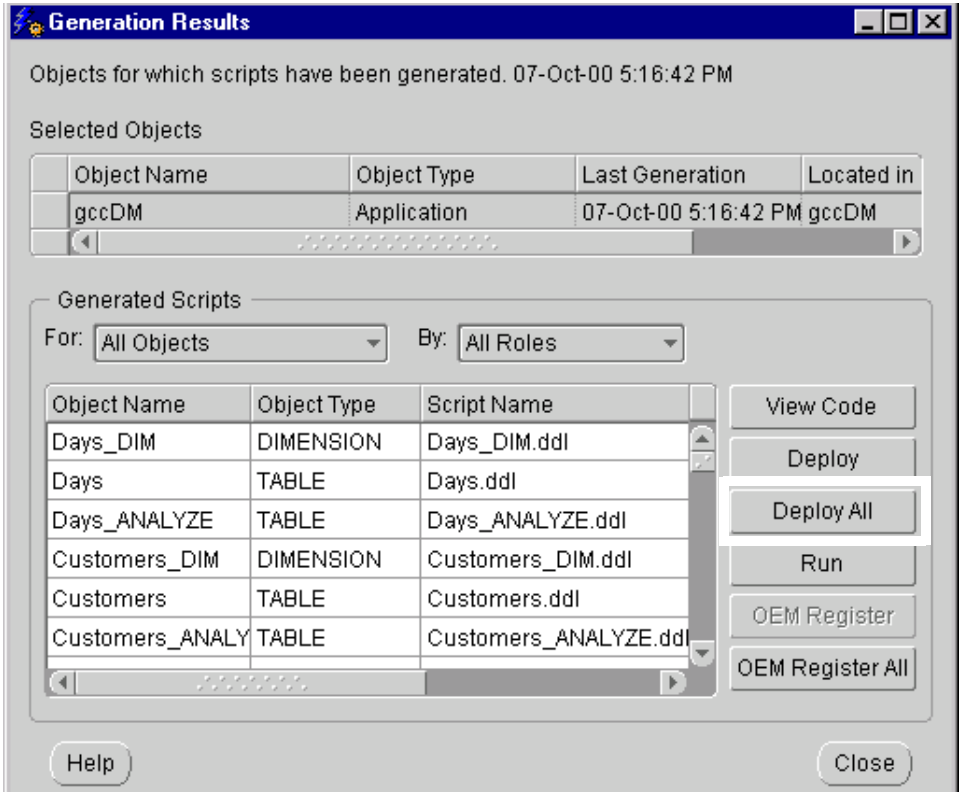
Configured Indexes

Builder generates a DDL script for each index defined during the configuration process. The following additional bitmap indexes were defined on the Sales table.



Deploy the Scripts

After you generate the scripts, you can complete the post-generation tasks using the buttons on the Generated Scripts window. This section uses the **Deploy All** button to copy the scripts from the warehouse module, which resides in the Builder repository, and store them in a local directory named the "Top Directory."



The Top Directory

When you install Builder, the Oracle Installer creates a Top Directory in the same Oracle8*i* home that hosts the Builder client. This is the default Top Directory.

For the case study's operating environment, the directory's name is:

D:\Oracle\Ora81\owb\codegen\

You must deploy all the scripts to this directory, and if you modify a definition and regenerate the scripts, you must also redeploy the scripts to this directory.

The Top Directory is divided into several subdirectories, and Builder writes generated scripts to subdirectories that correspond with the script's language:

Subdirectory	Contents
CTL	SQL*Loader control files that extract and transform data from file sources and load that data into target objects.
DDL	Scripts that create objects in the target schema including dimensions, fact tables, and materialized views.
PLS	PL/SQL scripts to create packages that extract and transform data from database sources and load that data into target tables.
TCL	Scripts that are registered with Oracle Enterprise Manager and used to initially load and periodically refresh the data mart

Deploy All the Scripts

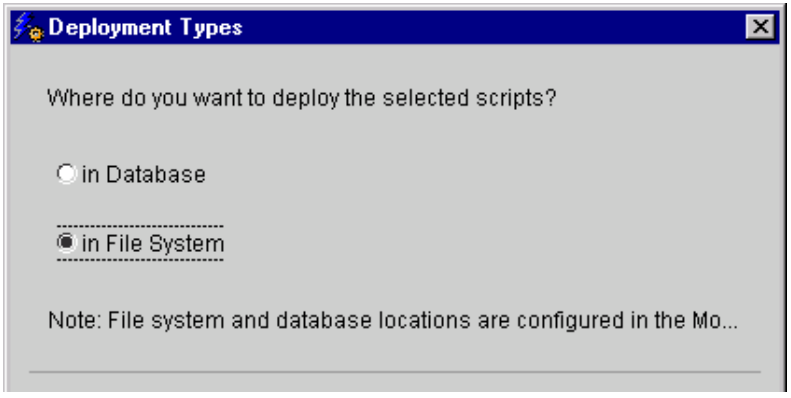
To deploy all the scripts to the Top Directory:

1. Display the Generated Scripts panel.

   Builder displays this panel following completion of the script generation process. To display the panel at a later time, open the warehouse module, switch to its physical navigation tree, click **View** on the editor's toolbar and then select **Generated Scripts**.

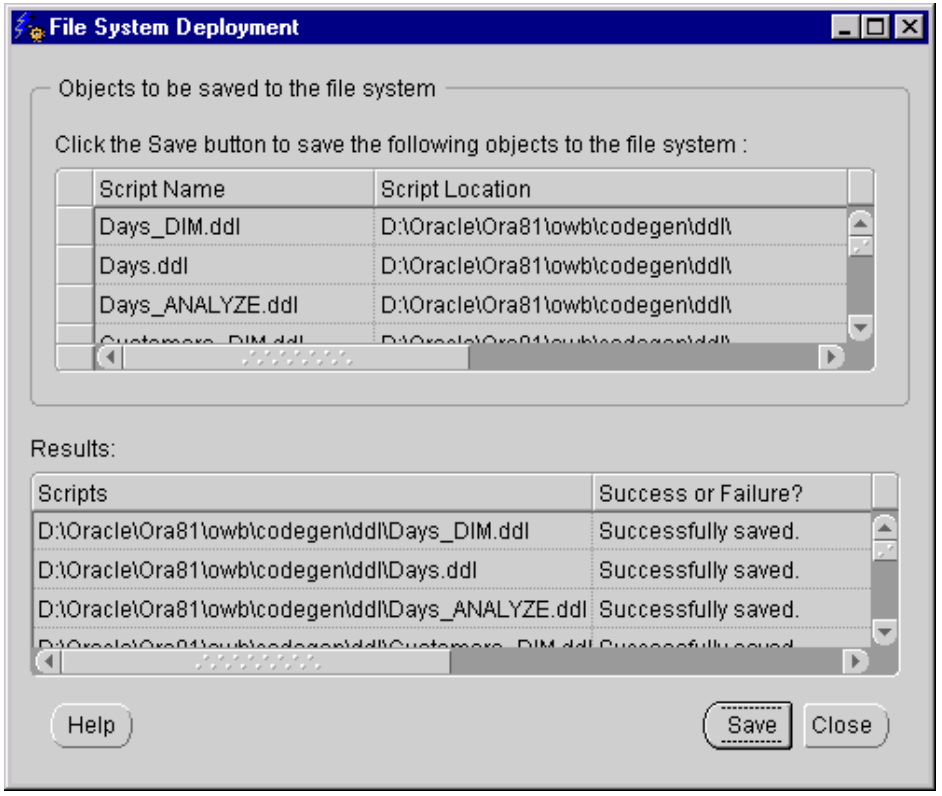
2. Click **Deploy All**.

   The editor returns the Deployment Types panel.



3. Select **in File System** and then click **OK**.

The editor returns the File System Deployment panel which summarizes all the scripts to be deployed and their destinations in the Top Directory.



Before you click the Save button, verify the Top Directory's name and the destination of the individual scripts.

4. Click **Save**.

   Builder returns a progress panel that continuously reports the percentage of scripts saved. When the operation is completed, Builder returns the individual status of each script saved (See the example above).

5. Verify that each script has been successfully saved and then click **Close**.

Deploy the Target Objects

After you deploy all the scripts to the Top Directory, you can deploy all the database objects to the target schema.

When you create database objects, you must create them in a strict sequence to insure that all object dependencies are satisfied. For example, the dimension tables referenced by a fact table must be created before the fact table is created so that the foreign key references made by fact tables can be resolved. Similarly, the detail tables and dimensions referenced by a materialized view must be created before the view is created.

The **Deploy All** button manages these object dependencies, and this section uses this button to create all the database objects in one fell swoop.

Deploy All the Objects

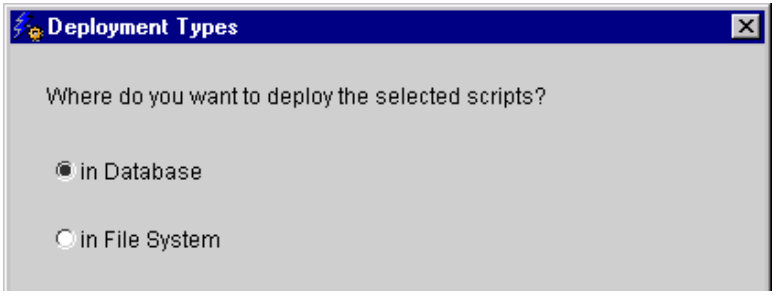
To deploy all the database objects to the target schema:

1. Display the Generated Scripts panel.

   If this panel is not visible, click **View** on the Warehouse Module Editor's toolbar and then select **Generated Scripts**.

2. Click **Deploy All**.

   The editor returns a the Deployment Types panel.

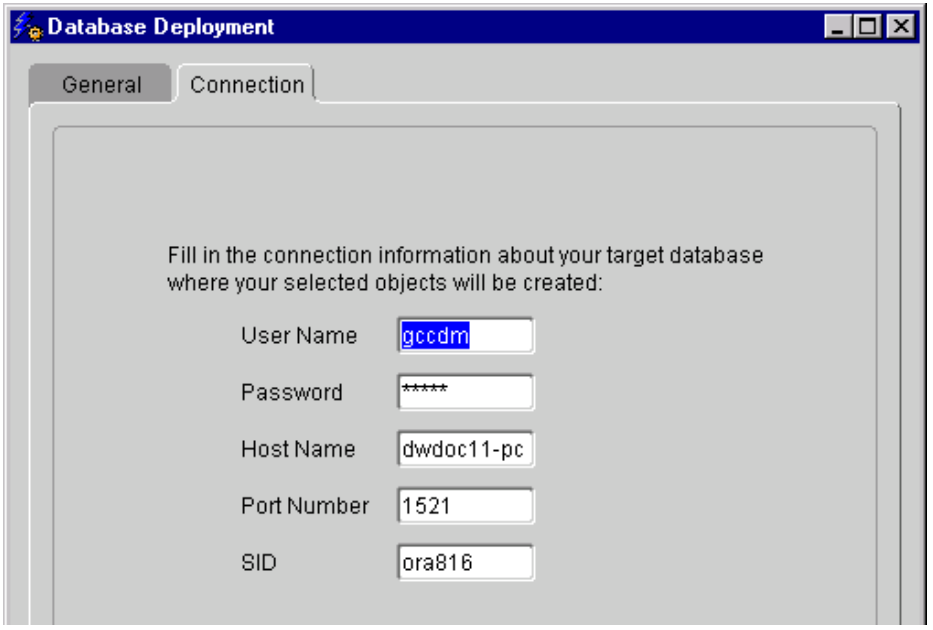


3. Select **in Database** and then click **OK**.

   The editor returns the Database Deployment panel which has two pages indexed by the General and Connection tabs. The General page summarizes the objects to be created in the target schema and the names of individual scripts used to create these objects. The Connection page solicits connection information for the target schema.

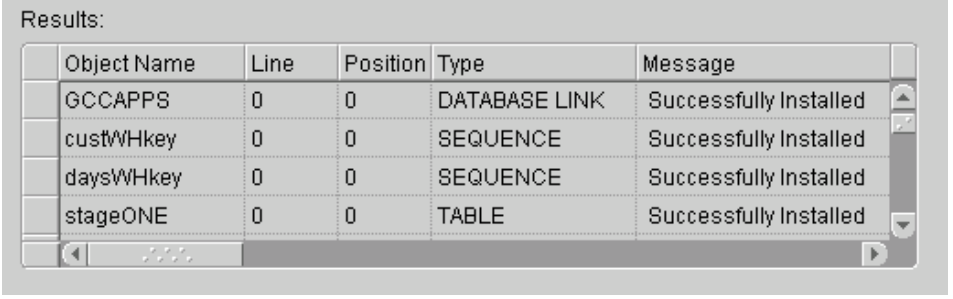
4. Click **Connection**.

   The editor returns a the Connections page.



5. Enter the solicited information and then click **Create**.

   The editor returns a panel that continuously reports the progress of the deployment and on completion returns the status of the operation in the Database Deployment panel (General tab).



   All the database objects now exist in the target schema. The final deployment step stores the job scripts in the Oracle Enterprise Manager Job Library.

Register the Jobs

This final deployment step registers and stores job scripts in the Oracle Enterprise Manager Job Library. After you complete this step, you can schedule and run these scripts using Enterprise Manager. The concluding Part of the cookbook shows you how to schedule and run these jobs.

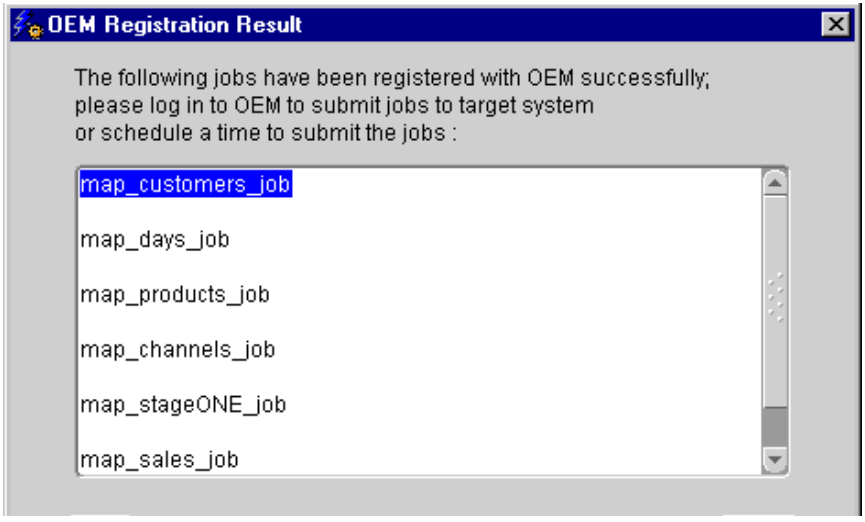
To register the jobs with Enterprise Manager:

1. Display the Generated Scripts panel.

   If this panel is not visible, click **View** on the Warehouse Module Editor's toolbar and then select **Generated Scripts**.

2. Click **OEM Register All**.

   The editor returns a panel that continuously reports the progress of the registration and on completion returns the status of the operation in the OEM Registration Result panel.



   To view these jobs as stored in the Job Library, start the Enterprise Manager Console, open the Job Library, select an individual job, and then click Edit.

Note: The Generation Results window has a **Run** button which can be used to execute the job scripts. This button is intended solely for testing a small sample of data.

Current State of the Project

The Global Computing Company Data Mart now exists but is empty. The jobs required to initially load and periodically refresh the data mart, however, are stored in the Oracle Enterprise Manager Job Library and are ready for scheduling.

The Next Step

This chapter showed you how to generate a set of scripts to create and load the physical instance of the Global Computing Company Data Mart. Builder generated these scripts from the set of configured definitions that reside in the *gccDM* warehouse module. The chapter also showed you how to deploy the generated scripts to the Top Directory, register the scripts with Oracle Enterprise Manager, and then create all the target schema objects.

The physical instance of the data mart now exists and the jobs that initially load and periodically refresh is reside in the Oracle Enterprise Manager Job Library. You are ready now to publish the contents of the data mart to the user community. The concluding part of this cookbook shows you how.

Part IV
Publication

This part of the cookbook, the concluding part, shows you how to publish the contents of the Global Computing Company Data Mart.

This is accomplished in four steps. First, you run the jobs that initially populate the data mart. Second, you create standard reports and make them available to the users. Third, you provide query access to the data mart with query tools. Finally, you set up the processes and scripts necessary to run the data mart on a daily basis.

The concluding chapter of this cookbook also describes how Oracle8*i* provides complete support for a data mart or data warehouse.

This part of the cookbook includes five chapters:

- Chapter 10, "Populate the Data Mart"
- Chapter 11, "Standard Reports"
- Chapter 12, "Ad Hoc Analysis with Discoverer"
- Chapter 13, "Manage the Data Mart"
- Chapter 14, "Preserve Data Mart History"

10

Populate the Data Mart

This chapter shows you how to set up and schedule jobs that initially populate the data mart. You must run these jobs according to a strict sequence and, after each job terminates, you must verify the completion status of each job. If a job terminates abnormally, you must take corrective action before you allow users access to the data mart.

This chapter covers the following topics:

- The Basic Steps
- Initial Population of the Data Mart
- Builder Runtime Audit Viewer
- Complete the Initial Load
- The Next Step

The Basic Steps

The jobs that initially populate the data mart already reside in the Oracle Enterprise Manager Job Library. These jobs are ready to run and you can schedule them immediately. Afterwards, you can use the Builder Runtime Audit Viewer to verify their completion status. Based on their status, you can take whatever corrective action is necessary, and then reschedule selected jobs.

After you initially populate the data mart, you can open the data mart up for business by providing the users with access tools. Chapter 11, "Standard Reports" shows you how to publish standard reports using Oracle Reports and Chapter 12, "Ad Hoc Analysis with Discoverer" shows you how to allow ad hoc access using Oracle Discoverer.

Initial Population of the Data Mart

The jobs that initially populate the case study data mart load the empty tables with a large amount of data. You can run these jobs immediately using Enterprise Manager but you must run them in a strict sequence so that all foreign key references can be satisfied. After each job completes, you should verify its completion status using the Builder Runtime Audit Viewer.

To insure that the jobs are correctly sequenced, you can schedule the jobs in batches, one batch at a time. After one batch completes, you can check the status of its individual jobs, and then schedule jobs in the next batch, and so on.

You can enforce the job sequence manually or define a process to automate the sequencing. The examples in this chapter show you how to manually schedule each job within a batch using Oracle Enterprise Manager.

You can also automate the entire schedule by defining an Oracle Workflow process. The Oracle Warehouse Builder User's guide shows you how. After you define the workflow process, you can then schedule the process rather than the individual jobs. Setting up Oracle Workflow and defining the workflow process is beyond the scope of this cookbook. See the Oracle Warehouse Builder User's guide for additional information and several examples.

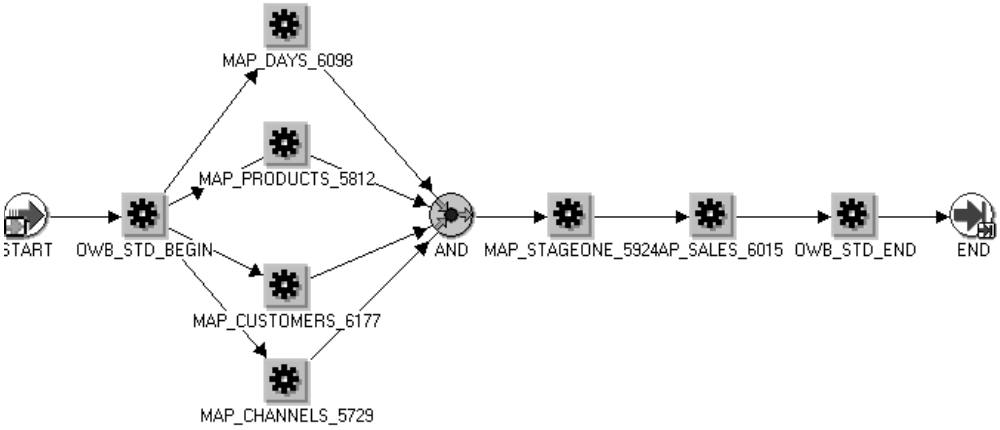
The Job Sequence

The jobs that load the case study schema must be run so that referenced tables are loaded before the tables or materialized views that contain the references to them are loaded. The case study is loaded in four batches.

Table 10-1 Four Batches of Jobs

Batch	Load	Dependencies
1	Dimensions	Completely independent and can be scheduled first.
2	stageONE	Depends on the normal completion of all jobs in the first batch because the staging table references each dimension over natural key columns.
3	Sales	Depends on the normal completion of the second batch of jobs because the *Sales* table references each dimension table over warehouse key columns and it is loaded with data from the *stageONE* table.
4	Materialized Views	Depend on the normal completion of the third batch of jobs because the materialized views are loaded with data from the detail table *Sales* and they both reference dimension tables. You only need to run this batch of jobs when you initially populate the data mart.

The dependency graph below describes an Oracle Workflow process defined with job scripts generated by Builder. This process can automatically manage these job dependencies. See the Oracle Warehouse Builder User's guide for additional information. The following examples show you how to manage the dependencies manually using Enterprise Manager.



Run the Jobs

After the job scripts that initially populate the data mart have been registered, you can run them manually using Oracle Enterprise Manager and then check the completion status of individual jobs using the Builder Repository Assistant. The examples in this section show you how.

Background Services

To schedule the jobs, the Oracle Management Service and the Oracle Intelligent Agent must be running on the data mart host (dwdoc11-pc), otherwise the console will not start. These services can be configured to start automatically or manually. If you have problems registering or scheduling a job, you need to check the status of each of these services.

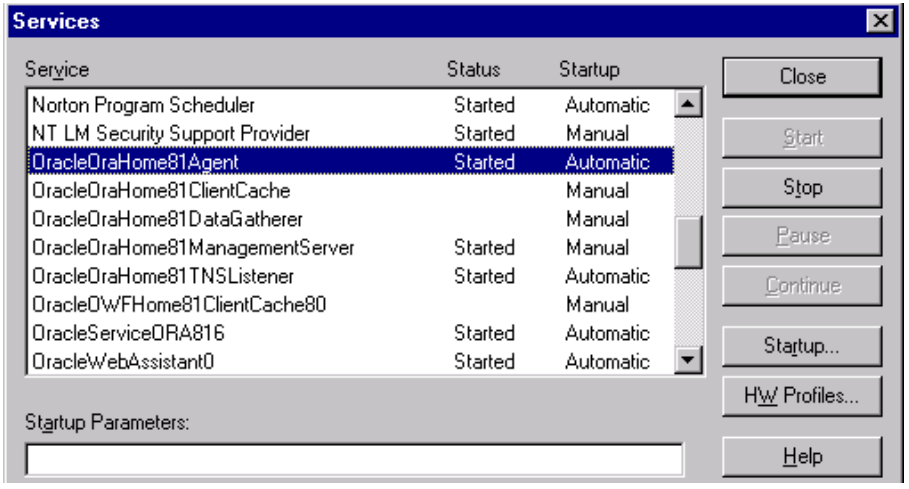
To Manually Start a Service

1. Select **Start > Settings > Control Panel**

   Windows NT opens the Control Panel directory.

2. Double-Click **Services**.

   Windows NT returns the Services window.



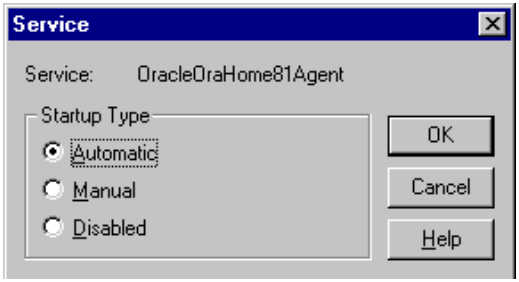
3. Select ***service-name*** **> Start**

   Windows NT starts the selected service.

After Windows NT marks the services as Started, you can proceed to the next step.

Start Service Automatically

1. Select the name of the ***service*** and then click the **Startup** button.



2. Check the **Automatic** button and then Click **OK**.

   The service now starts automatically as part of the Windows NT startup process.

Schedule

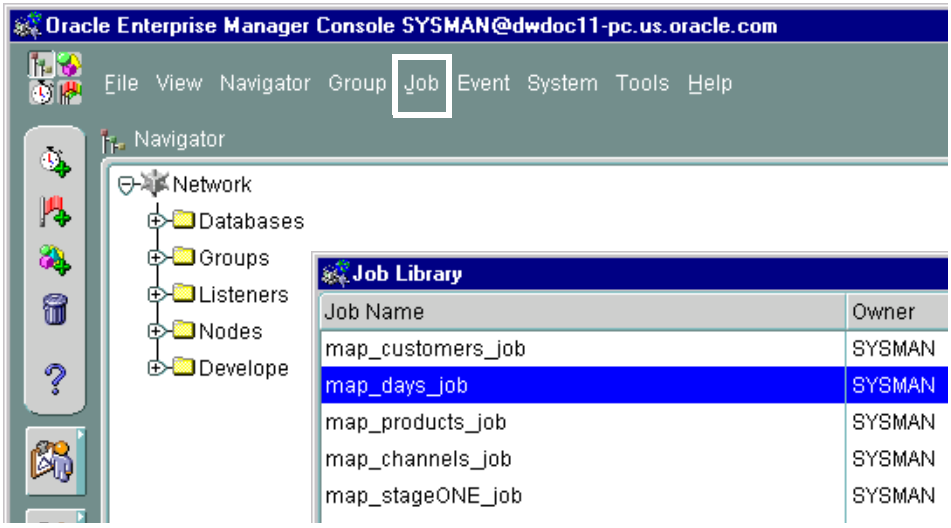
To schedule a job, call the Enterprise Manager console and open the Job Library:

1. **Start > Programs > Oracle > Enterprise Manager > Console**

   Windows NT returns the Enterprise Manager console in a separate window.

2. Click **Job** on the console toolbar and select **Job Library**

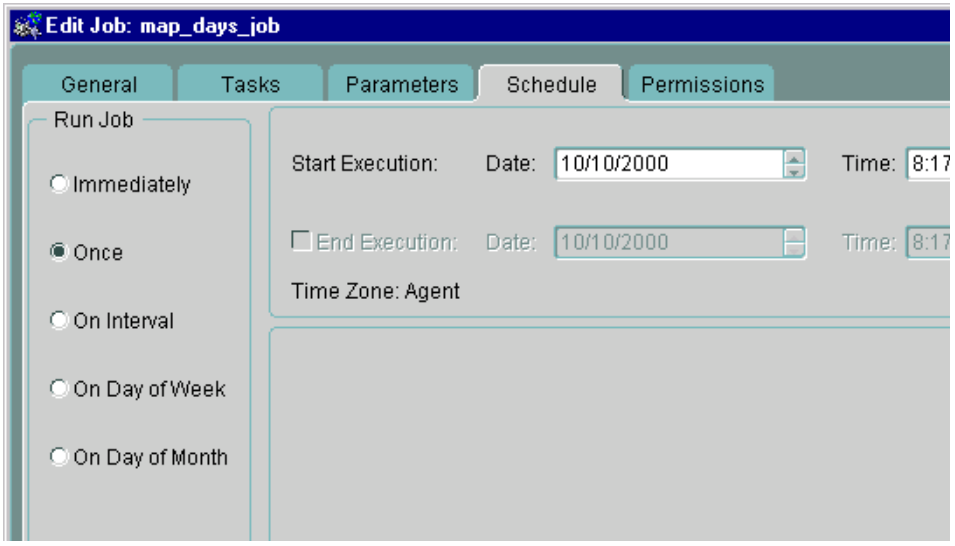
   Enterprise Manager returns the Job Library window.



3. Select the name of a job (***map_days_job***) and click **Edit**.

   Enterprise Manager returns a multi-page Edit Job window.

4. Select the **Schedule** tab.



   Enterprise Manager returns the Schedule page. You can schedule a job to run:

   - Immediately
   - Once
   - On Interval
   - On Day of Week
   - On Day of Month

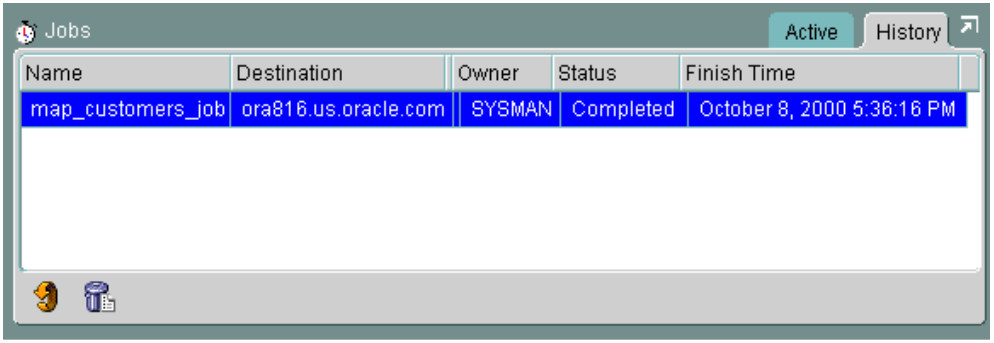
   Specify the schedule (**Once**) and close the window. You can submit the job from this panel now or later.

5. Select the **Submit** button at the bottom of the window and click **OK**.

   Enterprise Manager schedules the one time job according to the date and time entered in the Date and Time boxes. When the job starts, Enterprise Manager places an entry in the Active pane of the console's Job window. This entry remains in this pane until the job completes.

When the job completes its processing, Enterprise Manager places an entry in the History pane of the console's Job window.

You can track the status of a job from the Active pane of the Jobs window, and after the job completes, you can check its status in the History pane of this window.



To obtain a more comprehensive status of the job, use the Runtime Audit Viewer.

Builder Runtime Audit Viewer

The job scripts generated by Builder capture statistics and error information while the job is running and store this information in the set of runtime tables created for the target schema (see cross-reference). The runtime component captures audit and error statistics by job, by tasks within a job, and by detail mappings within tasks.

You can view this information using the Builder Runtime Audit Viewer. To call the Runtime Audit Viewer:

1. Select **Start > Oracle > Oracle Builder Run Time Audit Viewer**

   The Runtime Audit Viewer returns its Logon dialog box.

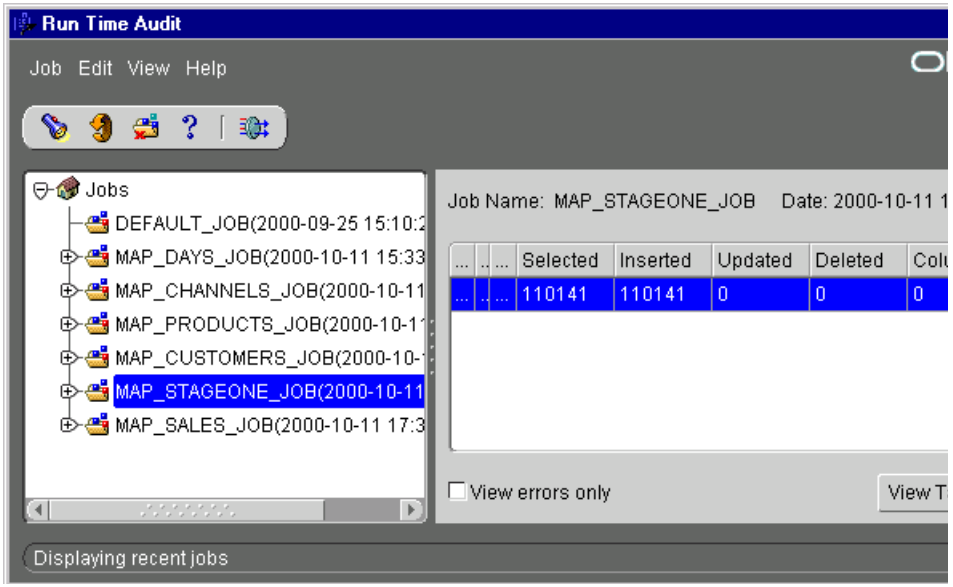
2. Enter log on information:

User name / password for target schema	*GCCdm/gccpword*
Hostname for database instance	*dwdoc11-pc*
Port number	*1521*
Oracle SID	*ora816*

Note: The Oracle8*i* instance must be active for the Viewer to start.

3. Click **OK**.

   The Audit Viewer returns a window with two panes. The left pane contains a navigation tree (*Jobs*) and the right a list of details that correspond with a navigation tree selection.



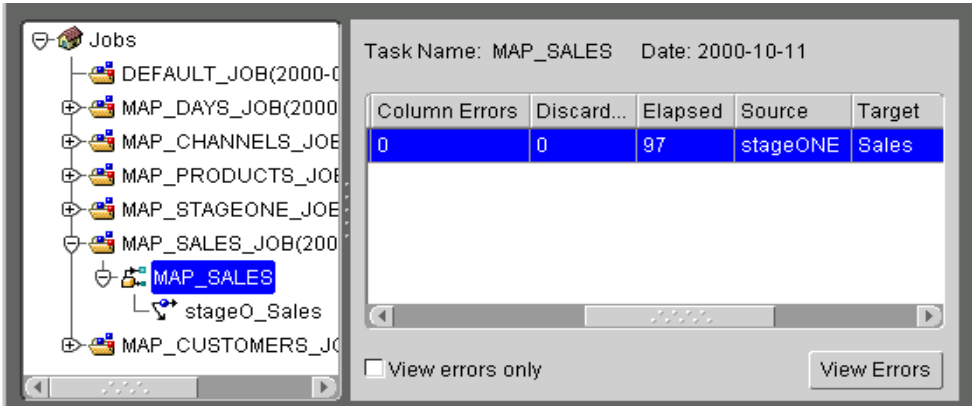
Two Kinds of Jobs

You can run a job directly or schedule the job with Oracle Enterprise Manager. When you run a job directly, the Audit Viewer stores the job's statistics under the "Default Job" node. When you schedule a job with Enterprise Manager, the Audit Viewer stores the job's statistics according to its name.

You can run a job that pulls data from a database source directly using the **Run** button on the Generated Scripts window. To run a job that pulls data from a flat file, however, you must execute the SQL*Loader and use the generated control file. The cookbook examples always schedules the jobs using Enterprise Manager.

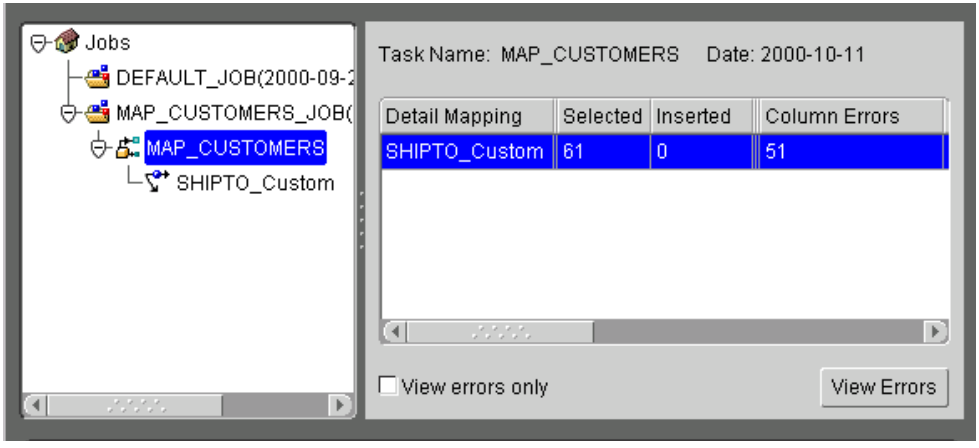
Scheduled Jobs To view the statistics and error messages for a scheduled job, select its name in the left pane. The Audit view returns summary information about the job in the right pane. The example above shows that 110141 rows were selected from a database source and all of them were inserted into the Sales table. The summary also shows that no rows were updated or deleted and no errors occurred.

To display additional statistics, click the plus sign next to the job's name and select the detail mapping's name. The Audit Viewer repeats the summary information, the job's elapsed time, and the names of the primary source and target tables.

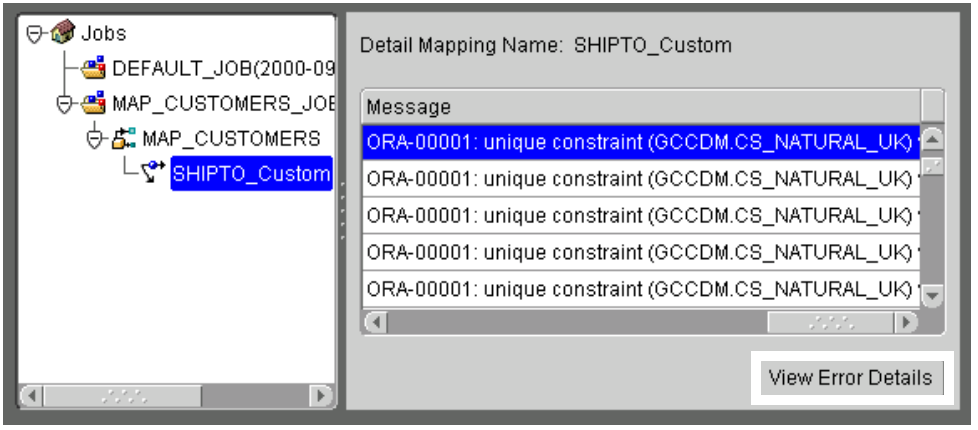


The additional information is specific to a detail mapping. Currently, Builder only supports one detail mapping per task.

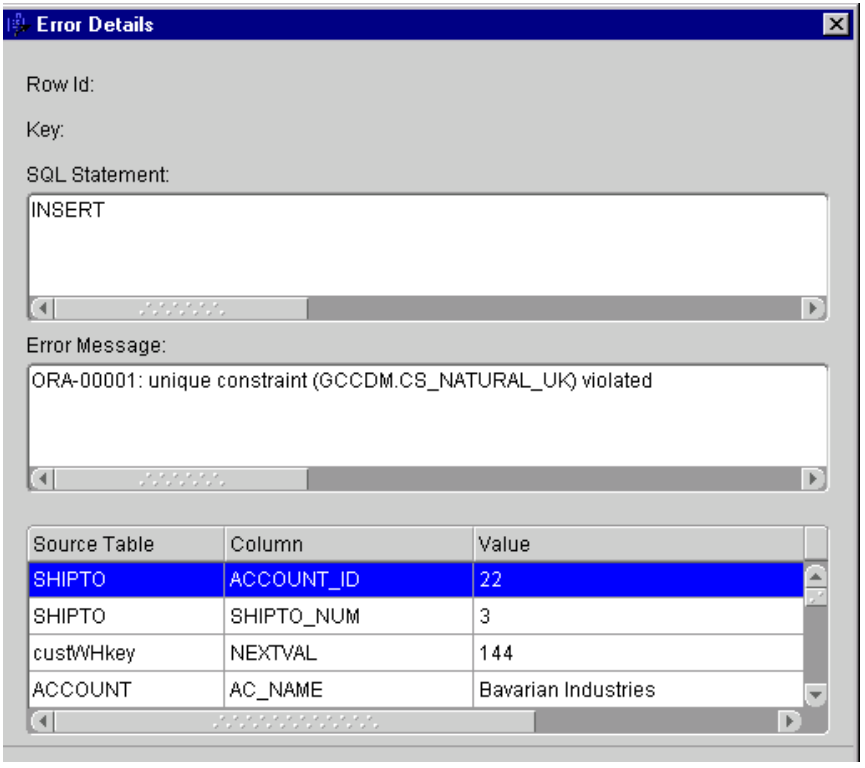
Error Messages Drill down to the next level (*stageO_Sales*) to view any error messages. The current job encountered no errors. The example below does show errors for an erroneous load of the Customers dimension.



The Customers table already contained data when this job was run. Consequently, each time the job attempted to append a row to the table, the Unique Key constraint CS_Natural_UK was violated and the input row was discarded.



For specific information on each error, click **View Error Details**.

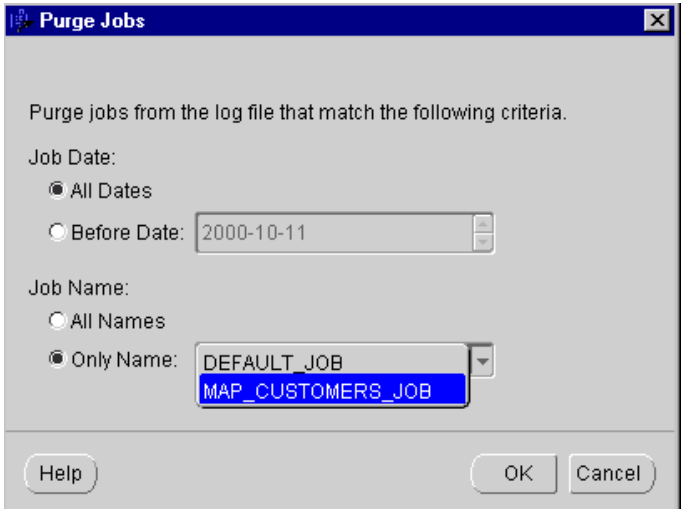


This panel returns all the column values for the offending input record.

Purge Run Time Entries You can purge jobs from the run time tables. From the Runtime Audit Viewer's tool bar:

Select Job > Purge

The Audit Viewer returns a pop-up dialog box.



You can purge jobs according to Job Data or Job Name.

Complete the Initial Load

To complete the initial load you must run all the jobs in the first batch and check the completion status of each job. If a job fails to complete normally, then you must figure out why it failed, correct the job, and run the job again.

After all the jobs in the first batch have completed normally, continue to the next batch. All the jobs in this batch must also complete normally before you move on to the next batch.

Continue this process until run all the jobs in each batch successfully. Once you reach this state, you can move on to the final steps of publication.

The Next Step

This chapter showed you how to set up, schedule, and run jobs that initially populate the data mart. The jobs all completed normally and the data mart is now populated with historical data.

You are now ready to provide users with access tools. Chapter 11, "Standard Reports" shows you how to publish standard reports using Oracle Reports and Chapter 12, "Ad Hoc Analysis with Discoverer" shows you how to allow ad hoc access using Oracle Discoverer. The publication process is not complete, however, until you setup a process to refresh and manage the data mart on a daily basis. That task is the subject of Chapter 13, "Manage the Data Mart."

11

Standard Reports

This chapter shows you how to generate, store, and distribute standard reports using Oracle Reports.

During the requirements phase of this project, the users identified a set of basic reports that were classified as indispensable. That is, if these reports can not be generated and distributed from the contents of the data mart, then the data mart is considered inadequate.

A basic set of indispensable reports were identified during the requirements phase of the data mart project, and the examples in this chapter show you how to implement these reports quickly with Oracle Reports. The examples also show you how to format the reports in a way that meets the users needs.

This chapter covers the following topics:

- The Role of Oracle Reports
- Build a Report
- Update the Report
- Deploy the Report
- The Next Step

The Role of Oracle Reports

Oracle Reports provides an easy-to-use, productive approach to developing and delivering reports in a timely fashion. The reports can range from the very simple to highly complex, and they can be formatted in a number of ways.

Oracle Reports includes the following features:

- A query builder for the graphical specification of SQL statements
- Online tutorials and wizards that guide users through report design
- A what-you-see-is-what-you-get (WYSIWYG) report previewer that lets you edit reports in place
- An integrated chart builder to graphically represent report data
- Comprehensive help

These features dramatically reduce the learning curve for new developers and increase the productivity of experienced staff members, enabling them to quickly and easily create complex sophisticated reports.

Scalable

Oracle Reports takes full advantage of a network computing model which supports a high degree of application scalability. You can use the Oracle Reports Server to easily deploy your applications in a multitier environment.

Oracle Reports allows you to readily capitalize on the promise of the Internet, the intranet, and the extranet. You can quickly give the extended reach of those technologies to your users, customers, and suppliers.

With Oracle Reports, you can confidently disseminate the data mart's information:

- In a secure environment that leverages the security of the database
- Via a Web browser, a thin client, or an ActiveX control
- Via industry-standard formats like HTML, HTML Cascading Style Sheets (CSS), Adobe PDF, PCL, PostScript, and ASCII

The possibilities are considerable.

Components

Oracle Reports includes the following components.

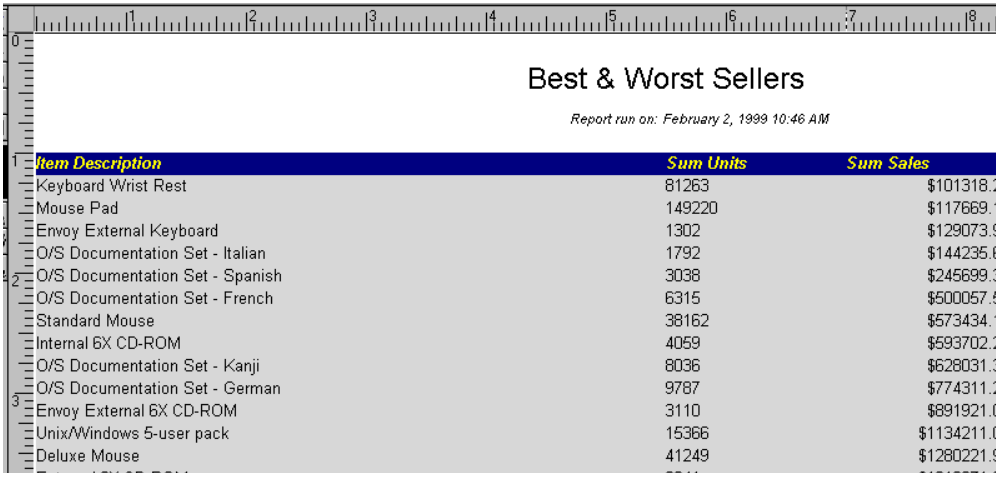
- Report Builder — An easy-to-use, wizard-driven tool for building simple and complex reports which are based on data they retrieve from the data mart
- Chart Wizard — An easy-to-use, wizard-driven tool for Builder charts which are based on data they retrieve from the data mart.
- Reports Runtime — A runtime engine for client/server deployment
- Reports Server — A server that enables multitier deployment of reports

You can also view and run reports generated by Oracle Reports using a web browser. Oracle Reports is tightly integrated with Oracle WebDB which allows additional control over who can access reports and when the reports can be run. Oracle WebDB provides the infrastructure for Internet access to an Oracle database.

Build a Report

The user requirements for the Global Computing Company Data Mart includes a standard report that identifies last quarter's best selling products. A best selling product is one that generates more than 1.3 million dollars in sales for the quarter. This section shows you how to build this report using Oracle Reports.

Figure 11–1 Best and Worst Sellers



Best & Worst Sellers

Report run on: February 2, 1999 10:46 AM

Item Description	Sum Units	Sum Sales
Keyboard Wrist Rest	81263	$101318.
Mouse Pad	149220	$117669.
Envoy External Keyboard	1302	$129073.
O/S Documentation Set - Italian	1792	$144235.
O/S Documentation Set - Spanish	3038	$245699.
O/S Documentation Set - French	6315	$500057.
Standard Mouse	38162	$573434.
Internal 6X CD-ROM	4059	$593702.
O/S Documentation Set - Kanji	8036	$628031.
O/S Documentation Set - German	9787	$774311.
Envoy External 6X CD-ROM	3110	$891921.
Unix/Windows 5-user pack	15366	$1134211.
Deluxe Mouse	41249	$1280221.

The Report Builder

You can build the report using the Report Builder wizard. This takes, more or less, three easy steps:

1. Call the Report wizard and name the report.
2. Define a query to retrieve the data necessary for the report.
3. Format the report.

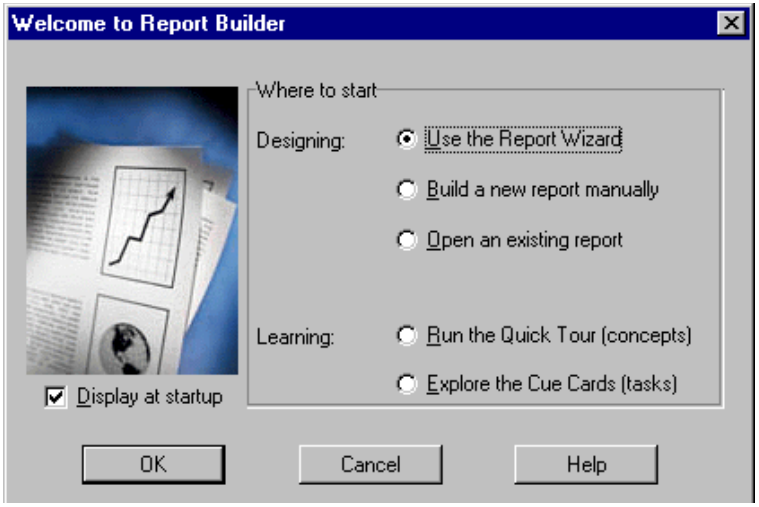
The following example walks you through each of these steps.

Call the Report Builder Wizard

To call the wizard, you must first run Report Builder:

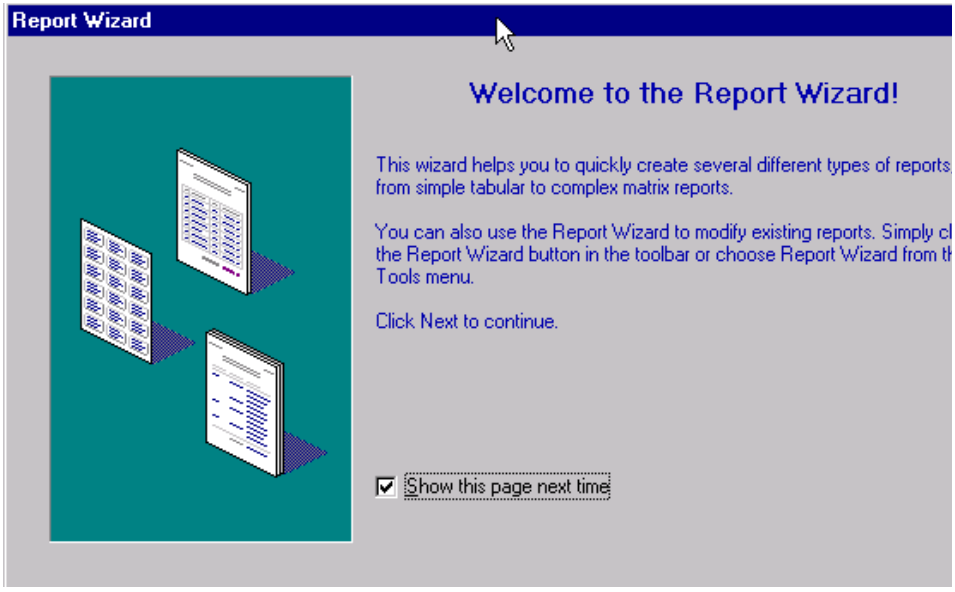
1. Select **Start > Programs > Reports 6i > Report Builder**

   Report Builder returns its welcome window which solicits a type of action: Designing or Learning.



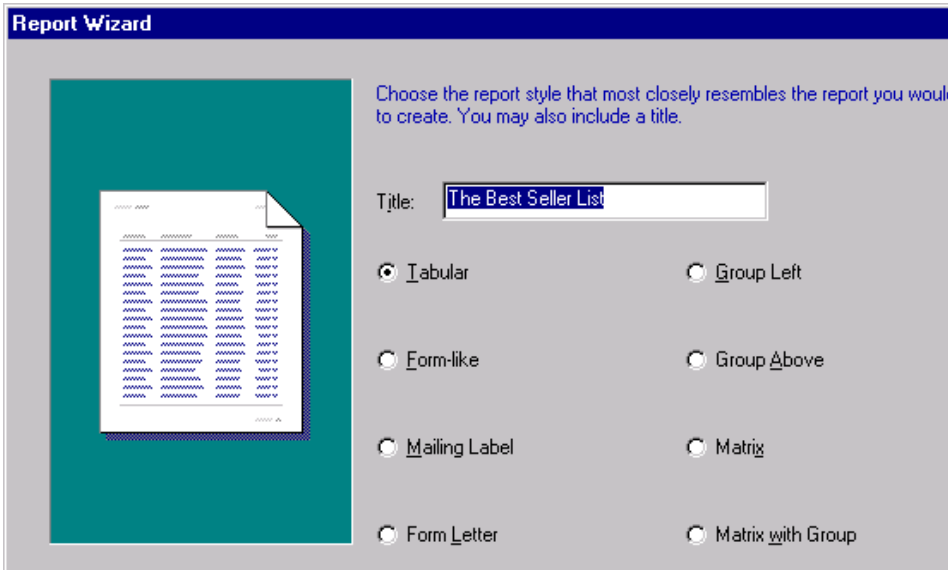
2. Click the button for **Use the Report Wizard** and then click **OK.**

Report Builder returns the welcome page for the Report Wizard.



3. Click **Next**.

The wizard returns its next page which solicits a title and a report format.

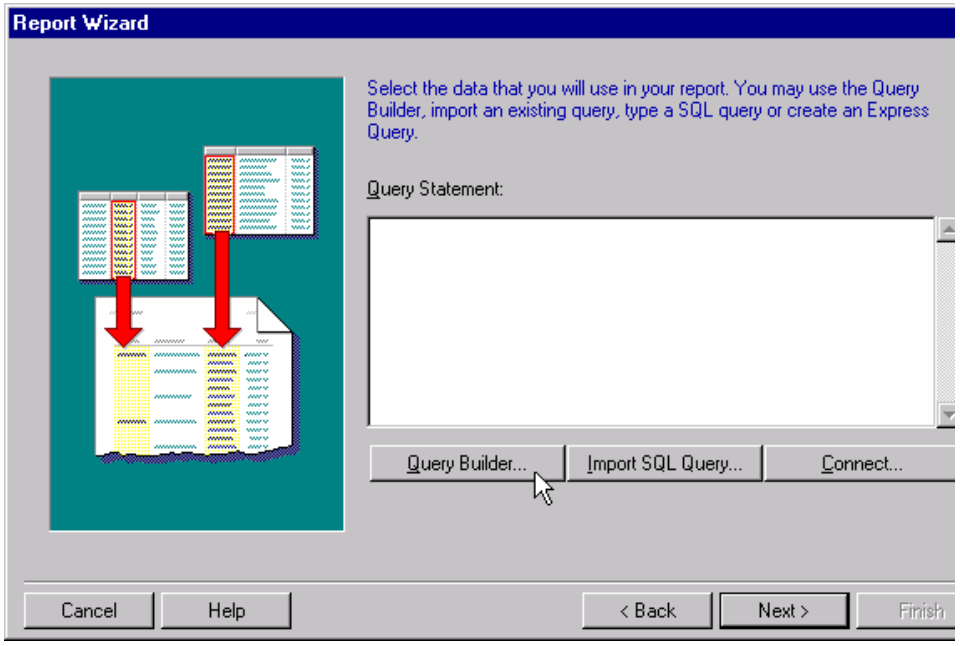


4. Enter the name of the standard report (***The Best Seller List***) and select a general format (***Tabular***). Then, click **Next**.

   The wizard returns the next page which solicits a "Query Type." The report can pull its data from an Oracle8*i* database using an SQL statement or from an Express data store using an Express query.

5. Select **SQL statement** and then click **Next**.

   The wizard returns the next page which solicits the text for an SQL query. Report Builder uses this query to pull data for the report from the data mart.



   You can enter the text for a query manually in the text box, or you can build a query using Query Builder. This example uses the Query Builder.

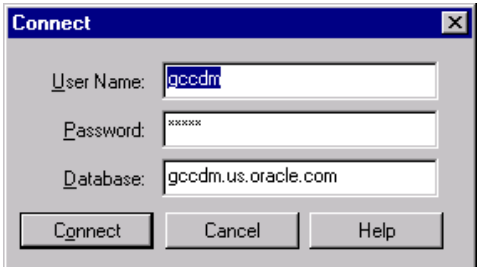
Identify the Data Sources

To build a query, you must specify the tables that contain the raw data and descriptions required for the report. Afterwards, you can define functions that calculate quarterly sales and units, and limit the result set to products whose sales exceed the 1.3 million dollar threshold. Finally, you can rank the values from the top to bottom sellers.

This process begins by calling the Query Builder from the last wizard page.

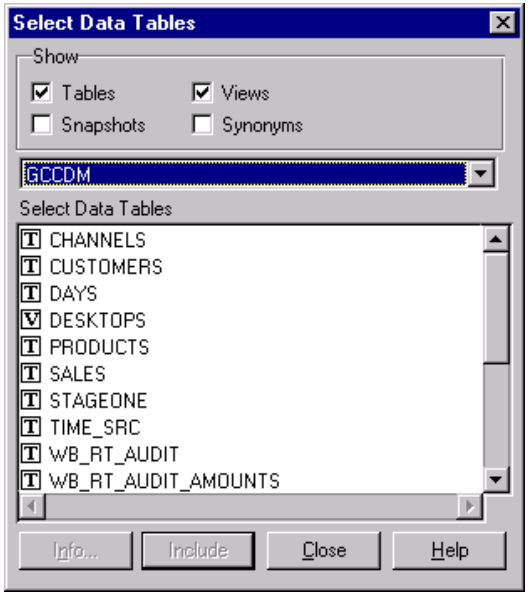
1. Click **Query Builder**.

   The wizard returns a logon panel which solicits a username, password, and connect string for the data mart. You must establish a connection with the data mart before you can build a query with Query Builder.



2. Enter the logon information for the data mart and then click **Connect**.

   Query Builder connects with the data mart, reads its data dictionary, and then returns the Select Tables panel. This panel lists all the tables owned by gccDM.



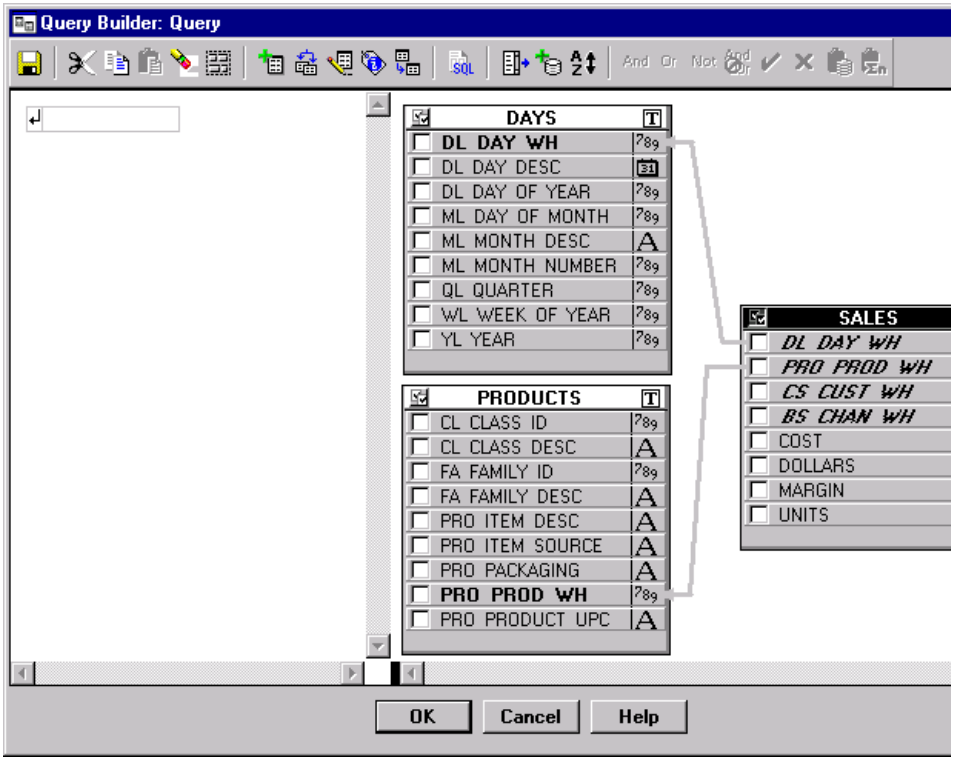
To identify the best selling products during the first quarter of 1998, your query must include the names of three tables:

- *Products* defines all the products sold by the Global Computing Company
- *Days* defines all the available time intervals
- ***Sales*** records the number of units sold, dollar amount of the sale, and cost.

These names will occur in the query's FROM clause.

3. Double-click each table's name and then click **Close**.

When you include a table name, Query Builder represents the table by placing a rectangle in the right pane of the Query Builder window. These rectangles include the names for all of the table's columns.



Query Builder automatically joins the Sales table to the dimension tables over the foreign key reference constraints defined on the Sales table. The lines connecting the tables represent the foreign key references.

Quarterly Aggregates

The report identifies best sellers according to their total sales during a specific quarter. The query also calculates the total units sold of each product during that quarter. You must define columns and calculate totals for both aggregates. The example calculates the totals for each aggregate using the standard SUM function.

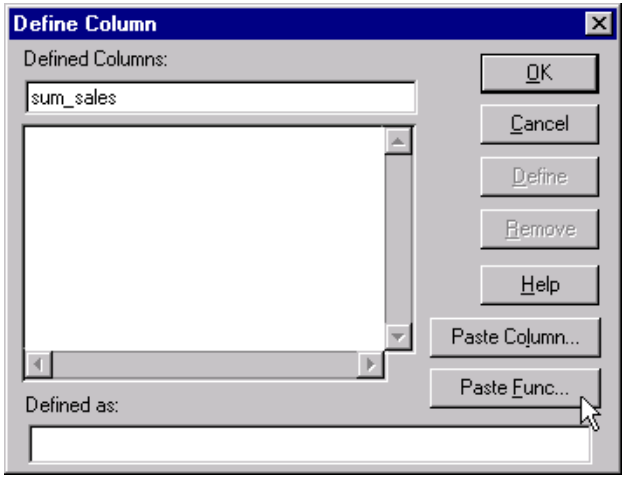
To define columns for the aggregate totals:

1. Click toolbar for the **SALES** table.

   Query Builder highlights the toolbar.

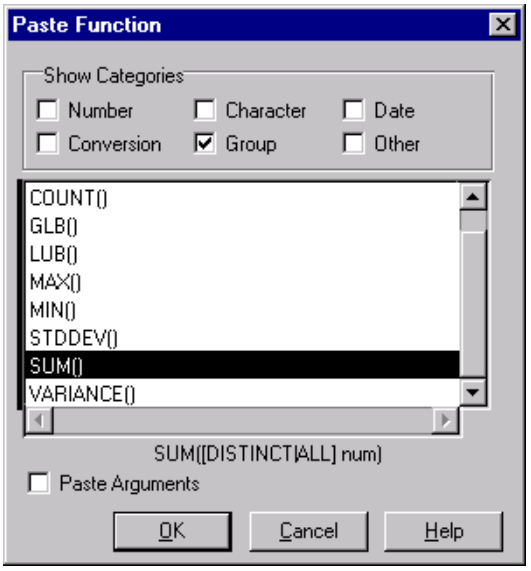
2. Click the **Define Column** icon on the Query Builder toolbar.

   Query Builder returns the Define Column panel.



3. Enter *sum_sales* in the **Defined Columns** field of this panel.
4. Move the cursor to the **Defined as** field and then click **Paste Function**.

Query Builder returns the Paste Function panel.

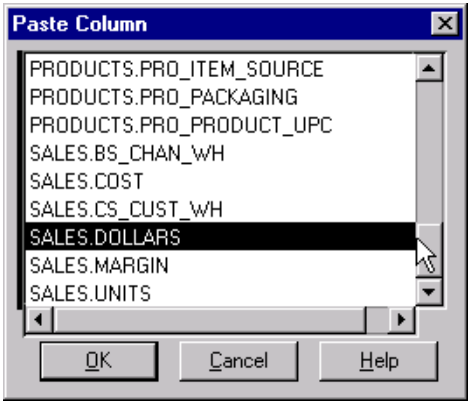


5. Select the **SUM()** aggregation function (Group category) and then click **OK**.

   Query Builder inserts this function in the Define as field.

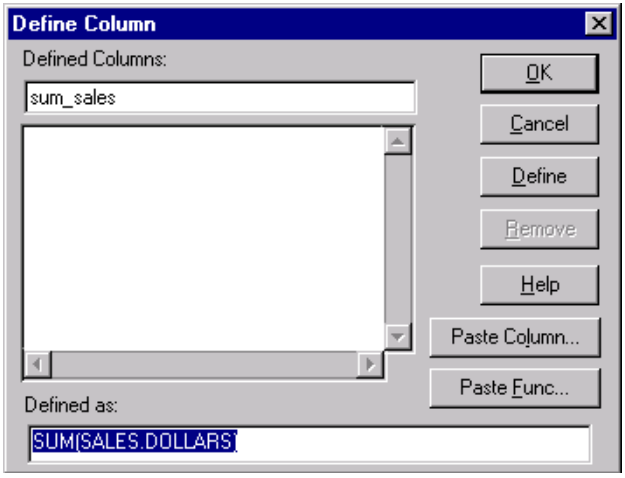
6. Click the **Paste Column** button on the Define Column panel.

   Query Builder returns the Paste Column panel.



7. Select the *Sales.Dollars* column and then click **OK**.

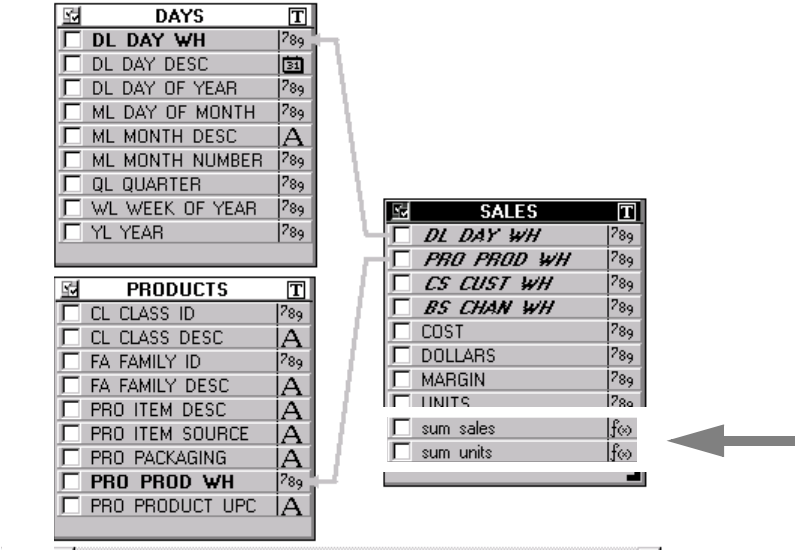
Query Builder inserts the column number in the Defined as field.



If you know the aggregation functions and the Sales table's column names, you can enter the function directly without using the Query Builder.

8. Define a sum_units column using the same procedure. The aggregation function for this column has the form SUM(SALES.UNITS).

Query Builder records the column names as entries in the Sales table.



Identify the Quarter and Sales Threshold

Products are ranked according to their sales performance in the last quarter. Thus, when you build the query, you need to limit the data to that quarter. Also, the report only includes the names of products whose quarterly sales exceed the 1.3 million threshold.

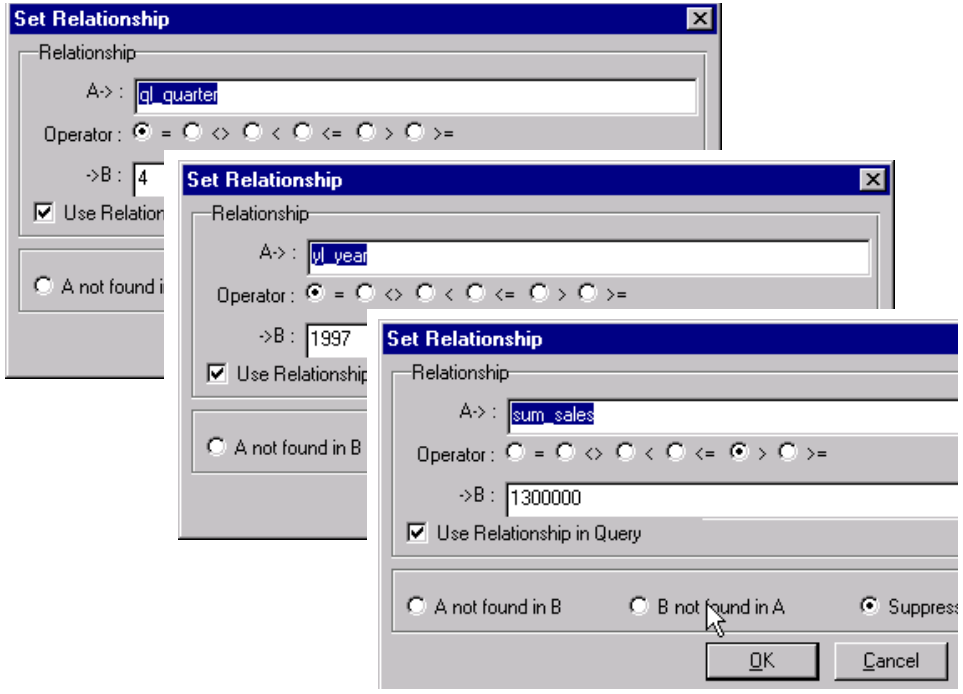
To identify the products which satisfy these conditions, your query must include the following three constraints.

ql_quarter = 1

yl_year = 1997

sales_sum > 1300000

To include these constraints, click the **Set Table Relationships** icon. Query Builder returns the Set Relationship panel. Use this panel to define the constraints.

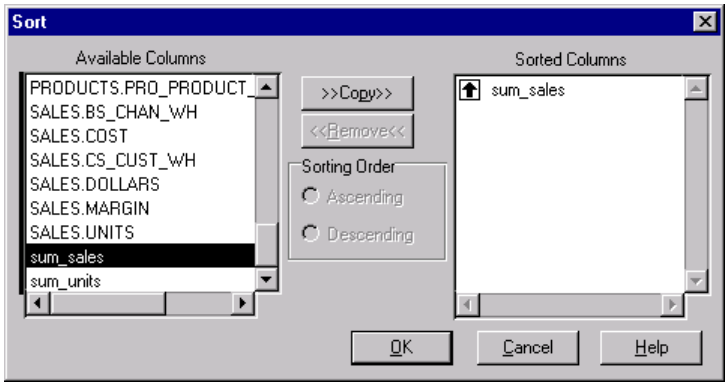


Order the Products Ascending

The report ranks products according to their quarterly sales from highest to lowest.

1. Click the **Sort** button on the Query Builder toolbar.

   Query Builder returns a list of all available column names.



2. Select the *sum_sales* column from the left pane and move it to the right pane.

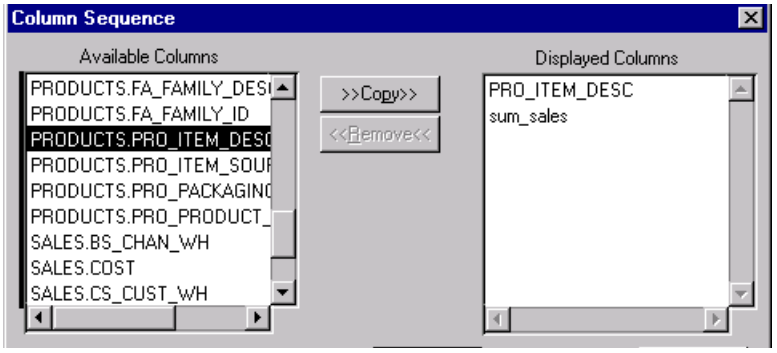
   Query Builder will then sort the products according to their total quarterly sales.

3. Select **Ascending** to order the sales from high to low and then click **OK**.

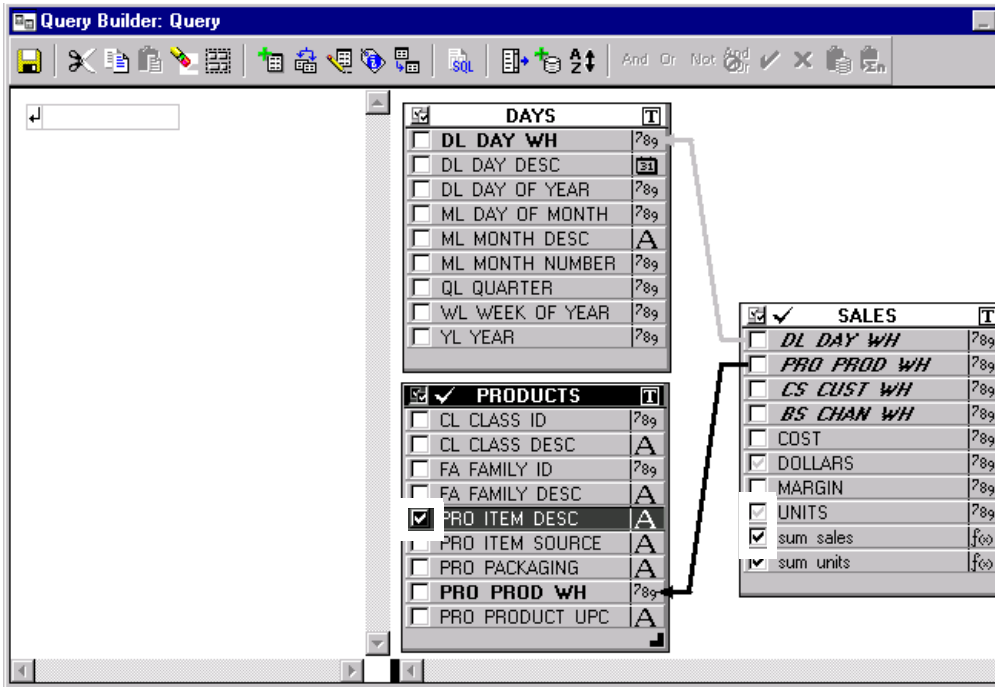
Query Builder will include an Order By clause in the query to sort the result set.

Display Columns

You can now select the columns to display in the report. Click the **Column Sequence** icon on the toolbar. Query Builder returns the Column Sequence panel.

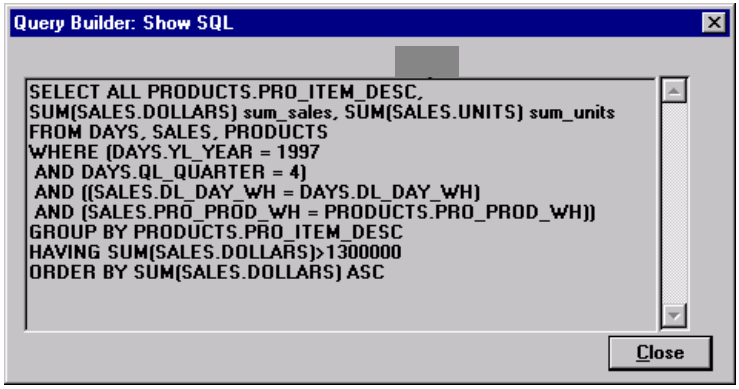


Select the columns that are to occur in the report. The report displays these columns in the order they occur in the Column Sequence panel. To change the sequence, select a column name and move it to another position. After you select the columns, Query Builder checks the box adjacent to their names.



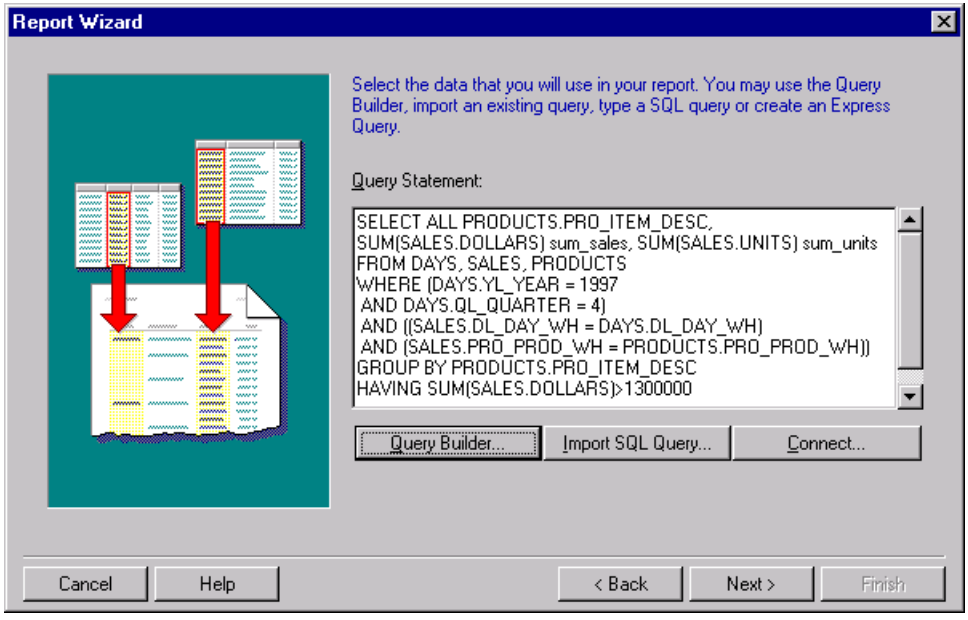
Generated Query Text

To review the query text generated by Query Builder, click the **Show Query** icon.



You can review the text for the query, close the panel, and then click **OK** to exit the Query Builder.

The Report Wizard now contains the text for the generated query in its SQL Query Statement pane.

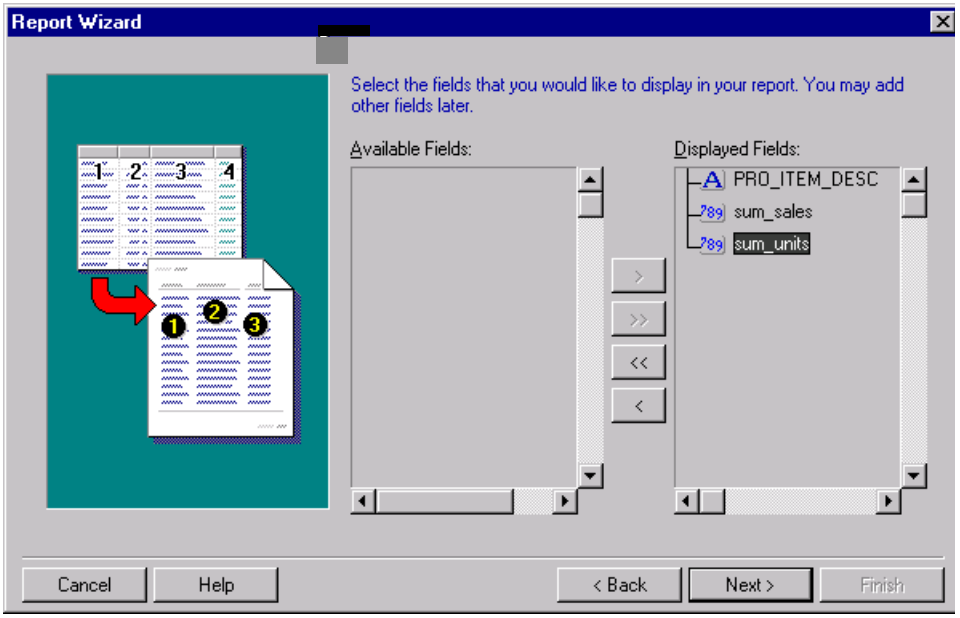


Complete the Report

Now that the generated query is available to pull data from the data mart, you can complete the definition of the report.

1. Click **Next** on the wizard page.

   The wizard returns the page where you choose the columns that are to occur in the report. For the case study, move all three columns from the Available Fields to the Displayed Fields pane.

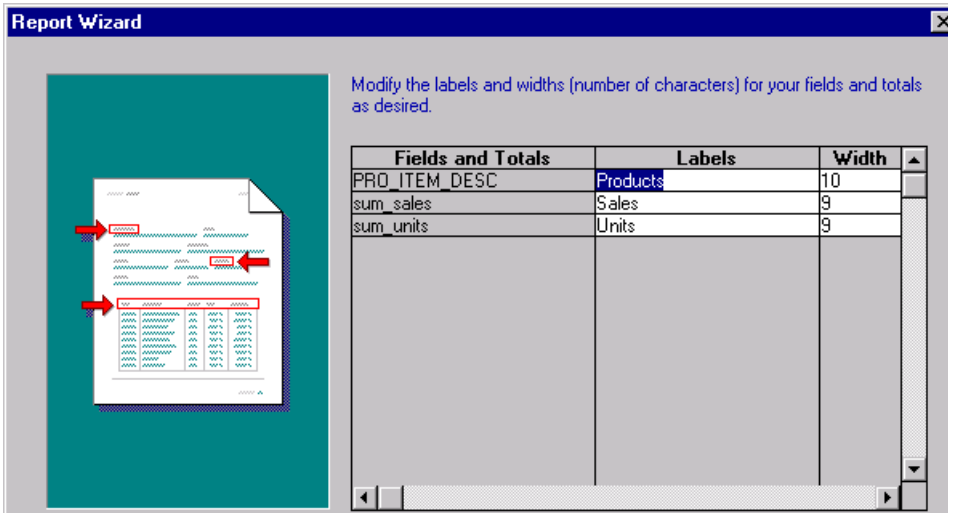


2. Click **Next.**

   The wizard returns a page that allows you to define subtotal calculations. The sum_sales and sum_units are both aggregate totals which are calculated when Oracle8*i* processes the query. You could define the calculations using this page. In that case, they would be performed by the Oracle Reports engine. This would not be efficient because the Reports engine would need to fetch all the detail rows and then perform the calculation. Using the database server in this case is far more efficient.

3. Click **Next** to skip this page.

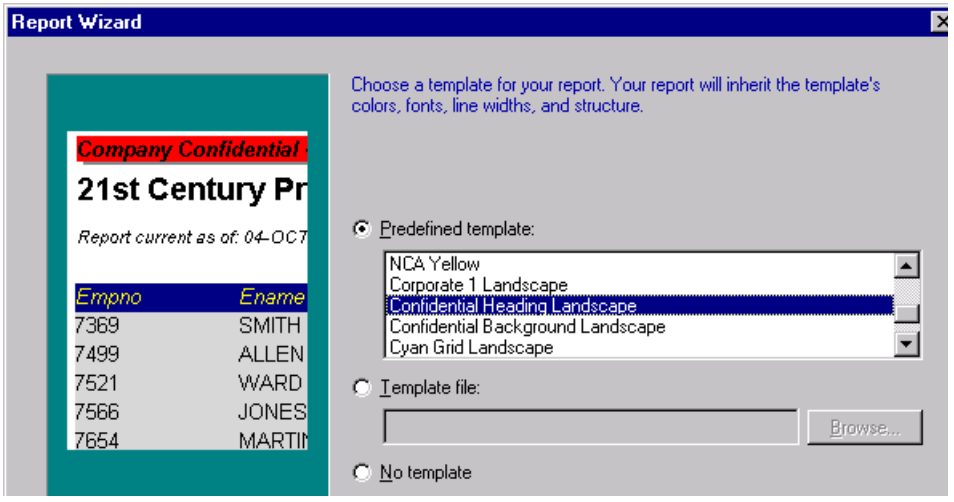
The wizard returns a format page that allows you to change the names of columns as they appear in the report as "Labels."



The example changes each name to an equivalent but shorter label. To change a name or column width, highlight the existing entry and then enter a new one.

4. Click **Next.**

The wizard returns a page that allows you to apply a predefined or custom template to the report.

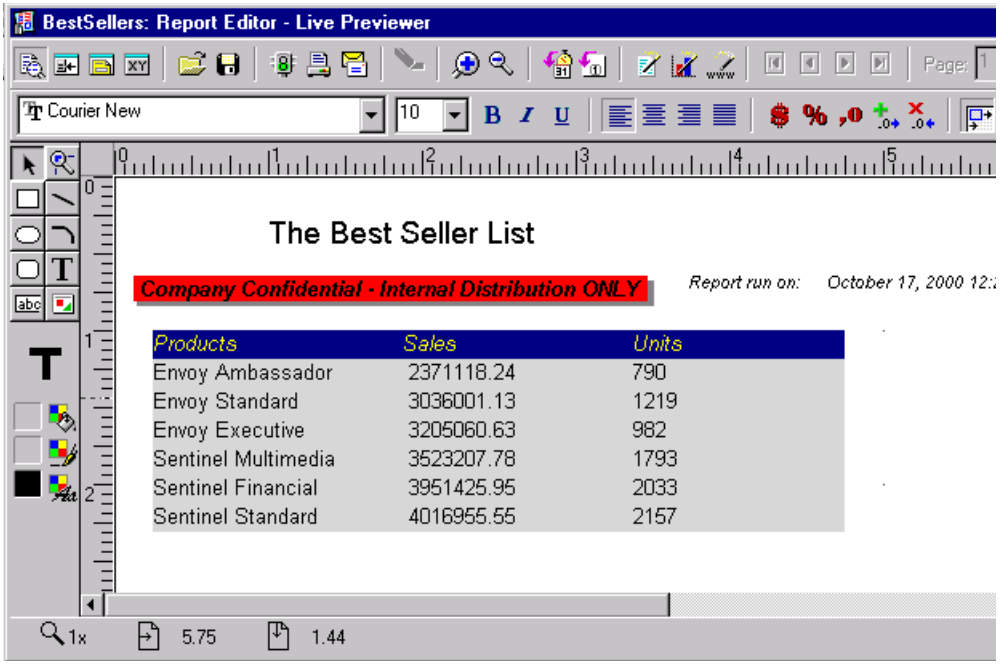


5. Click the name of a Predefined Templates from the list.

The example uses the **Confidential Heading Landscape** template.

6. Click **Next** and then click **Finish**.

The wizard returns the report in the Live Previewer:



Before you proceed, click **Save** from Report Builder's File menu. The case study report is saved as *BestSellers.rdf*.

Format the Report

The Live Previewer is an editor that lets you directly modify the format of the report from a live view of the report itself. For example, you can change the font, the font style (Regular, Italic, Bold, Bold Italic), and change the font size. Or, you can change how numbers are displayed (justification, currency, percent, number of decimal places, and so on). When you make these changes, the Live Previewer immediately applies the change to the report so that you can evaluate the change.

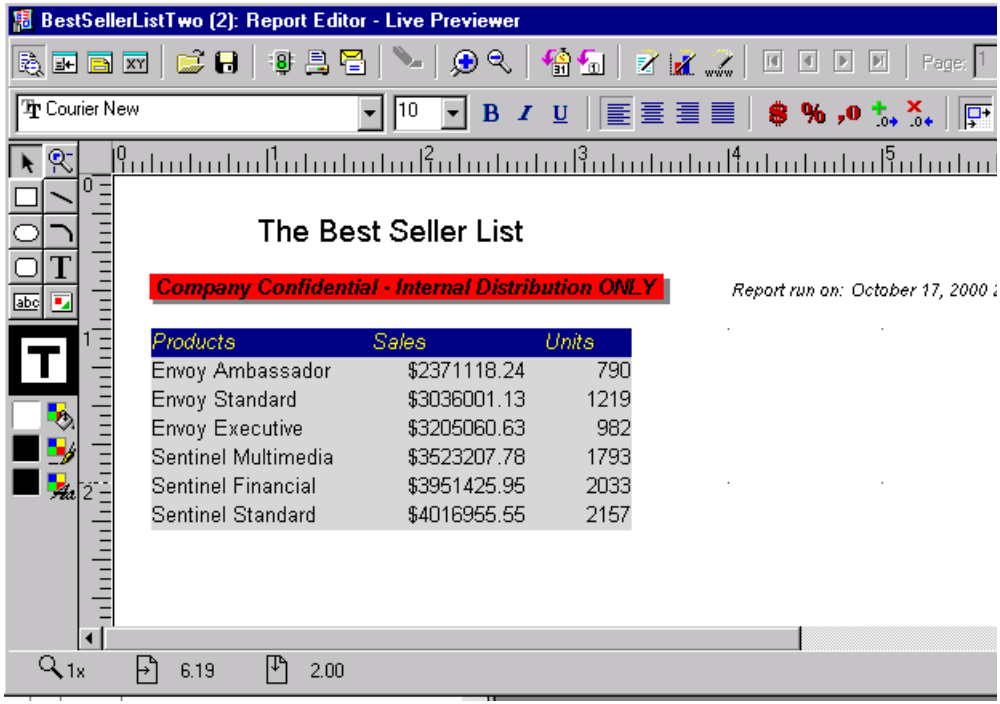
The following examples show you how to change the font size for the title, make the column headings bold for more emphasis, and display the Sales aggregates to a currency format that has two decimal places.

Font Size You can enlarge the font size of the title in a few steps. First, select the title. The Previewer displays eight handles around the title. Then, from the toolbar, select the down-arrow size button and click 18. The Previewer immediately enlarges the title font size to 18 and displays the result in the window.

Emphasize Column Headings You can change the font style of the headings from Regular to Bold in two steps. Select a heading and then click the **Bold** button in the format section of the toolbar. The Previewer immediately changes the style to bold and displays the result in the window.

Currency Format You can change the display format of a numeric column in a few steps. To format the Sales column, select the column and then, from the toolbar, click the **Currency** button, click the **Add decimal place** button twice, and then click the **End justify** button.

The Previewer immediately applies these changes and displays the result.



Update the Report

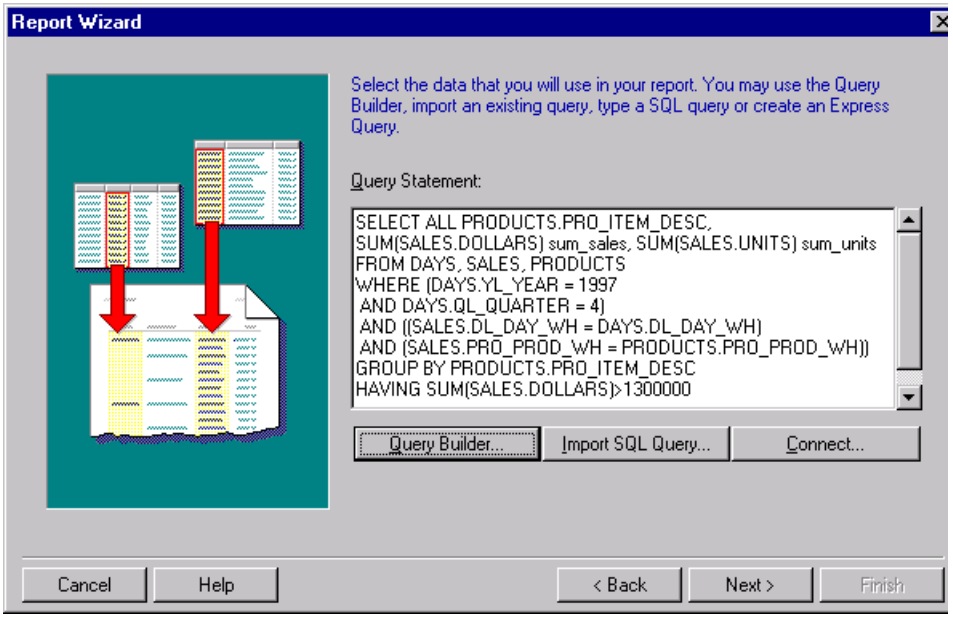
You can use the Report Wizard to update the report. For example, you could add a column for aggregate cost and calculate values for a margin column. Or, you could sort the column in another order. To update the report, open the report and then call the Report Wizard. The approach is then similar to the preceding example.

The following example changes the sort order of the aggregate sales column to Descending. To get started, open the report with Report Builder, and then from the toolbar call the Report Wizard.

1. Select **Report Wizard**.

   The Report Builder returns a window that contains tabs to all the information that defines the report.

2. Click the **Data** tab.

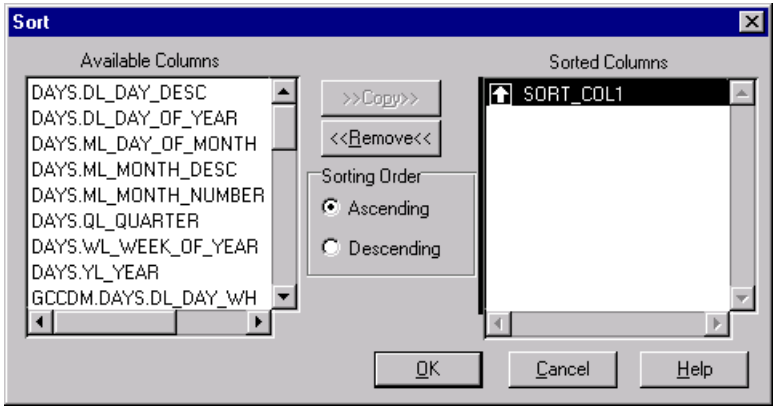


3. Click **Query Builder**.

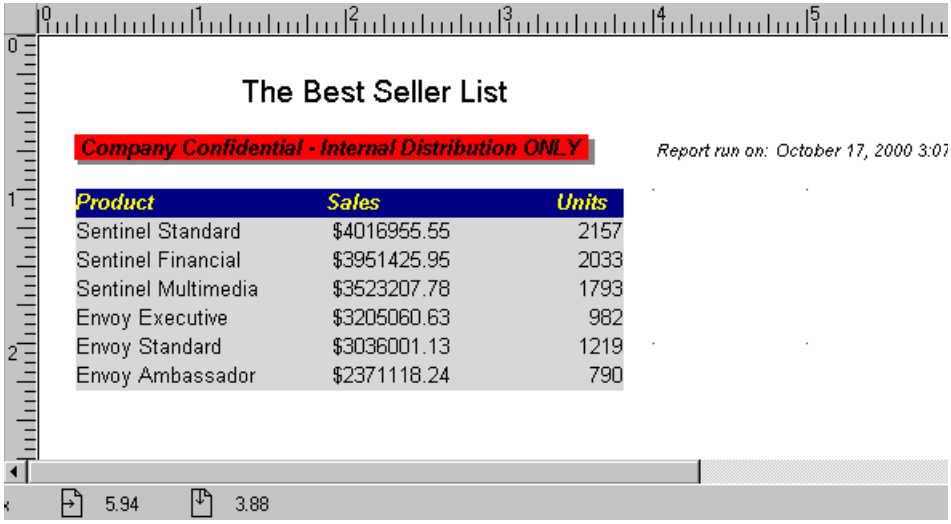
   The wizard returns the Query Builder window.

4. Click the **Sort** button on the toolbar.

   The wizard returns the Sort pane.



5. Select **SORT_COL1** in the Sorted Columns pane and then click **Remove**.
6. Double-click **sum_sales** in the Available Columns pane and then check the **Descending** button.
7. Click **OK** to exit the Sort window, and then click **OK** again to exit the Query Builder. You can now click Finish to execute and display the report.



The report rows now occur in order of sales descending from the top seller.

Deploy the Report

You can now disseminate the report. If you want to publish it on the Web, you can use the Oracle Reports Web Wizard to prepare the report for the Web. Click on the Web Wizard button in the Live Previewer toolbar to call this wizard.

It is not necessary to invoke Report Builder just to run a report. For a client / server installation, all that you need is the report definition file (with the .rdf extension), Reports Runtime, and SQL*Net access to the database. For a multitier configuration, you must install the Reports Server on an application server machine. See the *Oracle Reports Server Installation and Configuration Guide* for details.

The Next Step

This chapter showed you how to generate, store, and distribute standard reports using Oracle Reports. From this information, you should be able to define and distribute all the indispensable reports from the requirements phase plus additional ones that will be required in the future.

You are now ready to set up Oracle Discover so users can access the data mart's contents in an ad hoc fashion. Chapter 12, "Ad Hoc Analysis with Discoverer" shows you how to allow this kind of access.

12

Ad Hoc Analysis with Discoverer

This chapter shows you how to set up Oracle Discoverer so that users can access the data mart using Discoverer End User Edition, a powerful query and analysis tool. The chapter describes Oracle Discoverer, shows you how to create an End User Layer using either Oracle Warehouse Builder or Discoverer Administration Edition, reviews the fundamentals of querying and reporting, and then walks you through a typical exploration of data stored in the case study data mart.

The true measure of success of your data mart project is the value and utility of the data to the end users. Regardless of the technical expertise applied to the development of the data mart, if each end user cannot access the necessary information, the data mart is a failure.

At this point, you have built the database, and you want to let users explore the data and get information to support their decisions. If you understood the business problems and have successfully translated that understanding to the star schema, it will be easy for users to create and generate meaningful reports. Most importantly, you want every user to feel as though the interaction with the data is in appropriate business terms, not in unfamiliar database or system terms.

This chapter covers the following topics:

- Oracle Discoverer and the Data Mart
- The End User Layer
- Analyze the Data
- Investigate the Data

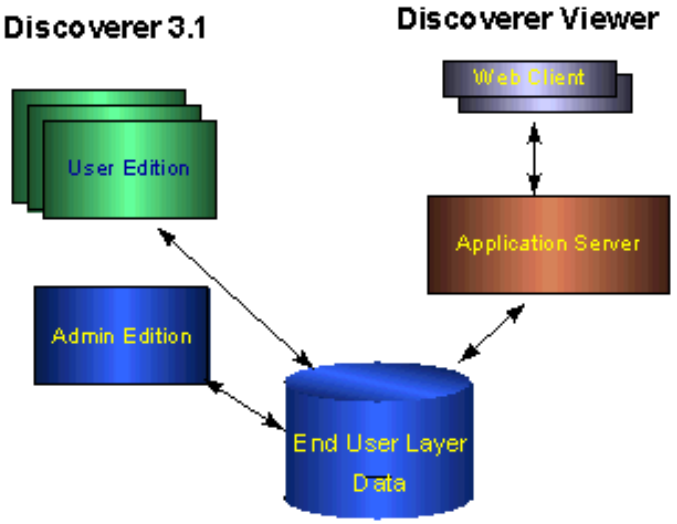
Oracle Discoverer and the Data Mart

Oracle Discoverer is an easy-to-use decision support tool that lets you query, analyze, and report data according to your business requirements. Oracle Discoverer reduces the complexity of managing many tools by providing a single administrative tool and one interface to support ad hoc analysis.

Discoverer 3*i* leverages your existing investments in hardware, software, and skills. The components for Discoverer constitute a business intelligence solution for client server and 'n'-tier architectures that deploy rapidly and are immediately available to all business users. Discoverer 3*i* and Discoverer for Windows have a common administration component.

Discoverer is the end user ad hoc query, reporting, analysis, and Web Publishing tool. Users gain rapid access to corporate data through Discoverer's easy to use interface. This in turn enables users to make more informed business decisions. Discoverer is deployed in either a Windows client-server architecture or Java / CORBA 'n'-tier architecture.

Oracle Discoverer consists of four components, each with a unique role: User Edition, Viewer Edition, End User Layer, and Administration Edition.



You can create ad hoc queries and reports using the User Edition, translate database code to business terms through the metalayer called the "End User Layer," and

manage the metalayer with the Administration Edition, an easy-to-use interface. Oracle Discoverer also provides several built-in features that optimize response to user queries.

With Discoverer viewer, users can access the data managed by Discoverer over the Internet and can view and manipulate reports using a Web browser interface.

Discoverer User Edition

Discoverer User Edition enables business users to query, graph and create reports. Users can perform drill and pivot analysis and publish results to the World Wide Web. Discoverer has a proven track record in ease of use, performance, navigational power and openness. The User Edition is available for the Web and Windows.

End User Layer

The Discoverer End User Layer (EUL) is a server based, low-maintenance, powerful mechanism for providing end users with a business-oriented view of the data warehouse or OLTP system. The End User Layer is shared by both the Windows and Web releases of Discoverer which significantly lowers maintenance and administration overhead.

Discoverer Administration Edition

Discoverer Administration Edition is used for the initial setup and ongoing maintenance of the End User Layer. The Administrator performs a variety of tasks such as maintenance of business areas, folders, and end user access. Discoverer provides extensive defaulting capability to enable administrators to rapidly implement their end user environment. The Administration Edition is currently available for Windows only.

The responsibility of the Discoverer administrator is to hide the complexity of the database from the end user. Administrator functions also include:

- Defining business visualization of the data
- Controlling user access
- Defining drill paths
- Defining summary tables

You perform these tasks with the Administration Edition.

Discoverer Viewer

Discoverer Viewer enables business users to run and view reports created by users on the Windows or Web releases of the User Edition. Discoverer 3i Viewer is a state of the art Business Intelligence tool, requiring no software other than a simple Web browser. Its Web interface is familiar to anyone who regularly uses a browser to view pages on the Internet, and its backbone uses the same high performance, scalable architecture as other Discoverer solutions. In addition, Discoverer 3i Viewer lets you embed the report within a standard Web Page to integrate Business Intelligence into your Web site.

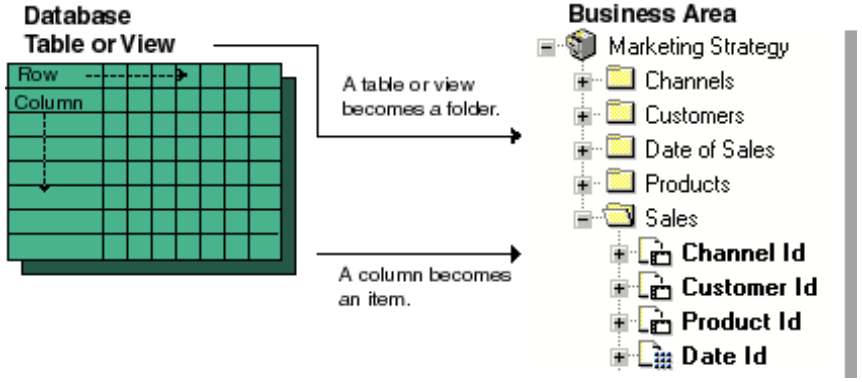
Design a Business Area

Simply stated, a **business area** is a logical grouping of data that applies to specific data requirements of a user group. Although some of the data needed by different departments may be the same, the exact combination of tables and views for each department is usually distinct.

Create and Manage a Business Area

Using Discoverer Administration Edition, you tailor the grouping of data to provide users with the proper access to the precise data they need for ad hoc queries, decision support, and presentation of results.

Business areas are designed to meet the data needs of the users. Data comes from different tables or views and is mapped to folders and items, as shown in the following figure. Folders can be used in more than one business area and do not have to be related to each other. Business areas can be allocated to many user IDs or roles; a user ID or role may be granted access to many business areas.



Business areas include conditions, joins, calculated items, formatting, hierarchical structures, and other custom features. Thus, a business area is a cohesive set of folders, items, and other functions designed specifically for professional business people so they can more effectively use the data in their company databases.

The time invested initially to create a well-designed business area will have a significant payback in reduced administration and maintenance, because you eliminate or greatly reduce the amount of time spent tailoring the environment for each user at the database level.

Manage End-User Access

The data administrator is responsible for assigning business area access and privileges to users. The data administrator can make business areas available to individual users or to all users.

The data administrator assigns privileges to users. Privileges include:

- Administration:
  - Manage Scheduled Workbooks
  - Create Summary Tables
  - Create / Edit Business Areas
  - Format Business Areas
  - Set Privileges
- User:
  - Collect Query Statistics
  - Create / Edit Queries
  - Drill Out
  - Grant Workbook Access
  - Save Workbooks to database
  - Item Drill
  - Schedule Workbooks

These privileges are independent of, but do not override, database privileges set directly through the database server.

Oracle Discoverer and Query Performance

Oracle Discoverer provides many built-in functions that can improve the performance of queries.

Query Prediction

Most decision-support tools provide a query governor to cancel long-running queries. Oracle Discoverer extends this capability to give you what you really need: an estimate of the retrieval time before the query execution and the ability to cancel that query.

The query predictor enables the user to make sensible decisions on whether to run a query immediately using the nonblocking capabilities, or refer it to a batch process for later calculation.

Query Governor

Oracle Discoverer lets you govern queries in the following ways:

- Voluntarily declining the query

  If the query predictor predicts a time longer than a specified time, a dialog box asks if you want to continue and run your query. If you decline the query, your original data is untouched. If you had no data, you retrieve no data.

- Automatically terminating the query after given time

  If the query is not completed within a given time, the query is automatically canceled. In this case, your original data is unchanged. If you had retrieved no data previously, no data is displayed.

- Automatically canceling the query after returning a specified number of rows

  Oracle Discoverer retrieves the number of rows up to the limit. Therefore, if you requested a tabular layout, you could get a partial list. In a crosstab, the results are calculated based on the partial results retrieved. Thus, some aggregations could be incorrect. Oracle Discoverer notifies the user when this is the case.

- Schedule the query to run in batch.

Client-Side Cubic Cache

As data is retrieved from the server, Oracle Discoverer stores it in a client-side cubic cache (the Resultsbase). The Resultsbase enables extremely rapid multidimensional analysis with any number of dimensions and measures, rotation, drill down, drill

up, and drill across functionality. An expert SQL query engine dynamically generates performance-optimized SQL queries.

The client-side cubic cache makes many operations, such as a pivot or the removal of a label or a column, instantaneous. If the new result can be calculated without sending a new query to the server, the whole step of processing a query and retrieving data is bypassed, thereby reducing overall network traffic.

The End User Layer

To give end users access to the data mart, you must perform a few procedures using Discoverer Administration Edition. Because Oracle Discoverer contains many features that let you customize the reports that the end users see, you may want to do more than the minimum and make the business area more appealing to the user.

Install the End User Layer

First, you must install a public End User Layer which is accessible to all users. You install this layer using Oracle Discoverer Administration Edition. Developers often install a private EUL for testing and application development.

To install an End User Layer, call the Administration Edition.

1. Select **Start > Program > Oracle Discoverer 3.1 > Administration Edition**.

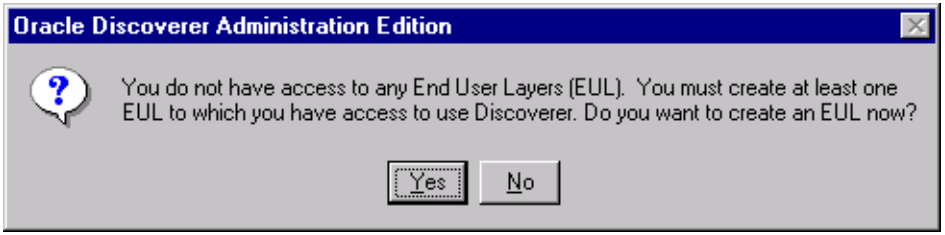
   If Oracle Discoverer displays the Welcome screen, select **Start**. Discoverer then returns a logon panel that solicits logon information.

2. Enter a username (***system***), a password (*manager*), and a connect string (*gccdm.us.oracle.com*).

   If you install the EUL using an account that does not have the DBA role, then you will not be able to schedule queries to run in batch mode.

3. Click *Connect*.

   Discoverer returns a dialog box.



4. Click Yes.

   Discoverer returns the EUL Manager panel. This panel solicits an action.

5. Click Create an EUL to call the Create EUL Wizard.

   Discoverer returns the first of the wizard pages.



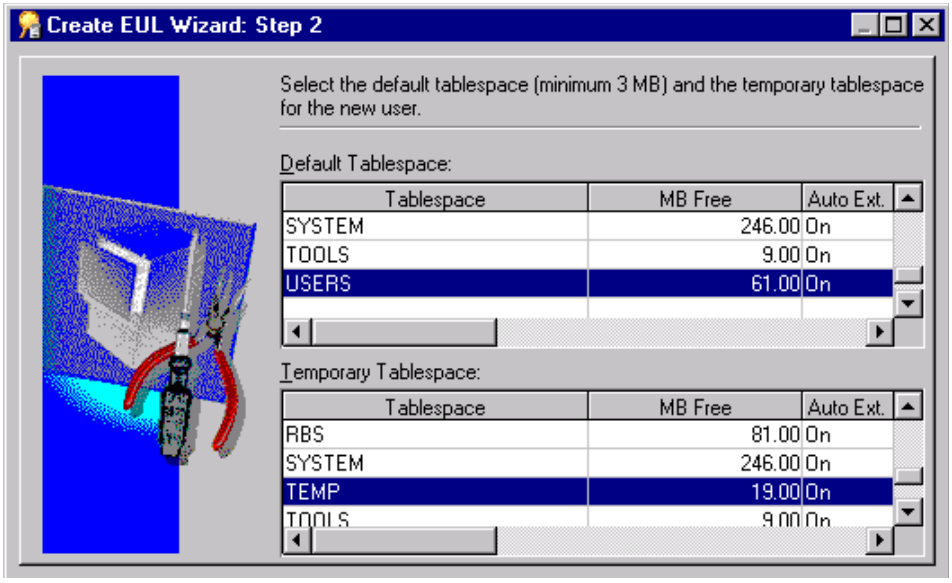
   This example creates a new owner for the End User Layer.

6. Complete the wizard page as follows:

   - Check **Create a new user** and **Grant access to PUBLIC** buttons.
   - Enter a username (*gccdmEUL*) and password.

7. Click **Next**.

   The wizard solicits the names for a Default and Temporary tablespace. When Discoverer creates the tables for this End User Layer, Oracle8*i* stores their data in the specified tablespaces.



8. Select **USERS** for the default tablespace and **TEMP** for the temporary tablespace. Then, click **Finish**.

   The wizard may require a few minutes to create the End User Layer. Once the operation is completed, the wizard returns a confirmation panel.

9. Click **OK**.

   Oracle Discoverer returns an option panel: Do you want to install tutorial tables in the End User Layer?

10. Click **NO**, you can install the tables at a later time.

    Oracle Discoverer returns another option panel: Do you want to connect with the End User Layer as its owner (*gccdmEUL*)?

    At this point, you can connect with the End User Layer and manually create a Business Area for the users or you can export a Business Area from the Oracle Warehouse Builder repository. The following example shows you how to define a Business Area in the Builder repository and then transfer it directly to the Discoverer End User Layer using the Builder Transfer wizard.

11. Click **NO**.

    The next section shows you how to define and transfer a Business Area using Oracle Warehouse Builder.

Create and Transfer a Business Area

Business areas are valuable when a data mart contains a large number of tables (and views) that are accessed by different organizations. You can use business areas to tailor access to the needs of individual organizations. Thus, when you know an organization's specific needs, you can develop a business area that meets *those* needs, eliminates unnecessary tables, and employs *their* terminology.

The case study data mart contains only a few tables, so it is difficult to illustrate the real value of business areas. The following example does, however, illustrate all of the necessary mechanics required to create a business area for a set of users. You can complete the task in four steps:

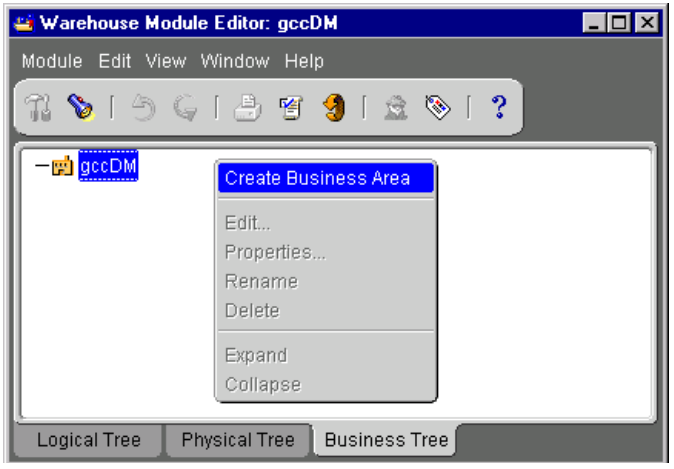
1. Create a business area using Builder
2. Transfer the definitions to a flat file using Builder and the Discoverer Bridge
3. Import the definitions in the End User Layer using Discoverer
4. Customize the business area using Discover.

After the business area has been customized, you can allow access to users.

Create the Business Area

The example creates a business area that includes only the case study's dimensions and facts. This results in no loss of information nor does it degrade query performance (Oracle8*i* still uses the materialized views to rewrite queries).

To define the Business Area, call the Builder client, open the GCC Data Mart Project, and then open the *gccDM* warehouse module.



Select the name of the warehouse module, right-click the **Business Tree** tab, and then select **Create Business Area** from the pop-up list.

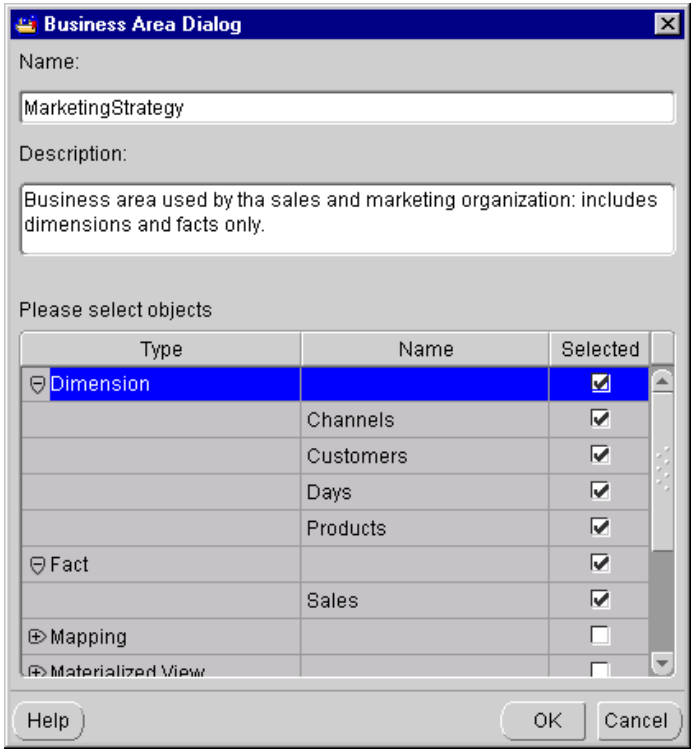
The editor returns the Business Area Dialog panel. This panel contains three areas:

- Text box for the Business Area's name.

  The name can be a logical or physical name.

- Text box for the Business Area's description.
- The names of each object defined in the warehouse module and a corresponding check box.

Use the text boxes to name and describe the Business Area and then select the check box for each object that is to be included within the Business Area.

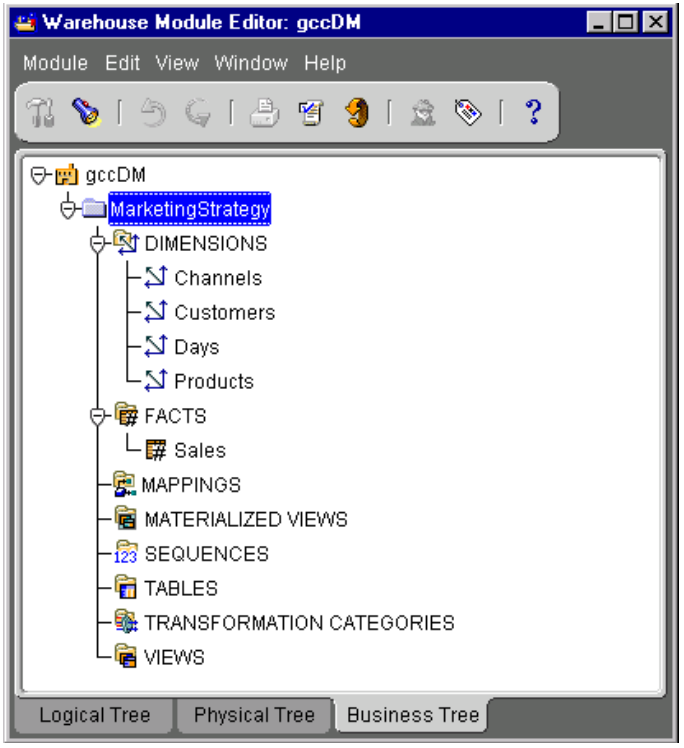


Enter a name, description, and select the objects to include within the Area and then click **OK**.

Builder creates the Business Area and stores it in the warehouse module.

Before you call the Transfer Wizard to export these definitions to a flat file, you should commit the definitions. Click the **Commit** button on the Builder toolbar.

To view the definition of the business area, fully expand the editor's navigation tree:



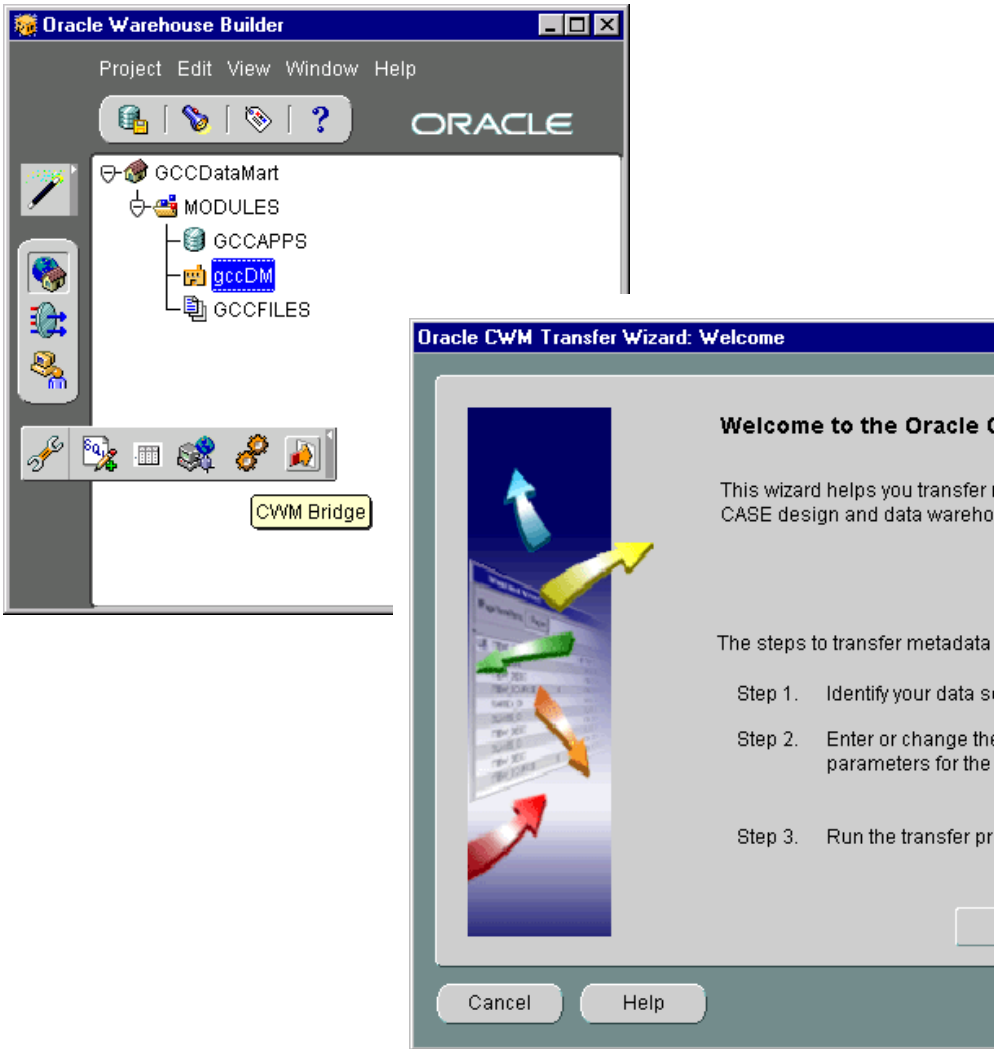
Transfer the Business Area

The definitions for the Marketing Strategy business area now reside in the Builder repository, and you can now export these definitions to a flat file. Use the Builder Transfer wizard to assign values to several parameters, and the wizard calls the CWM Bridge to transfer the definitions to a flat file.

To call the CWM Transfer Wizard,

1. Switch to the Builder console.
2. Click the icon for the **Utilities** drawer and then select the **CWM Bridge** entry from the drawer.

   Builder returns the wizard welcome page.



3. Click **Next**.

   The wizard returns the Source and Target Locations page. This page solicits the name of the product for the source and target repositories (Oracle Warehouse Builder and Oracle Discoverer) and a brief description of the transfer. Currently, you can enter only a description for the transfer.

4. Enter a brief description of the transfer, and then click **Next**.

The wizard returns a list of parameters. These parameters describe what you want to transfer, where it is located, and specific information about the operation. The wizard calls the Discover Bridge with these parameters, and the Bridge carries out the transfer operation.

The case study uses the following values to configure the Transfer Wizard configuration parameters.

Parameter	Value	Note
Warehouse Builder user	*gccrep*	
Password for Builder user	*gccrep*	
Builder connect string	*dwdoc11-pc:1521:ora816*	Node:Port:SID
Warehouse Builder Project	*GCCDataMart*	Case sensitive
Metadata Transfer Includes	*Dimensional Entities Only*	Dimensions and facts
Discoverer EUL Owner	*gccdmEUL*	Destination EUL
Discoverer Schema Owner	*gccDM*	Target schema
Dimensional Reuse	**False**	
Discoverer Output File	*e:\MarketingStrategies*	Discoverer imports this flat file
Log Level	**Errors**	

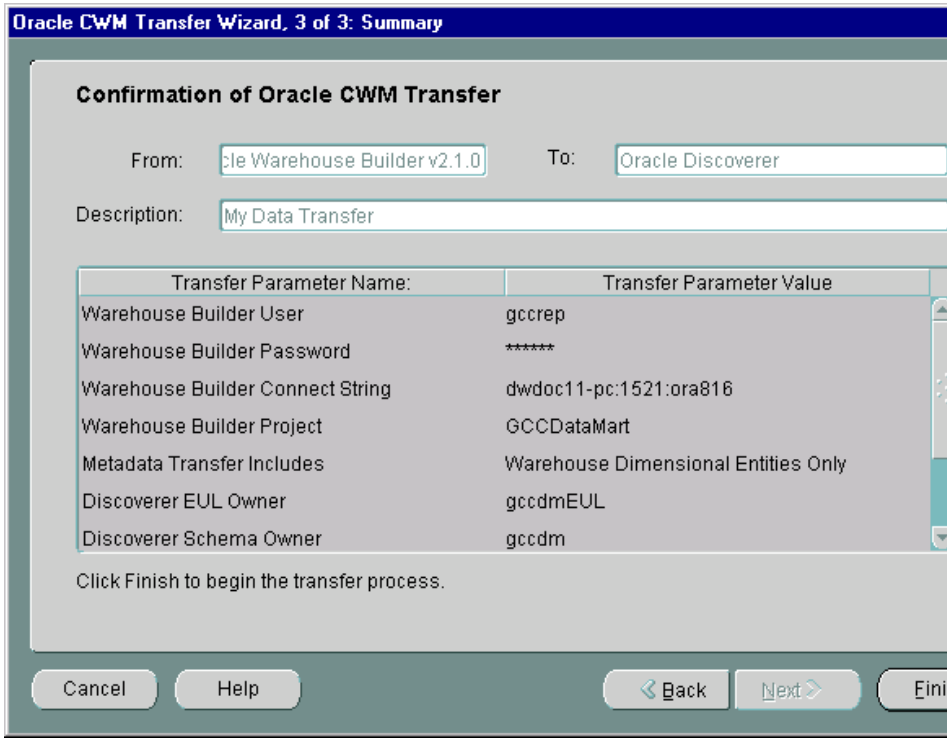
After you assign values to these parameters and click the Finish button on the next wizard page, the wizard calls the Discoverer Bridge. The Bridge logs on to the Builder repository and opens the GCCDataMart Project using the first four values. Then, the Bridge creates and writes definitions for dimensions and facts (Dimension Entities Only) to the output text file named by the Discoverer Output File parameter. This text file includes the name of the Discoverer EUL and the name of the data mart target schema. The wizard writes all error messages to its log file.

The Dimensional Reuse parameter must be set to False.

5. Click **Next** after you enter the parameters.

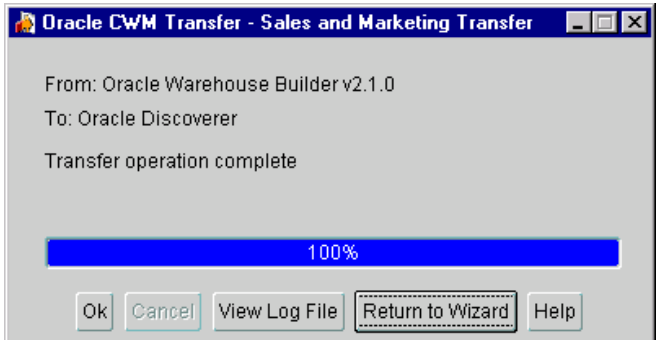
The wizard returns its finish page which summarizes the values assigned to each configuration parameter.

Note: The Dimensional Reuse parameter must be set to False.



6. Verify that the values assigned to the parameters are correct, and click **Finish**.

   The wizard starts the Discoverer Bridge which logs on to the Builder Project and writes the definitions for the business area to a flat file.



   At the conclusion of the operation, you can view the log file, return to the Wizard, or click **OK**.

7. Click **OK** to complete the transfer of the definitions to the flat file.

   The definitions for the MarketingStrategies business area now reside in the flat file e:\MarketingStrategies.eex.

In the next step, you use the Discoverer Administration Edition to import these definitions into the End User Layer *gccdmEUL*.

Import the Business Area

You import the definitions from the flat file and create the business area in the End User Layer using the Discoverer Administration Edition.

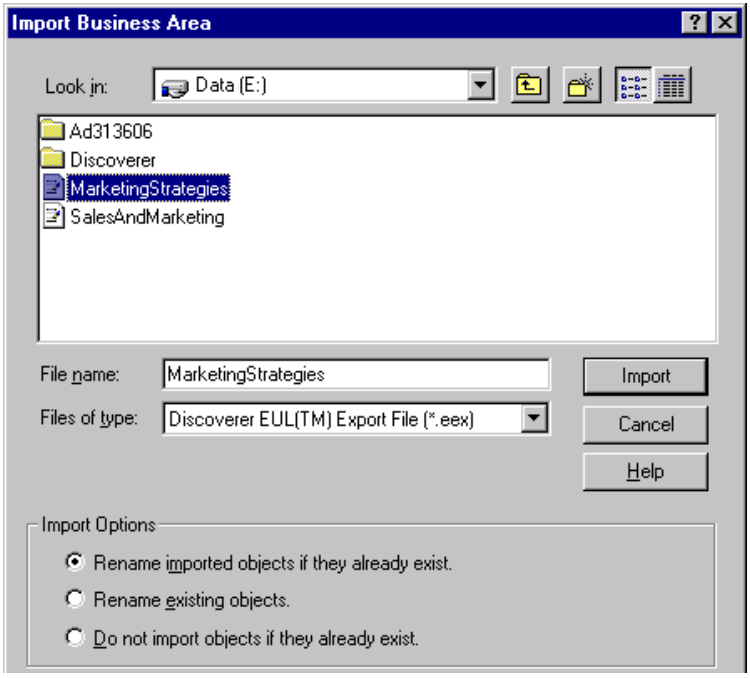
To import the definitions, call the Discoverer Administration Edition and log on to the End User Layer (*gccdmEUL*). After you complete the log on sequence, the Administration Edition returns the Load Wizard's window which solicits an action.

1. Click the **Cancel** button on the wizard window.

   The Administration Edition window now contains an Administration Task List.

2. Select **File** from the toolbar and then select **Import** from the drop-down list.

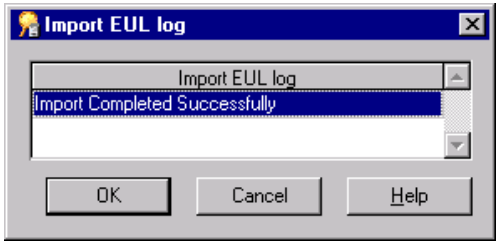
   The Administration Edition returns the Import Business Area window.



3. Select the name of the file which contains the exported definitions of business area (*MarketingStrategies.eex*).

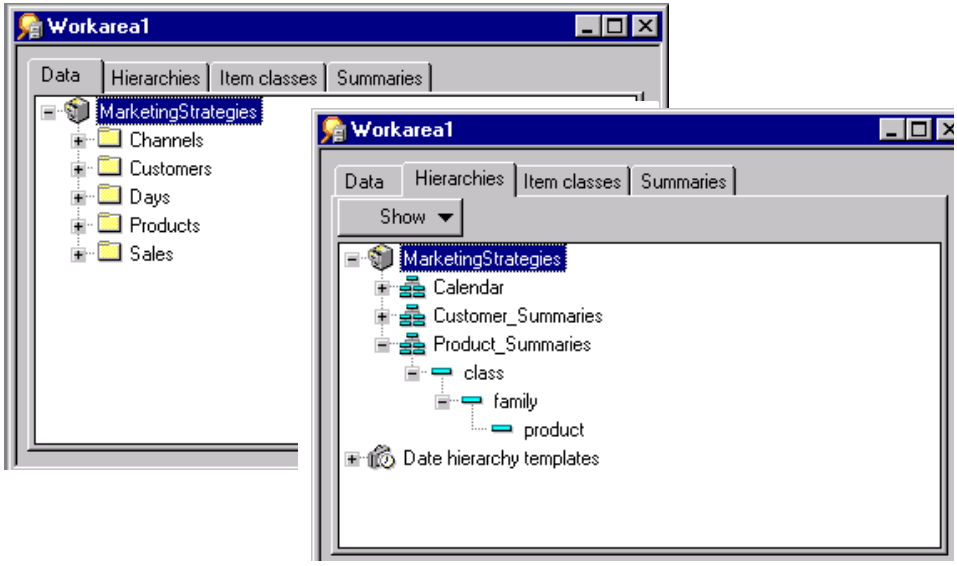
4. Click **Import**.

   The Discoverer Administration Edition imports the definitions from the flat file and returns a confirmation panel when the operation is completed.



5. Click **OK**.

   The Administration Edition returns its Workarea window. This window contains a navigation tree labeled *MarketingStrategies* that has a branch for each dimension or fact table imported. You can expand the tree and verify the contents of the business area. You can also click the Hierarchies tab and verify that all the hierarchies defined on dimension tables were also imported.



The additional hierarchy is a default object built by Administration Edition.

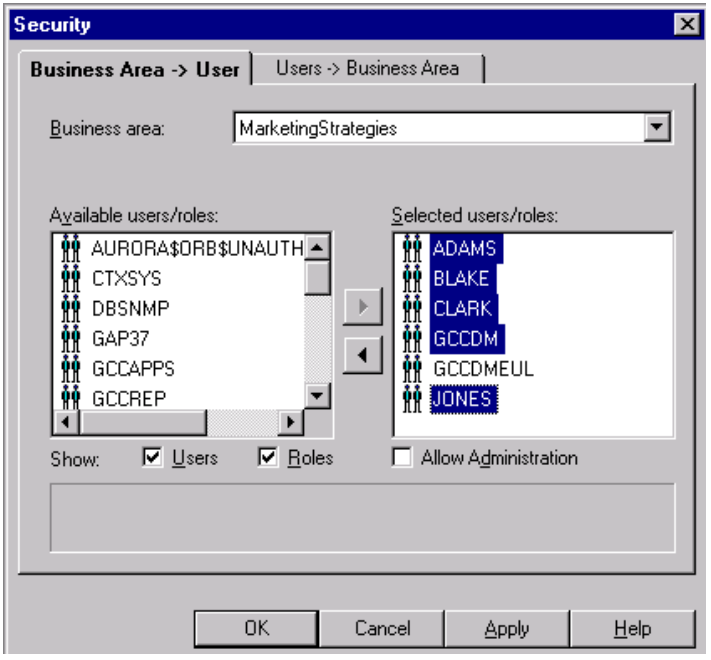
Grant Access to the Business Area

The End User Layer now contains the *MarketingStrategies* business area. You must grant access rights and privileges to the business area. This determines who can see and use data in the business area and requires only a few steps.

Grant Access

1. Select **Tools** > **Security** from the toolbar.

   The Administration Edition returns the Security panel.



2. Select the **Business Area->User** tab.

   From this panel, you can grant access to the selected business area. In the above example, the *MarketingStrategies* area is selected and one user (*gccdmEUL*) has already been granted access. This user was assigned to a Transfer Wizard parameter (Discoverer Schema Owner).

   To grant access to a user, double-click their name in the Available users / roles pane and Discoverer moves the name to the Selected users / roles pane.

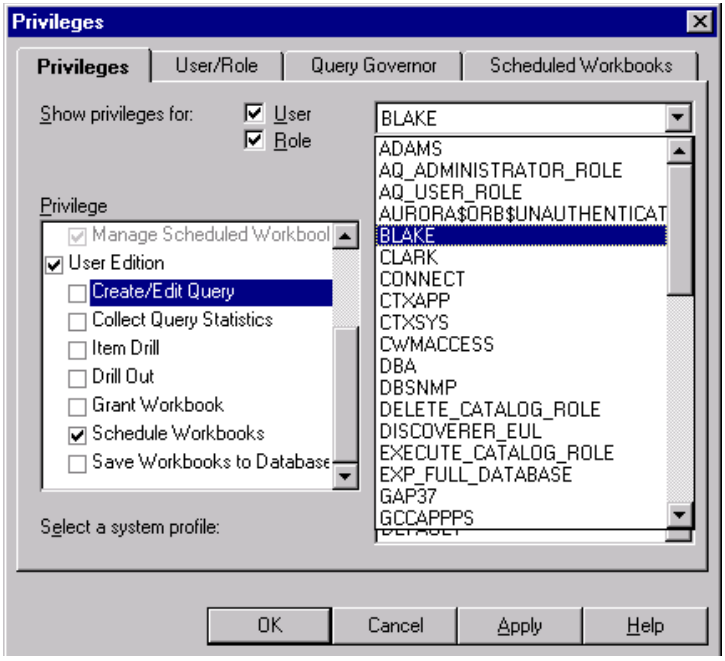
3. Click **OK** to save the changes and close the dialog box.

Grant Discoverer Privileges

You can also grant Oracle Discoverer privileges to selected users. For example, to grant *Blake* the privilege to schedule workbooks:

1. Select **Tools > Privileges** from the toolbar.

   The Administration Edition returns the Privileges panel.



2. Select the **Privileges** tab.

   From this panel, you can select a set of privileges and assign them to a user.

3. Check **Schedule Workbooks** and then select **BLAKE** from the down-arrow list.

4. Click **OK** to save the changes and close the dialog box.

The user Blake now has the privilege to schedule workbooks. Of course, this operation assumes that you allowed this user access to the business area.

Customize the Business Area

You can reformat items in the business area so they correspond more closely to the user's actual view of the business world. This simplifies the reports which are developed for users.

Many dialog boxes have an OK button and an Apply button. Click **Apply** and you put changes into effect, but this does not close the dialog box. You can continue making changes to other items using the same dialog box. Click **OK** and you apply the changes and close the dialog box. To save changes automatically, select the business area, right-click, and select **Properties.** Then, check the box **Automatically save changes after each edit**.

Rename Folders and Add Descriptions

You can create meaningful names for each business folder. Folders are the basic elements that the end user sees when working with the business area. For this reason, each folder should have a meaningful name as well as a description that explains the primary use of the folder.

To see the folders in the business area, click the plus sign (+) to the left of the **MarketingStrategies** business area.

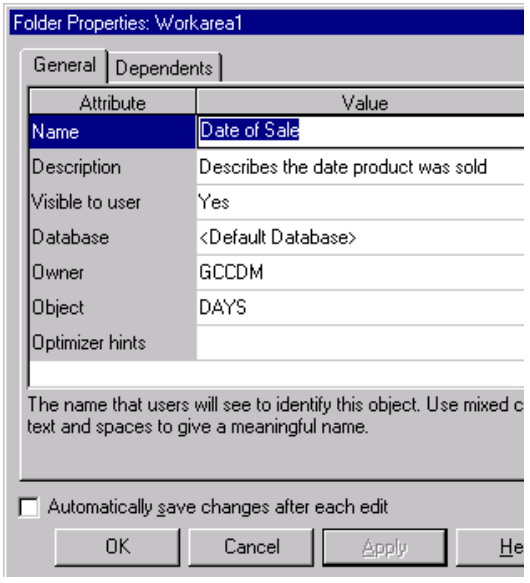


This example shows you how to rename the *Days* folder to the *Date of Sale* folder. This takes only a few steps:

1. Double-click the **Days** folder.

   Discoverer returns the Folder Properties window.

2. Click **Name** and enter the new name (*Date of Sale*).



3. Click **Description** and enter a description.
4. Click **OK.**

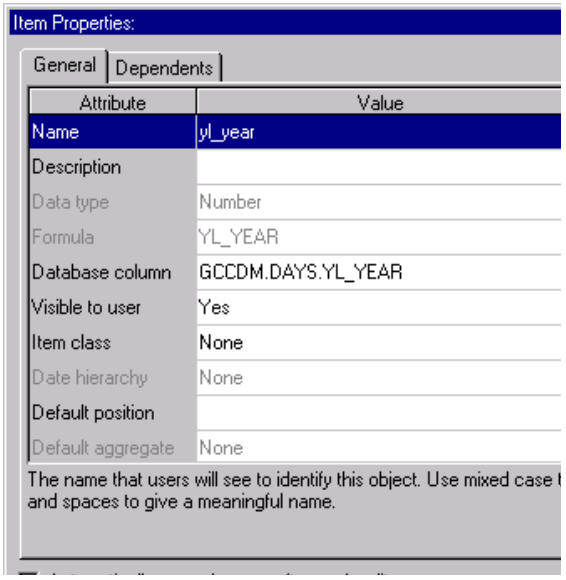
Rename an Item

Database columns often have names that bear too little meaning for the end user. You can change the names of columns to be more meaningful.

For example, to change the name of the column *yl_year* to *Year*:

1. Click the plus sign next to the Date of Sales folder.
2. Double-click ***yl_year*** to display its Item Properties window.

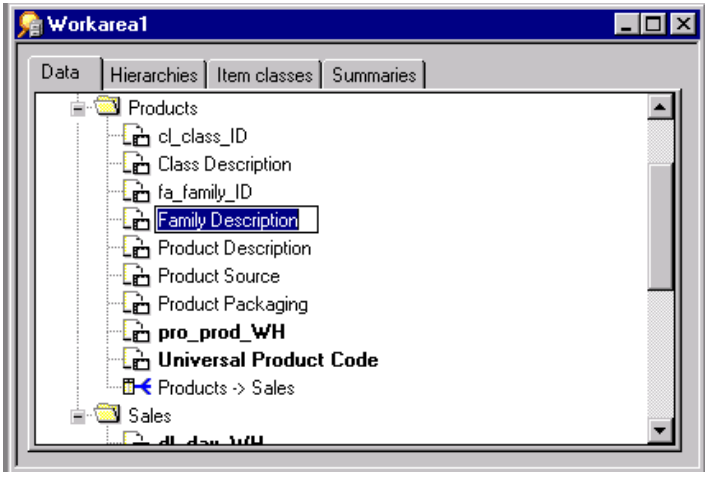
   Discoverer returns the Properties window for the column.



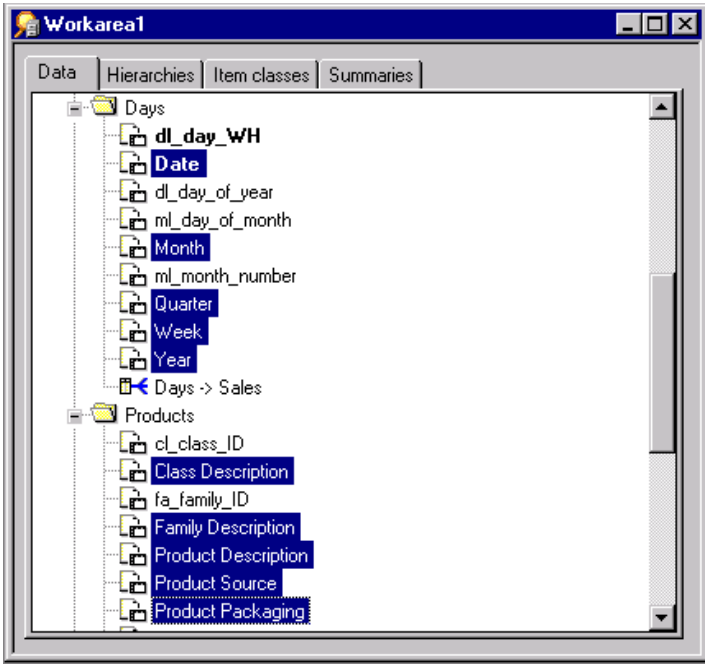
3. Click **Name** field and enter the new name.

4. Click **OK.**

You can also change a name in a manner familiar to most end users: select the name in the navigation tree, and then click it again. Discoverer highlights the name and you can type in a new name.



Several names in the example *MarketingStrategies* area have been changed so that they are more informative. The list below contains a few of the renamed columns.



Hide Items in the Business Area

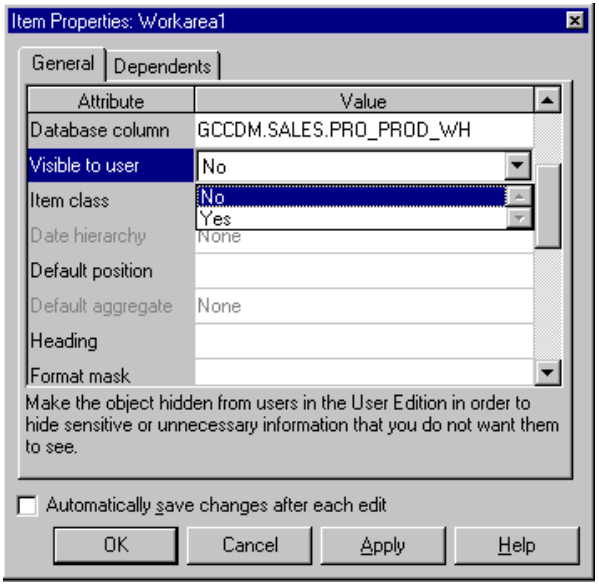
You can hide the database objects that are irrelevant to the end user provided they are not referenced by a hierarchy. For example, you can also hide calculations used in the business area. Hidden items are hidden and not deleted.

Warehouse and other key columns are good examples of column names that can be hidden. You can not hide the names of warehouse key columns in the case study's dimensions, however, because they all anchor at least one hierarchy. You can, however, hide the names of foreign key reference columns in the Sales fact table.

Table	Folder	Item
SALES	Sales	day_WH
		prod_WH
		cust_WH,
		chan_WH

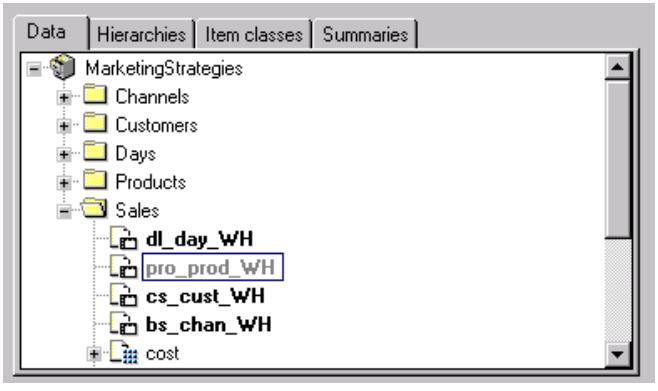
To hide the *prod_WH* column name in the Sales table:

1. Open the folder, **Sales**.
2. Double-click ***prod_WH***.
3. Discoverer returns its Item Properties window.



4. Select **No** from the down-arrow list next to the **Visible to User** parameter.
5. Click **OK** to close the dialog box.

   Discoverer makes the *prod_WH* entry grey to show that the item is hidden.



Format Currency

You can format the display of numeric values with a currency sign.

1. Expand the **Sales** folder, hold the Control key down, and then select the **Sales**, **Cost**, and **Margin** columns.
2. Right-click and select **Properties**.
3. Scroll until you see **Format mask**.
4. Enter **L999G999G990D99**.

   This mask precedes the value with the local currency sign and uses the default currency separators.
5. Click **OK**.

Create a New Item for Calculations

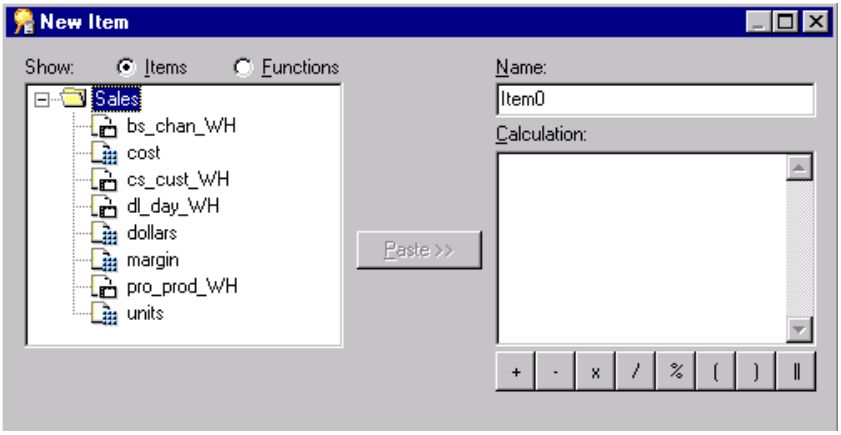
You can define calculated items for use by end users.

Typical business calculations include profits, profit margins, average revenues per month, expected sales, and percentage of profit. As the administrator, you can define the calculations that an end user is likely to need.

This example shows how to define an item so you can calculate percentage of profit:

1. Right-click **Sales > New Item**.

   Discoverer returns the New Item window.



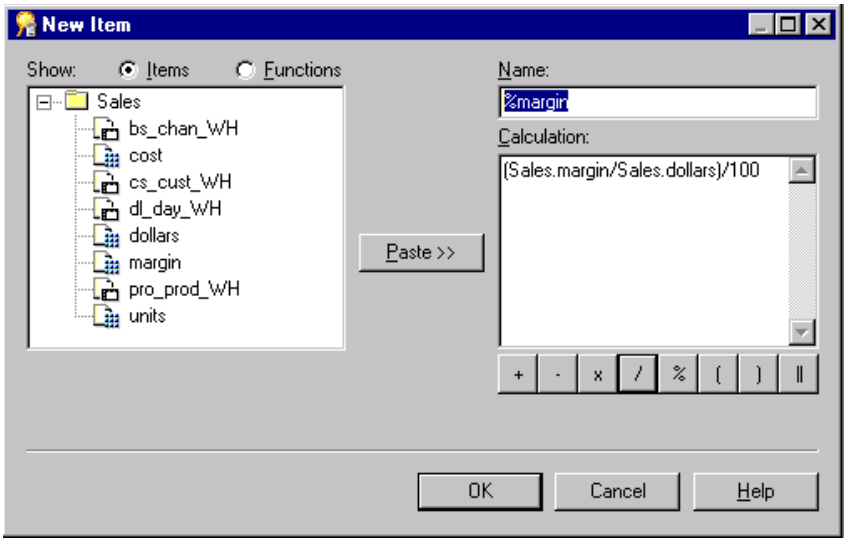
2. Expand the **Sales** navigation tree (click the plus sign).

To create a calculation, you can:

- Build an expression in the Calculation box one component at a time. For example, to move a column name to the Calculation box, double-click its name. Then, click the plus button to move the addition operator to the calculation. You can continue this process until you have completely defined the expression.
- Enter the calculation directly into the Calculation box.

The following example shows you how to calculate the percent margin as Sales minus cost.

3. Enter ***%margin*** in the **Name** field.
4. Enter ***(Sales.margin / Sales.dollars) * 100*** in the Calculation field.



5. Click **OK** to save the new calculation in the business area and close the panel.

   Calculations must conform the Oracle8*i* calculation syntax. For a full description of this syntax, see the Oracle8*i SQL Reference*.

6. Open the **Sales** folder and double-click the new **%margin** item. Set the Default position to **Data point** and the Default aggregate to **SUM.**
7. Click **OK** to close the Properties window.

Discoverer includes the *%margin* entry in the list of elements for the Sales table.

Defining a New Item Class

You can define a new item class. Administrators define these classes to:

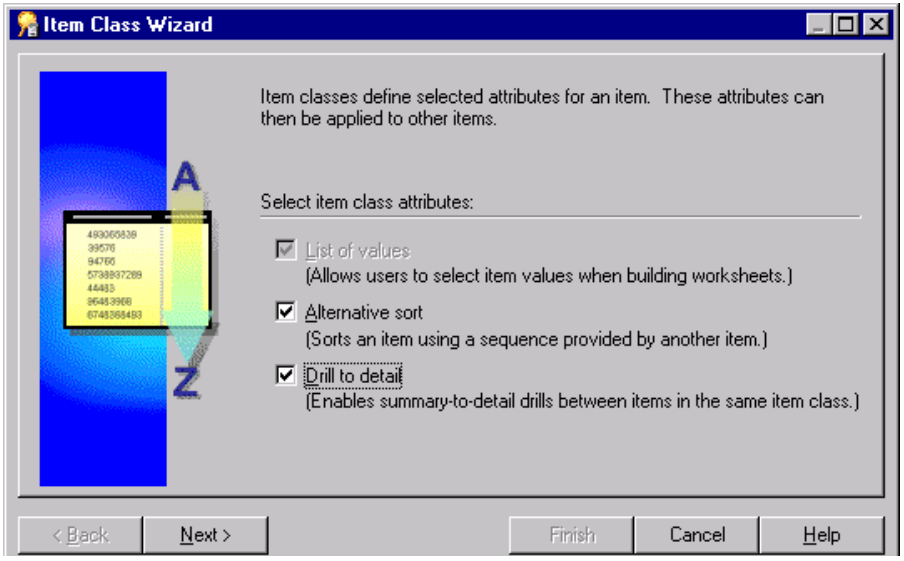
- Allow users to see a listing of the unique data values in an item.
- Sort values in a nonstandard fashion. Standard sorts include alphabetical, numeric, and date; a nonstandard sort might be North, South, East, West.
- Allow the user to see related information by hyperdrilling. **Hyperdrilling** allows jumping from summarized data to detail information. It differs from direct drilling, which allows end users to drill up or down to different levels of summarized data. (Hyperdrilling is also known as *drill to detail*.)

When you create a business area and choose to automatically generate lists of values, the Load Wizard creates item classes for you.

The following steps show how to create a new item class for drilling into related detail information from summary data and for viewing a list of unique values from the Item Desc column:

1. Select the **Item classes** tab.
2. Select **MarketingStrategies > New Item Class**.

   Discoverer returns the first page of the Item Class Wizard.



3. Click the check boxes for **Alternative sort** and **Drill to detail**.

Alternative sort lets you sort using a sequence provided by another item. Drill to detail lets a user drill from summary data into detail rows of an associated table for related information.

4. Click **Next**.
5. To create a list of detail values from a database column, open the *Products* folder, select ***pro_item_desc***, and then click **Next**.
6. To create an alternate sort, select ***pro_item_desc***.

   The report will be sorted by the data in this column.
7. Click **Next**.
8. To select the items that will use this item class, open the *Products* folder, select *pro_item_source* and click the **right arrow** to move it into the Selected items box. ***pro_item_desc*** is already listed in the box.

   Items that use the same item class define a relationship between those items. The relationship can be used to determine summary-to-detail drills.
9. Click **Next**.
10. Enter a name for this new Item Class, *Product Items class.*
11. Click **Finish** to create the item class.

If a list of values is available for a column, a plus sign (+) appears next to the item in the data tab in the work area.

The Item Desc item contains a list of values sorted in an alternate order. To see the list of values, click the plus sign **(+)** and click **Yes** to the informational message.

When you create an item class that includes an alternate sort order, a one-to-one relationship must exist between the column used for the list of values and the column used for the sort order. Also, the two items must be in the same folder.

Create a Condition

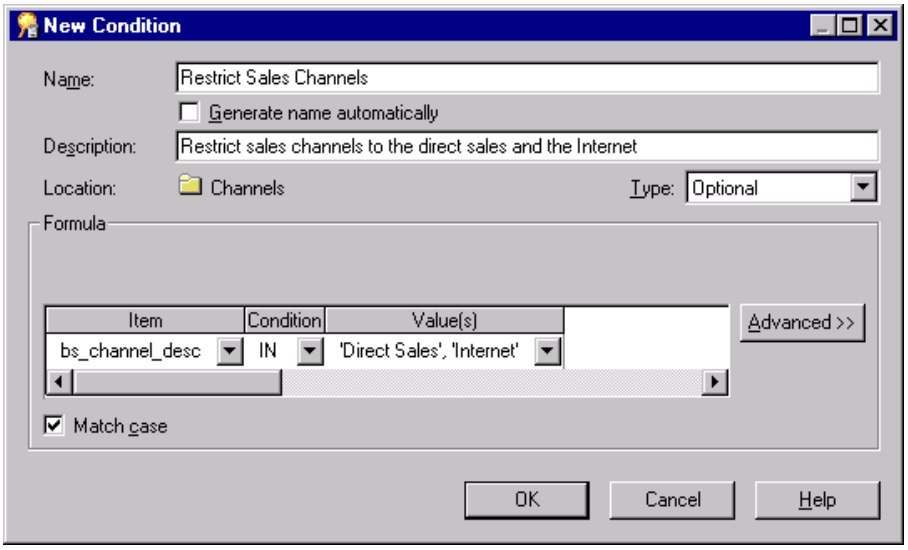
A condition limits the view of the data, restricting it to a predefined set of data.

To create a condition that limits the data to the Internet and Direct Sales channels, take these steps:

1. Expand the **Channels** folder and choose the item *bs_channel_desc*.
2. Select **Condition > New Condition**.

   (Or, from the Administration Task List, select **Create condition**.**)**

3. Clear the check box for **Generate name automatically**. Then, enter *Restrict Sales Channels* in the Name field.



4. From the Type drop-down list, select **Optional.** This lets the user choose whether or not to use the condition in a query.
5. Select the **IN** operator from the Condition drop-down list.
6. To limit channels to Direct Sales and Internet, enter the following in the Values text box;

   'Direct Sales', 'Internet'

7. Click **OK** and then click **OK** again to close the New Condition dialog box.

The *MarketingStrategies* business area is now ready for use, and you can exit the Discoverer Administration Edition.

The next section shows you, in a summary fashion, how to create a business area rather than import it from a Builder Project.

Create a Business Area

You can also create rather than import a business area within an End User Layer in a few steps using the Load Wizard, one of the Discoverer Administration Edition's wizards. The process takes three steps:

1. Identify the location of the database objects that will be loaded into this business area.
2. Identify specific objects to include in the business area.

   Identify user IDs and associated tables to group. You can specify access to private tables or public tables, meaning those tables that belong to a specific user ID or those that are available to every user ID.

Oracle Discoverer provides several options that let you filter the set of tables and views that are displayed. Typically, you use tables that are owned by someone other than the administrator. In this example, you use the Audit and Logging tables owned by the **gccDM** user.

To call the Load Wizard from the Discoverer toolbar:

1. Select **File > New**

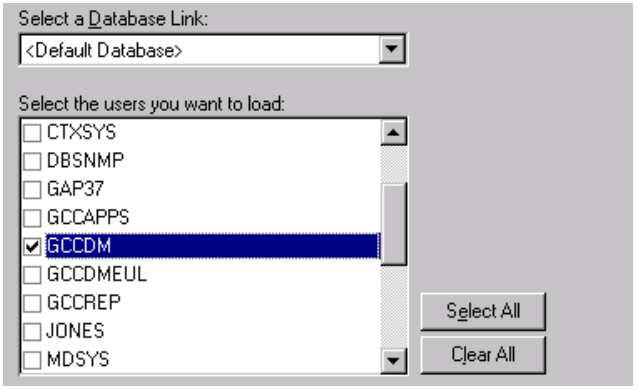
   Discoverer returns the Load Wizard.

2. Click **Create a business area**

   Discoverer solicits another action.

3. Click the button for **Online dictionary** and click **Next**.

   Discoverer returns a list of users.

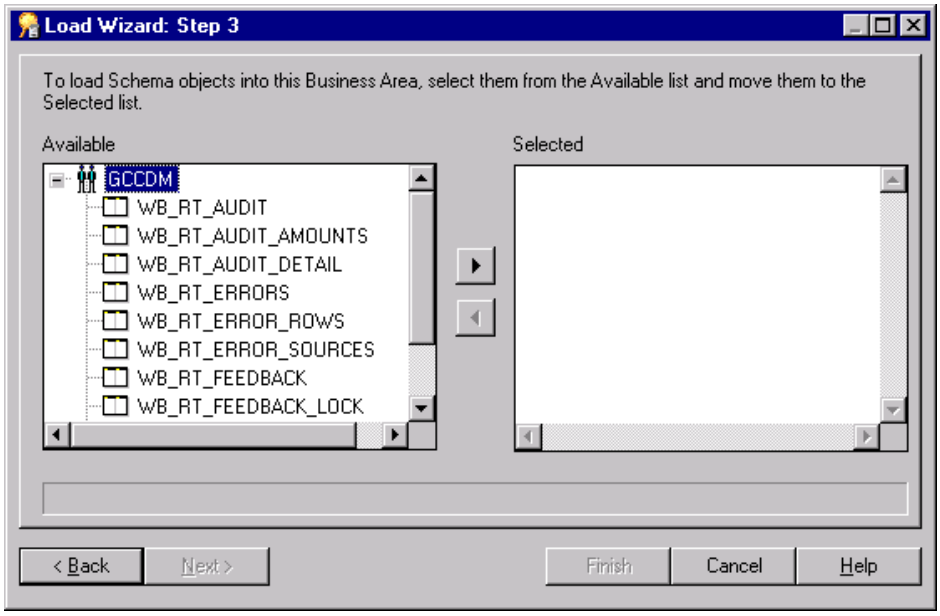


   Select **gccDM** and enter the wildcard pattern for the audit tables (*WB_%*).

All of the runtime Audit and Logging tables are prefixed with the three character string 'WB_' and the wildcard pattern will import definitions only for tables that match this search pattern

4. Click **Next**.

   Discoverer returns a window that has an Available and a Selected pane. The Available pane contains a navigation tree named **GCCDM**. Expand the navigation tree to see all the tables that matched the search pattern.



5. Select the owner name (**gccDM**) and then click the right-arrow button.

   Discoverer moves all the names to the Selected pane.

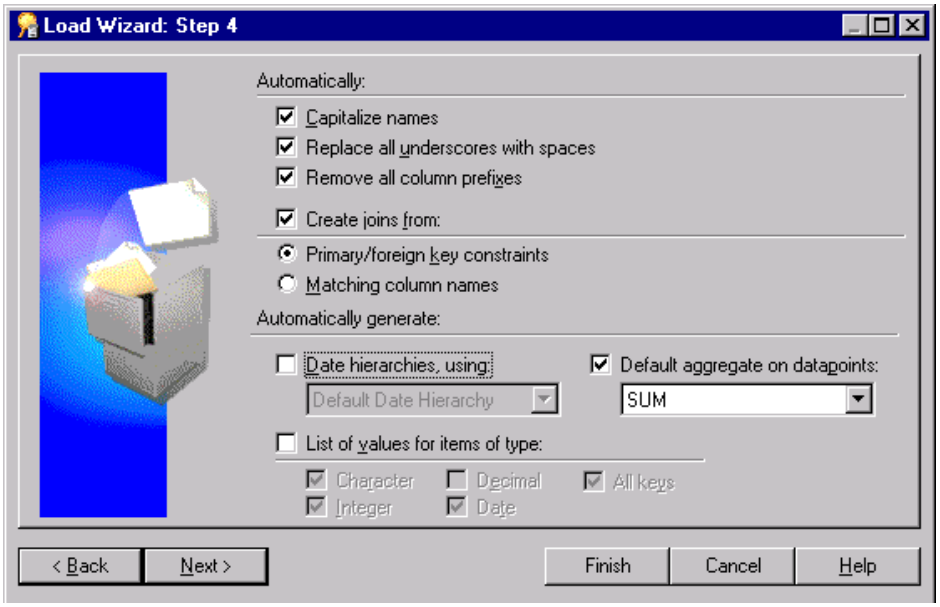
   You can actually move a name at a time, several names at a time, or all the names in one fell swoop:

   - Click one of the items on the left and drag it to the right.
   - Click one of the items on the left and click the arrow pointing right.
   - Select multiple items by holding down the Control key and clicking each item and then drag the items to the right side.

   The current example simply moves all the names to the Selected pane. All of the names that begin with 'WB_' should now be in the Selected pane.

6. Click **Next**.

Discoverer returns the next window which offers several options:



These options allow you to reformat the items during the load operation.

- **Capitalize names:** All folder names appear with initial capital letters.
- **Replace all underscores with spaces:** All folder names appear with spaces rather than underscores (WB AUDIT ERRORS instead of WB_AUDIT_ERRORS).
- **Remove all column prefixes:** Removes prefix characters that are the same for all columns (AUDIT instead of WB_AUDIT_ERRORS).
- **Create joins from:** Specifies how joins are created. Select **Primary/foreign key constraints.** With this option, joins are created if columns are related by a primary key and a foreign key constraint in the database dictionary.
- **Date hierarchies using:** All date items automatically use the date hierarchy specified in the drop-down list.
- **Default aggregate on datapoints:** All aggregates use this type as the default. Select **SUM**.
- **List of values for items of type:** Unique lists of values are generated for all items of the specified type.

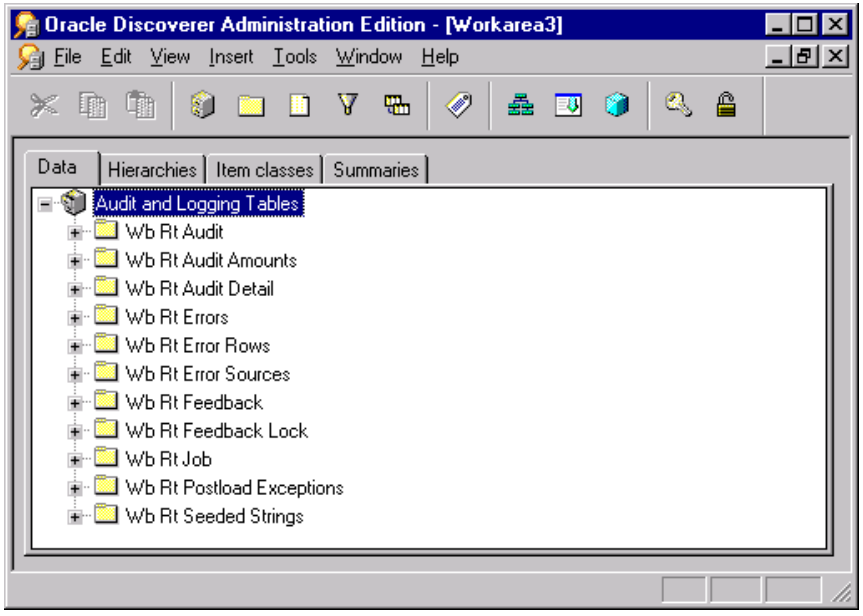
7. Click **Next**.

   The wizard returns a page that solicits a name and description for the business area. The example names the business area Audit and Logging Tables. The area is described as "Runtime tables that store audit and error information when a job is run with OEM."

8. Click **Finish.**

   The Load Wizard creates the business area and reports its progress in a small panel. When the operation is completed, the End User Layer will contain all the information that defines this business area.

After the Load Wizard creates the business area, it displays a Work Area, as shown in the following figure.



You must now grant access to users before they can access this business area. This process was described in section "Grant Access" on page 12-18.

The Data Mart Administrator can now use Discoverer User Edition to interrogate and analyze audit and logging data stored in the runtime library tables. Discoverer could also be used to generate routine management reports on the operation of the data mart.

Analyze the Data

An End User Layer now exists and it contains two business areas that simplify the contents of the data mart. Users can now use Discoverer User Edition to analyze data and generate routine reports. This section reviews the types of reports that Oracle Discoverer can generate.

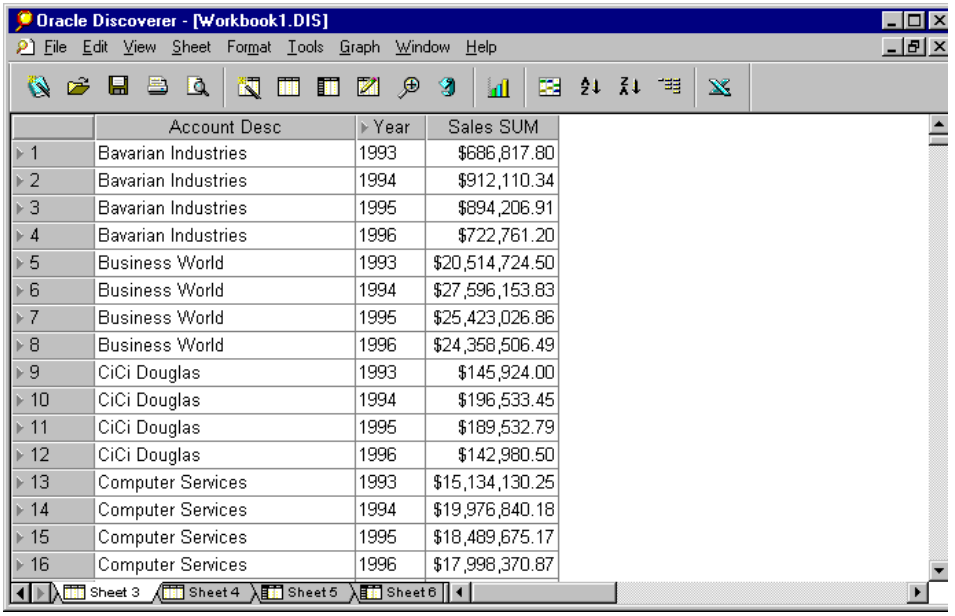
Types of Reports

Data selection and format are the keys to understanding the results. You, the business user, know the questions to ask. Understanding and utilizing the results becomes the challenge, and often it is facilitated or hampered by the structure of the report output.

First, you need to decide how you want the data displayed. Oracle Discoverer provides four report formats: table, page-detail table, crosstab, and page-detail crosstab. In addition, it lets you convert the reports to a graphical format.

Table Format

The **table format** provides a complete list of the data. For example, use the table format to list the total sales by account name by year.

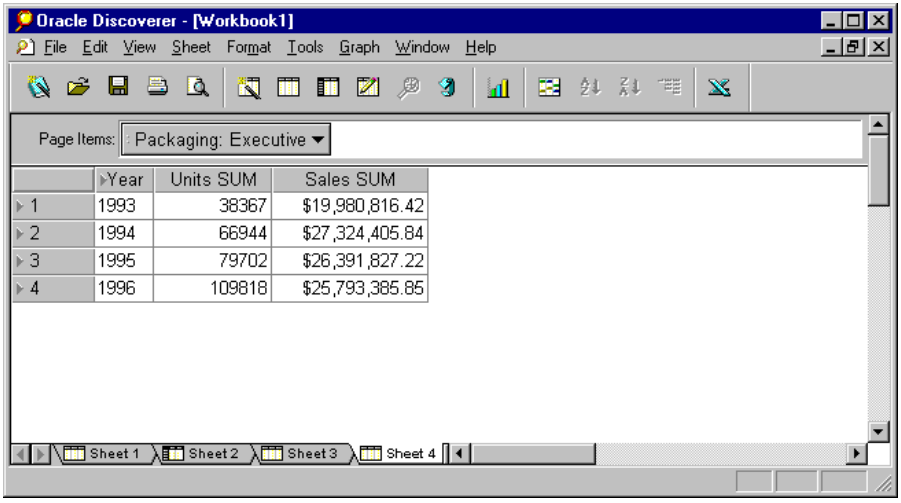


	Account Desc	Year	Sales SUM
1	Bavarian Industries	1993	$686,817.80
2	Bavarian Industries	1994	$912,110.34
3	Bavarian Industries	1995	$894,206.91
4	Bavarian Industries	1996	$722,761.20
5	Business World	1993	$20,514,724.50
6	Business World	1994	$27,596,153.83
7	Business World	1995	$25,423,026.86
8	Business World	1996	$24,358,506.49
9	CiCi Douglas	1993	$145,924.00
10	CiCi Douglas	1994	$196,533.45
11	CiCi Douglas	1995	$189,532.79
12	CiCi Douglas	1996	$142,980.50
13	Computer Services	1993	$15,134,130.25
14	Computer Services	1994	$19,976,840.18
15	Computer Services	1995	$18,489,675.17
16	Computer Services	1996	$17,998,370.87

Page-Detail Table Format

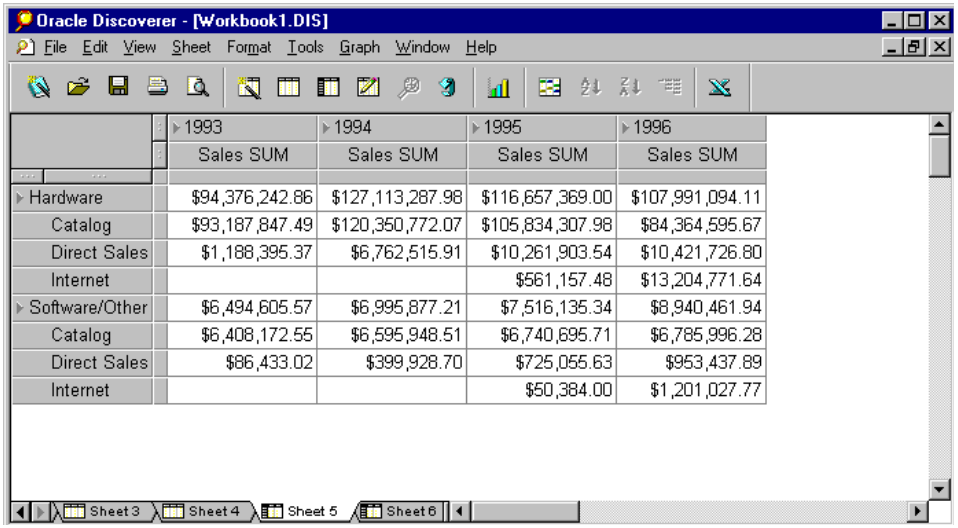
The **page-detail table format** adds a paging dimension, where one dimension becomes the header for the table. For example, you could add the customer's city to the page dimension. This allows users to cycle through the data, viewing their customers in each city, one city at a time.

The following page-detail table adds Packaging to the page dimension. Each page reflects the packaging option.



Crosstab Format

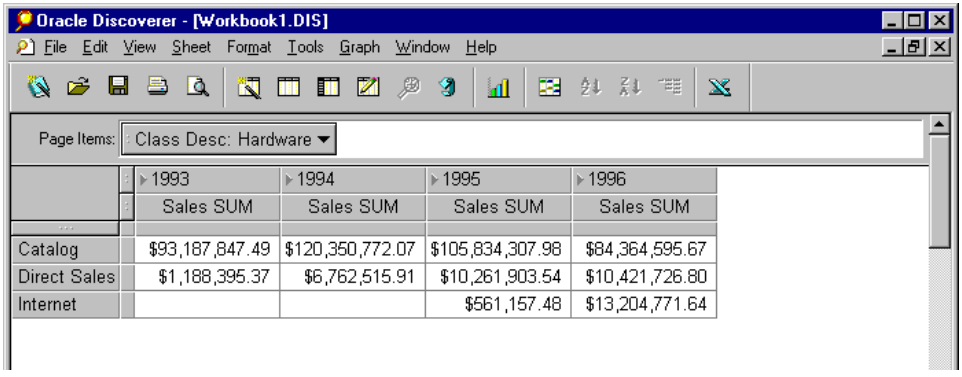
The **crosstab format** is most commonly used to analyze data. In the crosstab format, dimensions appear along either axis, and the facts appear in the intersection. You can arrange multiple dimensions along either axis by grouping one within another. The size of a crosstab report is dependent on the number of different values. For example, gender has two values, male and female, and year has only as many values as data. The result is a very compact way of displaying the data. The following chart shows the sales by year by product class by channel:



The key difference between crosstab format and table format is that table format has one column for each dimension and each fact, but crosstab format displays facts in the intersection of the values of the dimensions located around the axes of the page.

Page-Detail Crosstab Format

The **page-detail crosstab format** is also called master-detail crosstab format. It is a crosstab where one or more dimensions are placed at the top, above the crosstab. For each value in that dimension, you get a separate page. Essentially, this dimension becomes the Z axis, the third dimension. The page-detail crosstab format is like stapling all these pages together. The user can easily step through the pages by clicking on the pop-up list box.



Graphical Format

"A picture is worth a thousand words." Regardless of how good your report is, often a chart or graph can communicate the meaning in the data more effectively. Oracle Discoverer can instantly represent your reported data in a graphical format through various chart types, such as bar graph, area graph, and line-bar. You can display the graph and report simultaneously as you explore your data. The graph dynamically reflects data or orientation changes you make in the report.

Modifying Formats

Oracle Discoverer permits easy switching between various data report formats. Try different reporting formats by using the buttons on the toolbar to change to table or crosstab format.

Regardless of how you display the data, it is often easy to miss the outliers in your data. An **outlier** is an unusually large or small value. Scanning large lists of data can be difficult and error-prone. Oracle Discoverer can highlight these outliers with exception formatting. For any fact, you can define a range of values or a condition such as "greater than," and choose a different format to highlight values that meet this condition. The different format options include font, style, and color.

After you create a crosstab, you may want to change the orientation of the data for greater readability. You can do this by **pivoting**, which is moving a column heading to become a row heading, or vice versa. Pivoting aids in analyzing your data and makes the layout of the report so flexible that it really does not matter where you start.

You can also pivot within the same axis. For example, instead of showing the data by year, subgrouped by quarter, you can pivot to show quarter, subgrouped by year.

Drill Down and Drill Up

A drill-down is a navigation operation. **Drill-down** allows display of a lower level of detail. For example, if you are looking at Total Sales for a product for all of the United States and you want more detail, you can drill down to show sales by North, South, East, and West regions. An additional drill-down gives even more detail, such as states or cities.

Drilling requires that a hierarchy be defined. For example, the relationships among United States, Region, and State must be specified. The data administrator manages this definition through Discoverer Administration Edition.

A **drill-up** (or roll up or collapse) is the reverse of a drill-down. It is the return to the higher summary level of data within the hierarchy. Using the geographic example, states drill up to regions that drill up to Total United States.

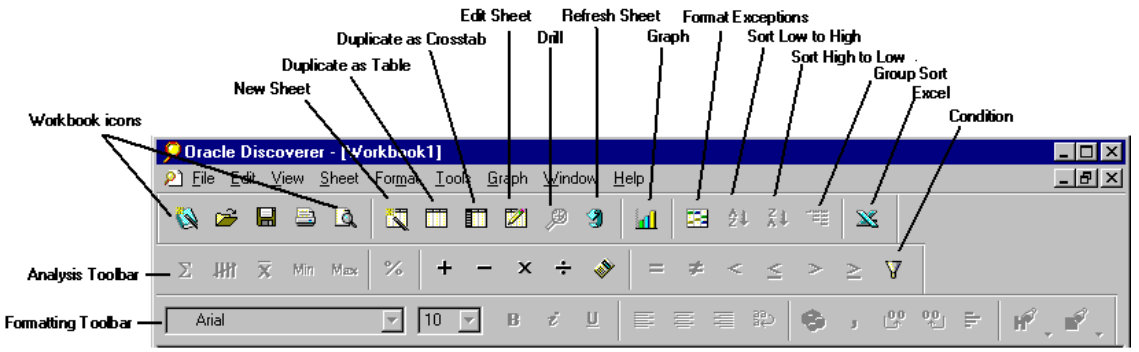
Investigate the Data

Ad hoc querying is an iterative process. Oracle Discoverer wizards aid in the creation and refinement of queries. To investigate your data, you:

- Build a table or crosstab using the wizard
- Modify a query layout using the wizard
- View the results in the results window
- Modify the results in the results window or using the wizard
- Do it again as needed

Besides the wizards, Oracle Discoverer has a unique set of toolbars designed to make your journey through your data easy and efficient. All functionality offered through a toolbar is also available through the menu.

The following figure shows the toolbar of Discoverer User Edition. The buttons unique to Oracle Discoverer are labeled.



To display all toolbars, select them from the View menu.

Where To Begin

The Workbook Wizard steps you through the selection and formatting of the data you wish to investigate. The process follows this order:

1. Select the report output format.
2. Select the items to include in the query. Think of the query as "Give me [fact] by [dimension] by [dimension]."

First, select the facts, then select the dimensions. Start with a few dimensions and add more as needed. It is usually easier to start with a little data and drill down into more data rather than going the other way.

3. Review the layout created by Oracle Discoverer. You can rearrange the data in the report before the query executes.
4. Apply any filters or conditions (scope) or select only the data that you want to view. For example, to study 1996 sales activity, limit the Days dimension to 1996.
5. Add any calculated items to your query. Oracle Discoverer displays all items available for calculation. Select the items, create the expressions, and format the output.
6. Click Finish. Oracle Discoverer executes your query and returns your requested data in a report format.

Access the Data Mart

You are the new marketing manager at Global Computing. You must find opportunities to support new business and to expand upon current trends. You need to put together a marketing plan for the year. You have looked at reports, but now you have a data mart that holds Global Computing sales for the past four years. You decide to explore the data using Oracle Discoverer.

To begin, log on and connect to the database:

1. From the Oracle Discoverer program group, select **User Edition**. If the Welcome screen appears, select **Start**.
2. For Username, enter **gccDM**.
3. For Password, enter **gccDM**.

4. For Connect, enter ***gccdm**NC**.us.oracle.com***.



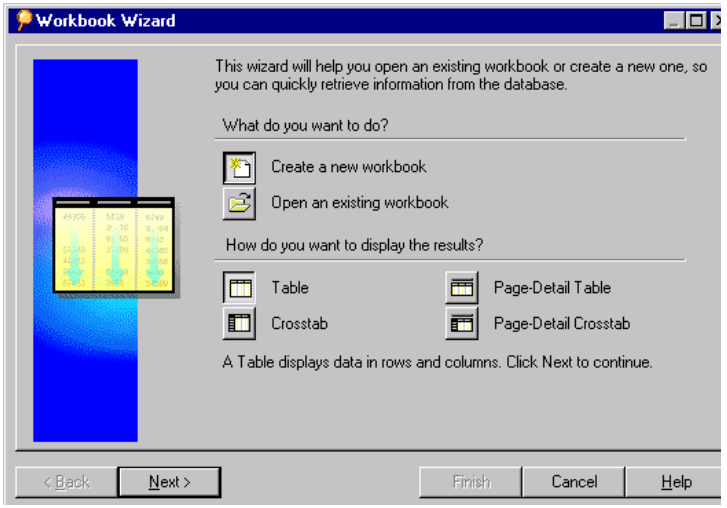
5. Click **Connect**.

Create a Report

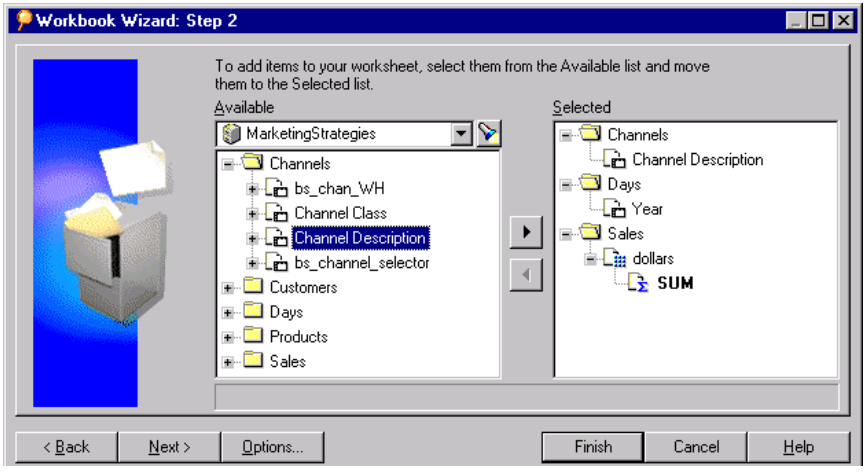
When you execute a query, Discoverer stores the output report in a worksheet. One or more worksheets and their queries comprise a workbook.

When you log in to Oracle Discoverer, the Workbook Wizard starts. With the help of the wizard, you can create a workbook for your reports:

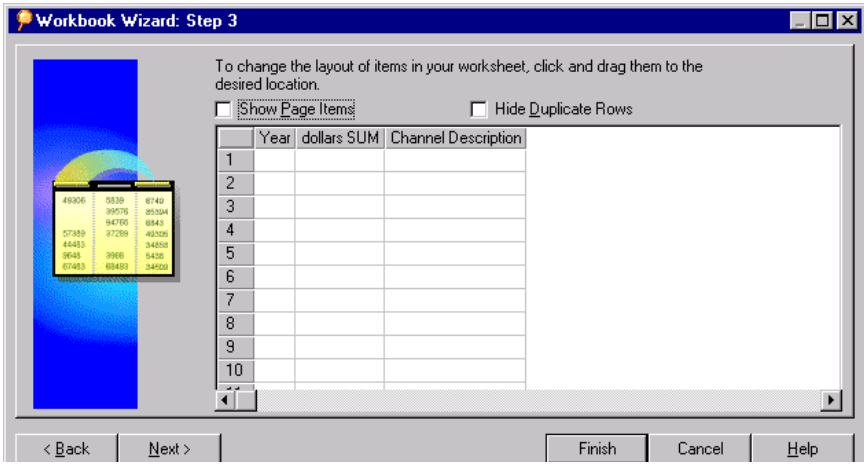
1. From the Workbook Wizard window, choose **Create a new workbook**.
2. To choose the report output format, select **Table**.



3. Click **Next**. The wizard returns a window which solicits items for the query.
4. Select ***MarketingStrategies*** from the Available list as the business area.
5. Select the following items by expanding each folder, then dragging the items from the **Available** to the **Selected** pane.
   - Select ***Dollars*** from the Sales folder.
   - Select ***Year*** from the Days folder.
   - Select ***Channel Description*** from the Channels folder.



6. Click **Next** to review the layout created by Oracle Discoverer:



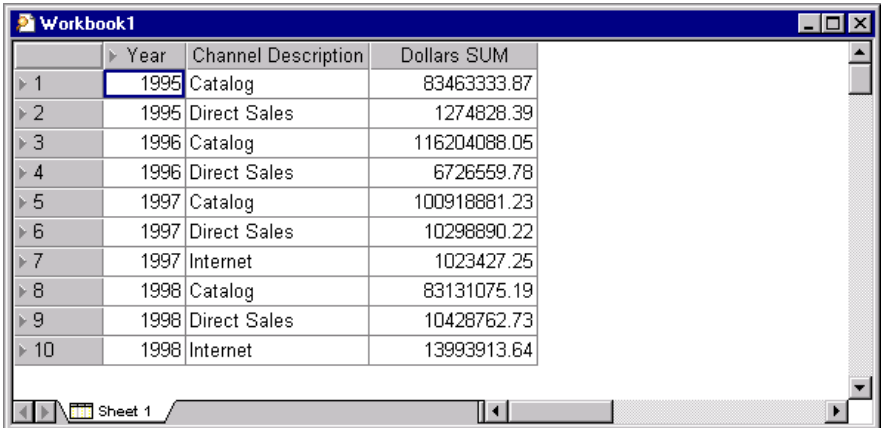
7. Looking at the order in which Oracle Discoverer has arranged your selected data, you decide that it would be helpful to have the year in the first column, and the channel description in the second, and Sales SUM in the third.

   Drag the **Sales SUM** header over the **Channel Description** header.

8. Click **Finish**.

The query predictor estimates the amount of time it will take to execute the query. At first, the estimate may be much higher than the actual time. The query predictor refines its estimate as it progresses in executing the query.

The report looks similar to the following figure:



You have your first report! You can make it more informative by reformatting it and adding subtotals.

Reformat a Report

To reformat the width of the columns, position your cursor in the column header between columns, so that it displays arrows pointing in both directions. Drag your cursor to make the column wider or narrower.

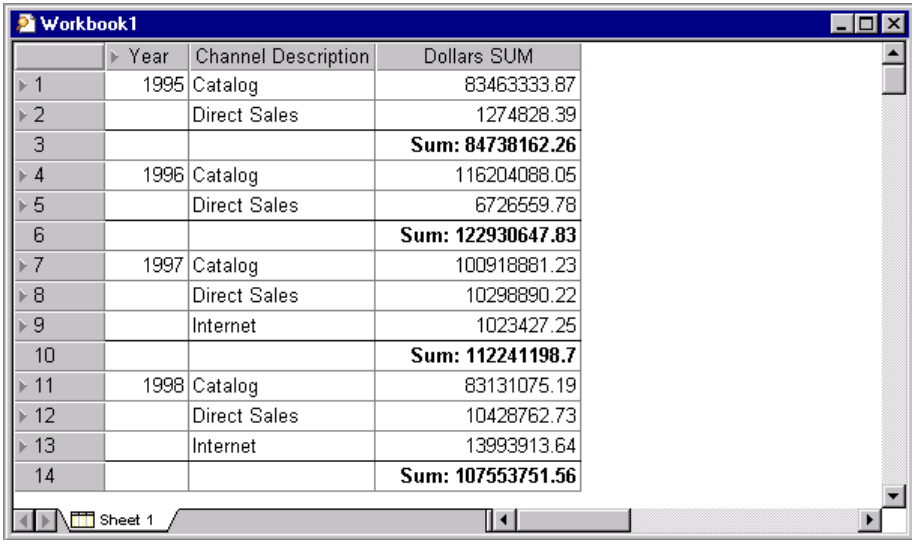
You can also reformat the width of columns by selecting **Columns** and then selecting **Auto-Size** from the Format menu.

You decide to add subtotals for each year:

1. Select **View** and then click **Analysis Bar**.
2. To suppress the redundant display of the year, select the year column name and click the **Group Sort** button from the toolbar.
3. To add the subtotals, select the column header for the year column and click the **Sum Total** button (the first button on the Analysis toolbar.)

   Oracle Discoverer adds subtotals automatically. The subtotal inherits the format of the column.
4. To format the subtotals for clarity:
   a. Select a Subtotal value.
   b. From the Format menu, select **Data**.
   c. Select the **Font** tab, select **Bold,** and click **OK**.

Your report should now resemble:



	Year	Channel Description	Dollars SUM
1	1995	Catalog	83463333.87
2		Direct Sales	1274828.39
3			**Sum: 84738162.26**
4	1996	Catalog	116204088.05
5		Direct Sales	6726559.78
6			**Sum: 122930647.83**
7	1997	Catalog	100918881.23
8		Direct Sales	10298890.22
9		Internet	1023427.25
10			**Sum: 112241198.7**
11	1998	Catalog	83131075.19
12		Direct Sales	10428762.73
13		Internet	13993913.64
14			**Sum: 107553751.56**

Immediately, you notice the trend in overall sales. Catalog sales peaked in 1996 and are now on a downward trend. Growth has slowed in the Direct Sales channel over the last year though the trend is still upward. And Internet sales have exploded!

Create a Page-Detail Crosstab Report

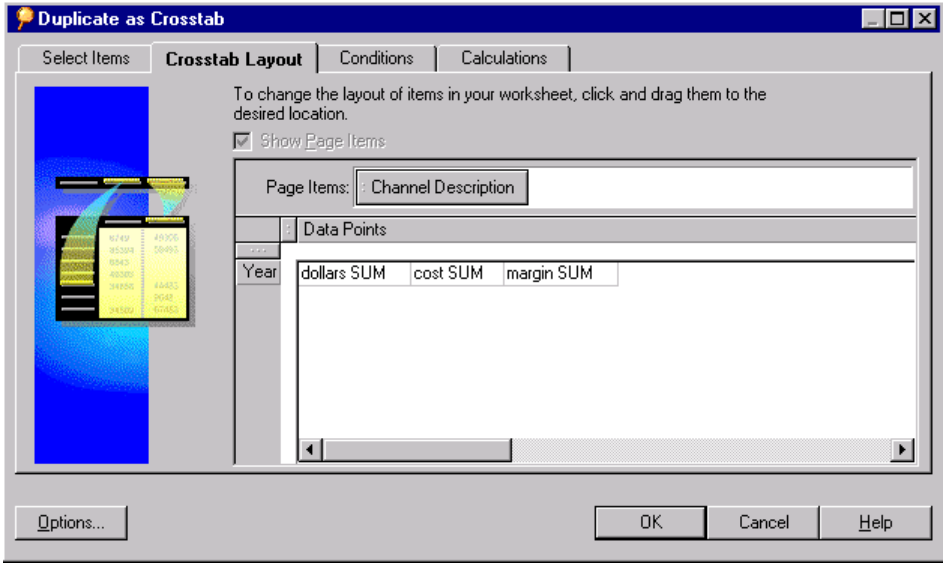
To create a page-detail crosstab report, take these steps:

1. From the Sheet menu, select **Duplicate as Crosstab** (or select the Duplicate as Crosstab button from the toolbar).

   Discoverer reports that no data can be displayed because there are no rows.

2. Click **OK**.

   The Duplicate as Crosstab window appears, with a default format. To move an item to another location, select its label and drop it on the desired location.



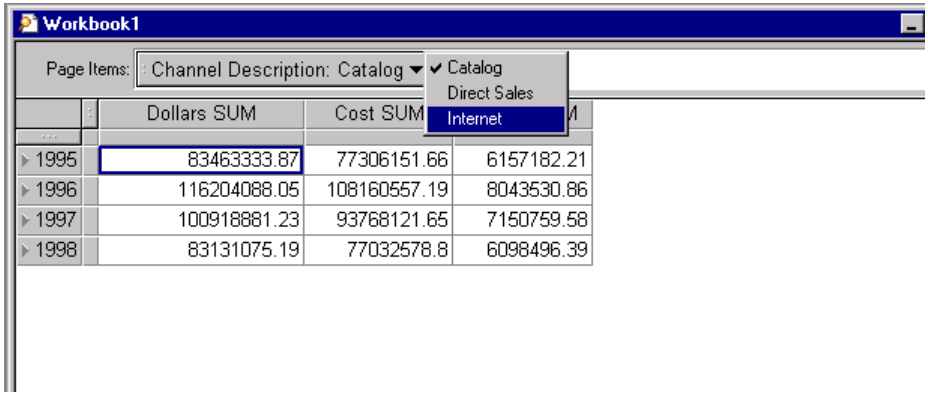
3. To view only 1997 and 1998, select the **Conditions** tab.

   Enter a condition on time where the year equals 1997 or 1998. In Oracle Discoverer language, this is: `Year IN '1997','1998'`.

   1. Select ***Year*** from the View Condition drop-down list, and then click **New**.
   2. Clear **Generate name automatically** and enter ***97 - 98*** in the Name box.
   3. Select ***Year*** from the Item drop-down list.
   4. Select **IN** from the Condition drop-down list and then enter ***'1997','1998'*** in the Value(s) box.

4. Click **OK**, then click **OK** again.

When your results appear in the report, notice that you can cycle through the Channel Desc in the page axis.



At this point, the report shows the years as row headers. You can drill down to provide more detailed information.

Drill Down

To provide more detailed information for your report, you can drill down to quarters, by using one of these methods:

- Highlight the column or row by selecting the gray bar above or to the right of the column or row. From the Sheet menu, select **Drill**. Then, select **Drill Up/Down** and select an item.
- Use the right mouse button to right-click the column or row. From the pop-up menu, select **Drill**. Then, select **Drill Up/Down** and select an item.
- Highlight the column or row. Move the cursor to the triangle next to **1998** until it appears as a magnifying glass and then click. Choose an item from the drop-down list.

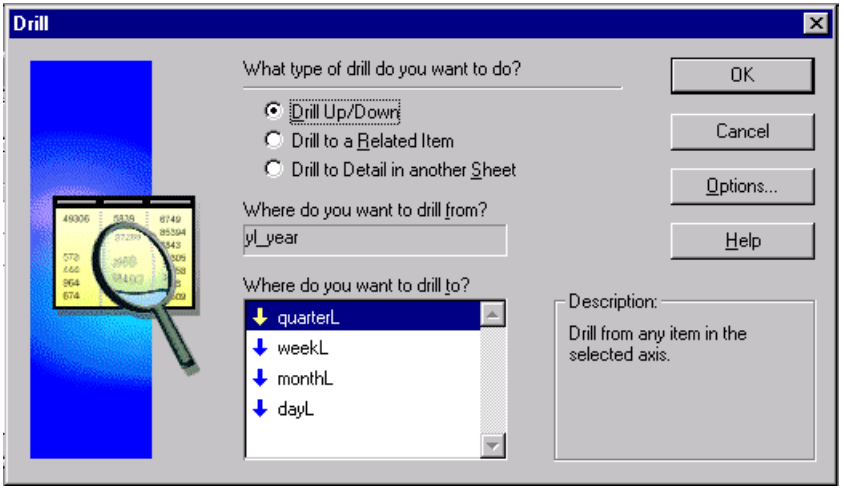
This example uses the first option.

1. Select the gray bar above the year column.

   Discoverer highlights each entry in the Years column.

2. Select **Drill** from the Sheet menu on the toolbar.

   Discoverer returns the Drill window.



3. Select **Drill Up/Down** and **QuarterL**, and then click **Options**.

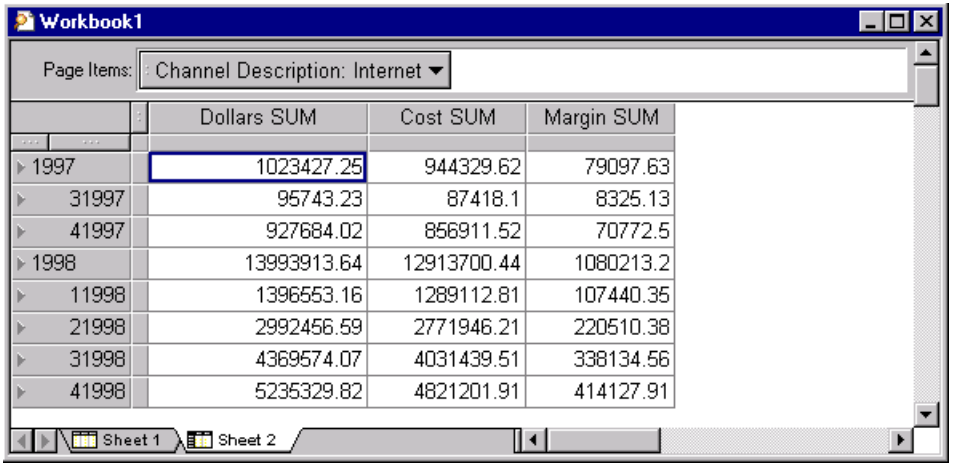
   Discoverer returns a Drill Options panel.

4. Select **Expanded to include new item** to add a column for quarters to the sheet.

   The other option replaces the existing item (Years column) with the new one.

5. Click **OK** and click **OK** again.

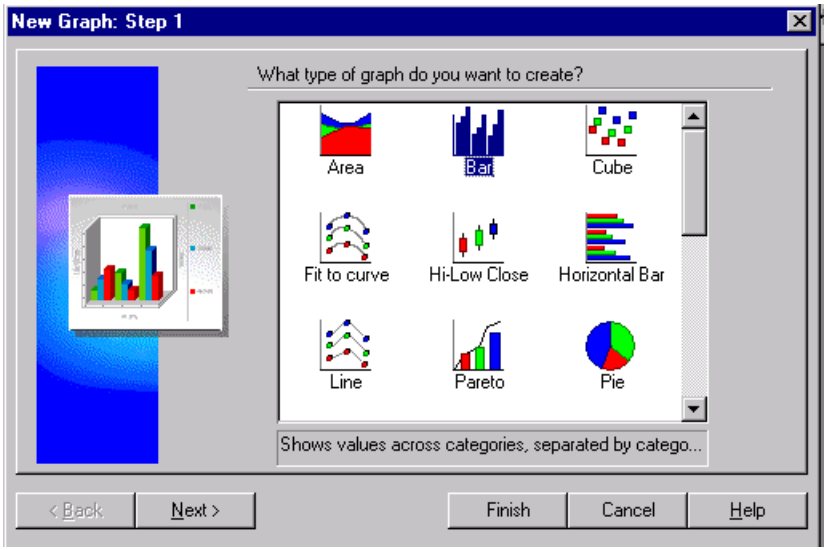
   Discoverer returns the drill down report.



Create Graphs

Whenever you are investigating trends, a graph can capture and communicate them very effectively.

To reformat the report into a graph, click the **Graph** button. Oracle Discoverer displays the New Graph Wizard.

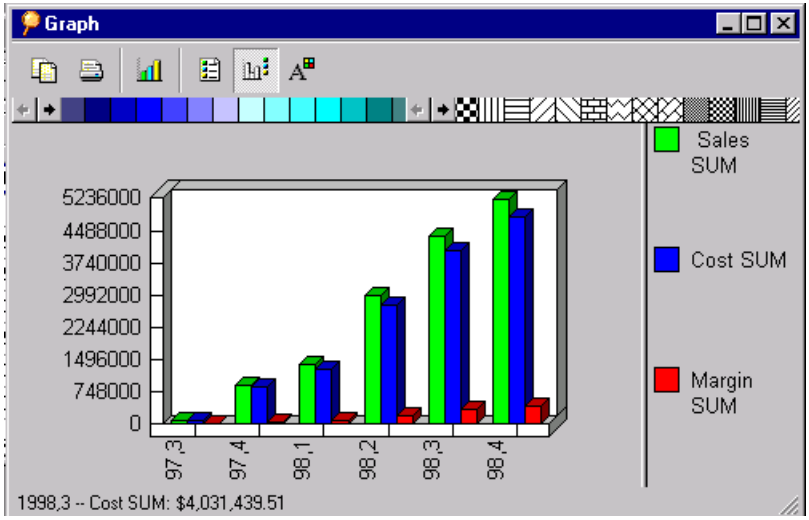


The wizard guides you in formatting your graph:

1. Choose the type of graph you want, such as a bar graph, area graph, or pie chart. Choose **Bar** and click **Next**.
2. Choose how you want to format the graph. Choose **3D** and click **Next**.
3. Click the **Show Legend** check box. You can title the top, left, bottom, and right axes, but for now, leave them blank. Click **Next**.
4. You can also format titles by positioning the cursor over the chart and clicking the right mouse button or by using the text button.
5. From the **Graph Series By** section, select **Column**.

   You can also specify maximums and minimums for the Y Axis Scale and you can pivot the axis by right-clicking and selecting Series by row (or column). Take the defaults.
6. Click **Finish**.

Discoverer returns the graph in a window.



You can format the graph in many ways.

- To change colors, drag and drop colors to different areas of the graph from the rainbow color bar across the top of the screen.
- To display the value of a particular data point on the bottom margin, move the cursor to that data point.
- To add text, click the letter button.

The graphing capabilities of Oracle Discoverer are flexible and extensive. When you reorient the underlying report and rotate the dimension values around the axis of the spreadsheet, the chart automatically reflects those modifications.

The example chart shows the sales trend very clearly over the last two years, highlighting the downward trend in the Catalog channel. You want to leverage the growth in sales through the Internet, while taking advantage of catalog sales. To develop an effective strategy, you need to know more about who is buying through the Internet and what they are buying. Is Global Computing making money from those items? In other words, how is the margin by product? Are there particular industries or market segments in which the company should invest? Are there any segments that Global Computing should completely ignore?

Now that you have been introduced to the basic capabilities of Oracle Discoverer, you can continue to explore the data and reach your own conclusions as you build the marketing plan.

Drill to Detail

Sometimes you want to look at a very low level of detail. For example, once you have identified a customer whose purchases have decreased, you might want to call the purchasing agent. You may need to access the specific account information, such as the purchasing manager and the correct phone number. Oracle Discoverer allows you to drill to this detail.

Drill Out

Oracle Discoverer lets you **drill out,** calling different applications to display various supporting documents. You can view the Bill of Materials for a product by looking at a Microsoft Word document. Similarly, you can navigate to your online Web page where you can see how the product is viewed on the Web, or to any other Web site on the Internet.

Custom Formats

You can alter the format of the heading and the data that appears in the report.

- To rename the sheet, double-click the **Sheet** tab at the bottom of the report.
- To format the headings:
  1. Select a range of columns by highlighting the heading of the first column, hold the Shift key, and then select the last column in the range.
  2. From the Format menu, select **Headings**.
  3. In the Font tab, and select Font **Times New Roman**, Style **Bold**, and Size **14**.
  4. Select the **Background Color** tab and select the white box.
  5. In the Alignment tab, select **Wrap text**.
  6. Click **OK** to apply the formats and to close the dialog box.
- To format the data, including font style, color, and size:
  1. Select the data.
  2. From the Format toolbar, select **Data**.
  3. Set the Font to **Arial**, the Style to **Bold**, and the size to **12**.
  4. Choose navy blue as the text color.
  5. Select the **Background Color** tab and choose a light gray.

Publish Your Work for Others to View

You can export worksheets and workbooks to a variety of formats for others to view. You can publish Oracle Discoverer query results on the Web and use them in other applications.

Export to Excel

An Excel button resides on the toolbar in the reporting module. By clicking on it, you copy the data from the sheet you are viewing into Microsoft Excel.

Export to HTML

From any worksheet you can export a sheet or your whole workbook to the Web by creating a hypertext markup language (HTML) file:

1. From the File menu, select **Export**.
2. Choose **All Sheets**.
3. Choose the **Hyper-Text Markup Language (*.htm)** export format.
4. Select a directory in which to save the export files.
5. Select **Finish**.

The resulting files are named using the following convention:

```
workbook_name_sheet_name.htm
```

If you have a Web browser installed on your machine, you can view these files by double-clicking on them from your browser or by typing the path and the file name in the Address or Location line of your Web browser.

The files are linked to each other, letting you click from one sheet to another.

Note:

You should apply word wrap to all of your headings if you use Netscape Navigator as a Web browser.

Export to Other File Formats

Similarly, you can export to other file formats, such as Lotus 1-2-3 and Adobe Acrobat PDF.

Schedule a Workbook

You can schedule a workbook to run at regular intervals. For example, to take advantage of the lighter load on weekends, you can schedule a workbook to run every Saturday at 11:00 PM.

To schedule Sheet1 of Workbook1:

1. Open the workbook from the User Edition.
2. From the File menu, select **Schedule**.
3. In the Schedule Workbook Wizard, select the sheet to schedule (***Sheet1***).

You can select all of the sheets in the workbook by clicking **Select All**.

4. Enter the time and date you want the sheet to run.
5. To specify how often to run this worksheet, select **Repeat Every**. Enter **1** in the first box and select **Weeks** in the second box.
6. Click **Next**.
7. Enter a name (***Sheet1***) and a description (***Weekly Run---Sheet 1***) for the scheduled worksheet.
8. Enter how long you want to save the results of the run.
9. Click **Finish**.

Because a scheduled report runs on the server, you do not need to leave your client machine running overnight (or whenever you schedule the report to run). The results of running the scheduled report are saved on the server and, therefore, are available when you log on to the server and start Discoverer.

You can view the scheduled workbooks and the results of those workbooks by selecting Open from the File menu and selecting Scheduling Manager. Then, select the workbook you want to view.

The Next Step

This chapter demonstrated some of the features of Oracle Discoverer, both for administrators and end users.

The data administrator is responsible for hiding the complexity of the database from the end user through the creation and design of a business area. In this chapter, the administrator learned how to:

- Define the business visualization of the data
- Control user access
- Define drill paths
- Define summary tables

The end user learned how to:

- Define and obtain business-specific data from the database
- Develop ad hoc report queries and reports
- Change reports to focus on the data relevant to the business
- Schedule workbooks to run at regular intervals

Chapter 13, " Manage the Data Mart," shows you how to manage the data mart. This includes refreshing the data mart on a periodic basis and looking after query performance.

13

Manage the Data Mart

This chapter gives you a few tips on how to manage the data mart. This includes showing you how to develop a set of jobs to periodically refresh the data mart, how to schedule those jobs in the correct sequence, how to keep an eye on the growth of physical storage, and how to keep a handle on performance. Several of these tasks are illustrated using Oracle Enterprise Manager.

This chapter covers the following topics:

- The Basic Steps
- Periodic Updates
- Schedule Jobs
- Physical Storage Growth
- Data Mart Performance
- The Next Step

The Basic Steps

The data mart is now built and initially loaded with three years of historical data. You can now turn your attention to several tasks required to run the data mart.

These tasks include the following:

- Develop jobs that periodically refresh the data mart
- Schedule jobs that refresh the data mart
- Manage the growth of physical storage
- Manage the performance of the data mart

The next few sections summarize how to carry out these tasks.

The first section shows you how to develop a set of jobs that periodically refresh the data mart. One set of jobs refresh the dimensions and the other applies updates to the fact table. You must select one refresh strategy from several available, create definitions for the jobs that implement the strategy, generate and deploy the jobs, and then register the jobs with the Enterprise Manager Job Library.

The first section shows you how to implement a simple refresh strategy which will work for those few cases that do not demand a complete accounting of the historical record. This case represents a straightforward task but is inadequate for many implementations. Unfortunately, refresh strategies that preserve the historical record present a great challenge. For that reason, a strategy that preserves the historical is described not in this but in the next chapter.

After you develop the refresh jobs, you can move on to the more mundane task of scheduling these jobs. You can schedule individual jobs manually in the proper sequence, or you can turn to Oracle Workflow to automate job scheduling. The second section summarizes how.

Most data marts grow over time, and the third section shows you how to monitor the growth of physical storage and add storage to manage the growth using Oracle Enterprise Manager.

The concluding section covers a few tasks that bear on the overall performance of the data mart. This includes materialized views, query rewrite, and the initialization parameters for the Oracle8*i* database instance.

Periodic Updates

This section shows you how to develop a set of mappings that periodically refresh the data mart so that it contains current as well as historical facts. Data marts are commonly refreshed in temporal increments such as daily, weekly, or monthly increments. The mappings developed in this section refresh the case study data mart on a monthly basis though you can easily change them to cover other periods.

Data Mart Refreshes

Two kinds of jobs are required to refresh a data mart: those that append new data to a fact table, and those that refresh dimension tables.

A mapping that applies monthly updates to a fact table appends a row to the table for each purchase made during a specific month. The fact table thus grows a month at a time, and the mappings that define the operation are easy to define.

As the fact table grows, administrators often remove aged data from the fact table to keep the table's size within manageable limits. You can easily implement a rolling window approach that appends the newest month and removes the oldest by including a post-map trigger in the mapping that defines the incremental load. When you use this approach, you must be on guard for performance issues when the fact table is large.

Before you can update a fact table, however, you must first refresh its dimensions so they reflect the addition of any new products during a previous month. For example, if the company has added a new monitor (M211300-19) during the previous month, then the sales of that product can not be recorded in the fact table until the product's natural key is stored in the Product dimension. Otherwise, fact table rows which reference the new product will be rejected during a \ load operation because they violate foreign key reference constraints.

A mapping that adds new products, customers, or channels to a dimension must also account for changes to the existing descriptions. Product names, colors, and sizes change as do customer addresses, ages, and income bracket.

Refresh a Dimension

Dimensions set the historical context for measures stored in the fact table, and they should preserve this context over time as new products are added and descriptions of existing products change. For example, when the speed of the Bushwhacker CD-ROM (*item_id* P34123-E48) changes from 24X to 48X, the data mart must reflect this

change in such a way that users can easily compare sales before and after the change. Preferably, the old product description does not disappear.

Basic Methods

Ralph Kimball spelled out three basic methods for refreshing dimensions in his first book *The Data Warehouse Toolkit* and these remain the standard methods. He labeled the methods Type 1, Type 2, and Type 3, and Oracle augmented the list with a fourth more generalized method.

The original three methods all represent a new product by appending a new row to the dimension. They differ on how they manage changes to the descriptions of an existing product. The three methods manage change as follows:

Type 1 Replace the existing description with the new description. This effectively destroys any historical trace of the original description. For example, if you change the product description 24X CD-ROM to 48X CD-ROM, then no trace of the 24X CD-ROM remains.

Type 2 Add a row to represent the new description. For example, the refreshed dimension would include a row for the 24X CD-ROM and another for the new higher speed 48X CD-ROM. Existing fact table rows reference the older product, and new fact table rows reference the faster product. This method preserves the historical context and is feasible to implement *because* the dimension table includes a warehouse key column.

Type 3 Add a column to represent the new description. The dimension includes a column for the product's previous and current descriptions. Unlike the Type 2 method, this method is limited to tracking a few changes and queries can be more complex to write.

In practice, multiple methods are often used to refresh a single dimension: selected columns are subject to Type 1 updates and other columns to Type 2. Naturally, the mappings can become quite complex and difficult to implement.

The methods chosen also determine how the mappings that append data to the fact table are implemented.

Type 1 updates are fairly easy to implement; the other two methods present more of a challenge. The following example shows you how to define a mapping that uses the Type 1 method to refresh the Products dimension. You can use this as an example to develop mappings that refresh the other dimensions. The example is simple and straightforward but highlights all the important points. The next chapter shows you how to implement a refresh mapping that uses the Type 2 method. This implementation requires a few fundamental changes to the target schema.

Refresh the Products Dimension

The example update routine refreshes the Products table on a monthly basis from updates stored in a flat file. Each record in the update file completely describes a new product added during the previous month or represents an update to an existing product during the same month. In both cases, the records include a complete description. The file's records are identified by the product's natural key (product UPC), and have the same format as records in the *product**t.fix*** file.

The refresh routine reads a record from the input file and then searches the dimension table for the record's natural key. If the natural key is found, the routine replaces the each non-key column with descriptions from the update record. Otherwise, the routine generates a new warehouse key, builds a new dimension row, and then appends it to the dimension.

The refresh routine is implemented with a staging table, two mappings, and a sequence object. Each of these objects reside in the target schema.

map_prodSTAGE	Extracts rows from the flat file and stores them in a staging table. This table is just like the Products dimension except that it does not include a column for warehouse keys.
refresh_Products_type_1	Extracts rows from the staging table and refreshes the dimension.

The staging table simplifies and generalizes the operation. The example assumes that the update file resides in the directory named by the *GCCFILES* source module.

Create and Load prodSTAGE The staging table is the same as the Products dimension except that it does not include a column for warehouse keys. The example uses the Create Like wizard to create definitions for the staging table and its mapping.

You can create definitions for the mapping and the table in a few steps:

1. Select **Mapping > Create Mapping**

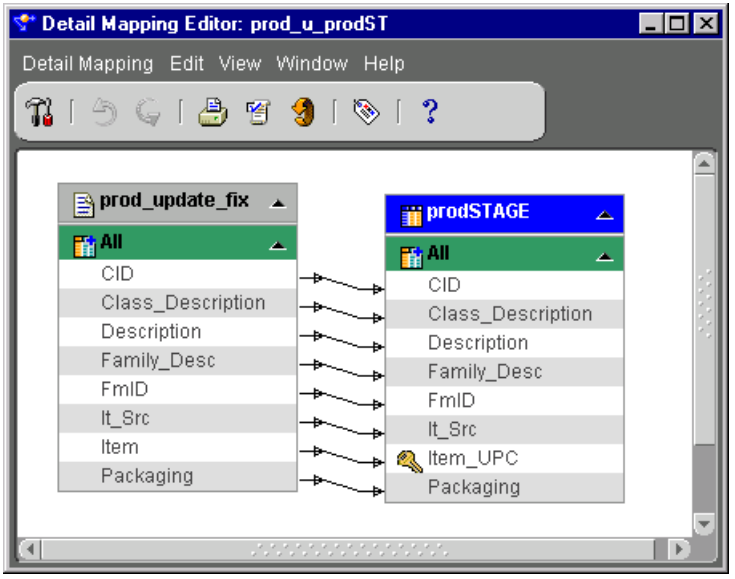
   Builder returns the New Mapping Wizard.

2. Use the New Mapping wizard to create a high-level mapping (*map_prodSTAGE*) that names *GCCFILES* as its source and *gccDM* as its target.

3. Open the high-level mapping and import *prod_update.fix* as the primary source.

4. Call the Create Like wizard to create a definition for the staging table (*prodSTAGE*) and a mapping from the update file to the staging table.

The final detail mapping is a straightforward field-to-column mapping.



Create a Sequence for the Products Warehouse Key The Products dimension was initially loaded using an SQL*Loader job. This job uses an explicitly named expression in its control file to generate values for the warehouse key column. The mapping that refreshes the Products dimension from the prodSTAGE table, however, uses a PL/SQL procedure, and this procedure uses an existing sequence object to generate warehouse key values.

Thus, you must create a definition for the sequence object, configure the definition, and then deploy the sequence to the target schema.

Use the New Sequence Wizard to create a definition for the sequence.

1. Select **Sequence > Create Sequence**.

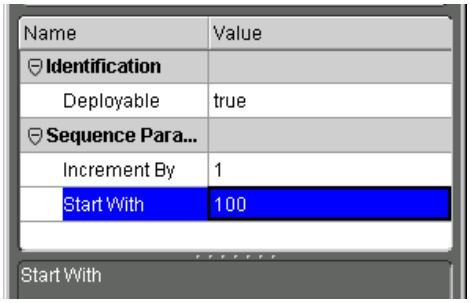
   Builder returns the welcome page of the wizard.

2. Enter a name for the sequence (*prodWHkey*) and a description, and then click **OK**.

   After you create the definition, you need to configure its initial value.

3. Switch to the Physical Tree, double-click the name of the sequence, and then enter a value (*100*) for the **Start With** parameter.

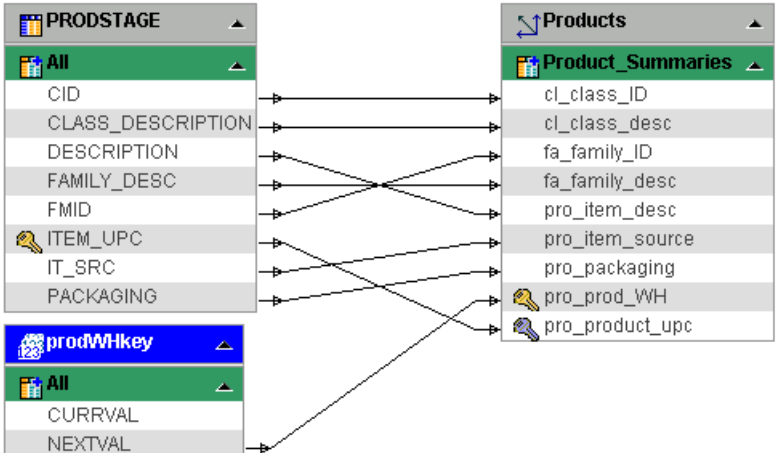
The **Start With** value must exceed the highest value now stored in the warehouse key column (36); otherwise, the update procedure will fail.



Create a Mapping to Refresh the Dimension The mapping describes how to extract data from the prodSTAGE table, generate warehouse key values, and then refresh the Products dimension.

To create the mapping:

1. Create a definition for a high-level mapping (*refresh_Products_Type_1*). This mapping names the *gccDM* warehouse module as its target.
2. Name the *prodSTAGE* table as the primary source and the *Products* dimension as the target table.
3. Open the detail mapping and map the respective columns.
4. Import the *prodWHkey* sequence into the detail mapping, and then map its output to the *pro_prod_WH* column.



5. Change the mapping's DML Type from **INSERT** to **INSERT/UPDATE**.

   The **INSERT/UPDATE** operation performs an insert command first. If the insert operation fails because the row already exists in the target table, the operation performs an update command.

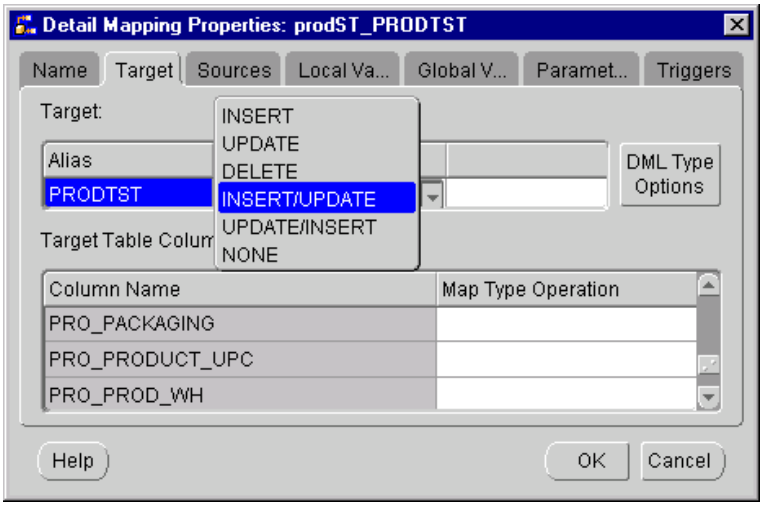
   To change the DML Type for a mapping:

   1. Right-click the editor's white space and select Mapping Properties from the pop-up list.

      Builder returns the mapping property sheet.

   2. Click the **Target** tab.

      Builder returns the a sheet that describes the target.



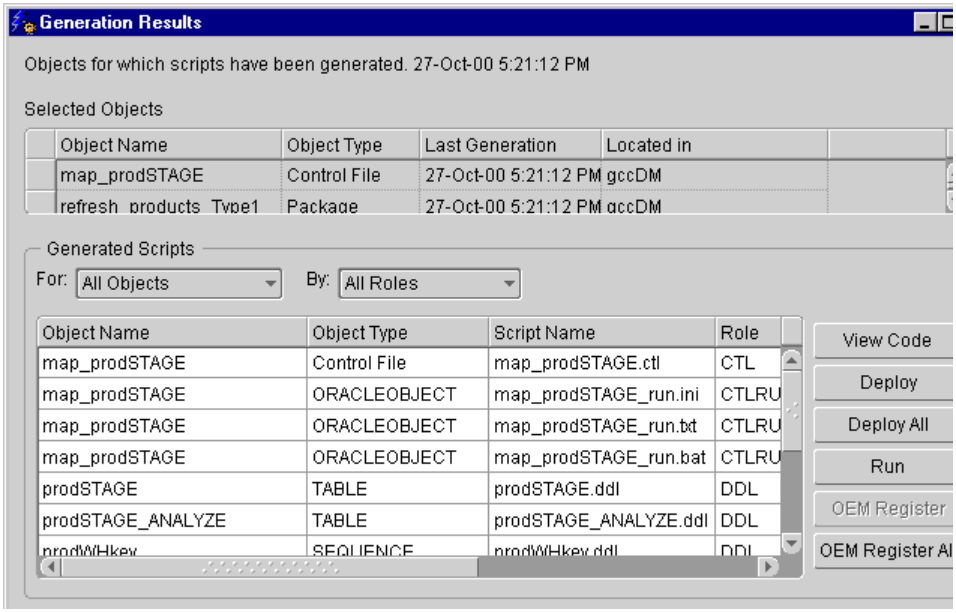
   3. Select **INSERT/UPDATE** from the down-arrow list and then click **OK**.

6. Close the Detail and Mapping Editors.

Deploy the Objects You can now generate scripts that create the staging table, the sequence object, and the two new mappings.

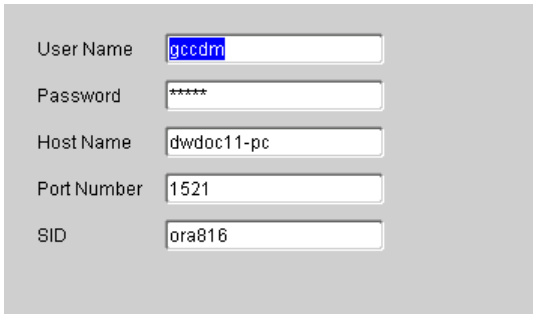
1. Switch to the Physical Tree and select the names of each object.
2. Select **Module > Generate** from the toolbar.

Builder generates scripts for the selected objects and then returns its Generation Results window.



You must deploy the staging table and the sequence before you deploy the two mappings. The easiest approach is to close the Generation Results window, select the name of the staging table and the sequence object on the navigation tree, and then display the Generated Scripts window. To display the Generated Scripts window, select **View > Generated** from the toolbar.

Builder returns a Generated Scripts window that includes scripts for the table and the sequence only. Click **Deploy**, enter connection information for the target schema, and then click **Create**.



After Builder creates the table and the sequence, you can select the names of the two mappings from the navigation tree and display the Generated Scripts window again. This time, deploy the scripts to the Top Directory and create the objects in the target schema. Then, register the mappings with Oracle Enterprise Manager.

> **Note:** You must deploy the scripts to the Top Directory and create the objects in the target schema *before* you can register the scripts with Oracle Enterprise Manager.

You can now log on to the Enterprise Manager console, open the job library, and schedule the two jobs that refresh the Products dimension. You must run the job that loads the staging table first, verify that its completion status is normal, and then schedule the job that refreshes the dimension.

After the Refresh

The refresh jobs added two rows to the Products dimension and updated the item descriptions for two existing jobs. The rows in the update file are listed below but only the natural key and the changed field are shown.

```
ITEM_UPC    DESCRIPTION
----------- --------------------------
M211300-19  Monitor- 19" LCD Flat Panel
M211300-21  Monitor- 21" LCD Flat Panel
A461245     The New Superior Mouse
P34123-E6   External 48X CD-ROM
```

You can now load data into the Sales fact table that references the two new products (the Flat Panel Displays) or the remaining thirty-six products defined by the Products dimensions.

Before you add data to the Sales fact table, it is instructive to look at its current state. A query that constrains on the product description 'The Standard Mouse' will no longer return any rows. To locate those rows, you must now constrain on 'The New Superior Mouse'. This is the expected side-effect of Type 1 updates which replace existing descriptions with new ones.

The next chapter shows you to modify the target schema and refresh dimensions without partially destroying the historical context using the Type 2 method.

Fact Table Updates

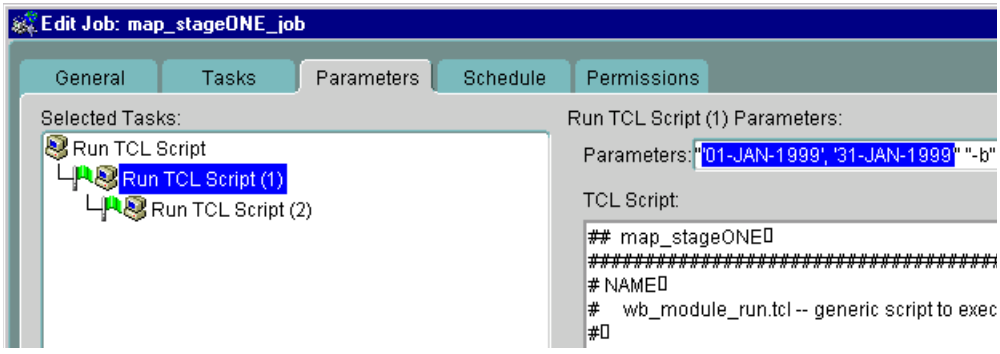
You can use the jobs that initially loaded the stageONE table and the Sales fact table to append monthly increments to the Sales fact table. When you schedule the job that loads data into the staging table, however, you must define the time increment to reflect "last month."

For the example data, edit the parameter values for the map_stageONE_job so that:

start_date '01-JAN-1999'

end_date '31-JAN-1999'

To assign these values, open Oracle Enterprise Manager Job Library, select the job that updates the staging table (*map_stageONE_job*), and the click **Edit**. Enterprise Manager returns the specific information about the job. Select the **Parameter** tab and then select the name of the second task (**RUN TCL Script(1)**).



Locate the parameter values that determine the end points of the extraction interval, and then enter new end points.

Now when you run the job, it extracts data for January from the Global Computing Company application, transforms the data, and then stores it in the stageONE table. After you check the completion status of this job, you can schedule the job that loads the Sales fact table. The two values in the mapping for the *start_date* and *end_date* parameters are key to making this work.

In practice, isolating the changed data is a much bigger problem and the issues it raises usually cuts across several organizations. Once the changed-data has been defined and captured, however, the mechanics for the incremental append of data to a fact table is straightforward. At least this is true for cases where the dimensions are refreshed using the Type 1 method. When you refresh dimensions using the other methods, developing the incremental appends requires a little more effort.

Schedule Jobs

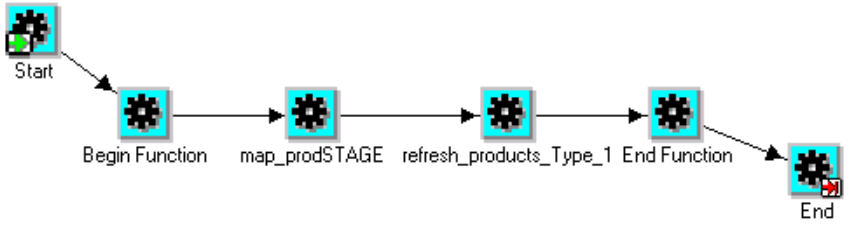
Jobs that refresh a data mart must be run in a strict sequence to insure that all foreign key references can be satisfied. Generally, this means that referenced tables must be refreshed before the tables making the references can be updated. For example, you cannot incrementally update the fact table with data that references new products and customers before you properly refresh each dimension. Otherwise, the fact table jobs will fail.

Job scheduling can be manually intensive (start a job, check its completion status, start the next job in the sequence) unless you have a tool that can manage job dependencies. You can manage dependencies for jobs that reside in the Enterprise Manager Job Library using Oracle Workflow but you must complete a few preparatory steps using Oracle Warehouse Builder and Oracle Workflow Builder.

1. Deploy the jobs generated by Oracle Warehouse Builder to the Generation Target Directories and the PL/SQL packages to the target database.
2. Register the generated jobs with Oracle Enterprise Manager.
3. Deploy the mappings as functions (identified by an Item Type and a Workflow Process name) to the Oracle Workflow server using the Warehouse Builder Workflow Deployment Wizard.
4. Define the workflow process using Oracle Workflow Builder.
5. Schedule the Oracle Workflow process using Oracle Enterprise Manager.

After you schedule the Workflow process, the Oracle Workflow engine executes the individual jobs in sequence. During the process, the engine checks the individual completion status of individual jobs before it starts any of the job's dependents. If a job abnormally terminates, the engine terminates the entire process.

The Workflow process below executes jobs that refresh the Products dimension.



For more information on managing job dependencies with Oracle Workflow, see the Oracle Warehouse Builder User's Guide.

Physical Storage Growth

Data marts tend to grow over time, especially as users become more familiar with data mart's contents and its value. The growth of data marts that contain clickstream data usually grow explosively.

To manage this growth, you will need to add physical space to the data mart's tablespaces. You can write your own scripts and execute them with SQL*Plus, or you can greatly simplify the task by turning to Oracle Enterprise Manager.

Enterprise Manager includes a suite of tools that simplify not just the management of physical storage, but the entire data mart. The following example shows you how to check the space currently used by the data mart tablespace (GCCDM) and how to increase its size using the DBA Studio, one of Enterprise Manager's many tools.

DBA Studio

DBA Studio is a set of tools that you can use to administer the Oracle8*i* database instance, database schemas, security, and physical storage. You can logon to this toolset in a standalone mode or from the Enterprise Manager console.

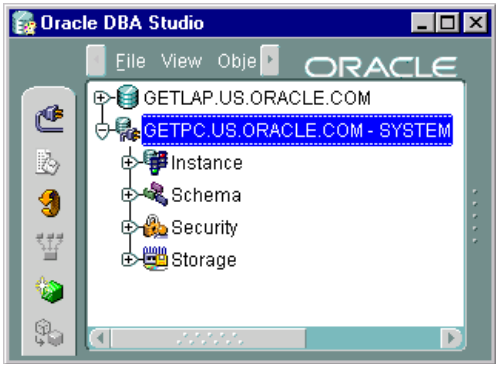
To logon on to this toolset:

1. Select **Start > Oracle > Database Administration > DBA Studio**

   Enterprise Manager returns a logon panel.

2. Select the button **Launch DBA Studio standalone** and then click **OK**.

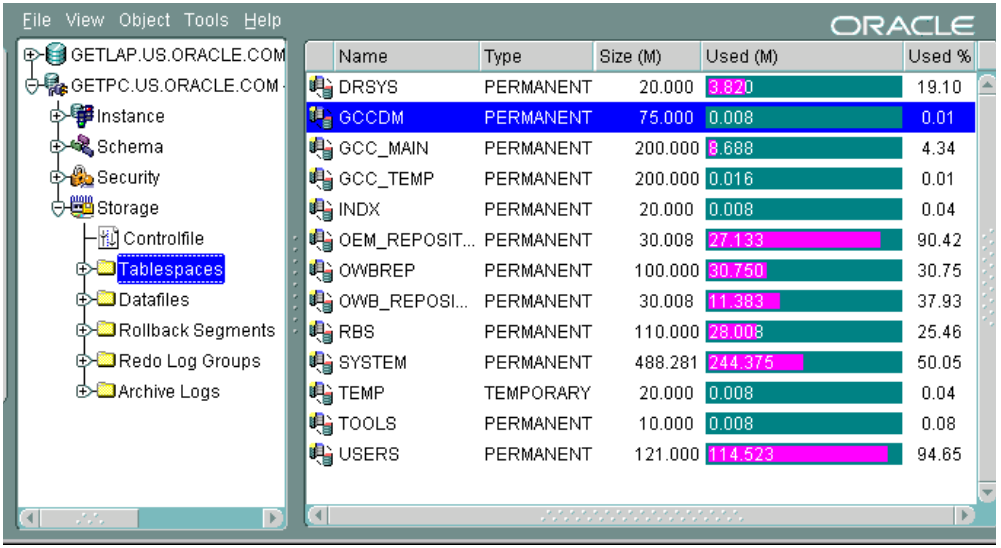
   Enterprise Manager returns its primary window. The left pane of the window contains a navigation tree that lists all the visible nodes.



To administer a database, expand the navigation tree that represents its host (*getPC*).

Space Utilization

To view the current space utilization for all of the tablespaces defined for the database, expand the **Storage** branch and then select **Tablespace**. Enterprise Manager summarizes basic information about each tablespace.



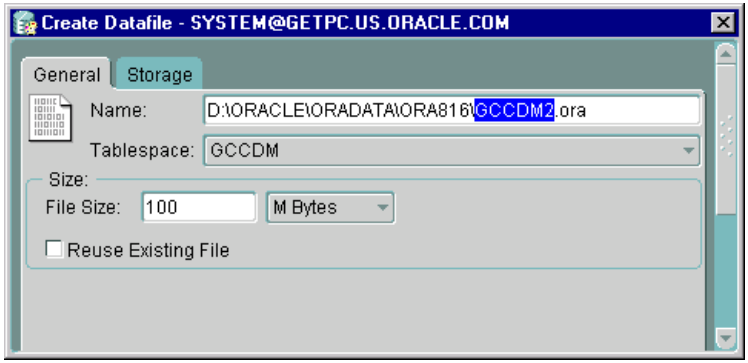
The right pane summarizes the size of the tablespace in megabytes, the total number of megabytes used, and space utilization (percentage used).

Add a Datafile

Although the GCCDM tablespace has plenty of space, the following example shows you how to increase its available space by adding another file.

To add a datafile to an existing tablespace:

1. Expand the **Tablespaces** branch
2. Right-click the name of the tablespace (*GCCDM*).
3. Select **Add Datafile** from the pop-up list.
4. Enterprise Manager returns a panel that solicits a complete pathname for the file (*D:\ORACLE\ORADATA\ORA816**GCCDM2**.ora*), the name of the tablespace (*GCCDM*), the maximum size of the file, and whether to reuse an existing file. You can also view the SQL statement that adds the file to the tablespace.

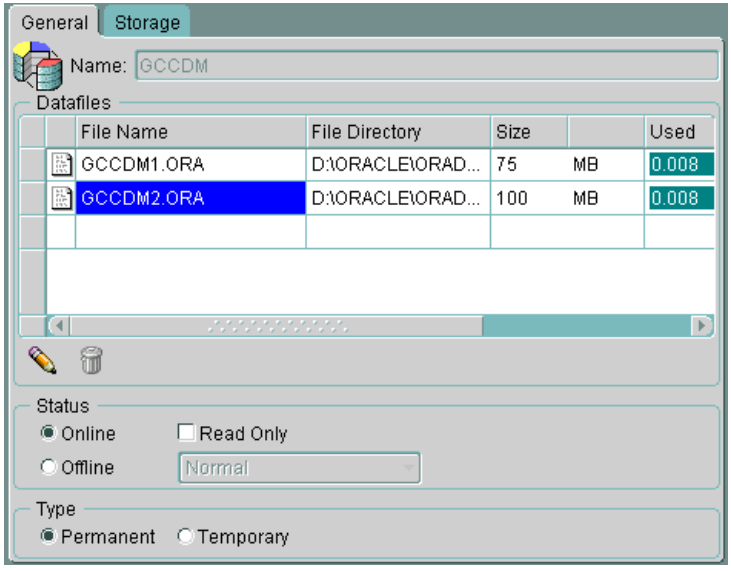


5. Enter the information and then click **Create**.

   Enterprise Manager adds the datafile to the tablespace and assigns several default values to its operating parameters.

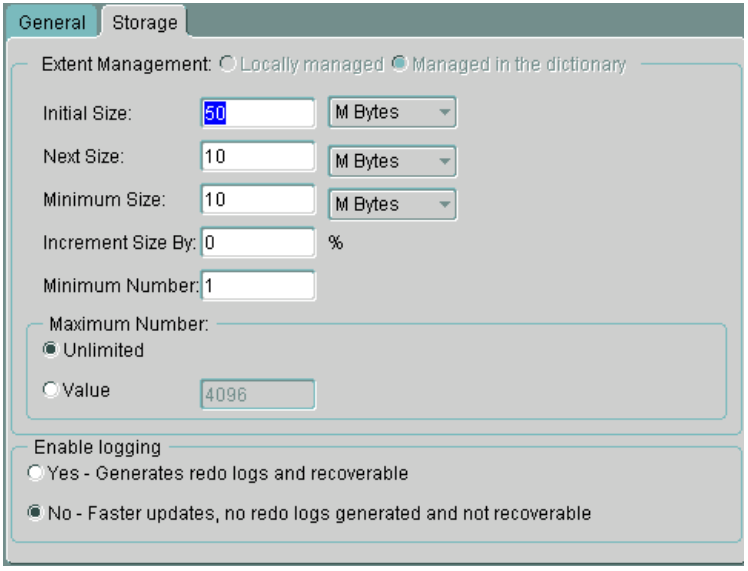
6. Select **View** from the toolbar, and then select **Refresh**.

   Enterprise Manager displays the default values assigned to selected operating parameters (**Online** and **Permanent**) and the current utilization of space:



7. Click **Storage** to display the parameters that govern the growth increments and enable logging.

Enterprise Manager returns the Storage pane.



8. Click the **No** button to disable the generation of logging, update the parameters that govern how the file grows, and then click **Apply**.

When you manage the database using DBA Studio it is usually instructive to display the generated SQL statements. Frequently, you can copy, save, and edit these scripts to make minor changes.

Data Mart Performance

The major factors that determine a data mart's query and load performance with respect to a given workload are well known and well understood. These factors are listed below in order of importance:

- Design of the target star schema
- Implementation of proper indexes
- Utilization of materialized views
- Allocation of physical storage and database parallelism
- Configuration of Oracle8*i*

While the details of star schema design and the detailed strategies that allocate physical storage are beyond the scope of this cookbook, the other factors are not. The following discussion summarize the most important topics so you will at least know what you need to look for and where. For an in-depth discussion of these topics, begin with the *Oracle8i Data Warehousing Guide Release 2 (8.1.6)* guide, and then refer to the *Oracle8i Designing and Tuning for Performance Release 2 (8.1.6)* guide.

Implementation of Indexes

Generally speaking, there are two broad classes of indexes that should be created for a data mart: those on dimensions and those on the fact table. User queries typically constrain dimensions, and indexes on dimension columns speed-up the evaluation of constraints. After the server evaluates constraints, it retrieves rows from the fact table. The correct indexes on the fact table can greatly improve the speed of the retrieval operation.

Dimension Table Indexes

An initial set of indexes can be created on the dimensions based on the expected query workload. Good candidates are those columns that are expected to be constrained the most frequently or are considered the most important (there may be a column that no one constrains except upper management).

Begin with bitmap indexes on low cardinality columns; that is, columns in which there are relatively few distinct values when compared with the total number of rows in the table. A column that contains state names is a good example. Then, create B-tree indexes on the remaining candidates. You can adjust these over time as you develop a profile of the *actual* query workload and gather evidence on which kinds of indexes yield the greatest payoff.

Fact Table Indexes

After the constraints on the dimensions have been evaluated, the *Oracle8i* server uses the constraint set to retrieve rows from the fact table. The star transformation algorithm is designed precisely for this kind of retrieval.

To take advantage of this specialized algorithm:

- Create a bitmap index on each of the fact table's foreign-key reference columns.
- Set the init.ora parameter STAR_TRANSFORMATION_ENABLED to true.
- Enable the cost-based optimizer.
- Analyze the fact and all dimension tables.

When you take these steps, the *Oracle8i* server will use a query-execution strategy called the star transformation. This algorithm is highly efficient for star queries.

As you develop your query profile and gather statistics on actual query performance, you may find a group or two of queries whose performance is unacceptable. For these groups, you can create concatenated B-tree indexes on subsets of the foreign key reference columns. The general guideline suggests that you create the concatenated index so that its leading column corresponds with the most tightly constrained column within the group of queries.

The process of finding the right set of indexes for the fact and dimension tables is a series of successive approximations, each approximation evaluated empirically.

Utilization of Materialized Views

You go through the effort and expense to create materialized views so that users can benefit from query rewrite. This specialized algorithm transforms a query that references detailed data into one that references materialized views defined on the detailed data. The materialized view contains pre-summarized data and thus greatly reduces processing time. The result is faster queries.

The transformation is transparent to the end user or application. Because query rewrite is transparent, materialized views can be added or dropped just like indexes without invalidating the SQL in the application code.

Initially, you can create materialized views according to the expected query profile. Look for queries that aggregate data through natural hierarchies. For example, queries that roll days into months, product codes into families. or product families into classes. After your data mart operates for awhile, you can use the Oracle Enterprise Manager Summary Advisor and the DBMS_OLAP package to make intelligent adjustments to your set of materialized views.

Query Rewrite

Before a query is rewritten, it is subjected to several checks that determine whether it is a candidate for query rewrite. If the query fails any of the checks, then the query is applied to the detail tables rather than the materialized view.

The Oracle optimizer uses two different methods to recognize when to rewrite a query in terms of one or more materialized views. The first method is based on matching the SQL text of the query with the SQL text of the materialized view definition. If the first method fails, the optimizer uses the more general method in which it compares join conditions, data columns, grouping columns, and aggregate functions between the query and a materialized view.

To Enable Query Rewrite To enable query rewrite for your data mart, you must take the following four steps:

1. Include the ENABLE QUERY REWRITE clause on individual materialized views.
2. Set the init.ora parameter QUERY_REWRITE_ENABLED to true.
3. Enable the cost-based optimizer by configuring one of the init.ora parameters:
   - OPTIMIZER_MODE to ALL_ROWS or FIRST_ROWS, **or**
   - Analyzing the tables and set OPTIMIZER_MODE to "CHOOSE"
4. Set the init.ora parameter COMPATIBLE to 8.1.0 or greater.

The mapping definitions used to generate the PL/SQL packages that load the data mart can be configured to analyze tables every time you perform a load operation with the script. See the mapping configuration parameter ANALYZE TARGET.

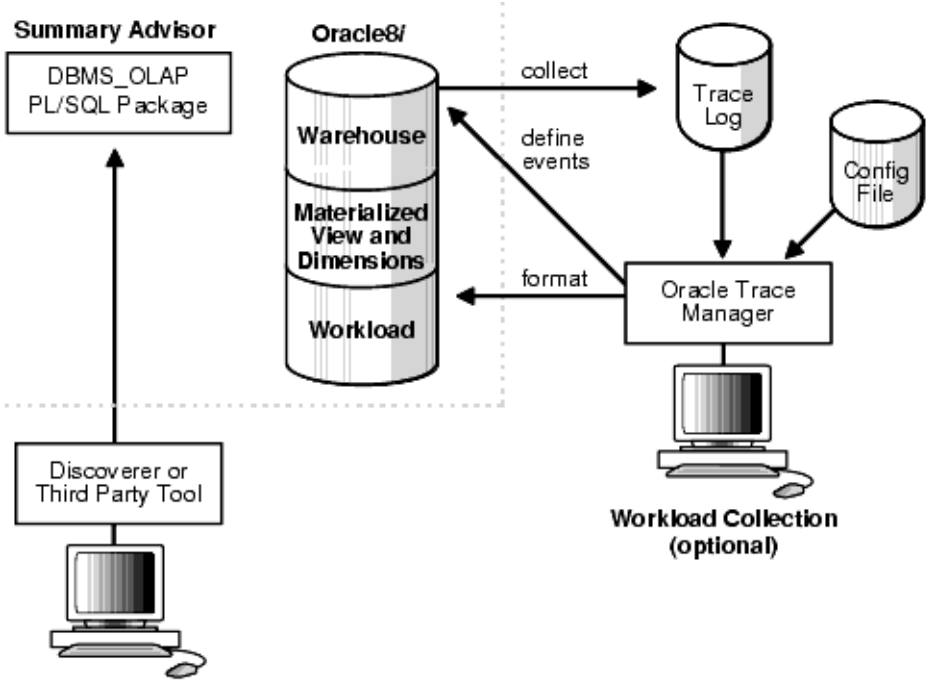
You can also configure an individual definition for a materialized view so that query rewrite is completely enabled for the view (default value). Open the configuration panel for a materialized view, and look for the QUERY REWRITE parameter. You need to set this parameter for each materialized view.

Dimension Table Columns To gain full benefit of query rewrite, insure that all columns within a dimension are declared not null. This is a fundamental design maxim for star schemas independent of materialized views. When you allow a dimension column to be valueless for some rows, you run the risk of inconsistent calculations. When source values are missing, you can insert a constant value ('Unknown').

Design Note: Declare all dimension columns NOT NULL to prevent inconsistent calculations and maximize query rewrite operations.

Enterprise Manager Summary Advisor

The Summary Advisor guides you through the numerous candidates for materialized views, and quantifies which ones are the most effective in theory and in practice. The Advisor is part of the DBMS_OLAP package.



The Summary Advisor Wizard is an interactive tool that recommends and builds materialized views. This wizard asks where the materialized views are to be placed, which fact tables to use, and which of the existing materialized views are to be retained. If a workload exists, it is automatically selected; otherwise, the wizard returns a set of recommendations generated from the functions RECOMMEND_MV or RECOMMEND_MV_W which are part of the DBMS_OLAP package.

By using the wizard, all of the steps required to maintain your materialized views can be completed by answering the wizard's questions and no subsequent DML operations are required. See the Oracle Enterprise Manager documentation set for further details.

Configuration of Oracle8*i*

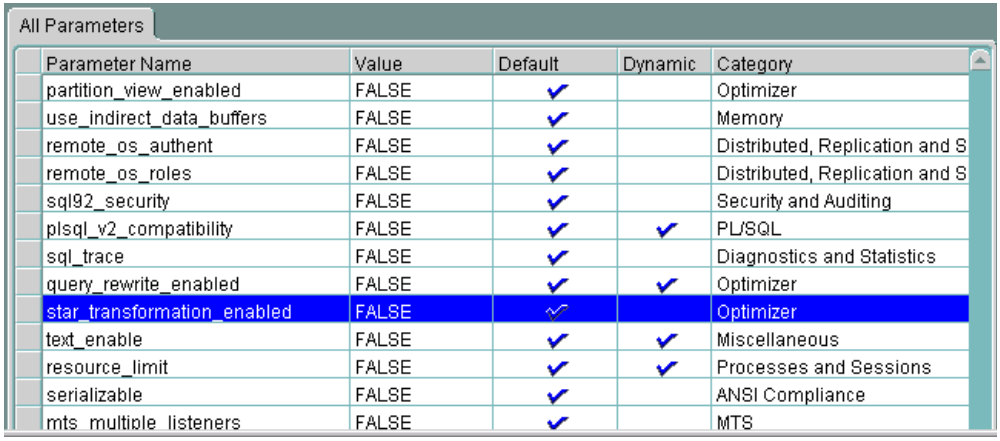
Initialization parameters stored in the init.ora file determine the operating characteristics of the Oracle8*i* instance which serves as the host for the data mart. Several of these parameters can greatly alter the performance of the data mart and they are summarized below. A few of these parameters can be set only when you create a database; others you can modify at any time.

The Initialization File

You can access and update the initialization file with a text editor or use the Database Instance tool from DBA Studio.

To display the initialization parameters, call DBA Studio, expand the navigation tree for the node that hosts the data mart, expand the branch for **Instance**, and then select **Database**. Enterprise Manager returns the General panel that describes the current state of the database.

At the bottom of this window, click the All Initialization Parameters button. Enterprise Manager returns a list of the parameters, their configured values, and the category of each parameter.



Click the **Description** button at the bottom of the panel to display a brief but helpful description of the parameter's function, a range of possible values for selected parameters, and its default value.

> **Note:** To edit these parameters, you must log on to DBA Studio as a user with SYSDBA privileges.

Input and Output Parameters These parameters define physical input and output characteristics for the database instance:

- DB_BLOCK_SIZE is the default block size. The value used for the case study database is 8 KB. For data marts and data warehouse applications, this value should be set to a minimum of 16 KB. This value cannot be changed after you create the database.
- DB_FILE_MULTIBLOCK_READ_COUNT determines how many database blocks are read with a single operating system call. In general, use 64KB / DB_BLOCK_SIZE as a guideline. For example, specify 8 if the DB_BLOCK_SIZE is 8 KB.
- HASH_MULTIBLOCK_IO_COUNT controls the number of blocks a hash join operation should read and write concurrently. The suggested value is 4.
- SORT_DIRECT_WRITES specifies how sorts execute. When set to AUTO, sorts execute very efficiently.

Resource Usage This list shows some of the parameters that affect system resource usage in a data mart environment.

- DB_BLOCK_BUFFERS determines the size of the database buffer cache and is specified in units of database block size. A reasonable starting value is 1000.
- SORT_AREA_SIZE specifies the amount of memory to allocate for sort operations. The cumulative sort area used by all server processes adds up fast because each server can allocate this amount of memory to perform a sort. If memory is abundant on your system, set the parameter SORT_AREA_SIZE to a large value but less than 2 megabytes. If memory is a concern, reduce its size so that paging does not result. A sample range of starting values is 256 KB to a 2 megabytes.
- HASH_AREA_SIZE indicates the memory used by each process for hash joins. Start with initial value of a few megabytes and adjust according to workload.
- PARALLEL_MAX_SERVERS specifies the maximum number of query servers that can be started on the system. Remember that most queries need at most twice the degree of parallelism attributed to any table referenced by the query.
- PARALLEL_MIN_SERVERS specifies the number of query servers to be started and reserved for parallel query operations when the instance starts.
- SHARED_POOL_SIZE is used primarily for storing shared SQL statements. When you use parallelism, the shared pool also provides space for a pool of message buffers that query servers use to communicate with each other. Start with 50 MB. Increase the value if you significantly increase the number of active query servers. If your data mart does not use parallel query, ignore this parameter.

Data Warehouse Functions The following parameters are commonly used in a data mart environment because they enable increased functionality or determine query performance to a large degree.

- STAR_TRANSFORMATION_ENABLED enables star transformation when set to true. Use the STAR_TRANSFORMATION hint to make the optimizer use the best plan in which the transformation has been used.
- QUERY_REWRITE_ENABLED to true.
- OPTIMIZER_MODE to ALL_ROWS or FIRST_ROWS; **or**, analyze all the tables in the data mart and then set OPTIMIZER_MODE to choose.
- COMPATIBLE to 8.1.0 or greater.
- ALWAYS_ANTI_JOIN determines how the NOT IN operator is evaluated. The recommended value is HASH, which enables the most efficient execution path.
- HASH_JOIN_ENABLED enables hash joins when you set it to TRUE.

Oracle Warehouse Builder Specific Set the following parameters to insure proper operation of the Builder client and the packages generated to implement a target data mart.

- GLOBAL NAMES set to false.
- PROCESSES set to 200, or higher.
- OPEN_CURSORS set to 200 or higher.
- OPEN_LINKS set to a maximum number of concurrent open connections to remote databases in one session. The value should equal or exceed the number of databases referred to by a single PL/SQL package.

Obvious Omissions

Partitioned tables simplify the management of large fact tables, especially during load and prune operations. Partitions also go hand in hand with parallel query. For medium and large scale data marts, you must understand these topics to take full advantage of Oracle8*i* features that are specialized for decision support and data warehousing applications. These features can make the difference between a data mart that meets its basic objectives and one that doesn't. This is especially true for a data mart that contains clickstream data. Unfortunately, these topics are far beyond the scope of this cookbook. For more information, see Chapter 15 of the *Oracle8i Administrator's Guide*.

The Next Step

This chapter gave you a few tips on how to manage the data mart. The chapter showed you how to develop a set of jobs to periodically refresh the data mart, how to schedule those jobs in the correct sequence, how to keep an eye on the growth of physical storage, and how to keep a handle on performance. Several of the tasks were illustrated using Oracle Enterprise Manager.

The data mart is complete and functioning at this point. The refresh strategy implemented for the data mart, however, does not completely preserve the historical record. While this is acceptable in a few cases, it is unacceptable in most. The next chapter shows you how to implement a refresh strategy that completely preserves history. This requires a few fundamental changes in the target schema.

14

Preserve Data Mart History

This chapter shows you how to revise the case study data mart so that it preserves the historical record over time. These revisions allow the Products dimension to track changes in the product descriptions by inserting a row in the dimension each time a description changes. The latest row inserted always contains the current descriptions of the product.

This chapter covers the following topics:

- The Basic Method
- Generation Steps

The Basic Method

This following examples show you how to implement a Type 2 method to refresh dimensions. The example implements this method for the Products dimension.

While the example implementation is straightforward, you must modify the Products dimension, and in the real world this change *should* be made before the initial load. This task should be considered as a design change and occur early in the development stages. You can modify the schema and a few of the mappings after the initial load, but that is a poor practice and is not recommended.

The Key Change

You must change the unique key constraint on the Products dimension so that multiple rows can describe the same product. The example modifies the current unique key constraint on the Products table so that it includes the natural and warehouse key columns (*pro_product_upc*, *pro_prod_WH*).

With this modification, you can insert a new row that contains a new description into the dimension each time a product description changes. Afterwards, the dimension contains the current as well as the previous description.

For example, when the product description changes from 'The Standard Mouse' to 'The New Superior Mouse', a new row is added to the dimension.

Natural Key	WH Key	Description	Packaging	…
A461245	25	The Standard Mouse	Laptop Value Pack	…
A461245	103	The New Superior Mouse	Laptop Value Pack	…

Subsequent updates of the fact table include the warehouse key for the new description. Thus, you must modify the mapping that loads data into the Sales table and devise an efficient method to pick out the row that represents the current description of a product.

Each time a product description changes, you insert a new row in the dimension, and thus you preserve all the changes over time.

This implementation preserves history but does make the dimension grow over time, especially when there are many changes. For moderately sized dimensions that change slowly, this approach works well. When the dimension table is large and the pace of change is furious, you should use another method such as that described in the *Oracle8i Data Warehousing Guide Release 2 (8.1.6)* guide.

Implementation Steps

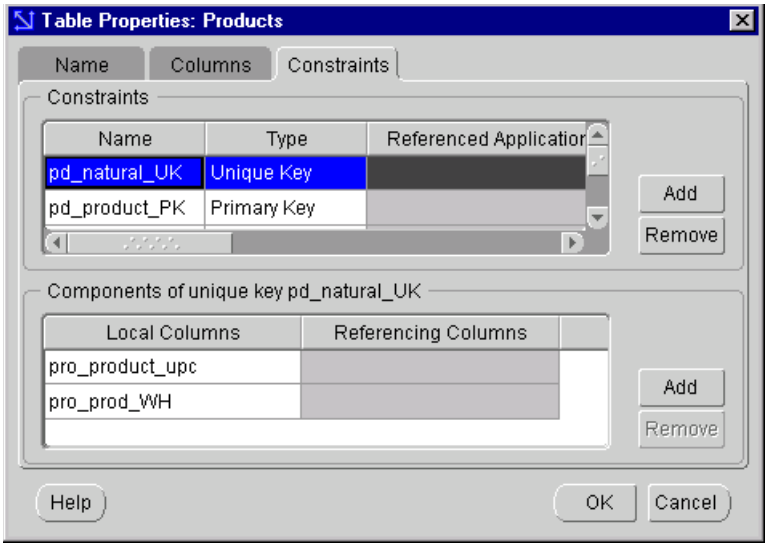
To implement this method, you must make a few changes to the data mart which make the Products dimension incompatible with previous mappings:

1. Change the unique key constraint on the Products dimension so that it is defined over the dimension's natural key (*pro_product_UPC*) and warehouse key columns (***pro_prod_WH***). Otherwise, the dimension can not retain the multiple descriptions which are necessary to track history.
2. Create a staging table (*prodSTAGE*) for the Products dimension and a mapping (*map_prodSTAGE*) that loads the updates into this table. These were created in the previous chapter.
3. Create a lookup table (*prodLOOKUP*) that associates each natural key value with the highest warehouse key value. This table efficiently defines the current product descriptions.
4. Create a post-processing transformation (*LoadLookUpTable*) that selects the natural key and the corresponding highest valued warehouse key from the Products table and inserts them into the lookup table. The mapping that refreshes the Products table uses a trigger to execute this transformation after each successful refresh operation.
5. Create a sequence object ***prodWHkey*** that generates values for the Product table's warehouse key column.
6. Create a mapping that refreshes the Products dimension. This mapping includes a post-map trigger that loads the lookup table when the refresh operation runs to a successful completion.
7. Update the foreign key constraint on the *stageONE* table so that it references the *prodLOOKUP* rather than the *Products* table.
8. Update the post-map trigger for the clean-up operations after a successful load of the Sales table.
9. Create a mapping that loads updates into the Sales table from the stageONE table. This mapping references the prodLOOKUP table instead of the Products table so that the new fact table rows reference current product descriptions.

Each step is straightforward, and the following sections show you the way.

Change the Unique Key Constraint

You must add the name of the warehouse key column (***pro_prod_WH***) to the UK constraint defined on the Products table. Open the table's property sheet and click the **Constraints** tab.



Select the name of the constraint and then click the lower **Add** button, select the name of the warehouse key column from the drop-down list, and click **OK**.

Create and Load prodSTAGE

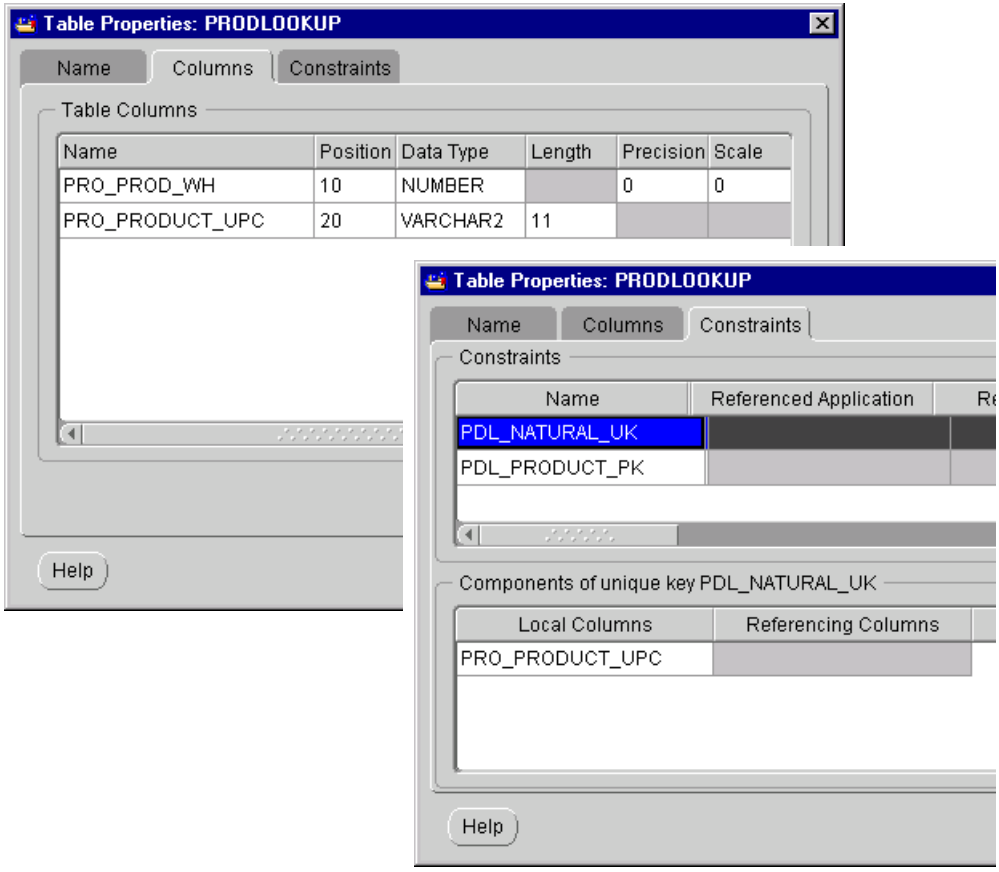
The previous chapter completely described these steps. See the examples that begin with the section "Create and Load prodSTAGE" on page 13-5. You can use the same table and the same mapping for the present case.

Create the Lookup Table

The lookup table contains two columns: *pro_item_upc* and *pro_prod_WH*. The values in these columns associate each natural key value with the maximum warehouse key value. The mapping for the Sales update joins the stageONE table to this lookup table rather than the Products dimensions and thus picks up the most current description for any product.

The datatypes for these columns must match the datatypes of the respective columns in the Products dimension. Use the New Table wizard to create a definition for the table.

Open the ***gcc****DM* warehouse module, expand its navigation tree, click **Table**, and select **Create Table** from the pop-up list. Builder returns the New Table wizard.



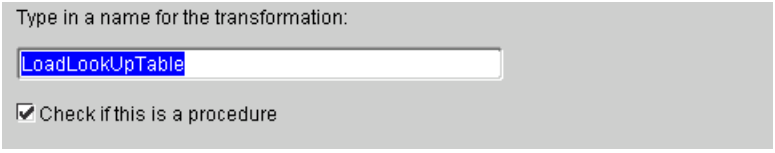
The unique key constraint for the lookup table includes only the natural key column. There is a one-to-one mapping between warehouse keys and the most current natural key. This also allows you to join the lookup table to the existing stageONE table over the natural key column.

Create a Transformation to Load the Lookup Table

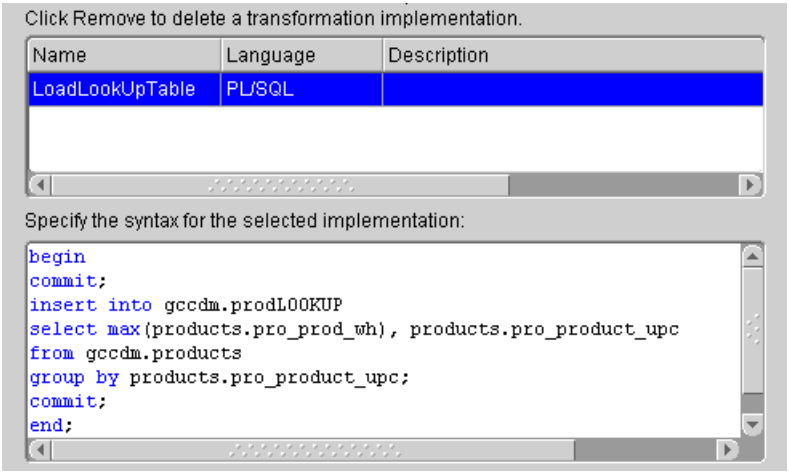
The mapping that refreshes the Products dimension uses a post-map trigger to call a transformation that loads the lookup table with data from the refreshed dimension. You must create a definition for this transformation before you create a definition for the mapping.

Create a transformation that updates the lookup table within the PostProcess transformation category. A definition for the category was created in Chapter 6. Name the transformation *LoadLookUpTable*.

The example on page 6-30 shows you how to create a definition for a transformation using the New Transformation Wizard. When you create this definition, select the **Check if this is a procedure** button.



The transformation defines a procedure that uses an Insert Into statement to retrieve and insert the required values into the lookup table.



The first statement commits the refreshes to the Product dimension before running the query to extract the required values.

Tip: You can and should generalize this transformation so that it can update a lookup table for any dimension. To generalize the transformation, define parameters for the schema, dimension, and lookup table names. Then, you can specify values for each parameter when you include the transformation in a mapping.

Create a Sequence for the Products Warehouse Key

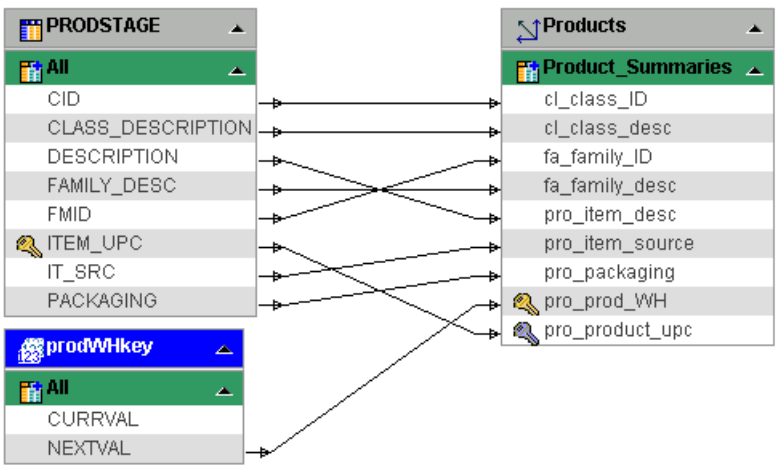
The Products dimension was initially loaded using an SQL*Loader job, but the current example uses a PL/SQL mapping to refresh the dimension which requires a sequence object. The previous chapter completely described these steps. See "Create a Sequence for the Products Warehouse Key" which begins on page 13-6.

Create a Mapping to Refresh the Dimension

The mapping describes how to extract data from the prodSTAGE table, generate warehouse key values, refresh the Products table, and load the lookup table.

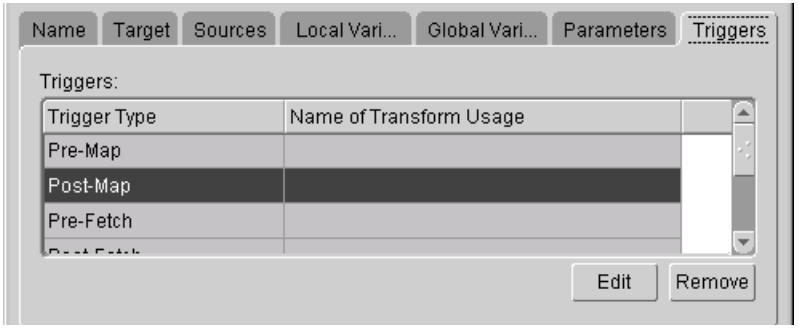
To create the mapping:

1. Create a definition for a high-level mapping (*refresh_Products_Type_2*) which references only the *gccDM* warehouse module.
2. Name the *prodSTAGE* table as the primary source and the *Products* as the target.
3. Open the detail mapping, map the respective columns, import the *prodWHkey* sequence into the detail mapping, and then map its output to *pro_prod_WH*.



4. Define a post-map trigger to call the LoadLookUpTable transformation.

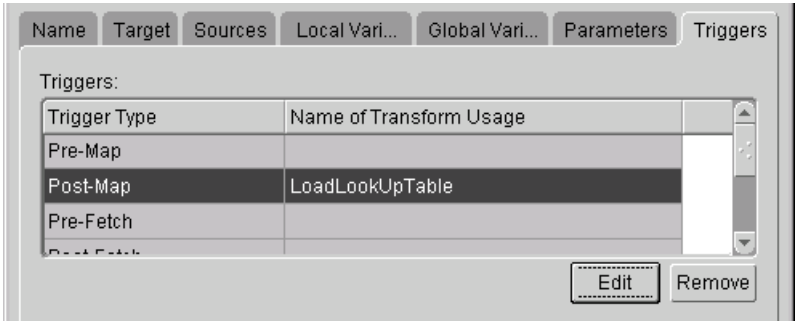
   1. Right-click the editor's white space and select **Detail Mapping Properties** and then select the **Triggers** tab on the mapping's property sheet.



   2. Select **Post-Map** as the Trigger Type.

   3. Click **Edit**.

      Builder returns the New Transformation Usage Wizard. Use this wizard to specify the LoadLookUpTable transformation.



   4. Click **OK**.

   The mapping now defines a post-map trigger that will execute the transformation only after the Products table has been successfully refreshed.

5. Close the Mapping Editor, switch to the Physical Tree, and generate the scripts that implement the mapping.

6. Deploy the mapping to the file system and the target schema, and then register the mapping as a job with Oracle Enterprise Manager.

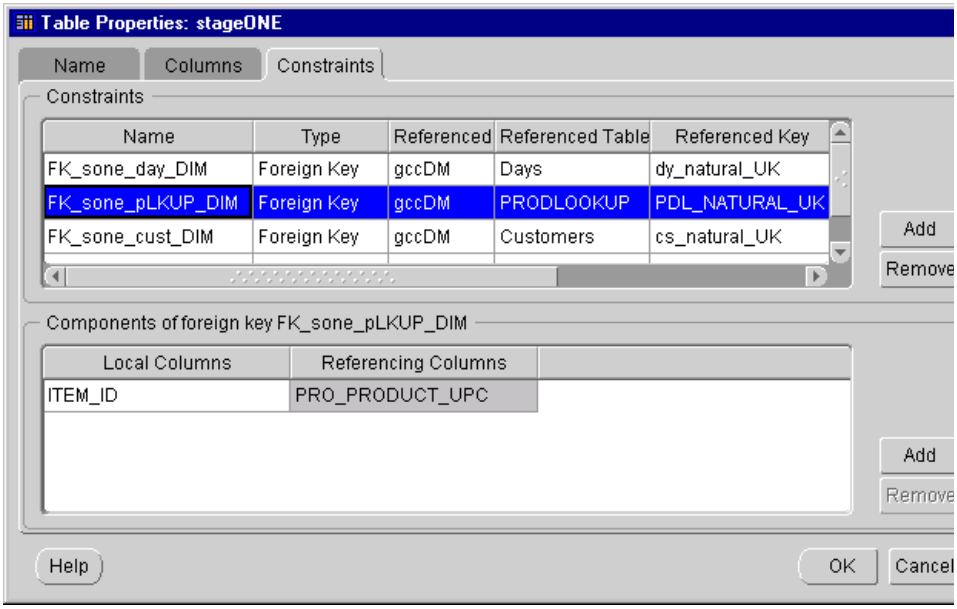
A mapping that refreshes the Products dimension and loads its lookup table now resides in the target schema and a job that executes the mapping resides in the Oracle Enterprise Manager Job Library.

Update stageONE Constraints

The mapping that updates the Sales table must join the prodLOOKUP table to the stageONE table so that it can pick up the warehouse key value that represents the current product descriptions. Thus, you must change the definition of the stageONE table so that it references the lookup table rather than the Products table.

To update the definition:

1. Open the properties sheet for the stageONE table.
2. Select the **Constraints** tab.



3. Remove the constraint for the Products dimension and then create a new constraint that references the lookup table over the *ITEM_ID* column of the *stageONE* table and the *PRO_PRODUCT_UPC* column of the lookup table.
4. Click OK.

Now when you update the mapping for the Sales table, you can reference the lookup table as a secondary source for the stageONE table.

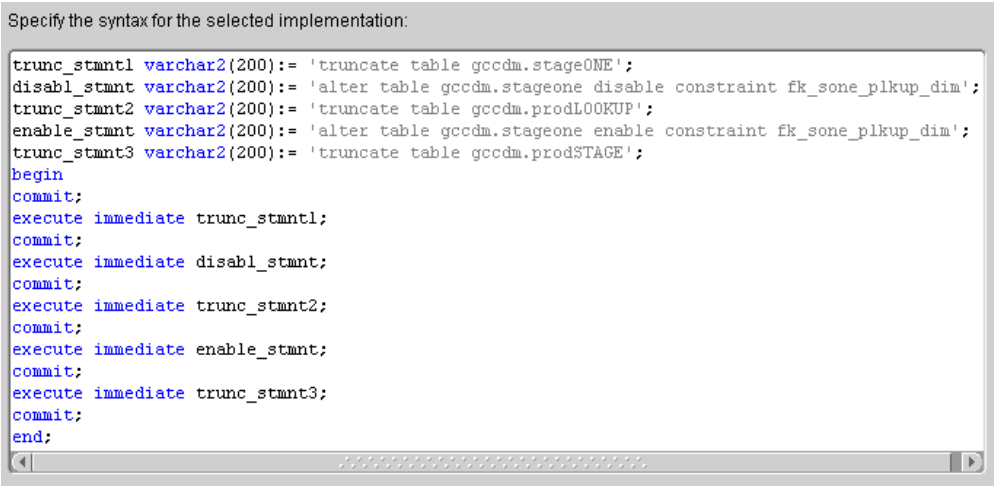
Update the Cleanup Transformation

The mapping that updates the Sales table uses a post-map trigger to perform cleanup operations after a successful update. The operations truncate:

- LookUpTable
- prodSTAGE table
- stageONE table

After a successful load operation, the cleanup operations prepare the way for the next periodic refresh of the data mart.

To update the cleanup transformation (*truncate_table*), open the property sheet for the transformation, select the Implementation tab, and then update the transformation so that it performs the following operations.



After a successful update of the Sales table, these statements clean up the staging and lookup tables so they can be used for the next update operation.

Create a Mapping that Updates the Fact Table

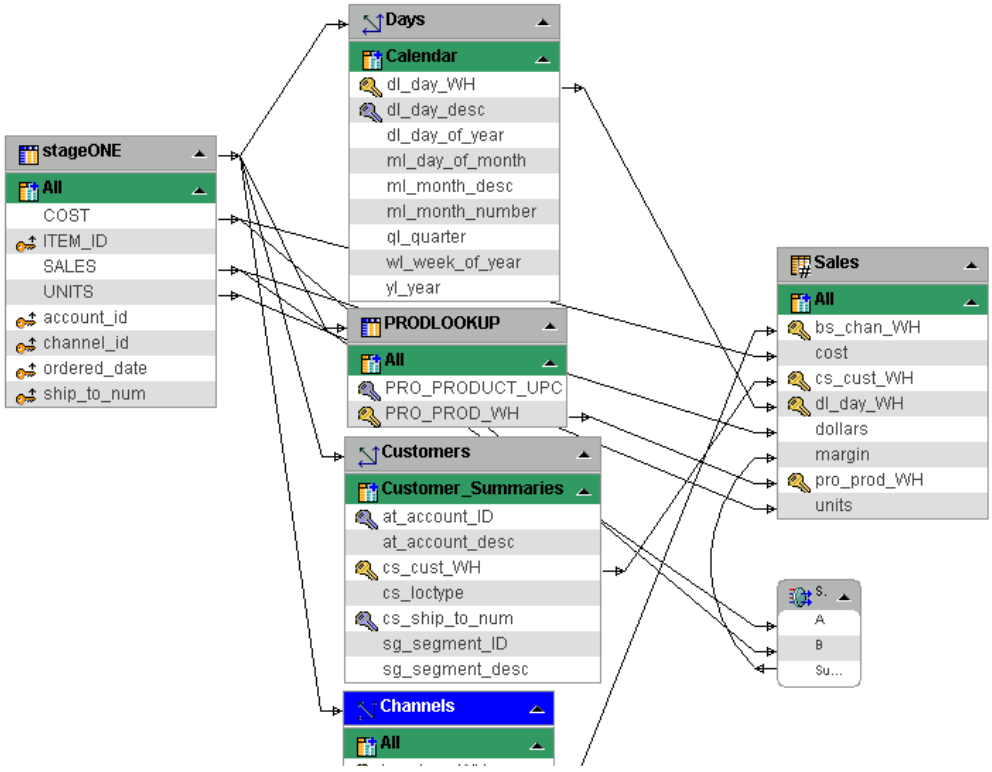
The update mapping for the Sales fact table is the same as the initial load mapping except that it joins the stageONE table to the prodLOOKUP table rather than to the Products table. This join selects the warehouse key from the most current description of each product.

The following example updates the existing mapping for the Sales table. You could also create a new mapping.

1. Open the *map_Sales* detailed mapping.
2. Right-click the Products dimension and then from the pop-up list select **Remove From Detail Mapping**.
3. From the toolbar, select **Detail Mapping > Add > Secondary Source**.
4. Select the **Foreign key related sources** button on the pop-up panel and then click **OK**.

   The Detail Mapping Editor returns a **Secondary Source Selection** list.
5. Select the *prodLOOKUP* table from the list and then click **OK**.

   The Detail Mapping Editor includes the look up table within the mapping.
6. Map the warehouse key column (*pro_product_upc*) of the lookup table to the corresponding column of the Sales table.



Generation Steps

You can now switch to the physical navigation tree, validate all the objects, and make whatever corrections are necessary. The working assumption here is that you have already configured the physical instance according to the examples in Chapter 8, "Configure a Physical Instance."

Once the logical definitions have been validated, you can generate the scripts and then use the scripts to deploy the physical instance. The following steps that deploy and load the data mart assume that no data mart has been deployed to the *gccDM* target schema.

Deploy the Data Mart

The deployment takes three steps. Display the Generated Scripts window, and then:

1. Click the **Deploy All** button and select **in File System** to write all the scripts to the Top Directory.
2. Click the **Deploy All** button and select **in Database** to create all the data mart objects (dimensions, facts, sequences, tables, packages, transformations, and materialized and conventional views).
3. Click **OEM All** to register all the jobs with Oracle Enterprise Manager. Before you take this action, make sure that the Oracle Management Server is active.

The data mart is now deployed to the database instance and all the jobs required to load the data mart have been registered with Enterprise Manager Job Library.

Load the Data Mart

You can now run the scripts that load the data mart. Run the initial load jobs first, verify that the fact table was successfully loaded, and then you can run the jobs that update the data mart with the data for January 1999. Before you run the job that loads the stageONE table, you must first set the start and end parameters.

The Final Step

This chapter showed you how to revise the definitions for the case study data mart so that it preserves the historical record over time. These revisions allow the Products dimension to track changes in the product descriptions by inserting a row in the dimension each time a description changes.

A

The Case Study Environment

This appendix shows you how to set up a Windows NT system so that you can work through all the examples in the cookbook. The directions for the product installations follow the simple and straightforward path rather than the path that would install a complete system optimized for a data mart or a data warehouse. For example, the section on the Oracle8*i* Database Instance directs you to perform a "Typical" installation. In practice, you would more likely select the "Data Warehousing" button for the type of installation.

The purpose of this appendix is to help you (or a database administrator) get a basic system up and running quickly so that you can work through the cookbook examples. After you work through the examples, you can turn to the major task of planning, sizing and implementing a database instance for a robust data mart.

This appendix covers the following topics:

- The Windows NT Host
- The Base Oracle8i Database Instance
- The Repositories
- Install Publication Products
- The Database Source

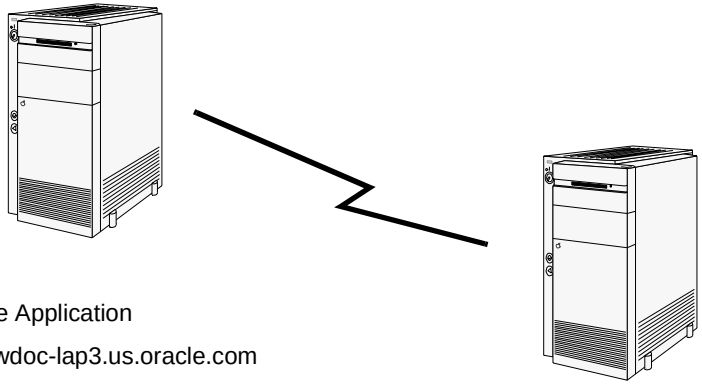
Note: This Appendix assumes you are installing these components on a system that does not contain any Oracle homes. If your system already has an Oracle home, talk with your database administrator.

The Windows NT Host

A single Windows NT system (dwdoc11-pc) serves as the host for the case study data mart. This system supports an Oracle8*i* database instance and all the components required to build, load, refresh, and operate the data mart.

The primary source data for the data mart resides on a separate Windows NT system (dwdoc-lap3). This small order-entry application (*GCCAPPS*) could reside on any database host, for example a UNIX system, provided an Oracle Transparent Gateway agent can access the database. If you just want to work through the cookbook, you can create the application database on the data mart host provided it has sufficient memory and disk storage.

The secondary sources for the data mart, two small text files, reside on the data mart host. They too could reside on a remote host provided that its file system or directory can be mounted on the data mart host.



Database Application

Node: dwdoc-lap3.us.oracle.com

Computer Name: dwdoc-lap3

Connect: dwdoc-lap3:1521:gcc816

Global Computing Data Mart

Node: dwdoc11-pc.us.oracle.com

Computer Name: dwdoc11-pc

Connect: dwdoc11-pc:1521:ora816

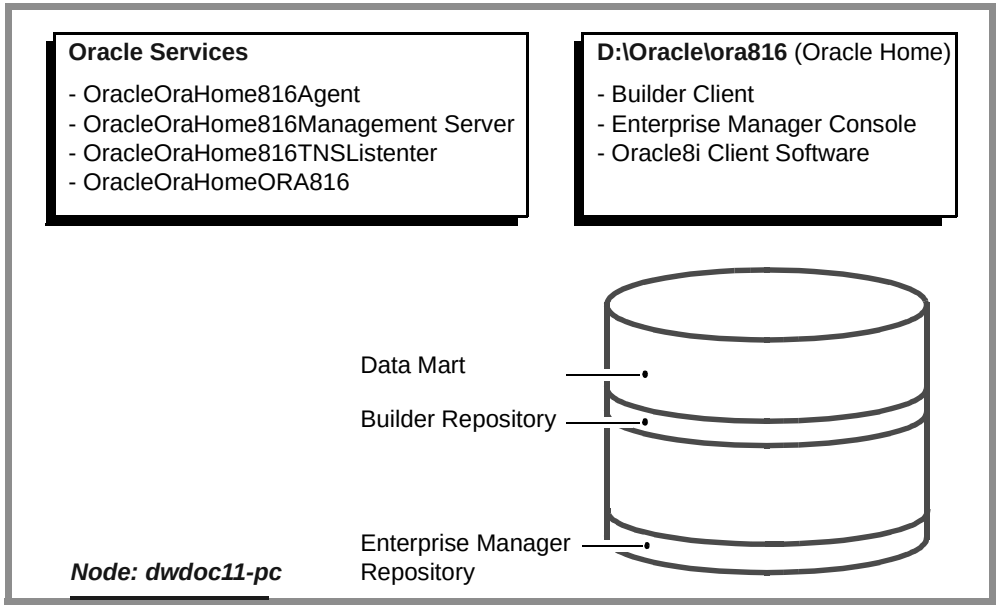
The case study mart data mart was built on a system that had a 667 mhz processor, 383 MB of memory, 10 GB disk storage, and operates under Windows NT 4.0 (Service Pack 4). The two systems are connected over a LAN using the TCP/IP protocol.

Tip: If you just want to work through the examples, you can run the data mart and the database source application on a single Windows NT system.

The Base Oracle8i Database Instance

The Oracle8*i* database instance that supports the data mart target schema (*gccDM*) also supports Oracle Enterprise Manager, Oracle Warehouse Builder, and their respective repositories. This instance also supports the publication components, Oracle Reports and Oracle Discoverer (User and Administration Editions plus a single End User Layer).

Figure A–1 The Basic Oracle8i Database Instance



The Oracle components are installed in three Oracle Homes, and each resides in the directory d:\Oracle. Oracle Reports *requires* the *default* home which must be installed first, and Warehouse Builder was placed in its own home to simplify long-term management installed software (when you move to a new Oracle8*i* release).

Table A–1 The Three Oracle Homes

Directory	Contents	Name
Oracle\orant	DEFAULT_HOME	Oracle Reports Runtime
Oracle\ora816	ora816	Oracle8*i*, Oracle Management Server, Listener, & Agent
Oracle\owb21	owb21	Oracle Warehouse Builder

The Default Home

The software for Oracle Reports requires a separate Oracle Home, so you must install Reports software in this home *before* you install any other components. For the base database instance, you can install the Oracle Reports Runtime component. Later, after the data mart has been built and initially loaded with data, you can install the software for Oracle Reports Developer and Oracle Discoverer in this directory. You could, as an option, install these software components at this time.

Table A–2 Oracle Default Home

Component	Description
DEFAULT_HOME	Name of the Home Directory
D:\Oracle\orant	Location of the Home Directory
Oracle Reports Runtime	Runtime component for Oracle Reports

Create the d:\Oracle\ora816 directory and install Oracle Reports 6i Runtime:

1. Select **Oracle Reports Developer** from the Oracle Tools Installation Options.
2. Select **Custom Install** from the Installation Options.
3. Expand the Oracle Reports Developer selection list on the Software Asset Manager panel, and install **Oracle Net8 Products** and **Reports Runtime**.

This lays the foundation for Oracle Warehouse Builder, Oracle Reports 6i, and Oracle Discoverer 3.1. Only the runtime component for Oracle Reports is installed at this time.

Do I Need To Install Oracle Reports:

When you install the Oracle software for the data mart, you must install Oracle Reports Runtime first. When you install this component, the Oracle Installer creates the proper default home. Warehouse Builder generates numerous metadata reports using the Oracle Reports Runtime.

If you do not plan on using the metadata reports, you can still follow the cookbook to build the data mart. The cookbook does not use the metadata reports. If you take this path, however, you will need to install Oracle Reports Developer on another Windows NT system in order to work through the examples in Chapter 11 that show you how to generate standardized reports for users.

The ora816 Home

The ora816 home contains software for the Oracle8*i* database instance and utilities required to maintain and operate the data mart. This database instance supports the target schema, its runtime libraries, the Reports runtime tables, and the repositories for Oracle Enterprise Manager, Oracle Warehouse Builder, and Oracle Discoverer.

Table A–3 *Oracle Home for Oracle8i*

Component	Description
ora816	Name of the Home Directory
D:\Oracle\ora816	Location of Home Directory for Oracle8*i* and OEM.
Oracle8*i* Database Instance	Data Mart Database
SID: ORA816	SID for the database instance
Oracle Enterprise Manager	Database Administration Tools
OEM Console	Oracle Enterprise Manager client
OMS	Oracle Management Server
Oracle Intelligent Agent	Executes batch jobs for Oracle Management Server
Oracle TNS Listener	Network database listener

Create the d:\Oracle\ora816 directory first, and then install the components listed above into that directory. To simplify the installation, choose a "Typical" install if you intend only to work through the cookbook examples. When you build an actual data mart, you should size the data mart, plan the Oracle8*i* database installation, and complete a "Custom" install with the "Data Warehousing" option. This enables specialized database features designed for decision support applications.

Typical Installation

When you install the software for Oracle8*i* Enterprise Edition for the purpose of working through the cookbook examples, a "Typical" installation suffices and simplifies the process:

1. Select **Oracle Enterprise Edition 8.1.6** from the Available Products page.
2. Select the **Typical (1001MB)** option from the Installation Types page.

   This option installs all the Oracle components listed in Table A–3 above and a few example schemas. This option does not, however, create a repository for the Oracle Management Server. See the discussion in "The Repositories" for an example that shows you how to create this repository as well as one for Builder.

Oracle Initialization Parameters

The installation process created a database instance (ora816) and an initialization file (init.ora) that defines the instance's operating parameters. When you restart your Windows NT system, the instance automatically restarted.

Before you can run Oracle Warehouse Builder on this instance, you must modify values of selected initialization parameters. Table A–4 lists these parameters and their respective values (Required or Suggested).

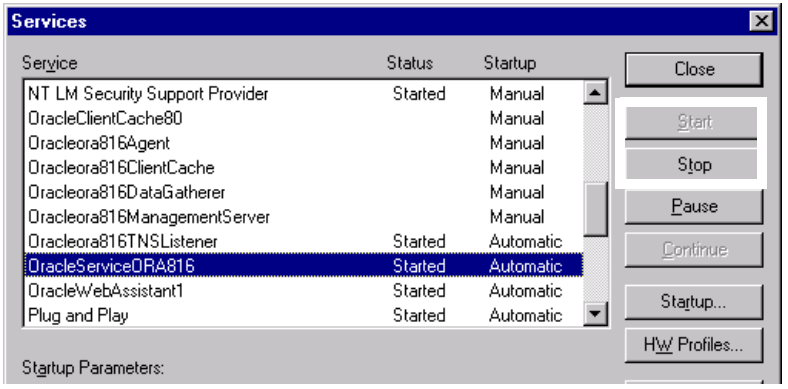
Table A–4 Builder Client Parameters

Parameter	Value	Required/Suggested Value
global_names	FALSE	Required
open_cursors	200	Suggested
open_links	20	Suggested
processes	200	Suggested
dbwr_io_slaves	Between 1 and two times the number of CPUs.	Suggested
utl_file_dir	*	Required

You can edit the initialization file using a text editor or with DBA Studio (an administration tool installed during a "Typical" install). The path name for the initialization file is:

d:\Oracle\admin\ora816\pfile\init.ora

These new values do not become effective until after you stop and then restart the instance. You can start and stop the database instance using DBA Studio or the Services panel in the Windows NT system's Control Panel.



The owb21 Home

The owb21 home contains the software for Oracle Warehouse Builder. The separate home directory isolates the Builder components from the Oracle database so the database can be upgraded without disturbing the Builder components.

Table A–5 *Oracle Home for Oracle Warehouse Builder*

Component	Description
owb21	Name of the Home Directory
D:\Oracle\owb21	Location of the Home Directory for Warehouse Builder.
Oracle Warehouse Builder	Builder client and three utilities
Builder client	Defines and creates scripts to implement data mart
Repository Assistant	Creates Builder repositories
Runtime Assistant	Creates target schema and runtime audit tables
Runtime Audit Viewer	Reads audit records stored by load and refresh routines

Create the d:\Oracle\owb21 directory and install Oracle Warehouse Builder:

1. Select **Custom** from the Installation Types page.
2. Select the following components and then click **Next**:
   - Net8 Client 8.1.6.0.0
   - Oracle Warehouse Builder Client 2.1.1
   - Oracle Warehouse Builder Repository Assistant 2.1.1
   - Oracle Warehouse Builder Runtime Assistant 2.1.1
   - Oracle Warehouse Builder Runtime Audit Viewer 2.1.1

   After installing these components, the Oracle Installer automatically starts the Repository Assistant.
3. Select the **Cancel** button on the Repository Assistant wizard's welcome page.

   The Oracle Installer automatically starts the Runtime Assistant.
4. Select the **Cancel** button on the Runtime Assistant wizard's welcome page.

   The Oracle Installer reports that both of these tools failed and both of their Status entries show *canceled*. This is OK. These tools are actually part of the development and implementation process and are discussed in that context.

The Repositories section of this chapter shows you how to create a Builder repository using the Repository Assistant, and how to initialize the repository with a Project for the case study. Chapter 7, "The Physical Environment," shows you how to establish the runtime environment for the case study using the Runtime Assistant. Finally, Chapter 10, "Populate the Data Mart," shows you how to interrogate the runtime audit tables using the Runtime Audit Viewer.

The Repositories

The Oracle8*i* database instance contains three schema objects: the target schema, the Builder repository, and the Oracle Enterprise Manager repository. This section shows you how to create and initialize the repositories; Chapter 7, "The Physical Environment," shows you how to create the target schema and its runtime environment.

***Table 14–1 ORA816** Schema Objects*

Owner Name	Description
gccDM	Target schema for the data mart
Target Tables	Dimensions, facts, stageONE table, and materialized views
PL*SQL Packages	Stored procedures generated and deployed by Builder that extract and transform data from database sources (*GCCAPPS*).
Runtime Audit Tables	Tables created by the Runtime Audit Viewer. The Builder generated jobs write log entries to these tables during a load operation. You can display their contents using the Runtime Audit Viewer or SQL*Plus.
gccrep	Builder repository contains definitions for the case study. These include definitions for all data sources, all target tables, and all the mapping definitions that describe how to extract, transform, and load the data mart. Currently, these definitions describe a logical model of the data mart which can be configured to describe a physical instance. The next chapter shows you how to configure a basic physical instance of the model.
OEM	Oracle Enterprise Manager repository contains generated TCL scripts in its Job Library. Enterprise Manager uses these scripts to schedule and run jobs that initially populate and periodically refresh the data mart.

Use the Enterprise Manager Configuration Assistant to create a repository for the Oracle Management Server, and the Builder Repository Assistant to create a Builder repository.

Oracle Enterprise Manager Repository

This repository is managed by the Oracle Management Server, and you must create this repository before you can use the Oracle Enterprise Manager Console.

To create the repository, begin with the Windows NT Start button:

1. **Start > Oracle > Enterprise Manager > Configuration Assistant**.

   Oracle Enterprise Manager returns the first page of the Configuration Assistant.

2. Select **Create a new repository** and click **Next**.

   The Assistant returns the Select Database for Repository page.

3. Enter the administrator username and password (*system/manager*) and the service name for the data mart's instance (*dwdoc11-pc:1521:ora816*), then click **Next**.

   The Assistant connects with the database instance and returns the Repository Login Information page which solicits a username and password pair for the owner of the repository.

4. Enter **REP_OEM** / oem and click **Next**.

   The Assistant returns the Select Repository User Tablespaces.

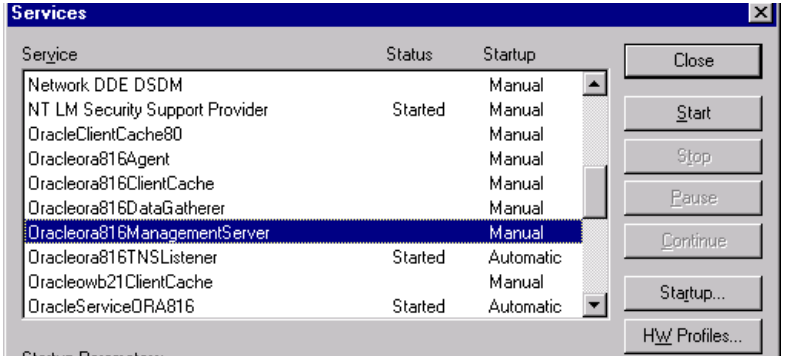
5. Select **Create a new OEM_REPOSITORY tablespace** and click **Next**.

   The Assistant returns the Create Repository Summary page.

6. Verify your entries before you click Finish. Click **Finish**.

   The Assistant creates the repository.

7. Open the Services window for your Windows NT system and then start the Oracle Management Server service.

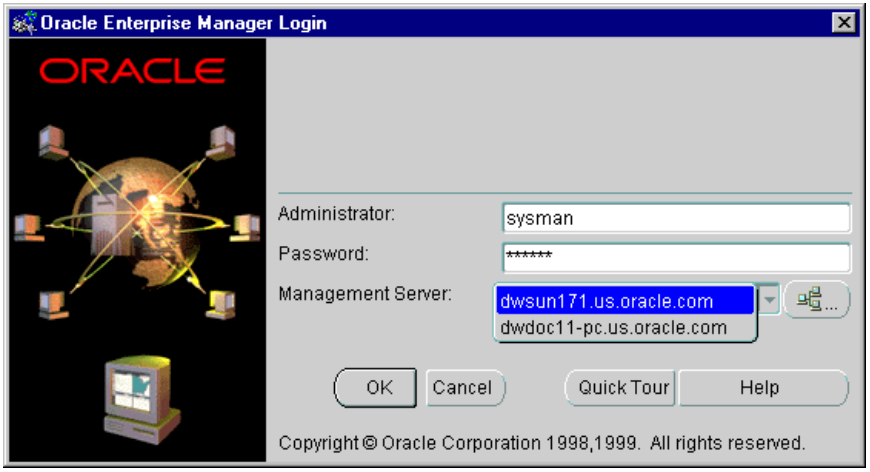


8. You can now log on to the repository using the Oracle Enterprise Manager console.

   Start > Oracle > Enterprise Manager > Console.

   The console returns its login panel. The initial username and password for a new repository is *sysman/oem_temp*.

Figure A–2 Oracle Enterprise Manager Console



   The Enterprise Manager returns a panel that allows you to change the username and password before it opens the repository and returns the root window for the console.

The information you use to login to this repository (Figure A–2) is critical when you configure your data mart to operate with Enterprise Manager. You will also need the "home directory" for Enterprise Manager. In the present case, it is the same as the ora816 database instance (d:\Oracle\ora816).

Oracle Warehouse Builder Repository

This repository is managed by Oracle Warehouse Builder, and you must create this repository before you can use Builder. The following example shows you how to create a Builder repository using the Oracle Warehouse Builder Repository Assistant and then how to create the GCC Data Mart Project within the repository using the Oracle Warehouse Builder client. This Builder Project is the starting point for the case study.

Create a Tablespace for the Repository

The Builder repository requires more disk storage space than is allocated to the USERS tablespace during a "Typical" install of the database instance. You can increase the size of the USERS tablespace or create a dedicated tablespace for the repository. The latter case simplifies storage and performance management. The following example shows how to create a dedicated tablespace with DBA Studio.

To create a tablespace, call DBA Studio:

1. **Start > Oracle > Database Administration > DBA Studio**.
2. Logon as *system* / *manager* and expand the Storage branch of the navigation tree.
3. Right-click **Tablespaces** and select **Create** from the pop-up list.
4. Enter the name of the tablespace (*REP_GCCREP*), its size in megabytes (*100*), and then click **Create**.

   DBA Studio creates the tablespace.

You can now close DBA Studio, and then create a Builder repository.

Create the Builder Repository

To create a Builder repository using the Repository Assistant, call the Assistant and then provide information solicited by its five wizard pages:

1. **Start > Oracle > Oracle Warehouse Builder Repository Assistant**.

   The assistant returns its welcome page.
2. Click **Next**.

   The wizard returns the Choose Operation page.
3. Select **Create a New Warehouse Builder Repository** and click **Next**.

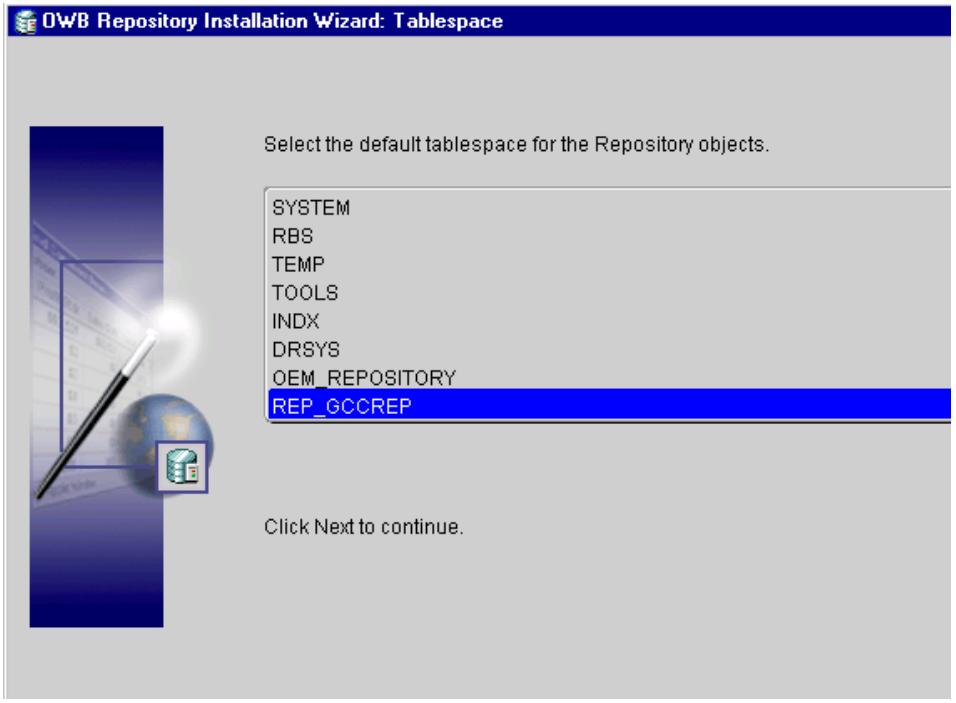
   The wizard returns the Server Information page which solicits connection information for the database instance where the repository will reside.

4. Enter the following values then click **Next**:
   - Host Name: *dwdoc11-pc*
   - Port Number: *1521*
   - Oracle SID: *ora816*

   The wizard returns the New or Existing Schema page.
5. Select **Create and install into new user Schema** and click **Next**.

   The wizard returns the Username and Password page.
6. Enter *System* / *Manager* as the System username and password, then click **Next**:

   The wizard returns the Tablespace page.
7. Select *REP_GCCREP* from the down-arrow list, accept the default for the temporary table space (**TEMP**), and then click **Next**.



The wizard returns the Username and Password page.

8. Enter a username and password for the schema that serves as the Builder repository for the case study (**gccrep** / **gccrep**), and click **Next**.

   The wizard returns a Summary page. Verify the information before you click the Finish button on this page.

9. Click **Finish**.

   The wizard creates the Builder repository used in the case study.

After the repository has been created, you need to create a Project within the repository named GCC Data Mart.

Create the Builder Project

To create a Project in the Builder repository for the case study, you must call the Builder client, switch to the administration environment, and then create the Project using the Create New Project wizard.

1. **Start > Oracle > Oracle Warehouse Builder Client**.

   Windows returns the Builder client's logon panel which solicits connection information for a repository.

2. Enter the following logon information and click **Next**.

   - User Name: *gccrep*
   - Password: *gccrep*
   - Host Name: *dwdoc11-pc*
   - Port Number: *1521*
   - Oracle SID: *ora816*

   All of this information was entered in the previous section. After you click Next, the wizard returns a Project Selection panel.

3. Select **My Project**, a built-in skeleton project, and click **Next**.

   The console opens the skeleton project and puts the console in the Project environment. To create a new project, you must switch to the Administration environment and call the New Project Wizard.

4. Left-click the Administration icon (the vertical toolbar on the left side of the console).

   The console returns the Administration navigation tree and a different set of menus on the horizontal toolbar.

5. Expand the navigation tree, right-click **Projects**, and select **Create Project**.

   The console returns the welcome page for the New Project Wizard.

6. Click Next.

   The wizard returns the Name page.

7. Name the project (***GCC Data Mart***), enter a brief description, and click **Next**.

   The wizard returns the Finish page.

8. Review the summarized information and then click **Finish**.

   The wizard creates the Project within the repository.



When you exit the console, Builder returns the Exit Confirmation panel. Click **Yes** to commit this change to the repository.

You can now call the Oracle Warehouse Builder client, logon to the ***GCC Data Mart*** project, and start building a data mart.

Install Publication Products

At some phase in a development and implementation project, you are ready to publish the data mart's contents either for sanity-check testing, usability testing, or further analysis and design sessions with the users. At that time, you need to install Oracle Reports to generate reports and Oracle Discoverer to support query access.

You must install both of these products in the DEFAULT_HOME directory.

Oracle Reports 6i

Install Oracle Reports in the DEFAULT_HOME directory. When you call the Oracle Installer, you can enter the following selections for a quick and minimal installation:

1. Select **Oracle Reports Developer** from the Tools Installation Options panel.
2. Select **Custom Install** from the Installation Options panel.
3. Select **Oracle Reports 6.0.8.6.8** and **Oracle Reports Documentation 6.0.8.6.8** from the Software Asset Manager panel.
4. Click **Install**.

This also installs the Reports Server and creates an Oracle Windows NT Service (Oracle Reports Server [Rep60_DWDOC11-PC]). You do not need this service to complete the cookbook examples. You can stop this service manually and reset it to start manually from the Windows NT Services window.

Oracle Discoverer 3.1

Install Discoverer 3.1.36 into the DEFAULT_HOME directory. When you call the Oracle Installer, enter the following selections for a minimal and quick installation.

1. Select **Minimal Installation without Tutorial and Quick Tour** from the Installation Options panel and click **OK**.
2. The Oracle Installer returns a few warning panels that say you have more recent versions of selected Net8 components already installed: do you want to replace the newer versions. Click **No**.
3. The Oracle Installer returns a warning panel that Microsoft support software is required. Click **OK**.
4. The Oracle Installer returns a warning panel that says the installed version of the Java runtime environment is more current: do your want to replace it with an older version? Microsoft support software is required. Click **No**.

After the installation is complete, you can create an End User Layer and go forward.

The Database Source

The database source tables for the case study are supplied in DMP format file. You can import these tables into an existing Oracle8*i* database using the Oracle imp utility. The directory that contains the DMP format file also contains a parameter file you can use to import the tables.

To import the tables on the gccapps.DMP file into the ora816 database instance:

1. Create a database user named gccapps on the Oracle8*i* database instance that serves as the source database.
2. Create a directory on source database system (e:\gccdata).
3. Copy the gccapps.DMP and gccapps.parf files to e:\gccdata.
4. Open a DOS window and navigate to e:\gccdata.
5. Call the Oracle imp utility with the following command:

   ```
   imp system/manager parfile=gccapps.parf
   ```

   The import utility will import the database tables into the schema.

The parameter file (gccapps.parf) assigns the following values:

```
file=gccapps.dmp
grants=y
rows=y
fromuser=gccapps
touser=gccapps
constraints=y
destroy=n
```

Note: The application database (*GCCAPPS*) can reside on the data mart's host or on another system.

B

Materials for the Case Study

This cookbook uses a set of examples that progressively walk you through the construction of a data mart from start to finish. These examples show you how to use Builder for specific tasks (such as create a dimension) but do not show you how to create every dimension required to implement the *GCC Data Mart*. Thus, many tasks are intentionally left as exercises for the reader. Moreover, many other examples remain at a summary level and omit details. This appendix is intended for those readers who would like to build the entire case study data mart using Oracle Warehouse Builder.

This appendix contains detailed information on the case study source and target:

- Data sources

  The Global Computing Company Applications database and the two flat files.

- The target schema
  - Dimensions
  - Facts
  - Views
  - Sequences

The Data Sources

The Global Computing Company Data Mart pulls data from two sources: the *GCC Database* Application and two flat files. This section describes these sources and the last section shows you how to import them into an Oracle8*i* database.

The GCC Database Application

The *GCC Database Application*, an order entry system, is the principal source of data for the Global Computing Company Data Mart. The application database runs on an Oracle8*i* instance but could run on any database system.

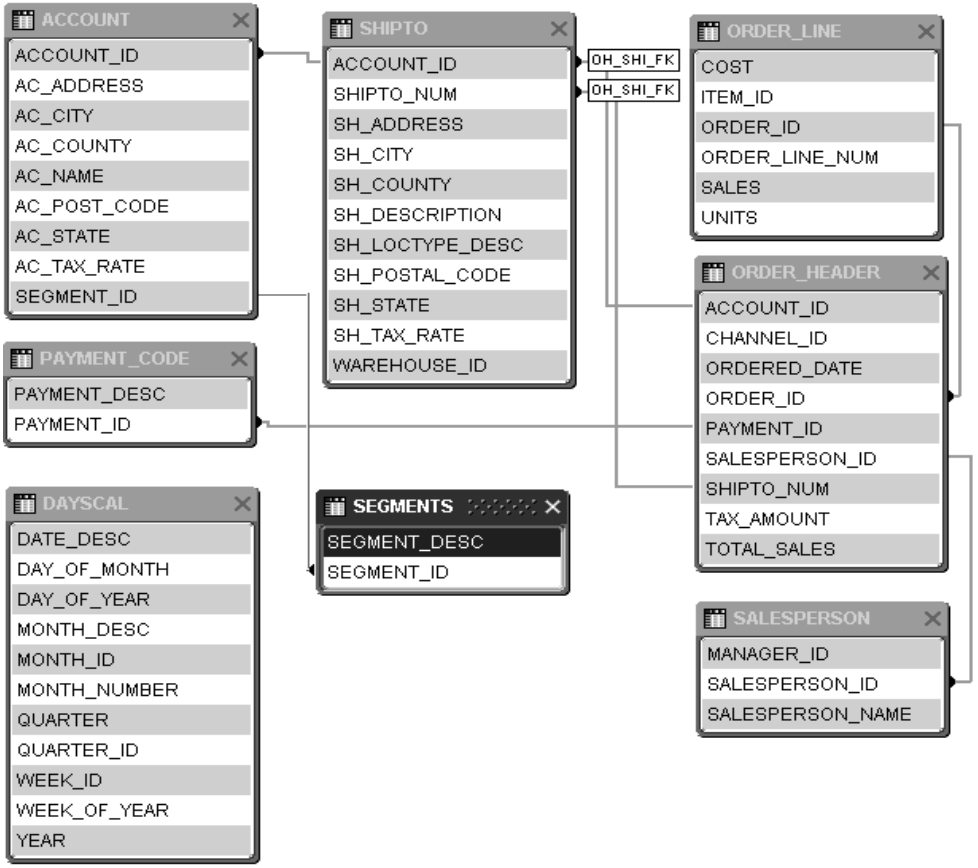


Table B–1, "GCC Application Database Tables" on the following page describes each of these tables in detail.

Table B–1 GCC Application Database Tables

Column Name	Data Type	Definition	Examples	Related to
ACCOUNT				
ACCOUNT_ID	NUMBER	Key field: Account number	23	
AC_NAME	VARCHAR2(35)	Customer name	Bavarian Industy	
AC_ADDRESS	VARCHAR2(20)	Customer street address	123 Main St.	
AC_CITY	VARCHAR2(10)	Customer city	Nashua	
AC_STATE	VARCHAR2(2)	Customer state	NH	
AC_COUNTY	VARCHAR2(10)	Customer county	Hillsboro	
AC_POST_CODE	NUMBER	Customer postal code	03062	
AC_TAX_RATE	NUMBER	Customer tax rate	0.06	
SEGMENT_ID	NUMBER	Industry identifier	4	
ORDER_LINE				
ORDER_ID	NUMBER	Key field: Order number	21000	
ORDER_LINE_NUM	NUMBER	Line number on the order form	11300	
ITEM_ID	VARCHAR2(11)	Product identifier	9700	
UNITS	NUMBER	Number of units purchased	8	
SALES	NUMBER	Total sales	8.4	
COST	NUMBER	Total cost	3.6	
ORDER_HEADER				
ORDER_ID	NUMBER	Key field: Order number	34590	Foreign key to ORDER_LINE
CHANNEL_ID	NUMBER	Channel identifier	2	
ACCOUNT_ID	NUMBER	Account identifier	23	
SHIPTO_NUM	NUMBER	Ship to location	145	Foreign key to SHIPTO
ORDERED_DATE	DATE	Date of order	3/15/95	
SALES_PERSON_ID	NUMBER	Salesperson	889	
PAYMENT_ID	NUMBER	Method of payment	4	
TOTAL_SALES	NUMBER	Total sales for the order	89	
TAX_AMOUNT	NUMBER	Total tax amount	4.5	

Table B–1 GCC Application Database Tables

Column Name	Data Type	Definition	Examples	Related to
SHIPTO				
ACCOUNT_ID	NUMBER	Key field: Account number	23	Foreign key to ACCOUNT
SHIPTO_NUM	NUMBER	Key field: Ship to number	45	
WAREHOUSE_ID	NUMBER	Warehouse identifier	345	
SH_ADDRESS	VARCHAR2(20)	Address of warehouse	244 Mill St.	
SH_CITY	VARCHAR2(10)	City of warehouse	Nashua	
SH_STATE	VARCHAR2(2)	State	NH	
SH_COUNTY	VARCHAR2(10)	County	Hillsboro	
SH_POSTAL_CODE	NUMBER	Ship location postal code	03062	
SH_TAX_RATE	NUMBER	Ship location tax rate	0.06	
SH_DESCRIPTION	VARCHAR2(35)	Description of ship location	RAF Blygh	
SH_LOCTYPE_DESC	VARCHAR2(30)	Description of location type	Plant	
PAYMENT_CODE				
PAYMENT_ID	NUMBER	Key field: Payment code	23	
PAYMENT_DESC	VARCHAR2(20)	Description of payment method		
SALESPERSON				
SALESPERSON_ID	NUMBER	Key field: Salesperson	23	
SALESPERSON_NAME	VARCHAR2(20)	Salesperson name	Greg White	
MANAGER_ID	NUMBER	Manager's identifier	45	
SEGMENTS				
SEGMENT_ID	NUMBER	Key field: Segment number	23	
SEGMENT_DESC	VARCHAR2(20)	Description of segment		

Table B–1 GCC Application Database Tables

Column Name	Data Type	Definition	Examples	Related to
DAYSCAL				
DATE_DESC	DATE	Key field: Date Stampr	29-NOV-1998	
DAY_OF_MONTH	NUMBER	Day of the month	1	
DAY_OR_YEAR	NUMBER	Day of the year	325	
MONTH_DESC	VARCHAR2(9)	Name of the month	SEPTEMBER	
MONTH_ID	NUMBER	Identifer for the Month level	101998	
MONTH_NUMBER	NUMBER	Month of the year	9	
QUARTER	NUMBER	Quarter of the year	4	
QUARTER_ID	NUMBER	Identifer for the Quarter level	41998	
WEEK_ID	NUMBER	Identifer for the Week level	471998	
WEEK_OF_YEAR	NUMBER	Week of the year	47	
YEAR	NUMBER	Year number	1998	

The Flat File Sources

The Global Computing Company Data Mart pulls data from a comma separated file that describes sales channels and a fixed length file that describes their products.

Table B–2 Descriptions of the Flat File Sources

Field Name	Data Type	Definition	Examples	Related to
CHANNEL.csv				
CHANNEL_ID	NUMBER	Key field: Channel ID	2	
CHANNEL_DESC	VARCHAR2(20)	Description of channel	Internet	
PRODUCT.fix				
ITEM_ID	VARCHAR2(11)	Key field: Product ID	L14-3467-EE	
PACKAGING	VARCHAR2(20)	Package identifier	Executive	Attribute of item
ITEM_DESC	VARCHAR2(35)	Product description	Sentinel Standard	
ITEM_SOURCE	VARCHAR2(10)	Domestic or foreign	D or F	
FAMILY_ID	NUMBER	Product family identifier	45	Grouping of items; subset of class
FAMILY_DESC	VARCHAR2(18)	Class description	Portable PCs	
CLASS_ID	NUMBER	Product class identifier	23	Grouping of product families
CLASS_DESC	VARCHAR2(20)	Class description	Hardware	

Table B–3 Sample of Data from the channels.csv File

channel_id	channel_desc	channel_class
2	Direct Sales	DIRECT
3	Catalog	INDIRECT
4	Internet	INDIRECT

Table B–4 Sample of Data from the products.fix File

Item ID	Packaging	Description	SID	FID	Family Desc	CID	Class Description
L132356-ESPack	Laptop Value Pack	Envoy Standard	D	4	Portable PCs	D	Hardware
L143467-EE	Executive	Envoy Executive	D	4	Portable PCs	D	Hardware
L154578-EA	Individual Pack	Envoy Ambassador	D	4	Portable PCs	D	Hardware
L165689-SS	Individual Pack	Sentinel Standard	D	5	Desktop	D	Hardware
L171245-SF	Individual Pack	Sentinel Financial	D	5	Desktop	D	Hardware
L182356-SM	Multimedia	Sentinel Multimedia	D	5	Desktop	D	Hardware
…							

The Target Schema

The Global Computing Company Data Mart includes a single target schema called *gccDM*. This star schema contains examples of the basic building blocks used to build a star schema: dimension objects, dimension tables, fact table, staging table, sequence objects, and materialized and conventional views.

---

1. The Process:

The data mart is based on the online order entry system (*GCCAPPS*).

2. The Grain: (What is a fact record, exactly?)

Product to customer sales recorded by daily line item. (Each daily shipment includes aggregate product sales, units, and cost.)

3. Dimensions:

- Days
- Products
- Customers
- Channels

4. Facts:

- Sales in dollars
- Units per sale
- Cost associated with the product

The Data Mart Objects

Table B–5 catalogs all the warehouse objects in the *gccDM* schema, and each subsequent section includes a DDL script for each object as well as a diagram. The DDL scripts and the diagrams were generated by Oracle Warehouse Builder from definitions stored in the *GCC Data Mart* warehouse module which is described in Chapter 4, "Target Definitions."

Table B–5 Data Mart Objects

Dimension Objects	Tables	Views
Days_DIM	Days	Product_Summaries (*MV*)
Products_DIM	Products	Customer_Summaries (*MV*)
Customers_DIM	Customers	DeskTops
Channels_DIM	Channels	
	stageONE	
	Sales (fact table)	

The Top Directory for the *gccDM* warehouse module was configured as the default case, and so Builder stored the generated DDL scripts for these objects in the Oracle Home directory for Builder:

d:\Oracle\owb21\owb\codegen\ddl

For example, the script that creates the Days table is stored as:

d:\Oracle\owb21\owb\codegen\ddl\Days.ddl

The following pages summarize the contents of the generated scripts stored in the Top Directory.

Note:

- These scripts are for the version of the target schema that uses the Type 1 method to refresh the data mart.
- The headers for the following scripts, Channels.ddl for example, are file names generated by Builder.
- *These scripts have been slightly modified to improve their readability.*

Dimension Objects and Tables

Builder generates three scripts for each dimension. Theses scripts create the:

- Dimension object
- Underlying table
- Analyze statement

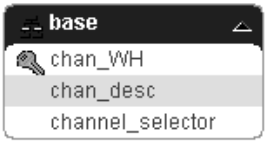
The analyze statement is generated only if the configuration parameter Analyze Table: Estimate Percent is set to a nonzero value.

Any constraints defined on the table, such as a PK or UK, are included in the script that creates the underlying table.

Channels Dimension

The Channels dimension has only the base level of aggregation and no hierarchies.

Channels_DIM.ddl




```
CREATE  DIMENSION Channels_DIM
LEVEL base IS  CHANNELS.bs_chan_WH
ATTRIBUTE base DETERMINES (bs_chan_desc,bs_channel_selector);
```

Channels.ddl

```
Create Table Channels (
bs_chan_WH NUMBER ,
bs_chan_desc VARCHAR2(20),
bs_channel_selector NUMBER )
TABLESPACE USERS PARALLEL NOLOGGING;

ALTER TABLE Channels ADD
CONSTRAINT ch_natural_UK UNIQUE(bs_channel_selector);

ALTER TABLE Channels ADD
CONSTRAINT ch_base_PK PRIMARY KEY(bs_chan_WH);
```

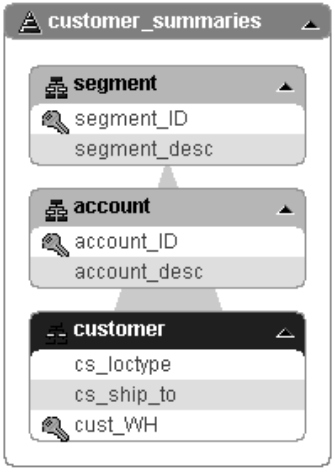
Channels_ANALYZE.ddl

```
ANALYZE TABLE Channels ESTIMATE STATISTICS SAMPLE 99 PERCENT;
```

Customers Dimension

The *Customers* dimension has three levels of aggregation and a single hierarchy called *customer_summaries*. The ***Customer_Summaries*** materialized view references the *Customers_DIM* and *Days_DIM* dimension and can exploit both of their hierarchies to rewrite queries for better performance.

Customers_DIM.ddl




```
CREATE  DIMENSION Customers_DIM
LEVEL account IS
        (CUSTOMERS.at_account_ID,CUSTOMERS.cs_cs_ship_to)
LEVEL segment IS
      CUSTOMERS.sg_segment_ID
LEVEL customer IS
      CUSTOMERS.cs_cust_WH
HIERARCHY customer_summaries (
  customer CHILD OF
  account CHILD OF segment )
ATTRIBUTE account DETERMINES (at_account_desc)
ATTRIBUTE segment DETERMINES (sg_segment_desc)
ATTRIBUTE customer DETERMINES
  (cs_cs_loctype,cs_cs_ship_to);
```

Customers.ddl

```
Create Table Customers (
at_account_ID NUMBER ,
at_account_desc VARCHAR2(35),
cs_cs_loctype VARCHAR2(20),
cs_cs_ship_to NUMBER ,
cs_cust_WH NUMBER ,
sg_segment_ID NUMBER ,
sg_segment_desc VARCHAR2(20))
TABLESPACE USERS PARALLEL NOLOGGING ;

ALTER TABLE Customers ADD
CONSTRAINT cs_natural_UK UNIQUE(at_account_ID,cs_cs_ship_to);

ALTER TABLE Customers ADD
CONSTRAINT cs_customer_PK PRIMARY KEY(cs_cust_WH);
```

Customers_ANALYZE.ddl

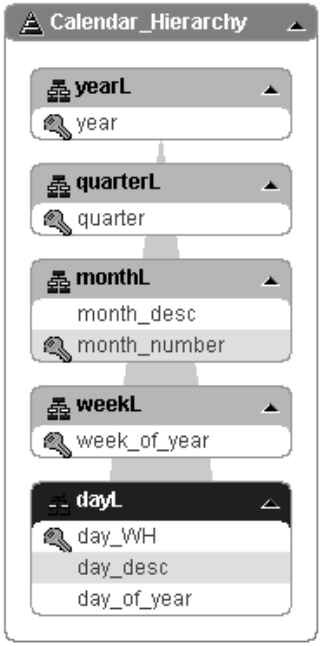
```
ANALYZE TABLE Customers ESTIMATE STATISTICS SAMPLE 75 PERCENT;
```

Days Dimension

The *Days* dimension has five levels of aggregation and a single hierarchy called *Calendar_Hierarchy*. Both the ***Customer_Summaries*** and *Product_Summaries* materialized views reference the *Days_DIM* dimension and can exploit its hierarchy.

Days_DIM.ddl

```
CREATE  DIMENSION Days_DIM
LEVEL dayL IS  DAYS.dl_day_WH
LEVEL weekL IS  DAYS.wl_week_of_year
LEVEL monthL IS  DAYS.ml_month_number
LEVEL quarterL IS  DAYS.ql_quarter
LEVEL yearL IS  DAYS.yl_year
HIERARCHY Calendar_Hierarchy (
 dayL CHILD OF
 weekL CHILD OF
 monthL CHILD OF
 quarterL CHILD OF yearL )
ATTRIBUTE dayL DETERMINES (dl_day_desc,dl_day_of_year)
ATTRIBUTE monthL DETERMINES (ml_month_desc);
```



Days.ddl

```
Create Table Days (
dl_day_WH NUMBER ,
dl_day_desc DATE,
dl_day_of_year NUMBER ,
ml_month_desc VARCHAR2(9),
ml_month_number NUMBER ,
ql_quarter NUMBER ,
wl_week_of_year NUMBER ,
yl_year NUMBER )
TABLESPACE USERS PARALLEL NOLOGGING :

ALTER TABLE Days ADD
CONSTRAINT dy_natural_UK UNIQUE(dl_day_desc);

ALTER TABLE Days ADD
CONSTRAINT da_dayL_PK PRIMARY KEY(dl_day_WH);
```

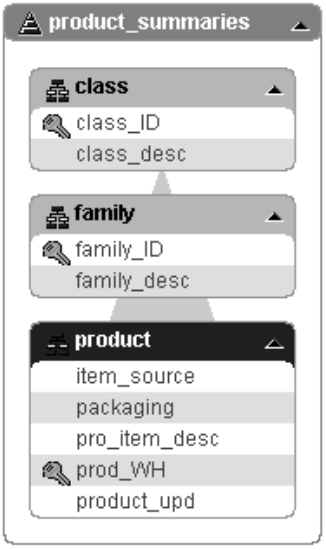
Days_ANALYZE.ddl

```
ANALYZE TABLE Days ESTIMATE STATISTICS SAMPLE 50 PERCENT;
```

Products Dimension

The *Products* dimension has three levels of aggregation and a single hierarchy called *product_summaries*. The ***Product_Summaries*** materialized view references the *Products_DIM* and *Days_DIM* dimension and can exploit both of their hierarchies to rewrite queries for better performance.

Products_DIM.ddl




```
CREATE  DIMENSION Products_DIM
LEVEL product IS  PRODUCTS.pro_prod_WH
LEVEL family IS  PRODUCTS.fa_family_ID
LEVEL class IS  PRODUCTS.cl_class_ID
HIERARCHY product_summaries (
  product CHILD OF
  family CHILD OF
  class )
ATTRIBUTE product DETERMINES (pro_item_source,pro_
packaging,pro_pro_item_desc,pro_product_upc)
ATTRIBUTE family DETERMINES (fa_family_desc)
ATTRIBUTE class DETERMINES (cl_class_desc);
```

Products.ddl

```
Create Table Products (
cl_class_ID NUMBER ,
cl_class_desc VARCHAR2(20),
fa_family_ID NUMBER ,
fa_family_desc VARCHAR2(20),
pro_item_source VARCHAR2(30),
pro_packaging VARCHAR2(20),
pro_pro_item_desc VARCHAR2(35),
pro_prod_WH NUMBER,
pro_product_upc VARCHAR2(11))
TABLESPACE USERS PARALLEL NOLOGGING ;

ALTER TABLE Products ADD
CONSTRAINT pd_natural_UK UNIQUE(pro_product_upc);

ALTER TABLE Products ADD
CONSTRAINT pro_product_PK PRIMARY KEY(pro_prod_WH);
```

Products_ANALYZE.ddl

```
ANALYZE TABLE Products ESTIMATE STATISTICS SAMPLE 50 PERCENT;
```

Fact and Staging Tables

Builder generates three scripts for each table defined within a warehouse module. These scripts create the:

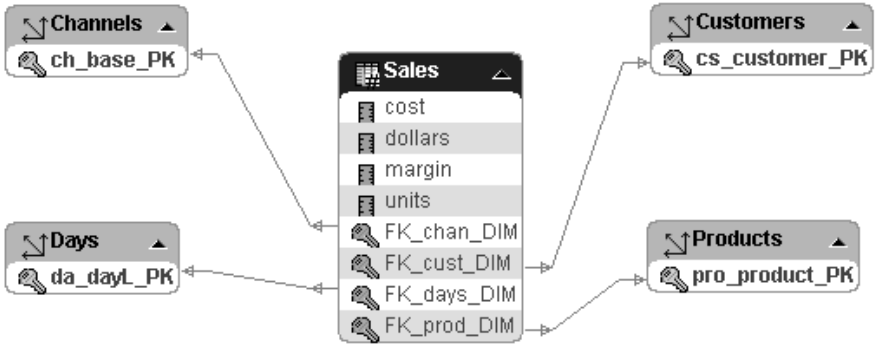
- Table
- Foreign Key constraints defined on the table
- Analyze statement

The analyze statement is generated only if the configuration parameter Analyze Table: Estimate Percent is set to a nonzero value.

Any non-foreign constraints defined in the table, such as a PK or UK, are included in the script that creates the table.

Sales Fact Table

The *Sales* fact table has foreign key reference columns and four measures.



Sales.ddl

```
Create Table Sales (
bs_chan_WH NUMBER ,
cost NUMBER(36,2),
cs_cust_WH NUMBER ,
dl_day_WH NUMBER ,
dollars NUMBER(36,2),
margin NUMBER(36,2),
pro_prod_WH NUMBER ,
units NUMBER )
TABLESPACE USERS PARALLEL NOLOGGING ;
```

Sales_FK.ddl

```
ALTER TABLE Sales ADD
CONSTRAINT FK_chan_DIM FOREIGN KEY (bs_chan_WH)
 REFERENCES Channels(bs_chan_WH);

ALTER TABLE Sales ADD
CONSTRAINT FK_cust_DIM FOREIGN KEY (cs_cust_WH)
 REFERENCES Customers(cs_cust_WH);

ALTER TABLE Sales ADD
CONSTRAINT FK_days_DIM FOREIGN KEY (dl_day_WH)
 REFERENCES Days(dl_day_WH);

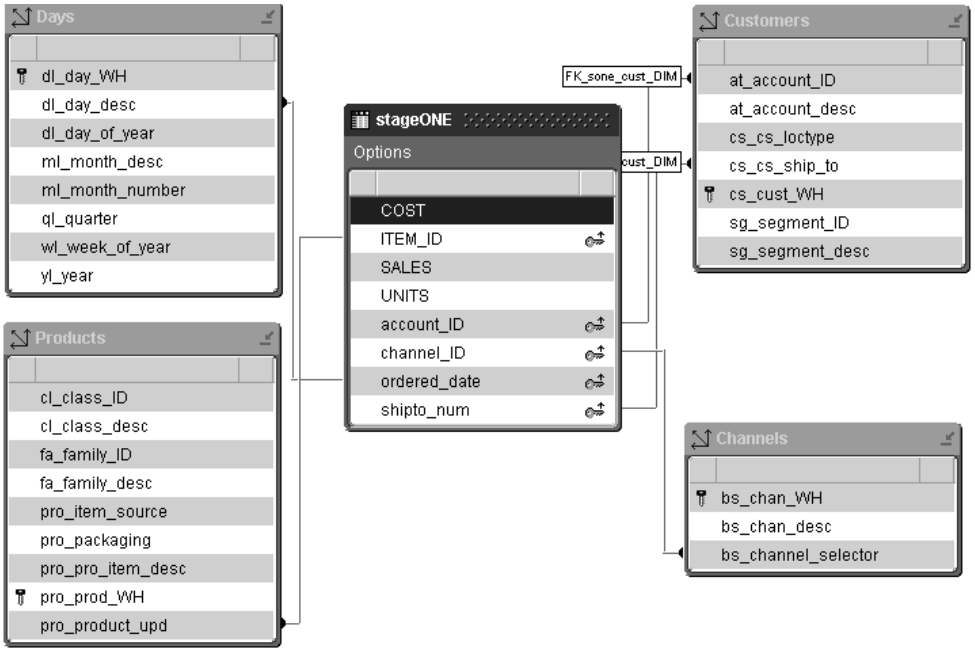
ALTER TABLE Sales ADD
CONSTRAINT FK_prod_DIM FOREIGN KEY (pro_prod_WH)
 REFERENCES Products(pro_prod_WH);
```

Sales_ANALYZE.ddl

```
ANALYZE TABLE Sales ESTIMATE STATISTICS SAMPLE 40 PERCENT;
```

stageONE Table

The *stageONE* fact table has foreign key reference columns and three measures. This table is used primarily to manage warehouse keys.



The format for the diagram of the *stageONE* table differs from the *Sales* table because the *Sales* table was a fact table.

stageONE.ddl

```
Create Table stageONE (
COST NUMBER ,
ITEM_ID VARCHAR2(11),
SALES NUMBER ,
UNITS NUMBER ,
account_ID NUMBER ,
channel_ID NUMBER ,
ordered_date DATE,
shipto_num NUMBER )
TABLESPACE USERS PARALLEL NOLOGGING ;
```

stageONE_FK.ddl

```
ALTER TABLE stageONE ADD
CONSTRAINT FK_sone_day_DIM FOREIGN KEY (
ordered_date)
 REFERENCES Days(dl_day_desc);

ALTER TABLE stageONE ADD
CONSTRAINT FK_sone_prod_DIM FOREIGN KEY (
ITEM_ID)
 REFERENCES Products(pro_product_upc);

ALTER TABLE stageONE ADD
CONSTRAINT FK_sone_cust_DIM FOREIGN KEY (
account_ID,shipto_num)
 REFERENCES Customers(at_account_ID,cs_cs_ship_to);

ALTER TABLE stageONE ADD
CONSTRAINT FK_sone_chan_DIM FOREIGN KEY (
channel_ID)
 REFERENCES Channels(bs_channel_selector);
```

stageONE_ANALYZE.ddl

```
ANALYZE TABLE stageONE ESTIMATE STATISTICS SAMPLE 60 PERCENT;
```

Sequence Objects

The *gccDM* warehouse module contains definitions for three sequence objects that generate values for warehouse key columns in the *Days*, *Products*, and *Customers* tables. Warehouse key values for the *Channels* table and the initial load values for the *Products* table are generated by sequences explicitly defined in the SQL*Loader control scripts.

```
Create Sequence custWHkey
INCREMENT BY 1 START WITH 1;

Create Sequence daysWHkey
INCREMENT BY 1 START WITH 1;

Create Sequence prodWHkey
INCREMENT BY 1 START WITH 100;
```

Views

Builder can generate three scripts which can be used to create a view. These scripts create the following:

- View

  Builder generates a script to create the view when the view's property sheet includes a query that defines the view or when a mapping for the view exists in the warehouse module.

- Constraints defined on materialized views

  Builder generates a script to create constraints on a materialized view when its property sheet includes definitions for constraints.

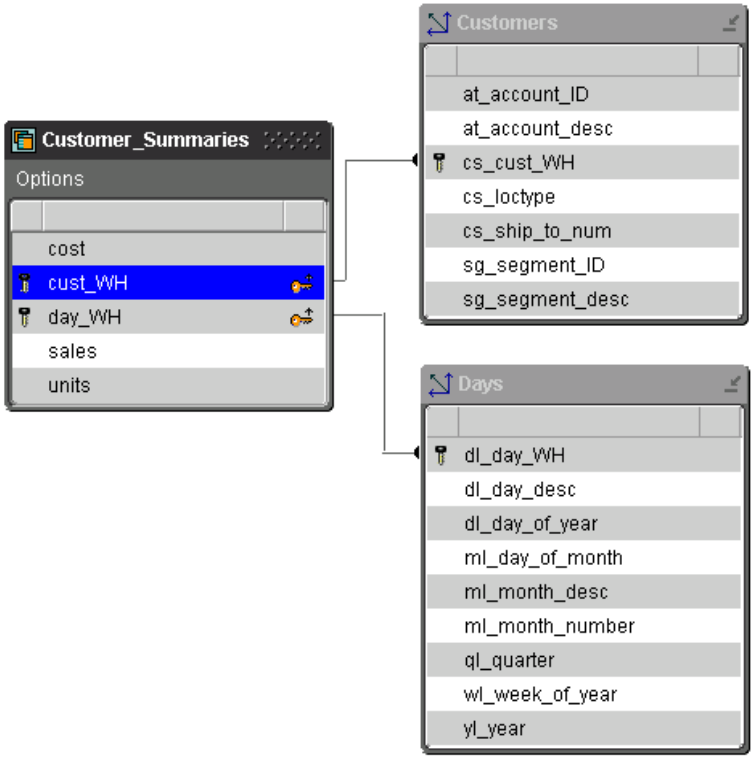
- Synonym

  Typically a materialized view is defined on tables within the target schema and a synonym is not required. If you define a mapping for the view that pulls data from tables in another schema object, then Builder generates a script to create the necessary synonym.

The *gccDM* warehouse module contains definitions for two materialized views and one conventional view, and each of these definitions include query text in their respective property sheets. Foreign key reference constraints are also defined on each view.

Customer_Summaries Materialized View

This materialized view references the *Customers* and *Days* dimensions.



Customer_Summaries.ddl

```
Create Materialized View Customer_Summaries
 TABLESPACE USERS PARALLEL NOLOGGING
BUILD IMMEDIATE REFRESH COMPLETE ENABLE  QUERY REWRITE
AS SELECT
  Sales.dl_day_WH as day_WH,
  Sales.cs_cust_WH as cust_WH,
  SUM(Sales.dollars) sales, SUM(Sales.cost) cost,  SUM(Sales.units) units
FROM
  Sales, Days, Customers
WHERE
  Sales.dl_day_WH = Days.dl_day_WH and
  Sales.cs_cust_WH = Customers.cs_cust_WH
GROUP BY
  Sales.dl_day_WH, Sales.cs_cust_WH;
```

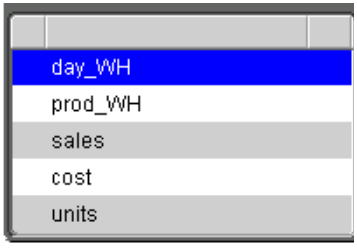
Customer_Summaries_FK.ddl

```
ALTER TABLE Customer_Summaries ADD
CONSTRAINT FKR_cs_dayDIM FOREIGN KEY (day_WH)
 REFERENCES Days(dl_day_WH);

ALTER TABLE Customer_Summaries ADD
CONSTRAINT FKR_cs_prodDIM FOREIGN KEY (cust_WH)
 REFERENCES Customers(cs_cust_WH);
```

DeskTops Conventional View

This conventional view limits access to a subset of products defined by the family description "Desktops."

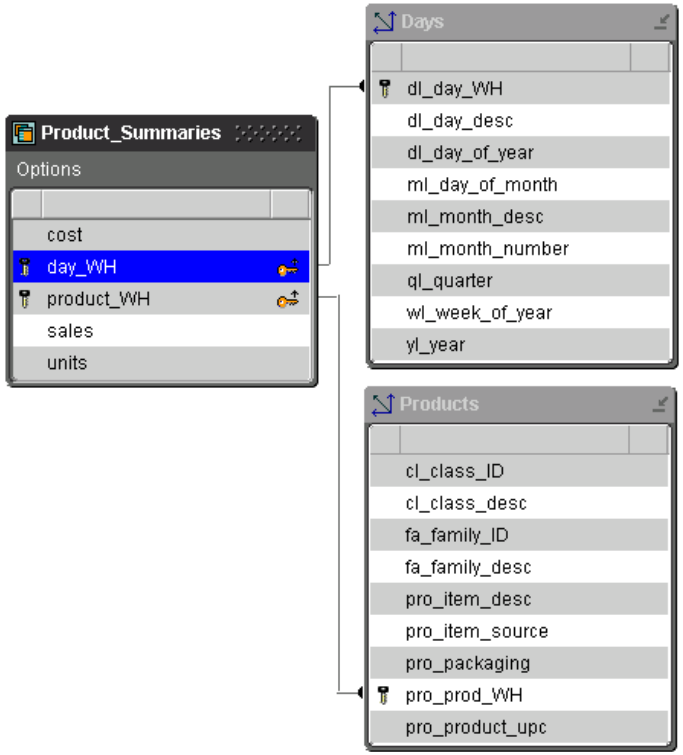


DeskTops.ddl

```
Create Or Replace Force View  DeskTops (
cost,
day_WH,
product_WH,
sales,
units)
AS select
  Sales.dl_day_WH as day_WH,
  Sales.pro_prod_WH as product_WH,
  SUM(Sales.dollars) sales,
  SUM(Sales.cost) cost,
  SUM(Sales.units) units
FROM
  Sales, Days, Products
WHERE
  fa_family_desc like 'DESK%' and
  Sales.dl_day_WH = Days.dl_day_WH and
  Sales.pro_prod_WH = Products.pro_prod_WH
GROUP BY
  Sales.dl_day_WH,
  Sales.pro_prod_WH;
```

Product_Summaries Materialized View

This materialized view references the *Products* and *Days* dimensions.



Product_Summaries.ddl

```
Create Materialized View Product_Summaries
  TABLESPACE USERS PARALLEL NOLOGGING
BUILD IMMEDIATE REFRESH COMPLETE ENABLE  QUERY REWRITE
AS select
  Sales.dl_day_WH as day_WH,
  Sales.pro_prod_WH as product_WH,
  SUM(Sales.dollars) sales,
  SUM(Sales.cost) cost,
  SUM(Sales.units) units
FROM
  Sales, Days, Products
WHERE
  Sales.dl_day_WH = Days.dl_day_WH and
  Sales.pro_prod_WH = Products.pro_prod_WH
GROUP BY Sales.dl_day_WH, Sales.pro_prod_WH;
```

Product_Summaries_FK.ddl

```
ALTER TABLE Product_Summaries ADD
CONSTRAINT FKR_ps_dayDIM FOREIGN KEY (day_WH)
 REFERENCES Days(dl_day_WH);

ALTER TABLE Product_Summaries ADD
CONSTRAINT FKR_ps_prodDIM FOREIGN KEY (product_WH)
 REFERENCES Products(pro_prod_WH);
```

Glossary

additive

Describes a fact (or measure) that can be summarized through addition. An additive fact is the most common type of fact. Examples include Sales, Cost, and Profit. (Contrast with *non-additive, semi-additive*.)

attribute

A descriptive characteristic of one or more levels. Attributes represent logical groupings that enable end users to select data based on like characteristics. Note that in relational modeling, an attribute is defined as a characteristic of an entity. In Oracle 8*i*, an attribute is a column in a dimension that characterizes elements of a single level.

aggregation

The process of consolidating data values into a single value. For example, sales data could be collected on a daily basis and then be aggregated to the week level, the week data could be aggregated to the month level, and so on. The data can then be referred to as *aggregate data*. *Aggregation* is synonymous with *summarization,* and *aggregate data* is synonymous with *summary data.*

aggregate

Summarized data. For example, unit sales of a particular product could be aggregated by day, month, quarter and yearly sales.

ancestor

A value at any level above a given value in a hierarchy. For example, in a Time dimension, the value “1999” might be the ancestor of the values “Q1-99” and “Jan-99.” (See also *descendant, hierarchy, level.*)

attribute

A descriptive characteristic of one or more levels. For example, the Product dimension for a clothing manufacturer might contain a level called Item, one of whose attributes is Color. Attributes represent logical groupings that enable end users to select data based on like characteristics.

Note that in relational modeling, an attribute is defined as a characteristic of an entity. In Oracle 8*i*, an attribute is a column in a dimension that characterizes elements of a single level.

child

A value at the level below a given value in a hierarchy. For example, in a Time dimension, the value "Jan-99" might be the child of the value "Q1-99." A value can be a child for more than one parent if the child value belongs to multiple hierarchies. (See also *hierarchy, level, parent.*)

cleansing

The process of resolving inconsistencies and fixing the anomalies in source data, typically as part of the ETT process. (See also *ETT.*)

Common Warehouse Meta Data

A repository standard used by Oracle data warehousing, decision support, and OLAP tools including Oracle Warehouse Builder. The CWM repository schema is a stand-alone product that other products can share—each product owns only the objects within the CWM repository that it creates.

console

The main window of the Oracle Warehouse Builder application. It contains a menu bar, launcher, and navigator.

data source

A database, application, repository, or file that contributes data.

data mart

A data warehouse that is designed for a particular line of business, such as sales, marketing, or finance. In a dependent data mart, the data can be derived from an enterprise-wide data warehouse (as a dependent data mart). In an independent data mart, data can be collected directly from sources. (See also *data warehouse.*)

data warehouse

A relational database that is designed for query and analysis rather than transaction processing. A data warehouse usually contains historical data that is derived from transaction data, but it can include data from other sources. It separates analysis workload from transaction workload and enables a business to consolidate data from several sources.

In addition to a relational database, a data warehouse environment often consists of an ETT solution, an OLAP engine, client analysis tools, and other applications that manage the process of gathering data and delivering it to business users. (See also *ETT, OLAP.*)

denormalize

The process of allowing redundancy in a table so that it can remain flat. (Contrast with *normalize.*)

derived fact (or measure)

A fact (or measure) that is generated from existing data using a mathematical operation or a data transformation. Examples include averages, totals, percentages, and differences.

dimension

A structure, often composed of one or more hierarchies, that categorizes data. Several distinct dimensions, combined with measures, enable end users to answer business questions. Commonly used dimensions are Customer, Product, and Time. In Oracle 8*i*, a dimension is a database object that defines hierarchical (parent/child) relationships between pairs of column sets. In Express a dimension is a database object that consists of a list of values.

dimension value

specialization of value

One element in the list that makes up a dimension. For example, a computer company might have dimension values in the Product dimension called “LAPPC” and “DESKPC.” Values in the Geography dimension might include “Boston” and “Paris.” Values in the Time dimension might include “MAY96” and “JAN97.”

drill

To navigate from one item to a set of related items. Drilling typically involves navigating up and down through the levels in a hierarchy. When selecting data, you can expand or collapse a hierarchy by drilling down or up in it, respectively. (See also *drill down, drill up.*)

drill down

To expand the view to include child values that are associated with parent values in the hierarchy. (See also *drill, drill up*.)

drill up

To collapse the list of descendant values that are associated with a parent value in the hierarchy.

element

An object or process. For example, a dimension is an object, a mapping is a process, and both are elements.

ETT

Extraction, transformation, and transportation. ETT refers to the methods involved in accessing and manipulating source data and loading it into a data warehouse. The order in which these processes are performed varies.

Note that ETL (extraction, transformation, load) and ETM (extraction, transformation, move) are sometimes used instead of ETT. (See also *data warehouse, extraction, transformation, transportation.*)

editor

A window in Oracle Warehouse Builder used to define or edit objects and their relationships to each other.

extraction

The process of taking data out of a source as part of an initial phase of ETT. (See also *ETT.*)

fact table

A table in a star schema that contains facts. A fact table typically has two types of columns: those that contain facts and those that are foreign keys to dimension tables. The primary key of a fact table is usually a composite key that is made up of all of its foreign keys.

A fact table might contain either detail level facts or facts that have been aggregated (fact tables that contain aggregated facts are often instead called *summary tables*). A fact table usually contains facts with the same level of aggregation.

fact/measure

Data, usually numeric and additive, that can be examined and analyzed. Values for facts or measures are usually not known in advance; they are observed and stored. Examples include Sales, Cost, and Profit. *Fact* and *measure* are synonymous; *fact* is more commonly used with

relational environments, *measure* is more commonly used with multidimensional environments.

file-to-table mapping

Maps data from flat files to tables in the warehouse.

hierarchy

A logical structure that uses ordered levels as a means of organizing data. A hierarchy can be used to define data aggregation; for example, in a Time dimension, a hierarchy might be used to aggregate data from the "Month" level to the "Quarter" level to the "Year" level. A hierarchy can also be used to define a navigational drill path, regardless of whether the levels in the hierarchy represent aggregated totals. (See also *dimension, level.*)Level

A position in a hierarchy. For example, a Time dimension might have a hierarchy that represents data at the "Month," "Quarter," and "Year" levels. (See also *family, hierarchy.*)

hub module

The metadata container for process data.

integrator

Software that works with Oracle Warehouse Builder to facilitate definition, design, and extraction of source data. Examples of integrators include the Oracle Applications Integrator, the PeopleSoft Integrator, and the SAP Integrator.

launcher

The panel in the Oracle Warehouse Builder console window that contains buttons to control the active environment. These environments include the Project environment, Administration environment, and Transformation Library environment. Environments act as separators between the separate functions of the Oracle Warehouse Builder product. For example, you use the Transformation Library to define the reusable transformation functions, and you use the Project environment to define the metadata used to create a data warehouse.

level

A position in a hierarchy. For example, a Time dimension might have a hierarchy that represents data at the "Month," "Quarter," and "Year" levels.

(See also *family, hierarchy.*)

level value table

A database table that stores the values or data for the levels you created as part of your dimensions and hierarchies.

mapping

The definition of the relationship and data flow between source and target objects.

metadata

Data that describes data and other structures, such as objects, business rules, and processes. For example, the schema design of a data warehouse is typically stored in a repository as metadata, which is used to generate scripts used to build and populate the data warehouse. A repository contains metadata.

Examples include: for data, the definition of a source to target transformation that is used to generate and populate the data warehouse; for information, definitions of tables, columns and associations that are stored inside a relational modeling tool; for business rule, discount by 10 percent after selling 1,000 items.

model

An object that represents something to be made. A representative style, plan, or design. metadata that defines the structure of the data warehouse.

module (source, target and hub)

The metadata containers for source data, process data, and warehouse data.

navigator

A panel in the Oracle Warehouse Builder console that displays the objects for the active environment in a tree structure.

nonadditive

Describes a fact (or measure) that cannot be summarized through addition. An example includes Average. (Contrast with *additive, semi-additive.*)

normalize

In a relational database, the process of removing redundancy in data by separating the data into multiple tables. (Contrast with denormalize.)

The process of removing redundancy in data by separating the data into multiple tables. (Kimball)

operational data store (ODS)

The cleaned, transformed data from a particular source database.

OLAP

Online analytical processing. OLAP functionality is characterized by dynamic, multidimensional analysis of historical data, which supports activities such as the following:

- Calculating across dimensions and through hierarchies
- Analyzing trends
- Drilling up and down through hierarchies
- Rotating to change the dimensional orientation

OLAP tools can run against a multidimensional database or interact directly with a relational database.

Oracle Warehouse

A data warehouse built using Oracle Warehouse Builder.

Oracle Warehouse Builder

The Oracle product used to build and manage an Oracle Warehouse.

Oracle Warehouse Builder Repository

Stores the Oracle Warehouse Builder metadata definitions used to build the Warehouse. A storage facility for metadata. For example, the Designer 2000 repository stores ER diagrams and business processes. (See also *metadata*.)

Oracle Warehouse Builder Repository administrator

The repository administrator is the information specialist that manages the Repository and other administration needs.

Warehouse administrator

The warehouse administrator is the information specialist that manages the warehouse database and warehouse management applications. For example, the warehouse administrator would be responsible for managing and monitoring periodic updates of the warehouse database.

Warehouse designer

The warehouse designer is the information specialist that uses Oracle Warehouse Builder to define the metadata that is used to create the data warehouse(s) for your enterprise.

parent

A value at the level above a given value in a hierarchy. For example, in a Time dimension, the value "Q1-99" might be the parent of the value "Jan-99." (See also *child, hierarchy, level.*)

project

A project contains all design definitions and information needed by the Oracle Warehouse Builder to build a warehouse. Oracle Warehouse Builder provides a project structure to help users organize their work, and support versioning.

report-to-table mapping

Maps PeopleSoft trees to tables in the warehouse.

schema

A collection of related database objects. Relational schemas are grouped by database user ID and include tables, views, and other objects. (See also *snowflake schema, star schema.*)

semi-additive

Describes a fact (or measure) that can be summarized through addition along some, but not all, dimensions. Examples include Headcount and On Hand Stock. (Contrast with *additive, nonadditive.*)

sequence

A database schema object supported by Oracle8i that is a series of sequential numbers generated by Oracle8i's sequence generator. Oracle stores sequences as rows in a single data dictionary table in the SYSTEM tablespace. A sequence definition indicates general information: the name of the sequence, whether it ascends or descends, the interval between numbers, and other information.

short name

The short name is a unique identifier that is used as the root name for all related objects created in Oracle Warehouse Builder.

snowflake schema

A type of star schema in which the dimension tables are partly or fully normalized. (See also *schema, star schema.*)

Software Library

Contains the integrators currently installed for Oracle Warehouse Builder. Accessed via the Administration environment.

source

A database, application, file, or other storage facility from which the data in a data warehouse is derived.

star schema

A relational schema whose design represents a multidimensional data model. The star schema consists of one or more fact tables and one or more dimension tables that are related through foreign keys. (See also *schema, snowflake schema.*)

subject area

A classification system that represents or distinguishes parts of an organization or areas of knowledge. A data mart is often developed to support a subject area such as sales, marketing, or geography. (See also *data mart.*)

table

A layout of data in columns.

target

Holds the intermediate or final results of any part of the ETT process. The target of the entire ETT process is the data warehouse. (See also *data warehouse, ETT.*)

transformation

The process of manipulating data. Any manipulation beyond copying is a transformation. Examples include cleansing, aggregating, and integrating data from multiple sources.

Transformation Library

Stores the reusable formulas for the transformation of data as it moves between source and target objects. Accessed via a button in the Oracle Warehouse Builder launcher.

transportation

The process of moving copied or transformed data from a source to a data warehouse. (See also *transformation.*)

tree-to-dimension mapping

A map between a PeopleSoft tree and an Oracle Warehouse Builder dimension.

validation

The process of verifying metadata definitions and configuration parameters.

versioning

The ability to create new versions of a data warehouse project for new requirements and changes.

Index

A

after the load, 13-24
aggregation
  define level, 4-5
ALWAYS_ANTI_JOIN parameter, 13-23
analysis
  drill down, 12-49
  drill out, 12-49
analyze data, 12-34
application
  generic file based, 3-22
attribute ID, 4-7
audit log, 10-7

B

batch user
  Oracle Enterprise Manager, 7-5
best and worst sellers, 2-3
bitmap index
  configure, 8-12
  define, 8-9
blocks
  database reads, 13-22
  default size, 13-22
buffer cache, 13-22
Builder
  configuration parameters init.ora, 13-23
  console, 3-4
  logon to repository, 3-2
  Oracle home, A-7
  Project environment, 3-5
  repository, A-8
  repository overview, 3-2
  Runtime Assistant, 7-9
  runtime tables, 7-9
business areas
  calculated items, 12-25
  create, 12-10
  define, 12-4
  hide items, 12-23
  identify sources, 12-16
  rename folders, 12-20
  selecting tables, 12-31
business process, 2-6
button
  commit, 3-5

C

calculated items
  business areas, 12-25
case study, 6-14
  accessing data, 12-39
  adding datafiles, 13-14
  base configuration, 2-22
  bitmap index, 8-9
  Channels dimension, 4-17
  Customer_Summaries materialized view, 4-29
  Customers dimension, 4-17
  data sources, 2-15
  Days dimension, 4-16
  default Oracle homes, A-4
  DeskTops view, 4-30
  fact table overview, 2-13
  flat file source module, 3-21
  GCCAPPS order entry system, 2-15

high level maps, 2-20
Indispensable Reports, 2-3
init.ora parameter values, A-6
mapping
Sales table, 6-28
materialized views, 4-24
naming mode, 3-11
operating environment, xxi, 7-2, A-1
Oracle8i database instance, A-3
principal dimensions, 2-8
Product_Summaries hierarchy, 4-8
Products dimension, 4-3
repositories, A-8
schema objects, 7-4
secondary sources, 2-19
software components, 7-3
source modules, 3-12
staging table mapping, 6-18
The Typical Questions, 2-3
truncate_table transformation, 6-40
warehouse key names, 2-12
warehouse module, 3-6
Windows NT host, A-2
character set
flat file, 3-22, 5-11, 5-12
charts, 12-37
cleanup
transform, 6-40, 14-10
cleanup transform, 14-10
client-side cubic cache, 12-6
clone a definition for a table, 6-19
column
add, 6-21
default order, 4-14, 5-6
generated, 4-5
position, 4-14, 5-6
properties, 4-14, 5-6
remove, 6-21
commit
button, 3-5
toolbar icon, 3-5
components
Oracle Warehouse Builder, 1-10
configuration
database link, 8-8
OEM user/password/domain, 8-5
Oracle Enterprise Manager, 8-4
Oracle8i instance, 13-21
panels, 8-2
connection information
configure database link, 8-8
New Module wizard, 3-7
OEM job connect string, 8-6
warehouse module, 3-7
wizard page, 3-15
console, 3-4
Builder, 3-4
constraint
add, 6-22
change UK to PK, 4-23
fact table
foreign key reference, 4-20
remove, 6-22
container
flat file source, 3-21
modules, 3-1
source module, 3-12
target schema, 3-6
warehouse module, 3-6
conventional view
DeskTops, 4-30
create
diagram of source definitions, 5-7
hierarchy, 4-9
runtime library, 7-9
warehouse module, 3-6, 3-12, 3-21
Create Like Wizard, 6-19
crosstab format reports, 12-35
creating, 12-44
custom transform
cleanup, 6-40, 14-10
load lookup table, 14-6
subtract, 6-31
customer support, xviii

D

data
analyze, 12-38
audit, 1-8

investigate, 12-38
data flow plans
scheduling, 13-12
data mart
access, 12-1, 12-3
compared with data warehouse, 1-3
dependent versus independent, 2-2, 2-18
growth, 13-13
hybrid, 1-5
performance, 13-17, 13-21
populate, 10-2
publish, 11-1
query rewrite, 13-19
refresh, 13-3
refresh Products dimension, 13-5
update
fact table, 13-11
data refresh
scheduling, 13-12
data sources
database tables, 2-15
flat file sources, 2-19
import tables, A-16
source module, 5-3
data warehouse functions, 13-23
databaase link
OEM job connect string, 8-6
database
block size, 7-11
buffer cache, 13-22
database link
configure, 8-8
connection information, 3-15
deploy, 9-4
GCCAPPS, 3-16
generated script, 9-4
New Database Link panel, 3-16
Days dimension
diagram, B-12
DB_BLOCK_BUFFERS parameter, 13-22
DB_BLOCK_SIZE parameter, 13-22
DB_FILE_MULTIBLOCK_READ_COUNT
parameter, 13-22
DBA Studio, 13-13
DBMS_OLAP package, 13-20
default Oracle homes, A-4
definition
attribute, 4-6
valid and invalid, 8-17
validate a set of definitions, 8-18
deploy
database link, 9-4
dimension object and table, 9-5
generated script
loader mapping, 9-7
PL/SQL Libraries, 9-8
PL/SQL mapping, 9-8
table, 9-9
design process
star schema, 1-6
detail mapping
Detail Mapping Editor, 6-8, 6-43
property sheet, 6-43
developmental phases, 2-6, 2-22
diagram
Customers dimension, B-11
Days dimension, B-12
GCCAPPS, 2-16
Global Computing Company database, B-2
Products dimension, B-13
Sales fact table, B-14
source definitions, 5-7
stageONE table, B-16
dimension
attribute definition, 4-6
attributes, 2-10
Channels, 4-17
generated scripts, B-10
column position, 4-14
create products dimension, 4-3
Customers, 4-17
generated scripts, B-11
Days, 4-16
generated scripts, B-12
define a level of aggregation, 4-5
deploy object and table, 9-5
describe a dimension, 2-8
generated scripts, 9-5
hierarchy, 2-10, 4-9
index, 13-17

mapping, 6-4
prefix, 4-5
primary key constraint, 2-11
Products
generated scripts, B-13
lookup table, 14-6
refresh
Type 2, 14-2
table
property sheet, 4-10
validate example, 8-18
dimensional design, 2-6
Discover 3.1, A-15
documentation
ordering, xviii
drill down
analysis, 12-49
in reports, 12-45
to detail, 12-27
drill out
analysis, 12-49

E

editor
Detail Mapping Editor, 6-8, 6-43
End User Layer (EUL), 12-4
install, 12-7
environment
Project, 3-5
error information
log, 10-7
exception formatting, 12-37

F

fact table
column position, 4-14
define foreign key reference, 4-20
general description, 2-13
grain, 2-7
index, 13-18
Sales, 4-18
table
property sheet, 4-22, 8-11
update, 13-11
filter
how to add, 6-25
flat file
case study sources, 2-19
channel.csv, 5-11
character set, 3-22, 5-11, 5-12
create format definition, 5-9
format
update, 5-18
generic file based application, 3-22
product.fix, 5-14
property sheet, 5-18
source container, 3-21
Flat File Sample Wizard, 5-11
format
channel flat file, 5-11
flat file, 5-14
formats of reports, 12-34

G

GCC Data Mart
materialized view
Customer_Summaries, B-19
view
desktops, B-20
GCC_MAIN tablespace, 7-9
GCC_TEMP tablespace, 7-9
GCCAPPS
data source, 2-15
database link, 3-16
diagram, 2-16
import definitions, 5-3
import tables, A-16
source module, 3-12
tables, 2-16
gccDM
dimension
products, 4-3
generate scripts, 9-3
generated
column name, 4-5
generated script
Channels dimension, B-10

Customers dimension, B-11
Customers_Summaries, B-19
database link, 9-4
Days dimension, B-12
desktops view, B-20
dimension, 9-5
loader mapping, 9-7
materialized view, 9-7
PL/SQL Libraries, 9-8
PL/SQL mapping, 9-8
Products dimension, B-13
Sales fact table, B-14
sequence object, 9-8
stageONE table, B-16
table, 9-9
generic file based application, 3-22
Global Computing Company
application database, B-2
brief description, xix, 2-1
Global Computing Company Application Database
description of flat files, B-3
description of tables, B-3
granularity, 2-7
graphs, 12-37, 12-47

H

hash joins
blocks, 13-22
enable, 13-23
memory, 13-22
HASH_AREA_SIZE parameter, 13-22
HASH_JOIN_ENABLED parameter, 13-23
HASH_MULTIBLOCK_IO_COUNT parameter, 13-22
hierarchy
create definition, 4-9
dimension, 2-10
order of levels, 4-9
Product_Summaries, 4-8
high level maps
case study, 2-20
description, 6-4
how to
add filter to mapping, 6-25
add parameters to mapping, 6-25
configure a physical instance, 8-2
configure Preferred Credentials, 7-7
create a Windows batch user, 7-5
create and configure definition for index, 8-9
create Builder runtime tables, 7-9
create conventional view, 4-30
create database link, 3-16
create detail mapping, 6-8
create dimension, 4-3
create fact table, 4-19
create format definition, 5-9
create format fixed-file, 5-14
create hierarchy, 4-8
create high-level mapping, 6-4
create local variable, 6-16
create materialized view, 4-24
create sequence object, 4-33
create source module, 3-12
create source module for flat file, 3-21
create transform, 6-40
create transformation, 6-44
create transformation category, 6-44
create warehouse module, 3-6
define level attribute, 4-5
define level of aggregation, 4-5
define post-map trigger, 6-40
deploy target objects, 9-13
edit database source definition, 5-6
generate scripts, 9-3
import database source tables, A-16
import database sources, 5-3
order levels within hierarchy, 4-9
register jobs, 9-15
rename ID column, 4-7
specify primary source, 6-5
update dimension definition, 4-10
update mapping, 6-13
HTML files
exporting data to, 11-2, 12-50
hybrid data mart, 1-5
hyperdrill, 12-27
hypertext markup language
exporting data to, 12-50

I

icon
  commit, 3-5
ID attribute, 4-7
Import Metadata Wizard, 5-3, 5-9
independent data mart, 1-5
index
  additional, 8-9
  data mart performance, 13-17
  fact table, 13-18
indispensable reports, 2-3
init.ora values
  case study, A-6
  performance, 13-21
Internet
  view data from, 12-4
I/O subsystem
  DB_BLOCK_SIZE parameter, 13-22
item classes
  define, 12-27

J

jobs
  dependency graph, 10-3
  dependency management, 13-12
  scheduling, 10-2, 13-2
joins
  Oracle Reports, 11-8, 11-9

K

key
  natural, 2-16
  production, 2-16

L

level
  define, 4-5
Live Previewer, 11-18
load jobs
  background services, 10-4
  dependency graph, 10-3
  manually schedule, 10-4
  sequence, 10-2
load lookup table transform, 14-6
Load Wizard, 12-30
loader
  generated scripts, 9-7
loadlookuptable transform, 14-6
local variable
  sequence object, 6-16
Local Variable Wizard, 6-16
log
  audit and error, 10-7
logical name, 3-11
logon
  Builder repository, 3-2
logon sequence for Builder, 3-3

M

manage
  DBA Studio, 13-13
  physical storage growth, 13-13
  space utilization, 13-14
mapping
  add filter, 6-25
  case study, 2-20
  Customers dimension, 6-4, 6-33
  definition, 1-9
  Detailed Mapping Editor, 6-8
  dimension, 6-4
  high-level, 6-4, 6-14
  map_prodSTAGE, 13-5
  PL/SQL generated scripts, 9-8
  PL/SQL scripts, 9-8
  primary source, 6-34
  Products dimension, 6-14
  property sheet, 6-13
  refresh_Products_type_1, 13-5
  Sales table, 6-28
  secondary source, 6-9
  specify primary source, 6-5
  staging table, 6-18
master-detail crosstab format reports, 12-36
materialized view
  column position, 4-14
  Customer_Summaries, 4-29

generated scripts, 9-7
New Materialized View wizard, 4-25
performance, 13-18
Product_Summaries, 4-24
property sheet, 4-28
memory
allocation for sorts, 13-22
module
containers, 3-1
GCCAPPS, 5-3
New Module Wizard, 3-6
source module, 3-21

N

names
Builder objects general description, 3-11
logical and physical, 3-11
naming mode, 3-11
natural key, 2-16
navigation tree
Run Time Audit Viewer, 10-8
search, 3-10
Network Computing model
Oracle Reports and, 11-2
New Database Link, 3-16
New Dimension Wizard, 4-3
New Fact wizard, 4-19
New Graph Wizard, 12-47
New Mapping Wizard, 6-4
New Parameter wizard, 6-25
New Transformation Category Wizard, 6-30
NLS character set, 3-22, 5-12
NOT IN operator
evaluate, 13-23

O

OEM Management Server, 7-9
operating environment
case study, xxi, A-1
Oracle Agent, 7-9
Oracle default homes, A-4
Oracle Discoverer, 12-2
Administration Edition, 12-3
components, 12-2
End User Layer (EUL), 12-4
logging in, 12-39
toolbar, 12-38
User Edition, 12-3
Viewer, 12-4
Oracle Enterprise Manager
batch user, 7-5
DBA Studio, 13-13
Job History Pane, 13-24
modify parameter, 13-11
overview, 1-11
repository, A-8
required services, 7-9
schedule jobs, 13-12
space utilization report, 13-14
summary, 13-2
Summary Advisor, 13-20
Oracle home
default, A-4
ora816, A-4
owb21, A-7
Oracle Intelligent Agent, 8-5
Oracle Reports, 1-13
components of, 11-3
features, 11-2
formats for, 11-2
formatting, 11-18
modifying, 11-20
Release 6i, A-15
sum of columns, 11-9
Oracle Warehouse Builder
overview, 1-10
Oracle Workflow
dependency management, 13-12
Oracle8i
case study instance, A-3
components, 7-3
configuration, 13-21
data warehouse components, 1-9
init.ora file, 13-21
outliers, 12-37
overview
Builder repository, 3-2
Oracle Enterprise Manager, 1-11

Oracle Warehouse Builder, 1-10
Oracle8i, 1-9
source module, 3-12
star schema, 2-6
wizards and navigation trees, 3-8

P

page-detail crosstab
create report, 12-44
format reports, 12-36
PARALLEL_MAX_SERVERS parameter, 13-22
PARALLEL_MIN_SERVERS parameter, 13-22
parameters
init.ora
Builder specific, 13-23
data warehouse functions, 13-23
input/output, 13-22
resource usage, 13-22
modify dynamically, 13-11
performance, 13-20
start and end date, 13-11
performance, 13-17
data mart, 13-17
DBMS_OLAP package, 13-20
Enterprise Manager Summary Advisor, 13-20
fact table, 13-18
index, 13-17
initialization parameters, 13-17
init.ora
data warehouse functions, 13-23
init.ora values, 13-21
materialized view, 13-18
Oracle8i configuration, 13-21
parameters, 13-20
query rewrite, 13-19
resource usage, 13-22
periodic updates
data mart refresh, 13-3
physical instance
configure, 8-2
physical name, 3-11
physical storage growth, 13-13
pivoting reports, 12-37
populate data mart
initial load, 10-2
position of columns, 4-14, 5-6
post-load tasks, 10-12, 13-24
post processing transform, 6-40
post-map triggers, 6-40
Preferred Credentials, 7-7
prefix
dimension, 4-5
preserve dimension history, 14-2
primary key constraint
dimension, 2-11
primary source, 6-34
specify, 6-5
Product_Summaries materialized view, 4-24
production key, 2-16
Products
mapping, 6-14
refresh, 13-5
Project
Builder environment, 3-5
property sheet
column position, 4-14, 5-6
database source table, 5-6
detail mapping, 6-43
dimension, 4-10
table, 4-10, 4-14
fact
object, 4-22
table, 4-22, 8-11
flat file format, 5-18
mapping, 6-13
materialized view, 4-28
publish
ad hoc query, 12-1
Oracle Discoverer 3.1, A-15
Oracle Reports, A-15
standard reports, 11-1

Q

query
governor, 12-6
prediction, 12-6
rewrite, 13-19
servers, specify maximum, 13-22

servers, specify minimum, 13-22

R

recipe
ingredients, 1-3
summary, 1-2
The Instructions, 1-14
refresh
basic methods, 13-4
data mart, 13-3
dimension
Type 2, 14-2
job scheduling, 13-12
Products dimension, 13-5
registration
Oracle Enterprise Manager, 8-4
rename
folders, 12-20
ID column name, 4-7
REP_GCCREP tablespace, A-11, A-12
report
space utilization, 13-14
Report Wizard, 11-20
reports, 1-13
adding subtotals to, 12-43
best and worst sellers, 2-3
changing, 11-20
creating, 12-34, 12-40
crosstab format, 12-35
creating, 12-44
customizing, 11-18, 12-49
formatting, 11-18, 12-20, 12-42, 12-49
changing width, 12-42
graphical format, 12-37, 12-47
headings
formatting, 12-49
hide items, 12-23
master-detail crosstab format, 12-36
modify formats, 12-37
page-detail crosstab
create, 12-44
page-detail crosstab format, 12-36
page-detail table format, 12-35
pivoting, 12-37
previewing, 11-18
renaming, 12-49
table format, 12-34
templates, 11-18
types of, 12-34
repositories
case study, A-8
create, A-11
rules
validation rules, 8-16
Run Time Audit Viewer
navigation tree, 10-8

S

Sales
fact table, 4-18
mapping, 6-28
scheduling
jobs, 10-2, 13-12
plans, 13-12
scripts
database link, 9-4
dimensions, 9-5
generated, 9-3
loader mapping, 9-7
materialized views, 9-7
PL/SQL library, 9-8
PL/SQL mapping, 9-8
sequence, 9-8
tables, 9-9
search
navigation tree, 3-10
wildcards, 3-10
secondary source
display, 6-10
mapping, 6-9
sequence object
daysWHkey, 4-33
generated script, 9-8
services
OEM Management Server, 7-9
Oracle Agent, 7-9
SHARED_POOL_SIZE parameter, 13-22
short name

defining, 8
sort
alternate, 12-27
SORT_AREA_SIZE parameter, 13-22
SORT_DIRECT_WRITES parameter, 13-22
sorting values, 12-27, 13-22
source module
flat file source, 3-21
overview, 3-12
sources
flat file, 5-9
how to import, 5-3
staging table
mapping, 6-18
star schema
basic dimensional design, 2-6
components, 1-6
design process, 1-6
fact table grain, 2-7
rationale, 1-5
STAR_TRANSFORMATION_ENABLED
parameter, 13-23
subtotals
adding to reports, 12-43
subtract transform, 6-31
Summary Advisor, 13-20
sums
Oracle Reports, 11-9

T

table
clone a definition, 6-19
column position, 4-14, 5-6
generate and deploy, 9-9
generated scripts, 9-9
property sheet, 4-14, 5-6
fact table, 4-22, 8-11
table format reports, 12-34
tablespace, 7-9
create, A-11
default, 7-11
GCC_MAIN, 7-9
GCC_TEMP, 7-9
repository, A-11, A-12
temporary, 7-11, A-12
templates
for reports, 11-18
toolbar icon
commit, 3-5
transform
cleanup, 14-10
load lookup table, 14-6
loadlookuptable, 14-6
post processing cleanup, 6-40
subtract, 6-31
truncate_table, 6-32, 6-40
transformation category, 6-44
New Transformation Category Wizard, 6-30
truncate_table transformation, 6-32
typical questions, 2-3

U

update
fact table, 13-11
format definitions flat file, 5-18
mapping, 6-13, 6-43
materialized view, 4-28

V

validation
messages, 8-19
rules, 8-16
view
column position, 4-14
conventional view, 4-30
desktops, B-20
materialized view, 4-24

W

warehouse module
case study, 3-6
connection information, 3-7
create, 3-6, 3-12, 3-21
how to create, 3-6
warning
database block size, 7-11

Web
exporting data to, 11-22, 12-50
view data from, 12-4
Web Wizard, 11-22
wildcard characters, 3-10
Windows NT host, A-2
wizard
Create Like, 6-19
Create Local Variable, 6-16
Flat File Sample, 5-11
Import Metadata, 5-3, 5-9
Load, 12-30
Local Variable, 6-16
New Dimension, 4-4
New Dimension Wizard, 4-3
New Fact, 4-19
New Graph, 12-47
New Mapping, 6-4, 6-14
New Materialized View, 4-25
New Module, 3-6, 3-12
New Parameter, 6-25
new project wizard, 3-5
New Sequence, 4-33
New Transformation Category, 6-30
New Transformation Category Wizard, 6-30
New View, 4-30
Report, 11-20
Summary Advisor, 13-20
Workbook, 12-39, 12-40
workbooks
creating, 12-40