Data Warehouse Administrator's Guide

IBM Informix MetaCube ROLAP Option

for IBM Informix Dynamic Server

Version 4.1
October 2002
Part No. CT1S9NA

Note:
Before using this information and the product it supports, read the information in the appendix entitled “Notices.”

Table of Contents

Introduction

In This Chapter

This manual contains information to help you use MetaCube data warehousing software.

Organization of This Manual

The chapters in this manual describe the process and the software components necessary to design, deploy, and manage a MetaCube data warehouse. Generally, this manual provides an overview of the many procedures a data warehouse administrator performs while creating and maintaining a data warehouse. Detailed descriptions of particular procedures can be found in other MetaCube manuals or in online help systems for each MetaCube product.

This manual includes the following chapters:

- This introduction provides an overview of the manual.
- Chapter 1, "Overview of the MetaCube System," introduces the concept of data warehousing and describes the software components of the MetaCube system generally used for managing a data warehouse. The chapter also describes the typical system architectures that MetaCube supports.
- Chapter 2, "Setting Up a Data Warehouse," outlines the process necessary to design and deploy a MetaCube data warehouse. Where appropriate, this chapter references other sources of detailed information.
- Chapter 3, "Secure Warehouse," introduces MetaCube Secure Warehouse, a tool that gives data warehouse administrators the ability to control user access to DSS Systems and other sensitive user information.

- Chapter 4, "Warehouse Manager," introduces MetaCube Warehouse Manager, a tool that enables data warehouse administrators to specify internal information about the data warehouse. This information enables the MetaCube components to access and graphically present the database for querying.
- Chapter 5, "Creating Metadata With Warehouse Manager," describes dimensions, dimension elements, attributes, fact tables, aggregate tables, and sample tables, and how to create them.
- Chapter 6, "Managing MetaCube Agents," introduces: MetaCube Scheduler, which controls the operation of the MetaCube Agents; MetaCube Agent Administrator, a tool for managing MetaCube Scheduler and the agents it spawns; and MetaCube Agents.
- Chapter 7, "Warehouse Optimizer," introduces MetaCube Warehouse Optimizer, a tool that helps data warehouse administrators design aggregate tables to optimize query performance.
- Appendix A describes the use of parameters in the MetaCube system.
- Appendix B describes the use of the **metacube.ini** file.
- Appendix C describes the use of the **mcrexec** utility.
- Appendix D describes how to configure a database connection.

Types of Users

This manual is written for data warehouse administrators who are responsible for creating and managing a data warehouse, the metadata contained in a data warehouse, and the end users who will access that metadata. Although, as data warehouse administrators, you should understand how all the components of a MetaCube system work, primarily you will use the following tools:

- MetaCube Secure Warehouse
- MetaCube Warehouse Manager
- MetaCube Agent Administrator
- MetaCube Warehouse Optimizer

Documentation

MetaCube product documentation contains two key components:

- Printed documentation
- Online help

Printed Documentation

The printed documentation for MetaCube products is divided into two distinct types:

- Manuals for products with graphical user interfaces
- Manuals for application development tools

Manuals for products with user interfaces contain information about the various features of the product. This information is fairly high-level and conceptual in nature. These manuals are designed to give you information about the purpose and capabilities of the product.

Manuals for application development tools are reference manuals that provide technical information about the tools.

In addition to this book, printed manuals for other MetaCube products include the following:

- *MetaCube Explorer User's Guide*. This manual is written for people who are responsible for analyzing data about their company's business. It describes the features of MetaCube Explorer, which querys a MetaCube data warehouse in multidimensional terms, returning meaningful reports to aid in making timely business decisions.
- *MetaCube for Excel User's Guide*. This manual is written for people who use Microsoft Excel spreadsheets for business analysis. After adding MetaCube for Excel to the Excel software, an Excel user can query a MetaCube data warehouse in multidimensional terms to obtain spreadsheet or PivotTable reports.

- *MetaCube Application Programmer's Manual*. This manual is written for the programmer who writes custom applications that interact with the MetaCube analysis engine. This manual describes the MetaCube OLE Automation programming interface.
- *MetaCube SDK for Snap-Ins Programmer's Manual*. This manual is written for the C++ programmer who will write custom measure calculations for MetaCube Explorer and MetaCube for Excel using the MetaCube SDK for Snap-Ins. The SDK Extension Wizard generates skeletal code that is a framework for adding custom C++ code for customized measure calculations.
- *MetaCube SQL Optimizer User's Guide*. This manual describes how to use the MetaCube SQL Optimizer for connecting third-party query tools or custom query applications to the MetaCube analysis engine to access a MetaCube data warehouse. Queries are optimized to run against aggregate and sample tables, thereby significantly improving query performance against very large data warehouses.
- *MetaCube Installation and Configuration Guide*. This manual describes how to install and configure the MetaCube software components on both the server and on PCs.
- *Introduction to New Features*. This guide describes the new features and enhancements for Version 4.10 of MetaCube. It provides information to existing MetaCube users.

Online Help

Each MetaCube product that employs a graphical user interface includes an extensive online help system that provides step-by-step instructions on the use of the product. The help system for most MetaCube products is accessed from the **Help** menu and consists of the following components:

- **Help Topics**: Displays a complete online help system that contains "how to" topics and procedural information on using the product.
- **Context Sensitive Help**: Allows the user to access help specifically on a feature of the user interface. Selecting this menu option changes the cursor into a help cursor. Clicking a feature of the window with the help cursor displays help about that specific feature.

- **Help on Help**: Contains instructions on using the MetaCube online help. For the user unfamiliar with using online help, this help system explains how to use the various features of the MetaCube online help system.

Tip: *For MetaCube for Excel, the help system is accessed from the MetaCube menu in Excel. For MetaCube Web Explorer, the help system is accessed from the* ***Help*** *button on the toolbar.*

Context-sensitive help can be accessed by clicking the **Context Sensitive Help** button on the toolbar in the main window. After clicking the button, clicking any feature of the window allows you to obtain help about that specific feature. This is the same as selecting **Context Sensitive Help** from the **Help** menu.

In a dialog box, pressing **F1** displays a help topic that describes the features of that dialog box. From that topic, the user may jump to the main help system or, in many cases, to a procedural help topic that describes how to use the dialog box. In the main window of an application, after activating an area of the window by clicking it, **F1** provides a help topic that describes that particular area of the window.

Readme Files

In addition to the printed manuals, readme files are distributed with MetaCube products. These files contain technical information, including last-minute changes to product capability or documentation. Please read these files, as they contain important information.

Compliance with Industry Standards

The American National Standards Institute (ANSI) has established a set of industry standards for SQL. Informix SQL-based products are fully compliant with SQL-92 Entry Level (published as ANSI X3.135-1992), which is identical to ISO 9075:1992, on INFORMIX-OnLine Dynamic Server. In addition, many features of OnLine comply with the SQL-92 Intermediate and Full Level and X/Open C CAE (common applications environment) standards.

Informix SQL-based products are compliant with ANSI SQL-92 Entry Level (published as ANSI X3.135-1992) on INFORMIX-SE with the following exceptions:

- Effective checking of constraints
- Serializable transactions

Informix Welcomes Your Comments

Please tell us what you like or dislike about our manuals. To help us with future versions of our manuals, we want to know about any corrections or clarifications that you would find useful. Please include the following information:

- The name and version of the manual that you are using
- Any comments that you have about the manual
- Your name, address, and phone number

Write to us at the following address:

Informix Software, Inc.
Technical Publications
300 Lakeside Drive, Suite 2700
Oakland, CA 94612

If you prefer to send electronic mail, our address is:

doc@informix.com

We appreciate your feedback.

Chapter

1

Overview of the MetaCube System

In This Chapter

This chapter introduces the MetaCube data warehouse, including its system architecture and software components.

What Is a MetaCube Data Warehouse?

The MetaCube software suite is Decision Support Software (DSS) that allows business analysts to query databases and generate reports and graphs with ease and efficiency unavailable with traditional data models. Traditional transactional databases, designed to house a massive number of discrete records, typically require complex joins in order to minimize the overall number of records *stored* in the database. To provide decision support, a database must be built to a simple, consolidated schema that, when queried, minimizes the number of records *processed*. This data model, which supports decision making business processes, is referred to as a *data warehouse*.

Data warehousing is the process of integrating data from a variety of sources to allow a single point of entry. A data warehouse may consist of a large, enterprise-wide database to which users and administrators can connect, or it may incorporate several smaller systems, often called *data marts*. Typically, each data mart addresses a specific subject area within a larger data warehouse.

MetaCube provides a sophisticated mechanism for querying data warehouses and data marts by integrating online analytical processing tools with relational database technology—Relational On Line Analytical Processing (ROLAP). MetaCube offers high-performance Structured Query Language (SQL) algorithms, mathematical summarization, cross-tabulation, pivoting functions, and decision-support query extensions. Moreover, MetaCube simplifies data warehouse searches by presenting data in everyday business language.

MetaCube Software Components

The MetaCube software components consist of the MetaCube analysis engine, the tools for querying a data warehouse, and the tools for managing a data warehouse.

MetaCube Analysis Engine

The MetaCube analysis engine provides a multidimensional, object-oriented programming interface to a relational database. It processes data requests submitted by query tools, such as MetaCube Explorer, MetaCube Web Explorer, or MetaCube for Excel, and generates SQL, which it sends to the database via an Open Database Connectivity (ODBC) driver. The MetaCube analysis engine extends the analytical functionality of the database by performing pivots and calculations on data sets before returning data to the query tool.

MetaCube applications connect to the database through the MetaCube analysis engine. Launching such a MetaCube application automatically launches the MetaCube analysis engine.

The application programming interface (API) for the MetaCube analysis engine is documented in the *MetaCube Application Programmer's Manual*.

Tools for Querying the Data Warehouse

A MetaCube system provides the following graphical, ad hoc data access tools that allow both novice and experienced decision support analysts to access and analyze a data mart or data warehouse:

- MetaCube Explorer—a graphical interface that allows users to submit queries to the database through the MetaCube analysis engine.
- MetaCube Web Explorer—the same interface as MetaCube Explorer except that MetaCube Web Explorer runs in a Web browser.
- MetaCube for Excel—an add-in to Microsoft Excel. This product incorporates multidimensional access into standard Excel features, returning data directly into an Excel spreadsheet or PivotTable.

MetaCube software also provides MetaCube SQL Optimizer, a library of routines that allows the optimization of SQL generated by any query tool or custom query application so that queries can run against a MetaCube data warehouse.

Tools for Managing the Data Warehouse

A MetaCube system provides the following graphical tools for managing a data warehouse:

- MetaCube Secure Warehouse
- MetaCube Warehouse Manager
- MetaCube Agent Administrator
- MetaCube Warehouse Optimizer

MetaCube Secure Warehouse

MetaCube Secure Warehouse provides you with a graphical user interface for controlling end user access to data. With Secure Warehouse, you can specify which users have access to a DSS System. (A DSS System is a view of a data warehouse, often corresponding to the requirements of a particular business unit, such as a finance or marketing department.) The data warehouse administrator can also control what information each user can see in a DSS System and when that user can access information by performing queries.

Users of any MetaCube product can only access a DSS System when authorized to do so in MetaCube Secure Warehouse. Web Explorer users must be properly defined in Secure Warehouse before they can even connect to the database.

Secure Warehouse is documented in this guide and in the product's online help.

MetaCube Warehouse Manager

MetaCube Warehouse Manager provides you with a graphical interface for creating a multidimensional description of tables and columns in the data warehouse. The MetaCube analysis engine stores that multidimensional description as a set of MetaCube system tables in the database. The information stored in these tables is known as *metadata*. Metadata can be used to configure applications that issue queries to the database through the MetaCube analysis engine, thus shielding users from the complexity of the physical model. Warehouse Manager can also describe summary tables, known as *aggregate tables*, and statistically representative sample tables. MetaCube can access these tables instead of the larger tables from which they were derived, thereby improving query performance.

Warehouse Manager is documented in this guide and in the product's online help.

MetaCube Agent Administrator

MetaCube Agent Administrator provides data warehouse administrators with a graphical interface for administering all the features and functions of MetaCube Scheduler and the various Agents.

MetaCube Scheduler is a server-side scheduling mechanism that controls the operation of the various MetaCube Agent processes. Those agent processes can execute SQL statements, background processing (QueryBack), full and incremental aggregation functions, sample tables, Alert jobs, Web Publisher jobs, and even operating system commands. Scheduler manages jobs or job sets that are executed by the MetaCube Agents. It evaluates each job's priority, the privileges of the user who submitted the job, and any dependencies or conditions that must be satisfied before the job can run. It then initiates the appropriate process to execute the job on the server.

MetaCube Scheduler is documented in this guide.

With Agent Administrator, you can manage the job queue, submit new jobs, store job sets, and perform other administrative duties. Agent Administrator also allows you to create aggregate and sample tables, which can improve query performance.

Agent Administrator is documented in this guide and in the product's online help.

MetaCube Warehouse Optimizer

MetaCube Warehouse Optimizer analyzes a data model (the logical representation of a data warehouse) and reviews the queries issued against it to recommend a strategy for creating aggregate, or summarization, tables. To improve performance for resource-intensive queries, aggregate tables group and sum detailed transactional records. Normally, queries requesting such summary-level data require the database to perform the sum while processing the query, but MetaCube automatically routes those queries to an aggregate table if one exists, retrieving the answer without further processing. If desired, MetaCube Warehouse Optimizer can generate the metadata necessary to create aggregate tables. You then submit a job with Agent Administrator to create the physical aggregate tables.

Warehouse Optimizer is documented in this guide and in the product's online help.

MetaCube Architecture

MetaCube system architecture consists of software components configured in a two-tier or three-tier design or a combination of both structures.

Three-Tier Architecture

A three-tier architecture typically supports all users running MetaCube Web Explorer through a Web browser. Figure 1-1 shows a typical three-tier architecture.

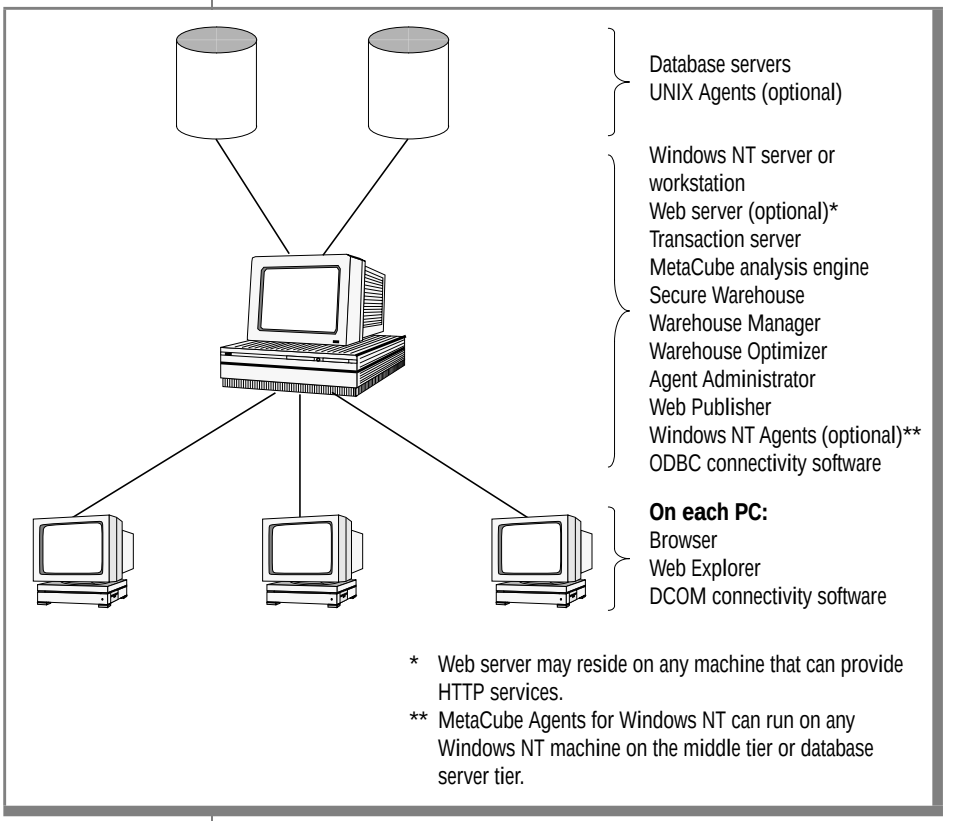


Figure 1-1 *Three-Tier Architecture*

In this configuration, the middle tier is a personal computer or server machine running the Microsoft Windows NT operating system. The middle-tier machine must run Microsoft Transaction Server, a system for managing shared OLE Automation Servers, such as the MetaCube analysis engine. Optionally, the middle tier can also serve as a Web server, although the Web pages for MetaCube Web Explorer can be served from any Web server on the local area network. The MetaCube analysis engine, which processes all queries from Web Explorer users, is installed on and runs on the middle tier.

All of the MetaCube components used to create and manage a data warehouse are installed on and run on the middle tier. These components include Secure Warehouse, Warehouse Manager, Agent Administrator, Warehouse Optimizer, Web Publisher, and the MetaCube Agents. Optionally, MetaCube Agents can be installed on the database server tier.

Typically, the client tier consists of personal computers equipped with Web browsers from which users can run MetaCube Web Explorer. In rare situations, the client tier may consist of personal computers running MetaCube Explorer served by the MetaCube analysis engine running on the middle tier.

Database servers constitute the server tier of this architecture. While Figure 1-1 shows two database servers, only one is necessary.

Two-Tier Architecture

MetaCube also supports a two-tier architecture, as shown in Figure 1-2.

In this configuration, the database server runs MetaCube Agents for either UNIX or Windows NT. While Figure 1-2 shows two database servers, only one is necessary.

The clients are personal computers running the MetaCube analysis engine and either MetaCube Explorer or MetaCube for Excel. In this type of configuration, at least one PC must support the MetaCube administrative tools, including MetaCube Secure Warehouse, which authorizes user access.

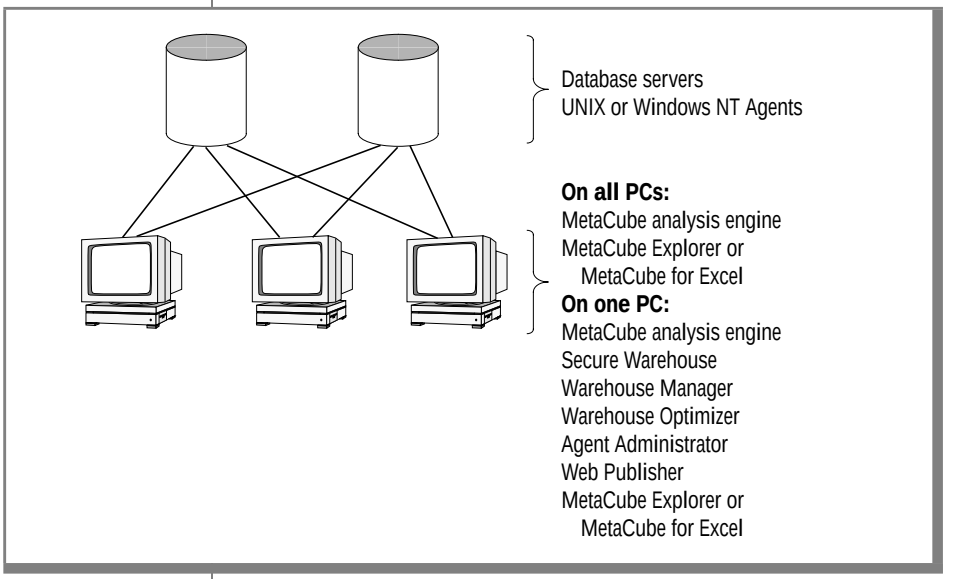


Figure 1-2 *Two-Tier Architecture*

Chapter

2

Setting Up a Data Warehouse

In This Chapter

This chapter describes how to set up a MetaCube data warehouse. The first section provides an overview of the process. Subsequent sections explain each of the steps.

Building a data warehouse requires you to use all of the components of the MetaCube system. Throughout this chapter, sources for more detailed information are referenced. Those sources may be other chapters of this book, online help systems, or other MetaCube manuals.

Overview of the Process

This section provides a high level description of the process necessary to build a data warehouse. Each of the tasks described in this section are discussed in greater detail in the subsequent sections of this chapter.

1. Design a schema.
2. Populate the data warehouse tables.
3. Install the MetaCube software.
4. Generate MetaCube metadata tables with the scripts provided with the MetaCube software.
5. Use MetaCube Secure Warehouse to create *secure users*—users with the necessary privileges to run the MetaCube data warehouse administrative tools: MetaCube Secure Warehouse, MetaCube Warehouse Manager, and MetaCube Warehouse Optimizer.
6. Create metadata with MetaCube Warehouse Manager.
7. Define public filters and queries, and design a folder structure with MetaCube Explorer.
8. Authorize user access with MetaCube Secure Warehouse.

9. Start MetaCube Scheduler.
10. Create metadata for aggregate tables with MetaCube Warehouse Optimizer, then use Agent Administrator to create the physical tables.
11. Create metadata for sample tables with MetaCube Warehouse Manager, then use Agent Administrator to create the physical tables.

Design a Schema

Before you begin designing your data warehouse, you must carefully study your data and your users to discover how your data is used and what the best way is to store it. After studying your data and interviewing users, the first step in creating a data warehouse is to design a schema, or *data model*. To provide decision support, the database tables used to support a data warehouse must be built to minimize the overall number of records processed. This is done using a *dimensional model* rather than an *entity-relationship model*.

Tip: *All of the schema designs discussed in this chapter are supported by MetaCube.*

Dimensional Model Versus Entity-Relationship Model

Traditional data models describe entities and relationships in what is called an entity-relationship (ER) model. An entity represents a type of physical object (for example, a product or a customer) or transaction (for example, a sale or a line item from an order). ER models store each entity in a different table, connecting each of the tables by a series of joins.

Dimensional models consolidate this information into *dimension tables*, so that characteristics such as product color, brand name, brand manager's name, and so forth are stored in a single table. The dimension table thus organizes data around a natural business concept such as a product, which includes products, product lines, and brands. In dimensional modeling, each of these organization levels within a dimension table is referred to as a *dimension element*.

A different type of table in dimensional models, the *fact table*, stores all of the atomic-level transaction information. All of the numerical data (called *measures* or *facts*), which quantify a single transaction, are stored centrally in this fact table, together with a set of keys joining the fact table to its related dimension tables.

ER models, although more complex, store data more efficiently than dimensional models. In a **Product** dimension, for example, brand information, such as **Brand Manager** or **Brand Name**, is repeated for each product, whereas the **Product** table and brand table in an ER model are joined by a single column containing a foreign key, such as **Brand Code**.

Traditional data models are designed to provide efficient data access for large numbers of transactions involving very few records. The ER is designed for On Line Transaction Processing (OLTP). Because data processing in OLTP systems is highly structured, complex data models work well in this environment. Transactions generally involve only one or two tables, often updating only a single record. This means that complex table relationships do not significantly hamper performance.

In decision support systems, there tend to be relatively few concurrent transactions, each accessing a very large number of records. Decision support systems are designed for On Line Analytical Processing (OLAP). In contrast to OLTP systems, DSS processing can involve accessing hundreds of thousands of rows for each query. Hence, complex joins can seriously compromise performance.

Poor performance aside, typical ER models are often inappropriate for decision support because their complexity makes them difficult to navigate. In OLTP systems, such complex models work well because usage and access paths are very well known, and the data structures used for various individual transactions can be coded directly into the applications. By contrast, DSS usage is very unstructured, as users often decide what data to analyze only moments before submitting a query; therefore, access paths change often. Data warehouses for decision support therefore rely on simpler data models that can offer faster access to large amounts of data.

The fundamental differences in the two models reflect the different purposes for which they are designed: ER provides excellent operational functionality, whereas dimensional modeling is well-suited for decision support. The table below lists some of the different requirements met by each data model, suggesting why traditional ER models are not appropriate for decision support.

Design Aspect	Entity-Relationship Model	Dimensional Model
On Line Processing	OLTP	OLAP
Data content	Current values	Archival data, summarized data, calculated data
Nature of data	Dynamic	Static until refreshed
Data structure	Complex, suitable for operational computation	Simple, suitable for business analysis
Frequency of data access	High	Moderate to low
Data update	Updated on a field-by-field basis	No direct update
Type of data access	Highly structured repetitive processing	Unstructured analytical processing
Response time	Subsecond or 2 to 3 seconds	Seconds to minutes

Dimensional Modeling

Dimensional models structure data around natural business concepts and enable sophisticated OLAP. Basic dimensional models have a star schema with the following elements:

- Fact tables and dimensions
- Dimension elements
- Dimension attributes

Star Schema

The basic architecture of the dimensional model is described as the *star schema*. The star schema is designed with a fact table at the center of the star and surrounding dimension tables.

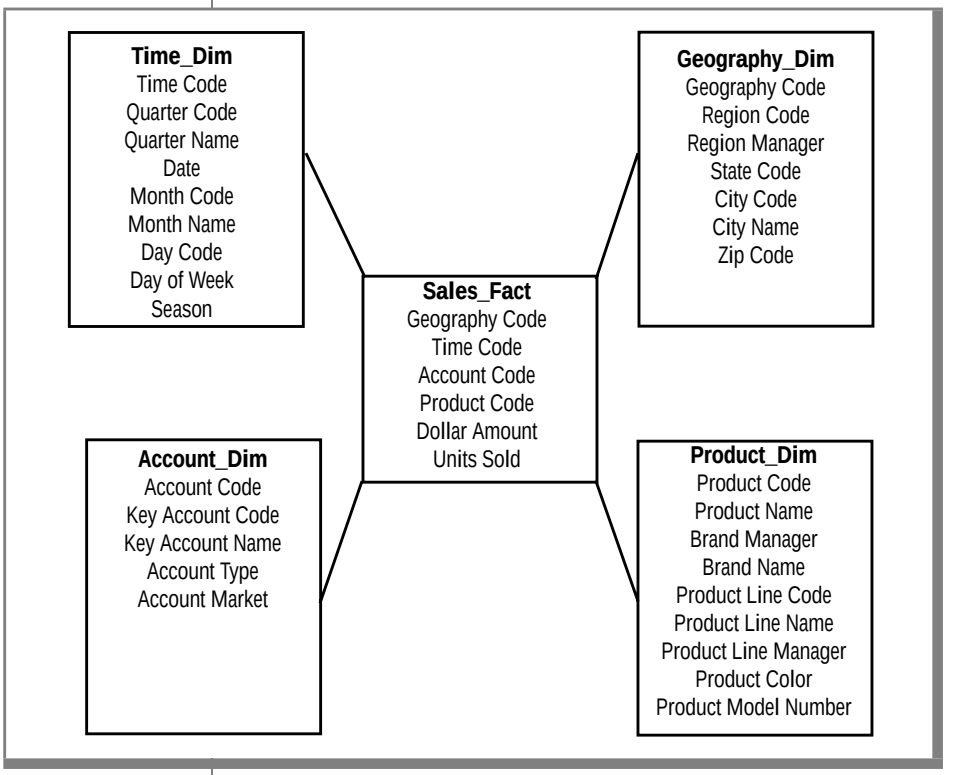


Figure 2-1
Star Schema

Fact Tables and Dimensions

In dimensional modeling, data structures are organized to describe facts (or measures). Facts, stored in a single central fact table, contain numeric values that quantify business transactions. Dimensions are the natural business terms used to describe these transactions. They are stored in dimension tables, that join to the central fact table. Examples of information that might be stored in fact tables include sales, inventory, expenditures, or gross revenue and profit data. Typically, dimension tables contain categories of descriptive elements, such as time, geography, account, and product.

In the simplified fact table below, **Item #** and **Quantity** are stored as facts. Every sales record also contains key columns joining it to the dimension tables. The fact table thus stores measures as well as foreign key columns. In this case, the joining columns are **Time Code** and **Product Code**.

Product Code	Time Code	Item #	Quantity
101	2045	100	1
102	2045	225	2
103	2046	200	20
104	2046	20	1

A fact table and its associated dimensions make up a MetaCube data source. There can be one or more data sources in a DSS System.

Dimension Elements

The days, weeks, months, and so on that make up the **Time** dimension are said to be elements of the time dimension. Each is a logical object called a *dimension element*. A dimension element represents a particular level in the dimension hierarchy, so that, for each hierarchy level, there is a corresponding dimension element. In the database table, a dimension element is a column storing codes that uniquely identify the various values for that element. For example, the **Month** level in the **Time** dimension hierarchy could be a column in the time dimension table that stores a different code for each month.

Dimension Hierarchies

Dimensions are typically organized hierarchically, with each level of the hierarchy consolidating information below it and rolling up to the next-highest level. For example, within a **Product** dimension, **Product** could roll up to **Product Line**, which could roll up to **Brand**.

Not all hierarchies roll up uniformly. Dimension elements can roll up along several paths, resulting in complex branching hierarchies. For example, the **Time** dimension shown in Figure 2-2 features days rolling up to weeks and months separately. Since months do not divide evenly into weeks, weeks cannot roll up to months. Similarly, months can roll up to quarters; however, weeks cannot. This causes the time dimension to have more than one consolidation path for summarizing data.

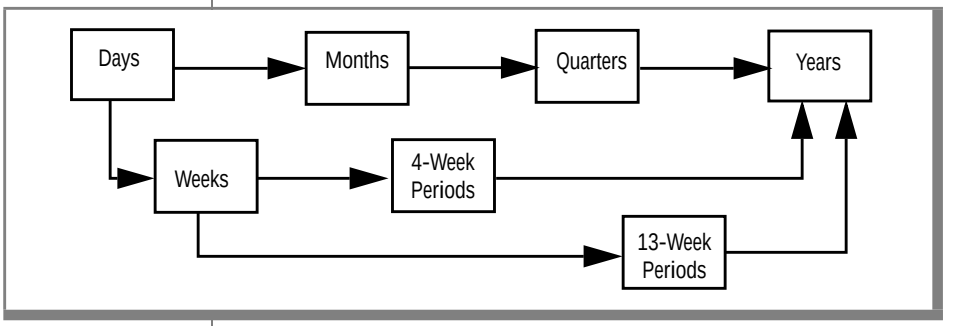


Figure 2-2 *Time Hierarchy with Multiple Consolidation Paths*

The hierarchical relationships among dimension elements provide the framework for *drill-down* and *drill-up* functionality for retrieving data. Drilling down retrieves a more detailed view of a result set. For example, in a report on sales by region, a user can drill down to see how sales in that region break down by state. Drilling down provides a greater level of detail. Conversely, drilling up is the opposite, producing a more summarized view of data.

Dimension Attributes

Dimension attributes describe a given dimension element. For example, **Brand Manager** is an attribute, along with **Brand Name**, that describes the dimension element brand. Color, size, and weight are all attributes that describe the dimension element **Product**. Attributes of the dimension element **Month** are the names of the months—January, February, and so on.

Attributes describe a particular dimension element. MetaCube Explorer users incorporate attributes into their queries to categorize facts along the consolidation paths of the dimension element hierarchy.

Tip: *For a more detailed description of fact tables, dimensions, dimension elements, dimension hierarchies, and dimension attributes, refer to Chapter 5, "Creating Metadata With Warehouse Manager."*

Advantages of Dimensional Modeling

The simplicity of the star schema provides some important advantages:

- It allows a very simple data model to define a complex, multidimensional data structure. The star schema allows you to easily define hierarchical relationships within dimensions and eliminates the need to define complex joins across multiple tables.
- It reduces the number of joins required to process a query, greatly improving performance.
- By presenting a simplified view of data to users, they find it easier to formulate queries to accurately return the information needed.
- It allows a data warehouse to expand and evolve with relatively low maintenance. The simple dimensional design of the star schema provides a flexible foundation for data warehouse growth.

Denormalization of Dimensions

Denormalization is a database design approach in which tables are consolidated and data is repetitively stored for the sake of design simplicity and performance. This design is a defining characteristic of multi-dimensional data models. Thus, in the star schema, attribute values are stored redundantly in the dimension tables.

Compare the two tables below.

The normalized tables shown below contain product data. The two tables join via the **Brand Code** column, thus minimizing the number of records stored.

Prod. Code	Prod. Name	Prod. Color	Brand Code
101	Widget	Blue	XYZ
102	Gadget	Blue	XYZ
103	Snicket	Orange	ABC
104	Graplit	Orange	ABC

Brand Code	Brand Mgr.
XYZ	J. Smith
ABC	T. Jones

The table shown below represents the same product data, denormalized. brand manager information is stored in the same table with the corresponding brand code. The join is eliminated, simplifying the schema.

Prod. Code	Prod. Name	Prod. Color	Brand Code	Brand Mgr.
101	Widget	Blue	XYZ	J. Smith
102	Gadget	Blue	XYZ	J. Smith
103	Snicket	Orange	ABC	T. Jones
104	Graplit	Orange	ABC	T. Jones

Use of denormalized dimension tables has implications for performance. For example, consider a company with one hundred separate products, rolling up to a total of five brands. For every product listed in the dimension table, its corresponding brand is also listed, as are all attributes of the brand. In a denormalized product dimension table, brand manager information, for example, is stored in virtually every record as are all other brand attributes. In a normalized data model, brand attributes would be stored once in the brand table, and only the foreign key would be stored multiple times.

Aggregation Levels

Because, in a data warehouse, dimension tables are denormalized, many of the values in columns that represent either a dimension element or an attribute are non-unique. Consider, for example, a query, based on the tables shown below, requesting sales by brand manager. After retrieving sales data from a **Sales By Brand** aggregate table, the query must retrieve Brand manager attribute values from the denormalized dimension table. The aggregate joins to the dimension table using the **Brand Code** column.

Prod. Code	Prod. Name	Prod. Color	P. Line Code	P. Line Name	Brand Code	Brand Mgr.
101	Widget	Blue	805	Widget	XYZ	J. Smith
102	Widget	Red	805	Widget	XYZ	J. Smith
103	Gadget	Blue	890	Gadget	XYZ	J. Smith
104	Snicket	Orange	770	Misc. Prod	ABC	T. Jones
105	Plunket	Orange	770	Misc. Prod	ABC	T. Jones

Dimension Table

Brand Code	Sales	Qty.
XYZ	$1500	2
ABC	$1720	5

Aggregate Table

There are three dimension elements in the table above: **Product**, **Product Line**, and **Brand**. **Product** is the lowest (or base) element in the product hierarchy. The code that identifies the products is contained in the **Product Code** column and every value in that column is unique; this column is used to join to the fact table. However, for each row of product data, the **Product Line Code** column contains duplicate values since several products belong to the same product line. Similarly, the **Brand Code** column also stores non-unique values for each product and product line. A query that summarizes by brand must return only unique values for as many products as there are within that brand; otherwise, summarization calculations will be incorrect.

When retrieving unique values, MetaCube avoids the use of the SQL DISTINCT keyword and SQL syntax that requires sub-selects, because of performance considerations. Instead, MetaCube requires that all dimension tables contain an additional column containing row values that flag the unique values for each dimension level. Each unique flag corresponds to an *aggregation level*—a single dimension element in the hierarchy. MetaCube uses the aggregation level flag to quickly identify unique values for each dimension element in the hierarchy, thereby assuring good performance for queries returning summarized data.

Prod. Code	Prod. Name	Prod. Color	P. Line Code	P. Line Name	Brand Code	Brand Mgr.	Agg. Level
101	Widget	Blue	805	Widget	XYZ	Smith	1
102	Widget	Red	805	Widget	XYZ	Smith	1
103	Gadget	Blue	890	Gadget	XYZ	Smith	1
104	Snicket	Orange	770	Misc. Prod	ABC	Jones	1
105	Plunket	Orange	770	Misc. Prod	ABC	Jones	1
NULL	*NULL*	*NULL*	805	Widget	XYZ	Smith	2
NULL	*NULL*	*NULL*	890	Gadget	XYZ	Smith	2
NULL	*NULL*	*NULL*	770	Misc. Prod	ABC	Jones	2
NULL	*NULL*	*NULL*	*NULL*	*NULL*	XYZ	Smith	3
NULL	*NULL*	*NULL*	*NULL*	*NULL*	ABC	Jones	3

Dimension Table

Brand Code	Sales	Qty
XYZ	$1,500	2
ABC	$1,720	5

Aggregate Table

The dimension table shown above contains the **Aggregation Level** column and the associated rows of unique data for that aggregation level. The **Aggregation Level** column contains arbitrary numbers that uniquely identify each dimension element in the table:

- The rows at aggregation level 1 contain unique values for the base element, **Product Code**.

- The rows at aggregation level 2 contain unique values for the dimension element, **Product Line Code**. Data in two columns, **Product Line Code** and **Product Line Name**, is unique for that aggregation level.
- The rows at aggregation level 3 contain unique values for the dimension element, **Brand Code**; there are two columns of unique data.

In the dimension table pictured above, aggregate tables can join to the dimension table using the **Product Line Code** column and the **Brand Code** column. This dimension table contains rows listing the attributes for each product line and for each brand only once, allowing a query to join directly to the appropriate level of distinct attribute values.

Each aggregate table corresponds to a particular aggregate level in the dimension table. Consequently, an aggregate table can join to a denormalized dimension table and obtain the unique attribute values. In the example above, the **Brand** dimension element has an aggregation level of 3. A query for sales by brand manager retrieves brand manager information from the bottom two rows, where the **Aggregation Level** column stores the value 3.

Using the **Aggregation Level** column in addition to the standard join allows the query to process fewer rows and eliminates the need for the DISTINCT keyword, thereby improving query performance.

In the example above, rows with an aggregation level of 2 do not include product data, and rows with an aggregation level of 3 do not include product data or product line data. Consequently, a query to retrieve sales by brand manager *by product* must run against the fact table for product information, and subsequently against rows with an aggregation level of 1 in the **Product** dimension table to consolidate product to brand manager. This is because all queries that run against the fact table join dimension tables at the lowest aggregation level, whereas all other aggregation levels are only referenced when a query runs against an aggregate table.

When creating dimension tables in the data warehouse, you will need to include an aggregation level column and assign an aggregation level value to every row in the table. You will also need to include the additional aggregate level rows containing the unique dimension element and attribute data for each aggregation level.

Then, when creating metadata using Warehouse Manager, you will enter the aggregation level value for each dimension element; this value will correspond to the actual value entered into the aggregation level column of the database table.

The aggregation level column can contain any unique identifier. The numeric values used in the example above bear a convenient and intuitive correspondence to the hierarchy level, but this is not required.

For dimension elements with a single consolidation path, aggregation levels can be numbered sequentially, where the base dimension element is assigned an aggregation level value of 1, and the elements succeeding it in the hierarchy are assigned aggregation level values of 2, 3, and so on. This scheme corresponds directly to the hierarchy.

In complex hierarchies with multiple consolidation paths, this logic becomes ambiguous. Although aggregation level and hierarchy level are conceptually similar, the number representing a dimension element aggregation level does not necessarily correspond directly to its position in the hierarchy. MetaCube only requires that you assign each dimension element a unique aggregation level value, an arbitrary value with which it can locate the element's distinct values.

Advanced Modeling Techniques

Under certain circumstances, the simplest implementation of dimensional modeling, the star schema, is not ideal. Denormalized tables require too much disk storage, and their size adversely affects performance, offsetting benefits gained by eliminating joins.

For example, consider a **Product** dimension in which 100,000 **Products** roll up to fifteen **Product Lines** and five **Brands**. In a star schema, the corresponding dimension table would have 100,000 rows, with each row tracking **Brand Manager**, **Product Line Category**, and other dimension elements. The dimension table thus stores every attribute in 100,000 records.

In some cases, the number of attributes stored for each dimension element can be substantial. In the case of a product dimension table with 100,000 rows, every kilobyte of attribute or element data costs 100 megabytes of disk space.

Normalizing the dimension table alleviates this disk storage burden by limiting the number of redundant columns in a dimension table. In a normalized model, the primary dimension table would have 100,000 rows, but might have only three columns—**Product Code**, **Product Line Code**, and **Brand Code**—each of which joins to a separate table containing the attribute values for that element. In this case, the **Product** dimension would consist of the dimension table and three additional tables, one for brand attributes, one for product line attributes, and one for product attributes.

The **Brand** attribute table stores the **Brand Code**, **Brand Manager**, and all other brand attributes. If the **Product** dimension table stores data for 100,000 products, but only 50 brands, the brand attributes are stored in a table of only 50 rows. Similarly the **Product Line** table stores **Product Line Code**, **Product Line Category**, and all other product line attributes. In a dimension table of one million rows, saving just one kilobyte per row through normalization saves a full gigabyte of disk space.

Snowflake Schema

Star schemas with normalized dimension tables are called *snowflake schemas*, named for their added structural complexity.

In a snowflake schema, shown in Figure 2-3, each dimension table stores one key for each level of the dimension hierarchy (that is, for each dimension element). The lowest-level key joins the dimension table to the central fact table; this key (or another) also joins the dimension table to the attribute table that contains the descriptive information for that lowest level dimension element. The rest of the keys join the dimension table to the corresponding attribute tables.

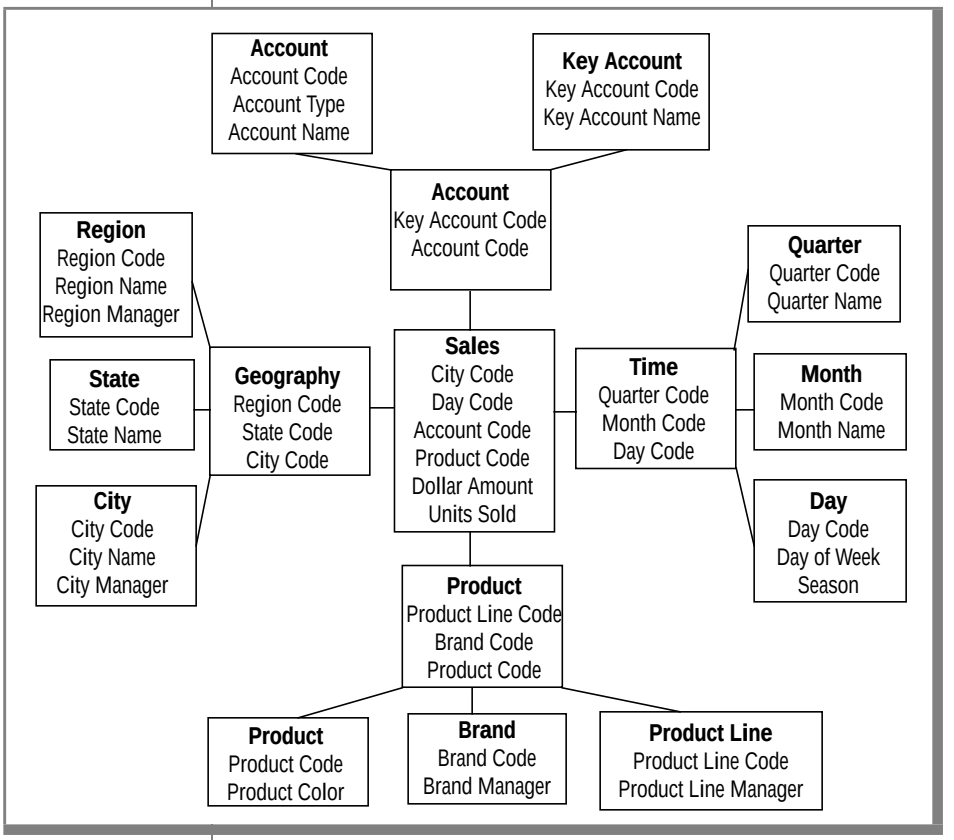


Figure 2-3
Snowflake Schema

Snowflake schemas produce faster query results by reducing the total number of rows processed to retrieve attribute information for queries from aggregate tables; queries using the fact table will actually run slower. The snowflake design also facilitates the Lookup feature in Explorer, where a list of attribute values is displayed for the user. Lookups can be performed faster when attributes are stored in an isolated attribute table and not in the dimension table.

Under normal circumstances, the main disadvantage of the snowflake schema is the relative complexity of the structure. Also, overall maintenance of a data warehouse designed in a snowflake schema is more difficult to manage since the data model is more complex.

You can implement variations of the snowflake schema that are designed for improved performance while still conserving disk space. The biggest single advantage for incorporating attribute tables into your design is to improve query processing for summarized reports when aggregate tables are used. Therefore, for any dimension, it is efficient to store the lowest-level data in the dimension table itself, while higher-level, summarization information can be stored in the attribute tables that join to the dimension table.

Partial Snowflake Schema

Another design intermixes star and snowflake tables within a single schema. That is, some dimension tables contain all codes and attribute values, while others contain only codes that join to separate attribute tables. The MetaCube demonstration database is an example of this design. Three dimensions—**Time**, **Product**, and **Channel**—join to attribute tables, thus forming a snowflake arrangement. The fourth dimension—**Geography**—contains attribute columns within the dimension table, thus forming a star arrangement.

This schema design variation is called a *partial snowflake*. Figure 2-4 illustrates this type of schema design; it includes attribute values at the base levels in the dimension tables themselves and joins to separate attribute tables at the higher levels of the dimension hierarchy.

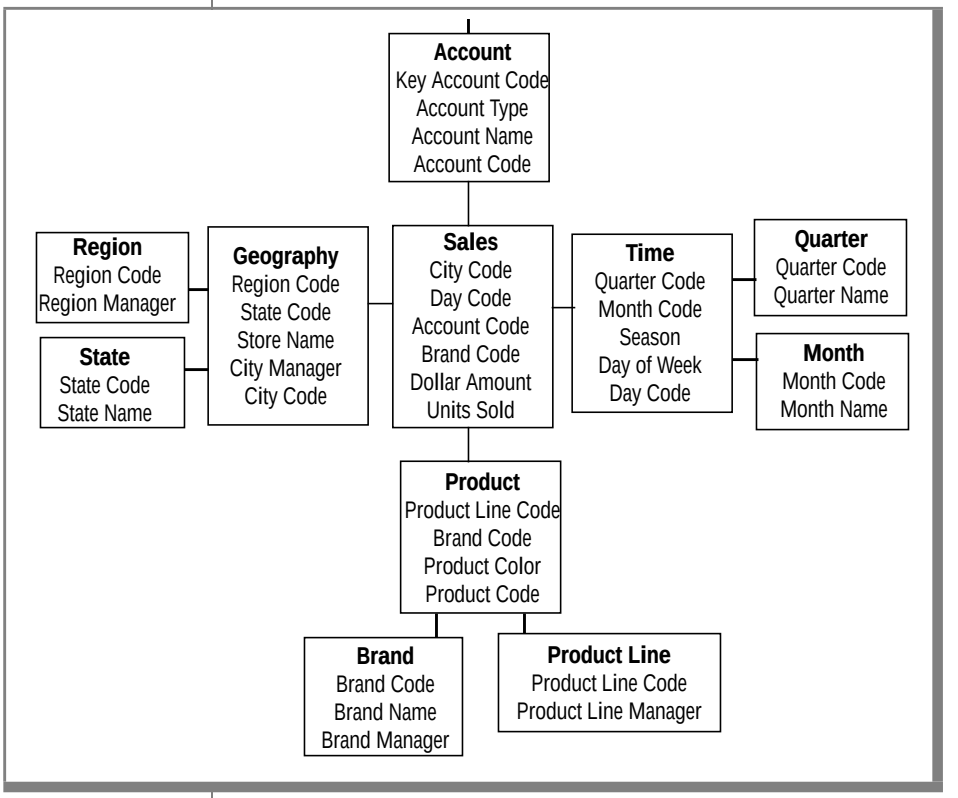


Figure 2-4 *Partial Snowflake Schema*

When to Use Snowflake Tables

The value of snowflake tables is greatest for dimensions in which:

- there are many rows.
- there are many attributes stored at the higher levels of the dimension hierarchy, and disk space is a concern.
- there are aggregate tables at a higher dimension element level and many queries that access data at those levels.

- you have dimension element tables. Refer to "Dimension Element Tables" on page 2-20.

Dimension Element Tables

A MetaCube table containing dimension element and hierarchy rollup information is called a *dimension element table* (DET). Dimension element tables are specifically designed for data warehouses residing in an Informix Dynamic Server with Advanced Decision Support and Extended Parallel Option to take advantage of the General Key (GK) indexing option.

The support of GK indexes requires a MetaCube schema in which tables contain a column of unique data values (primary key column) that allow a foreign column join to the fact table. Aggregate rows cause all columns in the dimension table to contain duplicate data. Therefore, with an aggregation level implementation, a MetaCube data warehouse does not support GK indexing.

To create dimension tables with a primary key column, the aggregate row information must be removed from the dimension table so that the table contains a column of unique values that can serve as the primary key column. With this type of implementation, the hierarchy and rollup information contained in aggregate rows is stored in separate tables.

Information in a single DET is associated with a single aggregate level in a dimension table. When DETs exist in a MetaCube data warehouse, the dimension table no longer contains aggregate rows. Use of DETs also removes the need for an aggregate level column in the dimension table.

Dimension Element Table Examples

The MetaCube demonstration database is used for the following discussion to illustrate the difference in the schema design of a data warehouse that contains DETs and one that does not.

Branching Hierarchy: The Product Dimension

The *Product* dimension hierarchy contains five elements in a branching hierarchy. Figure 2-5 shows this hierarchy as it appears in MetaCube Warehouse Manager.

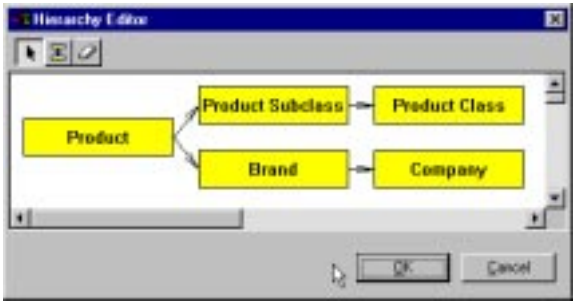


Figure 2-5
Product Dimension Branching Hierarchy

The base element, **Product**, rolls up to **Product Subclass** and **Product Class** along one branch. **Product** also rolls up to **Brand** and **Company** along another branch. Two dimension elements, **Product Class** and **Company**, are the top elements of the two branches.

The **Product** dimension table below shows the aggregate rows that are above the base element (that is, levels 2 through 5). (Although, in reality, the demo database table columns contain only codes, to aid in your understanding, this diagram shows the corresponding values for the codes in the shaded columns.) The blank fields in the aggregate rows contain NULL values.

Subclass Code		Class Code		Brand Code		Company Code		Agg_ Level
1	Speakers	1	Audio					2
2	Tape Decks	1	Audio					2
3	Compact Disc Players	1	Audio					2
4	Graphic Equalizers	1	Audio					2
5	VHS Recorders	2	Video					2
6	Laser Disc Player	2	Video					2
7	Video Rewinder	2	Video					2
8	Television Sets	2	Video					2
9	IBM Compatible PCs	3	Computer					2
10	Mac Compatible PCs	3	Computer					2
11	CDROM Drives	3	Computer					2
12	Memory Chips	3	Computer					2
13	Hardware Boards	3	Computer					2
		1	Audio					3
		2	Video					3
		3	Computer					3
				1	Delmore	1	Electrotron, Inc	4
				2	Techno Components	1	Electrotron, Inc	4
				3	Extreme	2	Soundbyte, Inc	4
				4	Suresound	2	Soundbyte, Inc	4
				5	NVD	2	Soundbyte, Inc	4
				6	Barton	3	Montel Tech	4
				7	Onetron	2	Soundbyte, Inc	4
				8	Lasertech	2	Soundbyte, Inc	4
				9	Alden	3	Montel Tech.	4
						1	Electrotron, Inc	5
						2	Soundbyte, Inc	5
						3	Montel Technology	5

The aggregate rows reflect the branching nature of the **Product** dimension hierarchy. Aggregate level two (2) contains two code columns, one for the unique subclass codes and one for each subclass code's rollup to a class code. Aggregate level four (4) also contains two columns: the unique column of brand codes and their rollups to company codes. The **Class** column, aggregate level 3, and the **Company** column, aggregate level 5, contain only the single unique values for class and company, respectively. These two elements are the top of the hierarchy.

Figure 2-6 shows the **Product** dimension, implemented as a snowflake schema. The **Product** dimension table contains aggregate rows and an aggregate level column.

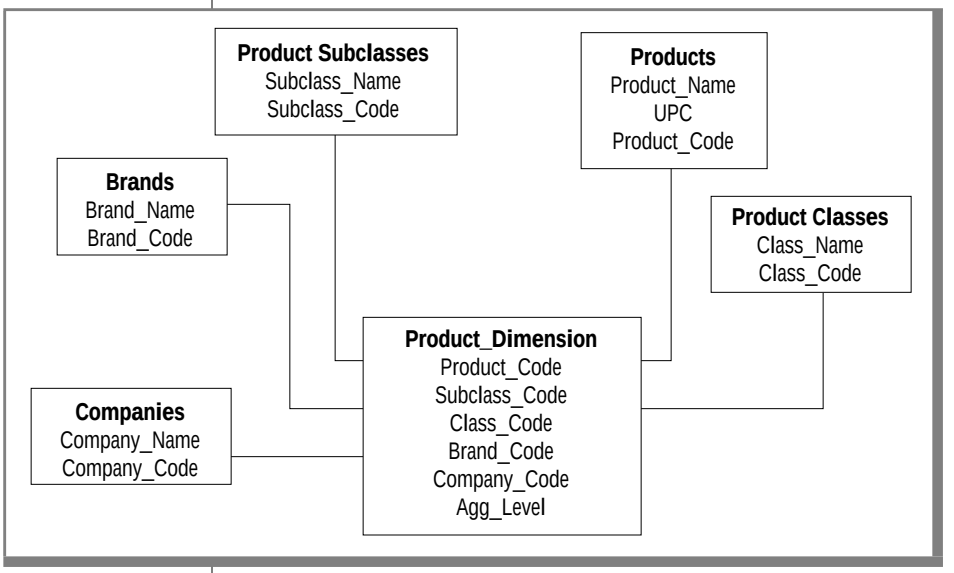


Figure 2-6 *Product Dimension with Aggregate Level Column in Snowflake Schema*

Because of the presence of aggregate rows in the **Product** dimension table, there is no column that contains unique information; therefore, this table cannot have a primary key column.

When using GK indexing in a data warehouse, aggregate rows as well as aggregate level columns are eliminated from dimension tables. Instead, the unique information for each dimension element is placed in a separate DET.

Figure 2-7 shows the **Product** dimension implemented to allow the use of GK indexing. The dimension table and all dimension element tables contain primary key columns and hierarchy (or rollup) information for subclass and class, brand and company. The **Product** dimension table contains no aggregate level column.

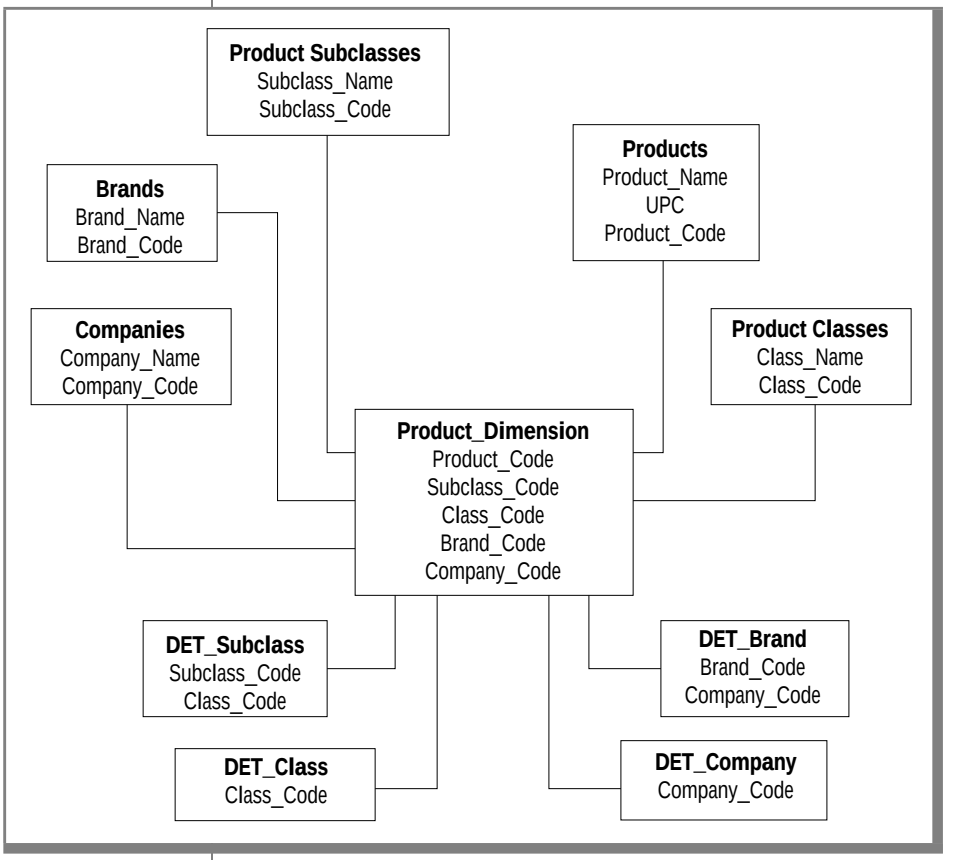


Figure 2-7 *Product Dimension with DETs Schema Design*

Tip: *To easily identify a dimension element table name, you can apply prefix DET_ to the filename; this is not required, however.*

In a DET implementation, each DET contains the information that is contained in the aggregate rows of a dimension table in a non-DET implementation. Therefore, the **DET_Subclass** table stores the information previously contained in the aggregate level 2 rows. The **DET_Class** table stores the information previously contained in the aggregate level 3 rows of the **Product** table. The **DET_Brand** table stores information previously contained in the aggregate level 4 rows. And, the **DET_Company** table stores information previously contained in the aggregate level 5 rows.

Non-Branching Hierarchy: the Geography Dimension

The **Geography** dimension contains a single, non-branching hierarchy.

Figure 2-8 shows the hierarchy as it appears in MetaCube Warehouse Manager. The base element in the **Geography** dimension hierarchy is **Postal Code** and the hierarchy rollups are **City**, **District**, **Region**, and **Country**.

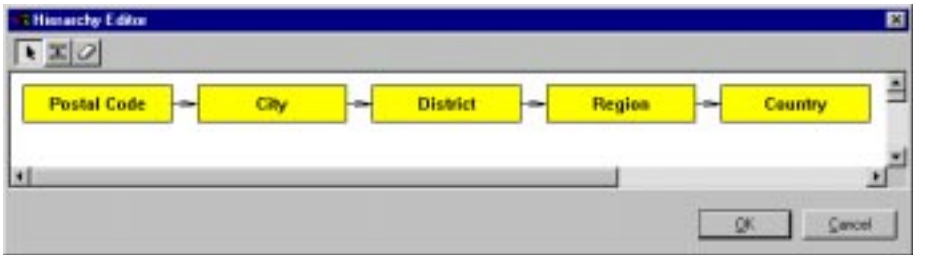


Figure 2-8 *Geography Dimension Hierarchy*

The table below shows the aggregate rows above the base element level in the **Geography** dimension—that is, aggregate levels 2 through 5. The aggregate rows reflect the non-branching nature of the **Geography** dimension hierarchy, with all levels rolling up to the single top level element—**Country**.

City	District	Region	Country	Agg_Level
Oakland	California	West	USA	2
Palo Alto	California	West	USA	2
San Francisco	California	West	USA	2
Boston	New England	Northeast	USA	2
Sudbury	New England	Northeast	USA	2
New York	New York	Northeast	USA	2
	New England	Northeast	USA	3
	New York	Northeast	USA	3
	California	West	USA	3
		Northeast	USA	4
		West	USA	4
			USA	5

Figure 2-9 illustrates the **Geography** dimension implemented with DETs. For each dimension element above the base element, a DET contains hierarchy and rollup information for the dimension elements: **City**, **District**, **Region**, and **Country**. The dimension table contains no aggregate level column.

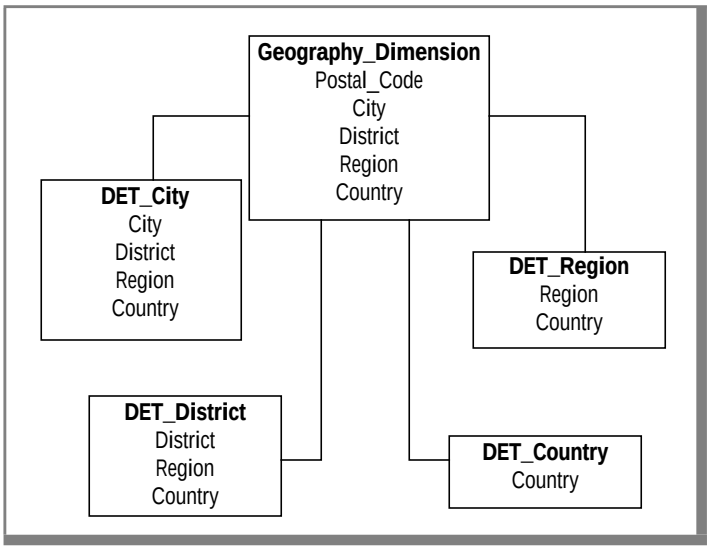


Figure 2-9 *Geography Dimension with DET: Schema Design*

Populate the Data Warehouse Tables

The next step is to create and populate the tables that make up your data warehouse. Central fact tables contain the factual data you want to track and report, such as sales information. Dimension tables contain descriptive data, such as product or geographic information, that allow the MetaCube analysis engine to present multidimensional views of data. The needs of your particular business and data warehouse will dictate the tables that must be created.

Relative Time

MetaCube tracks time on both absolute and relative scales, enabling Explorer users to build queries requesting, for example, sales from August 1997 through October 1997 (an absolute time interval), as well as queries requesting sales for the current month and the same month last year (a relative time interval). MetaCube requires the following additions to the **Time** dimension to support queries and filters involving relative time intervals:

- A current period column containing codes that indicate the current time period and the same time period in the previous year
- Sequential time codes that uniquely identify time elements (day, month, quarter, year, for example)

Current Period Column

To define a particular element in the time dimension as the current period, include a **Current_Period** or similarly named column in the **Time** dimension table. Then, after updating the table with the most recent data, insert a `Y` into the **Current_Period** column at the row representing the most recent time period, thereby designating that period as the current period. (The Y that previously marked the most recent time period must, of course, be deleted.) The designated row must have the lowest possible aggregation level, so that the most granular element in the time dimension table defines the current period.

For example, suppose a time dimension consists of the dimension elements day, month, quarter, and year. The current day is flagged by a Y in the **Current_Period** column at the row corresponding to that day. The Y in the last row of the simplified **Time** dimension table below identifies the current day as the 21st, the current month as July, the current quarter as Q3, and the current year as 1997.

Day	Month	Quarter	Year	Current Period	Agg. Level
20	July	Q3	1996		1
21	July	Q3	1996	P	1
20	July	Q3	1997		1
21	July	Q3	1997	Y	1

Flagging one row in the table tells MetaCube which day, month, quarter, and year comprise the current time period. Similarly, a P in the **Current_Period** column identifies a time interval as the same period as the current period, only in the previous year. If the current period is flagged as July 21, 1997, then July 21, 1996, is the same period last year, and a P should be placed in the column representing that day. You do not have to repeat these flags at higher aggregation levels.

Sequential Time Codes

MetaCube's relative time features also require code columns associating each time element in the table with codes that increment sequentially for each new day, new week, new month, and so on. The numeric code columns set the first value of each time element to an arbitrary value and then increment that number sequentially for each row in the table. For example, if the month code 12 indicates the month of December in 1996, 13 should represent the code for January in 1997.

For each dimension element, choose an arbitrary number, such as 1, to represent the first value in that series of time elements. The following time period code is then incremented by one. The actual number chosen is irrelevant as long as subsequent time element codes increase sequentially by one.

The table below illustrates a simplified example of implementing sequential time codes.

Day Code	Day	Month Code	Month	Quarter Code	Quarter	Year Code	Year	Current Period	Agg. Level
1340	20	546	July	209	Q3	50	1996		1
1341	21	546	July	209	Q3	50	1996	P	1
.	.	.	.	.	.	.	.	.	.
1502	1	552	January	211	Q1	51	1997		1
1503	2	552	January	211	Q1	51	1997		1
.	.		.		.		.	.	.
1705	20	558	July	213	Q3	51	1997		1
1706	21	558	July	213	Q3	51	1997	Y	1

Numeric codes for different dimension elements are independent of one another and can, in fact, overlap. You may find it useful to include a description column for each **Time** dimension record, specifying the actual date associated with the record.

Install the MetaCube Software

The *MetaCube Installation and Configuration Guide* describes how to install and configure MetaCube software.

Generate MetaCube Metadata Tables

The MetaCube installation process provides you with many SQL scripts. One of the functions of these scripts is the creation of metadata tables, which store descriptions of the data model used for the data warehouse. The metadata tables also store definitions and characteristics of jobs submitted to the MetaCube Agents, user properties, and saved queries and filters.

The *MetaCube Installation and Configuration Guide* describes how to create metadata tables.

Create Secure Users

Only secure users can access the data warehouse administrative tools: MetaCube Secure Warehouse and MetaCube Warehouse Manager. When setting up a data warehouse, many tasks require you to use these two applications. Only secure users can use MetaCube Warehouse Optimizer to create metadata descriptions of aggregate tables, although any MetaCube user can access the other capabilities of Warehouse Optimizer.

By default, the user **metapub** is a secure user and can access Secure Warehouse, Warehouse Manager, and Warehouse Optimizer. The password for this user is determined during the installation process, as described in the *MetaCube Installation and Configuration Guide*.

The first time you access Secure Warehouse, you should log in as **metapub**. Then you can define other secure users who will serve as data warehouse administrators.

Refer to Secure Warehouse online help for step-by-step instructions on how to define secure users.

Important: *After you have created additional secure users in Secure Warehouse, Informix strongly recommends you revoke **metapub**'s secure user privilege. Because it has been noted in this documentation that **metapub** has secure user privilege, your data warehouse is not secure until you revoke this privilege. Do not revoke **metapub**'s secure user privilege without first granting secure user privilege to at least one other user. If you revoke **metapub**'s privilege without first granting secure user privilege to another user, no one will be able to access Secure Warehouse or Warehouse Manager.*

Create Metadata

Metadata is a logical, multidimensional description of relational data. Essentially, metadata serves as a map to the data warehouse. Metadata completely describes the data warehouse and contains information about the physical structure of the underlying database—its tables, columns, and joins. Metadata also enables the MetaCube analysis engine to automatically generate the SQL statements needed to access data in the optimal way.

A big benefit of metadata is that all information about the DSS Systems in a data warehouse is located in a single, central set of database tables. When you enlarge your data warehouse or add new dimensions to it, you specify these changes in the metadata, using Warehouse Manager, and do not need to modify any other MetaCube applications.

For the most part, you create metadata using MetaCube Warehouse Manager, although you also generate user-related metadata with MetaCube Secure Warehouse. When you are setting up a data warehouse, you must use Warehouse Manager to create objects representing elements of all the DSS Systems necessary for the data warehouse and all the MetaCube data sources necessary for each DSS System.

You may design and implement one or many DSS Systems in your data warehouse. Each DSS System may contain a different set of measures and dimensions. Reasons for including more than one DSS System in your data warehouse are:

- DSS Systems will be deployed in various departments within your company whose data retrieval requirements are different.
- Various users in your company, need access to different data. This can be related to security considerations, need-to-know issues, or simply relate to the size of the DSS System itself.

A DSS System can contain one or more data sources—the organization of a single fact table and its associated dimension, attribute, aggregate, and sample tables. A data source contains closely related information about a single aspect of your business.

When you initially create the objects for a DSS System, you have the option of creating metadata describing aggregate and sample tables. You may want to wait until users begin querying your DSS System before you create either of these types of tables. Actual usage may influence your decisions concerning sample and aggregate tables.

After you have created metadata for a DSS System, you should verify the integrity of that data, a one-step procedure using Warehouse Manager.

Chapter 5, “Creating Metadata With Warehouse Manager,” describes all the tasks necessary for creating metadata for a DSS System. Online help gives step-by-step instructions on how to use Warehouse Manager.

Define Public Filters and Queries and Design a Folder Structure

All users of MetaCube Explorer, MetaCube Web Explorer, and MetaCube for Excel have access to public queries and filters stored in metadata for a DSS System. As the data warehouse administrator, you should create all necessary public queries and filters. The procedure for creating a public filter or query is the same as the procedure for creating any filter or query in Explorer except that the user performing these actions is **metapub**. The user **metapub** can also designate default filters. For more information about creating queries, filters, and default filters, see the *MetaCube Explorer User's Guide*.

Because public queries are typically accessed by many users, modifications to or deletions of public queries should be done carefully. Be sure to notify all users of changes made to these objects.

You can define public filters and queries any time after metadata tables have been created for a DSS System and the **metapub** user has been granted access to that DSS System using MetaCube Secure Warehouse.

When users save queries and filters, they are stored in folders and subfolders organized in a tree structure. The top-level folder is named **ROOT**. You may want to create subfolders if your users can create and store filters and queries, especially if you anticipate that a DSS System will require a large number of filters. Many MetaCube installations create a folder for each dimension so that a user can easily locate filters associated with a particular dimension, such as **Geography**.

All folders in the data warehouse are visible to all users of Explorer, Web Explorer, and MetaCube for Excel. That is, a single set of folders and subfolders are accessible to all users. For more information about folders, see the *MetaCube Explorer User's Guide*.

Define User Access

Use MetaCube Secure Warehouse to control user access to DSS Systems. With the Secure Warehouse graphical interface, you can specify which users have access to a DSS System, what information each user can obtain from the DSS System, and when that user can access information by executing background queries and other types of jobs (only administrators can execute other types of jobs).

Essentially, you accomplish these tasks by identifying the users who will be managed, assigning DSS System access to those users, and applying mandatory filters to users on a per-DSS-System basis. Mandatory filters limit what information a user can actually see in each DSS System.

You can define user access with Secure Warehouse any time after you have created metadata for a DSS System. Chapter 3, "Secure Warehouse," introduces MetaCube Secure Warehouse, and online help for Secure Warehouse provides all necessary procedural descriptions.

Start MetaCube Scheduler

MetaCube Scheduler is a server-side scheduling mechanism that controls the operation of the various MetaCube Agent processes. Scheduler allows users of MetaCube Agent Administrator to submit jobs to the server and MetaCube Explorer, MetaCube Web Explorer, and MetaCube for Excel users to submit background queries. It evaluates the job priority, the privileges of the user who submitted the job, and any dependencies or conditions that must be satisfied before the job can run. It then generates the appropriate process to execute the job on the server.

You can start MetaCube Scheduler any time after the MetaCube software is installed, but you *must* start Scheduler before you run any jobs using MetaCube Agent Administrator or MetaCube Warehouse Optimizer. The *MetaCube Installation and Configuration Guide* describes how to install Scheduler. In this book, Chapter 6, "Managing MetaCube Agents," describes how to start Scheduler.

Build Aggregate Tables

To improve performance, most data warehouses include aggregate tables, which contain summarized data. The MetaCube analysis engine automatically routes queries for summarized information to an aggregate table if a suitable one exists.

Using MetaCube Warehouse Optimizer, you can automatically determine the best aggregate tables to build, and you can generate the necessary metadata descriptions for those tables. If you prefer, you can use Warehouse Manager to create the metadata descriptions for those aggregate tables yourself. After the aggregate table metadata descriptions exist, you can use MetaCube Agent Administrator to submit a job that builds the aggregate tables.

Any MetaCube user can access Warehouse Optimizer, but to create metadata descriptions of aggregate tables, you must be a secure user.

Although you can generate aggregate tables any time after their metadata descriptions have been created and the MetaCube Scheduler is running, you may want to activate query auditing and wait until users begin querying the data warehouse to determine the most strategic use of aggregate tables.

Chapter 7, “Warehouse Optimizer,” describes MetaCube Warehouse Optimizer and aggregation, and Chapter 6, “Managing MetaCube Agents,” describes MetaCube Agent Administrator. Online help for these products provides all necessary procedural descriptions.

Build Sample Tables

To improve performance in large data warehouses, one or more sample tables can provide statistically accurate results for queries in much less time than it would take to query a large fact table. You may wish to build several sample tables in your data warehouse, each with a different level of accuracy. This offers users flexibility when specifying the degree of confidence they need for a particular report.

Using Warehouse Manager, you can generate the necessary metadata descriptions of sample tables. After the metadata descriptions exist, you can submit a job to generate the physical sample tables using MetaCube Agent Administrator.

Although you can create sample tables any time after you build their metadata descriptions and the MetaCube Scheduler is running, you may want to wait until users begin querying the data warehouse to determine the most strategic use of sample tables.

Chapter 5, "Creating Metadata With Warehouse Manager," describes how to create metadata for sample tables. Chapter 6, "Managing MetaCube Agents," describes the use of MetaCube Agent Administrator, and online help for Agent Administrator describes how to submit a Sample job.

Depending on the product families you have purchased for your database server, you may not need to use the MetaCube Agent, MetaCube Sampler, to create physical sample tables. Some databases, including the Informix Dynamic Server with Advanced Decision Support and Extended Parallel option, provide dynamic sampling that the MetaCube analysis engine can use in place of physical sample tables.

Data Warehouse Maintenance

After you have created a data warehouse, you will need to perform additional procedures to maintain it. Some of the most common tasks are listed below:

- Use Secure Warehouse to add new users and remove users who are no longer authorized.
- Update aggregate tables to reflect new information added to a fact table.
  1. Create an incremental fact table to house information that is added periodically. Use Agent Administrator to submit two jobs: one creates incremental aggregate tables and the other incorporates those tables into the existing fact and aggregate tables.
  2. Check the validity of the incremental aggregate tables before incorporating them into the existing aggregate tables.

- Update sample tables to reflect new information added to a fact table. Use Agent Administrator to submit a job that generates new sample tables.
- Use MetaCube Warehouse Manager to modify metadata.
- Use Agent Administrator to submit other types of jobs, such as SQL Select and SQL (Non-Select) jobs. For example, you could schedule a job that automatically generates monthly reports.
- Use Agent Administrator to control MetaCube Scheduler.

  You can use Agent Administrator to define a job set that automates many of the tasks described above. You can define dependencies between jobs in a job set, run the Job Set job at a specific time or schedule it to run on a periodic basis. For more information on dependencies and job sets, see Chapter 6, "Managing MetaCube Agents," and Agent Administrator online help.

Chapter

3

Secure Warehouse

In This Chapter

This chapter introduces MetaCube Secure Warehouse, a tool that gives data warehouse administrators the ability to control user access to DSS Systems.

Secure Warehouse online help provides step-by-step procedures for all the tasks described in this chapter.

About MetaCube Secure Warehouse

MetaCube Secure Warehouse is a graphical tool for controlling access to DSS Systems. With it, you, as data warehouse administrator, can easily specify which users have access to a DSS System, what information each user can see in the DSS System, and when that user can see it.

Essentially, you accomplish this by identifying the users who will be managed, giving those users access to DSS Systems and, optionally, applying mandatory filters to users on a per-DSS-System basis. The mandatory filters limit what information a user can actually see in each DSS System.

Most of the actions you perform with Secure Warehouse consist of defining information about users. That information is stored in metadata, although some connection information for users is stored in the registry of the computer running the MetaCube analysis engine.

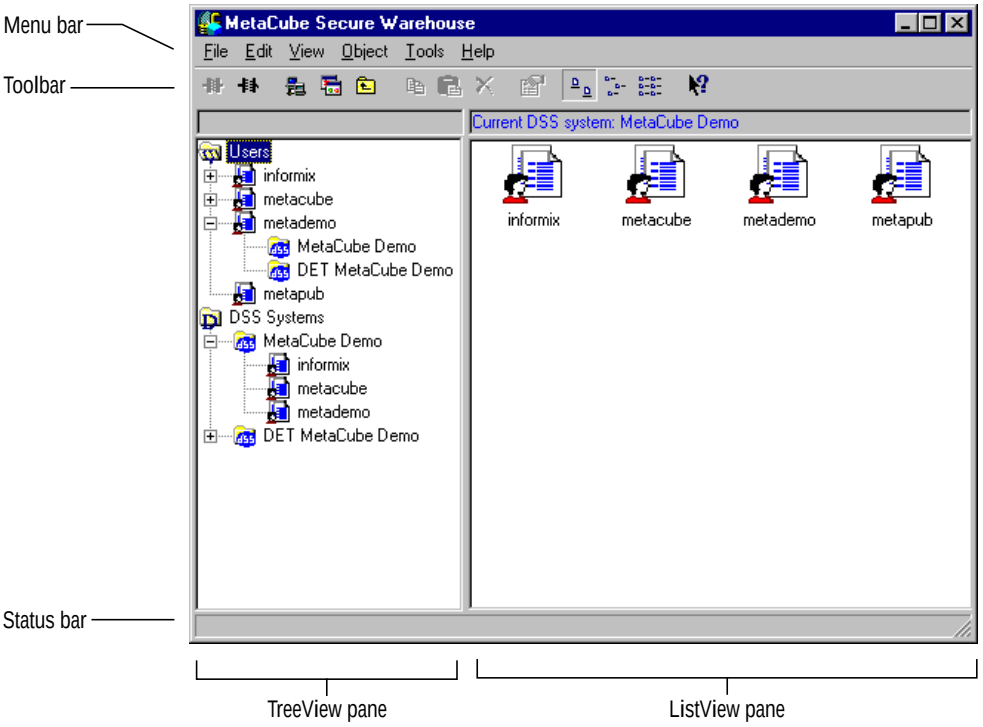
When you use Secure Warehouse, you do not perform a save as you would in many computing applications. Instead, almost all of the changes you make are instantly applied to metadata tables.

The Secure Warehouse User Interface

The Secure Warehouse user interface consists of two panes:

- The TreeView pane shows two hierarchical tree lists: one presents nodes that represent users, and the other presents nodes that represent DSS Systems available for the current database connection. The highest-level node in the user list is called **Users**. The highest level node in the DSS System list is called **DSS Systems**.
- The ListView pane shows the contents of the node currently selected in the TreeView pane.

Figure 3-1
Secure Warehouse User Interface



Using Secure Warehouse

When you use Secure Warehouse, you must establish a connection between Secure Warehouse and the database, via the MetaCube analysis engine. If necessary, you may need to change the current database connection. You may also want to choose a different DSS System.

The first time you log in to Secure Warehouse after installing MetaCube software, log in as **metapub** (when a system is initialized, the user **metapub** is by default the only secure user). Then grant your own user ID secure user privileges.

Important: *Because it has been noted in this documentation that* ***metapub*** *has secure user privilege, your data warehouse is not entirely secure until you revoke this privilege. After you have created additional secure users in Secure Warehouse, Informix strongly recommends you revoke* ***metapub****'s secure user privilege. Do not revoke* ***metapub****'s secure user privilege without first granting secure user privileges to at least one other user. If you revoke* ***metapub****'s privilege without first granting secure user privilege to another user, no one will be able to access Secure Warehouse or Warehouse Manager.*

After establishing the appropriate connections, you are ready to define the following parameters:

- Who (which database users) can access data?
- What data can those users access?
- When can those users access data?
- How do those users access data?

Who Can Access Data?

The first step in using Secure Warehouse is identifying which users you intend to manage. The following users are visible in Secure Warehouse:

- **Informix databases:** any user who has been explicitly granted CONNECT, RESOURCE, or DBA privileges to the database. Users granted access implicitly through the PUBLIC keyword are not visible to Secure Warehouse.
- **Oracle databases:** any user granted access to the database.

For more information on granting connection privileges to users, see your database system administration documentation.

Tip: *Although users are visible in Secure Warehouse, no user will be able to log in and connect to the data warehouse until the database administrator has provided that user with a login ID and password, and the data warehouse administrator has granted the user access privileges in Secure Warehouse.*

You can give a user access to a DSS System by manipulating a user or DSS System node or by defining the properties of the user or DSS System. When you assign a DSS System to a user, both trees in the user interface reflect the change. In the **Users** tree, a node representing the DSS System is displayed beneath the user. Similarly, in the **DSS System**s tree, a node representing the user is displayed beneath the corresponding DSS System node. Figure 3-2 illustrates this duplicate display of information.

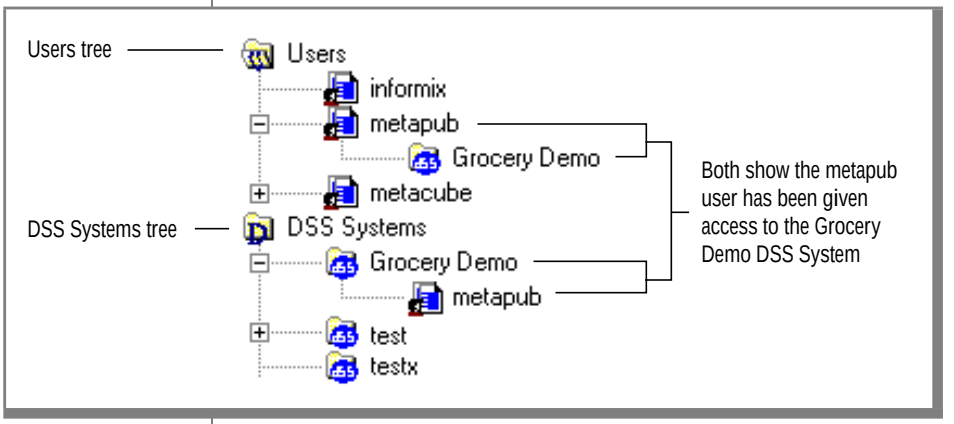


Figure 3-2
TreeView of Secure Warehouse

Active and Inactive Users

The ListView and TreeView panes of Secure Warehouse present two kinds of user icons:

- Colored user icons represent active users.
- Gray user icons represent inactive users.

Active users must be defined in metadata, and the connection information that is stored in the MetaCube registry for them must indicate they are associated with the current connection.

Inactive users are defined in metadata, but the connection information that is stored in the MetaCube registry for them indicates they are *not* associated with the current database connection. They are probably defined in multiple database connections, and their connection information matches one of those other database connections. You cannot use Secure Warehouse to manipulate an inactive user. First you must change the user into an active user by either switching to the default database connection for that user or by changing the user's default connection information to match the active database connection.

For every user managed in Secure Warehouse, the following connection information is stored as part of the registry information for the MetaCube analysis engine to which Secure Warehouse is connected:

- User ID
- Connect string (the string identifying the ODBC Data Source)
- Metaschema (the prefix identifying the schema or owner of the metadata tables in the database)
- Database type

Typically, the MetaCube analysis engine employs this user-connection information to facilitate connections between Web Explorer users and MetaCube DSS Systems, but Secure Warehouse also uses this connection information to determine whether a user is active or inactive. This is accomplished by comparing a user's connect string, as stored in the registry, with the connect string that you provide when you establish the database connection.

Secure User Privileges

Because MetaCube Secure Warehouse and MetaCube Warehouse Manager control the data warehouse and manipulate metadata, only authorized users should be granted secure user privileges. Secure user privileges are granted using the user properties dialog box.

What Data Can the User Access?

Beyond assigning a user access to DSS Systems, you can further control user access by defining mandatory filters that limit user access to certain categories of detailed information. For example, a user may be allowed to access a DSS System for summary sales data and a mandatory filter may be assigned to that user so he can only query sales data from the Eastern Region.

When Can the User Access Data?

Users can have unlimited ability to query a database, either by submitting real-time queries or QueryBack jobs (jobs that are queued on the server and processed in background). By defining user properties, you can limit the user to QueryBack jobs exclusively. You can also assign a priority level to the user and restrict the hours when the user's jobs are executed. For most users, a restriction on hours applies only to QueryBack jobs, but if the user is an administrator, the restriction also applies to any jobs submitted using MetaCube Agent Administrator.

How Does the User Access Data?

By defining user performance properties, you can specify the following aspects of how a user's query is processed:

- A threshold value for slow queries. A warning will be issued for queries that require a computational cost greater than this value. The cost is determined by the fact, aggregate, or sample table that the query will access. When using MetaCube Warehouse Manager to define metadata for the data warehouse, as discussed in Chapter 5, "Creating Metadata With Warehouse Manager," each of these data sources is assigned a cost.
- A maximum number of rows that can be retrieved.

The following user properties are specific to Informix databases:

- Auditing of user's queries. MetaCube Warehouse Optimizer can analyze data from user queries and recommend an aggregation strategy, as described in Chapter 7, "Warehouse Optimizer."

- PDQ Priority. You can enable PDQ Priority and specify a PDQ Priority value.
- Data skipping.

Managing Groups of Users

There are two ways to manage groups of users:

- Select multiple users and assign user properties
- Assign roles to users

Assigning User Properties to Multiple Users

In the ListView pane of Secure Warehouse main window, select multiple users by holding down `Ctrl` or `Shift`. Define user properties as described in Secure Warehouse online help.

Assigning Roles to Users

You can manage groups of users by assigning database roles to individuals. A *database role* is a classification of privileges. For example, you can assign a set of privileges to a certain role, such as *engineer*, and then assign that *engineer* role to many users rather than performing the repetitious work necessary to grant the same set of privileges to many different users.

Assigning a role to a user causes the MetaCube analysis engine, when connecting to the database, to issue a SET ROLE statement on behalf of the user, thus allowing any MetaCube application to take advantage of database-level security.

Tip: *When you assign roles, you are granting access to tables in the database. Assigning roles does not allow you to control access to DSS Systems for groups of users, nor does it allow you to apply mandatory filters to those groups.*

Use the database to assign privileges to a role. Use Secure Warehouse to assign roles to users. For more information on defining a role, refer to the SQL syntax documentation for your database.

Managing Snap-ins

You can use Secure Warehouse to manage the Snap-Ins available for each instance of the MetaCube analysis engine. Any users logged into a MetaCube data warehouse through that MetaCube analysis engine will have access to the Snap-Ins that are currently enabled for that engine.

Chapter

4

Warehouse Manager

In This Chapter

This chapter introduces MetaCube Warehouse Manager and describes the functions it performs.

Warehouse Manager online help system provides step-by-step procedures for all the tasks described in this chapter.

About MetaCube Warehouse Manager

MetaCube Warehouse Manager is a data warehouse administration program that enables you to create and manage the underlying metadata that allows other MetaCube applications to access the data warehouse. With Warehouse Manager, you can describe the underlying tables of your data warehouse as well as the associated aggregate and sample tables. This metadata model provides a logical, multidimensional system for MetaCube Explorer and other query tools to analyze data in a data warehouse.

Tip: *You may view up to 1,000 database tables, with up to 1,000 columns per database.*

Using Warehouse Manager, you can customize the interface for Explorer users. You incorporate icons into the Explorer interface that represent each of the data sources and dimensions of the data warehouse. Explorer users never see table names or column names because you change these names to familiar terms that are easily associated with your business.

Before you create metadata using Warehouse Manager, the tables that make up your data warehouse must be created and populated. For more information on setting up your data warehouse refer to Chapter 2, “Setting Up a Data Warehouse.”

The Warehouse Manager User Interface

Tip: *The DSS System for the MetaCube demonstration database is named MetaCube Demo; it is the source of the examples in this guide. Open this DSS System to follow along as you read and learn about Warehouse Manager.*

You must log in to Warehouse Manager and specify a DSS System to open (or create a new DSS System) before the main screen displays. The main screen, shown in Figure 4-1, is divided into three panes, a toolbar, drag functionality, and drop-down list boxes

Important: *You must be a secure user to access Warehouse Manager. For information on setting up secure users refer to Secure Warehouse online help.*

.

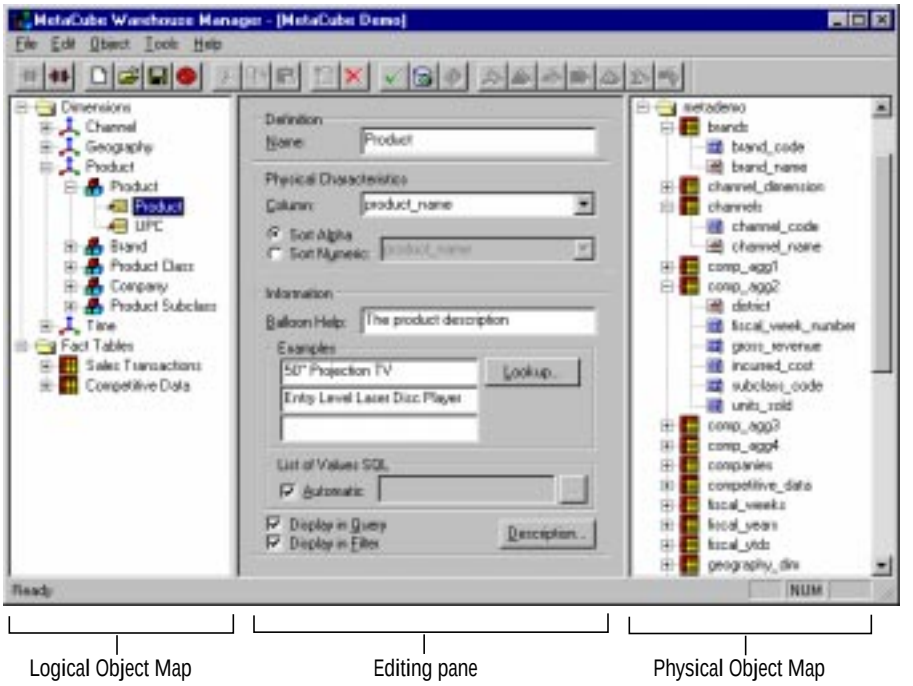


Figure 4-1 *Warehouse Manager's Main Screen*

The *Physical Object Map* is the right pane; it contains folder icons that represent the schemas / table owners and other icons that represent tables and columns in the database to which you are connected. One folder and its set of icons represent the metadata tables; usually, these tables are owned by the user **metacube**. As you use Warehouse Manager, you populate these metadata tables.

The *Logical Object Map* is the left pane; it displays folders and other icons representing the objects in the current DSS System. When you expand the folders in the Logical Object Map, icons appear that represent such objects as dimensions, dimension elements, dimension attributes, fact tables, aggregate tables, and sample tables. For a new DSS System, the only icons in the Logical Object Map are the empty **Dimension** and **Fact Table** folders. As you define the metadata using Warehouse Manager, the icons representing the objects you define appear in the Logical Object Map.

Between the Logical Object Map and the Physical Object Map is the *editing pane*, where you do the primary work of defining a DSS System. Selecting an icon in the Logical Object Map displays the details of that object in the editing pane. For illustrations and descriptions of the editing pane, refer to Chapter 5, “Creating Metadata With Warehouse Manager.”

Physical Object Map

The Physical Object Map serves two functions:

- It is a visual reference of the underlying database tables.
- It is a dynamic list from which you can drag database objects into the editing pane.

Figure 4-2 shows the hierarchy of the schemas / owners, tables, and columns in the MetaCube Demonstration Database. A distinct icon represents each object type.

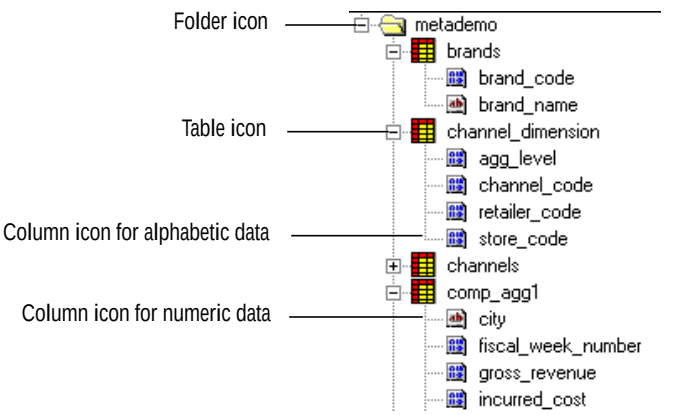


Figure 4-2
Icons in the Physical Object Map

Logical Object Map

The Logical Object Map displays two folders:

- The **Dimensions** folder, containing icons that represent each dimension, dimension element, and attribute.
- The **Fact Tables** folder, containing icons that represent each data source in the DSS System. Each data source expands to show its associated measures, aggregate tables, and sample tables.

For information on metadata objects contained in the **Dimensions** and **Fact Tables** folders, refer to Chapter 5, "Creating Metadata With Warehouse Manager."

The Dimensions Folder

Figure 4-3 shows the hierarchy of the dimensions, dimension elements, and attributes in the MetaCube Demonstration Database. A distinct icon represents each object type.

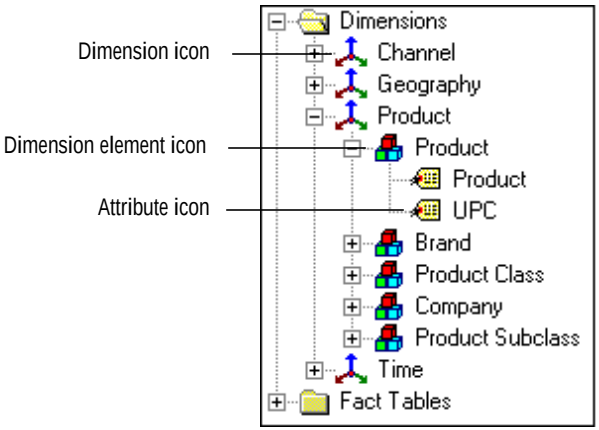


Figure 4-3 *Icons in the Dimensions Folder, Logical Object Map*

The Fact Tables Folder

At the center of every decision-support data model is the fact table. For every fact table in a data warehouse, Explorer offers a separate query environment or data source. The fact table, dimensions, attributes, aggregate tables, and sample tables, make up a single data source. Each data source represents a configuration of tables that completely defines a particular decision support environment.

The **Fact Tables** folder contains icons representing the data sources in the DSS System. Under each of these icons are folders containing icons for measures, aggregate tables, and sample tables associated with the fact table.

Figure 4-4 shows the hierarchy of the data sources, measures, aggregate tables, and sample tables in the MetaCube Demonstration Database. A distinct icon represents each object type.

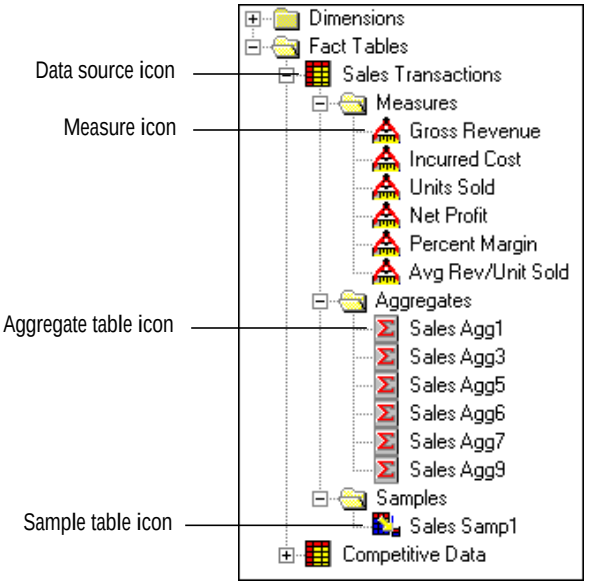


Figure 4-4
Icons in the Fact Tables Folder, Logical Object Map

Dragging Objects

You can drag the name of any schema/owner, table, or column from the Physical Object Map into the editing pane as you specify the physical characteristics of the DSS System. Use this technique anywhere in Warehouse Manager when you must provide the name of an object in the physical database.

List Boxes

If you prefer, you can use lists of available objects in the physical database associated with each of the fields in the editing pane. The lists display the same schemas, tables, and columns that appear in the Physical Object Map. A drop-down list displays only objects available for the selected physical object; for example, only tables for the current schema are listed, only columns for the current table are listed.

Chapter

5

Creating Metadata With Warehouse Manager

In This Chapter

This chapter describes the metadata objects that can be defined using Warehouse Manager, how to define the metadata, and how to verify the metadata.

To simultaneously create more than one of any type of metadata object described in this chapter, refer to "Warehouse Manager Assistants" on page 5-38.

Warehouse Manager online help provides step-by-step procedures for all tasks described in this chapter.

Tip: *The examples in this chapter are based on the MetaCube demonstration database. You may want to open the demonstration database to follow the examples and view Warehouse Manager on your PC.*

Guidelines for Metadata

Informix recommends the following guidelines for metadata objects:

- Schema, table, and column names are derived from the physical database components. MetaCube supports standards for the databases to which it connects.
- Names may contain up to 240 alphanumeric characters. Informix databases require at least one alphabetic character (or underscore) in a name. Other restrictions on naming are governed by requirements from SQL and from the database where the data warehouse is stored.
- Metadata names can not include SQL reserved words or keywords. Special characters, such as dots (periods), commas, single or double quotes, parentheses, brackets, carriage returns or tabs, are not allowed. Metadata names can not contain leading or trailing spaces.

- An aggregate table can not have a name that duplicates a fact table name.
- A dimension element table can not have a name that duplicates a dimension table name.
- Each physical table can be mapped to only one metadata object, unless you use synonyms. Without the use of synonyms to reference the same physical table more than once, you risk the MetaCube analysis engine generating incorrect SQL.

Dimensions

To fully define the metadata for a dimension, you specify characteristics of the dimension itself, then its dimension elements, and finally its attributes. You cannot define a dimension element before specifying metadata for the dimension to which the element belongs. Similarly, you cannot define an attribute before specifying metadata for the dimension element that the attribute describes.

For a given DSS System, the **Dimensions** folder in the Logical Object Map contains every possible dimension that can be incorporated into the data sources that make up a DSS System. The DSS System in the MetaCube demonstration database includes four dimensions—**Channel**, **Product**, **Geography**, and **Time**. Each fact table in this DSS System can join to any combination of these dimensions, or to all of them.

Before you can join a new dimension to a fact table, you must first define the dimension so that its icon appears in the **Dimensions** folder. The specification for a join from a dimension table to a particular fact table is part of the fact table definition.

After you add a new dimension to the **Dimensions** folder in the Logical Object Map, enter metadata for the new dimension in the Dimension editing pane.

The Dimension Editing Pane

Figure 5-1 shows the completed editing pane for the **Time** dimension of the MetaCube demonstration database. Compare the fields in the editing pane with the database objects displayed in the Physical Object Map to the right.

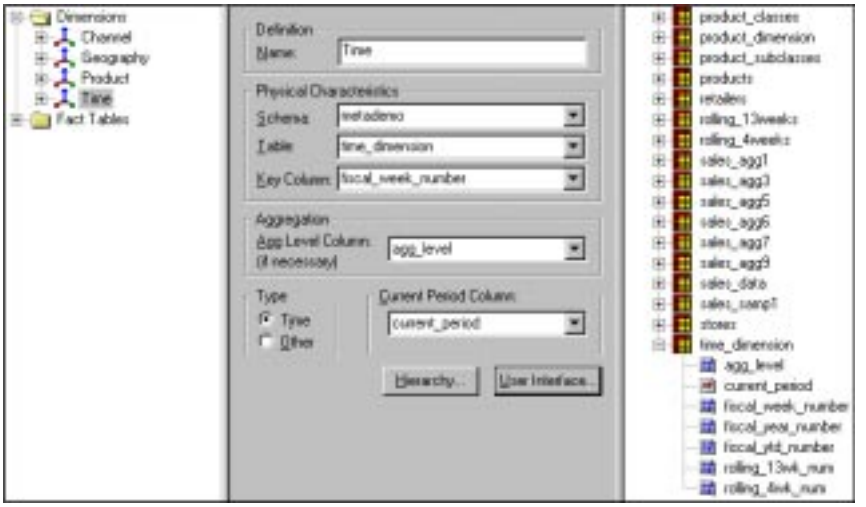


Figure 5-1
Dimension Editing Pane

The Dimension editing pane contains the following fields and buttons and has the following metadata defined for this example:

- **Name:** the name of the dimension is **Time**. This name appears in the MetaCube query applications.
- **Schema:** the schema owning this table is **metademo** (**metademo** owns all the tables for the MetaCube demonstration database).
- **Table:** the name of the underlying table is **time_dimension**.
- **Key Column:** the column used to join the fact table and the dimension table is **fiscal_week_number**.
- **Agg Level Column**: the name of the aggregation level column is **agg_level**. This field is left blank if you are using dimension element tables.
- **Type:** the dimension contains **Time** information. Since the type is **Time**, you must also specify the name of the current period column.

- **Current Period Column: current_period** is the column that contains information about what the current period is. This field is only activated if you specify **Time** as the type of information.
- **Hierarchy** button: brings up the Hierarchy Editor, described on page 5-11, in which you can define the dimension hierarchy. You cannot define the hierarchy until you have defined all the dimension elements for the dimension, but you must define the hierarchy before you define attributes for the dimension.
- **User Interface** button: brings up the Dimension User Interface editor, described on page 5-15, in which you specify the icon for the dimension and the order in which the attributes display in the MetaCube query applications. You cannot use the Dimension User Interface editor until all attributes are specified.

Objects included in your dimension definition must correspond to the actual structure of the physical tables in your data warehouse. That is, table names must belong to the designated schema/owner, and key, aggregate, and current period column names must exist and be correct for that table. Although you can incorrect information in the editing pane fields, Warehouse Manager's Verify feature identifies these errors and rejects the object definition. For more information about Warehouse Manager's Verify Feature, see "The Verify Feature" on page 5-40.

Dimension Elements

Dimension elements are metadata definitions of columns within a dimension table. The dimension element columns in a dimension table enable access to the attribute data associated with each dimension element. Thus, dimension elements serve to join data stored in the central fact table with the attributes that define a particular view of that data.

Metadata for a dimension element includes information about whether the dimension table contains attribute data (as in a star schema) or whether attribute data is stored in a separate table (as in a snowflake schema). In either case, there must be at least one attribute associated with each dimension element code.

For example, in the MetaCube demonstration database, the dimension element **Brand** corresponds to the **brand_code** column in the **Product** dimension table. Dimension element columns store short codes, since it is most efficient to reference data and join tables with short code fields, rather than long text fields. The MetaCube demonstration database stores the **Brand** dimension element code in the **brand_code** column and stores the **Brand Name** text in the **brand_name** column.

You cannot define a dimension element until you have created the dimension to which it belongs.

After you define all dimension elements for a dimension, return to the Dimension editing pane and specify the dimension hierarchy using the Hierarchy Editor. Do this *before* defining attributes, since one feature of the Attribute editing pane will not function if the dimension hierarchy is not defined. For information about the Hierarchy Editor, see "The Hierarchy Editor" on page 5-11.

The Dimension Element Editing Pane

Figure 5-2 shows the completed editing pane for the **Product** dimension element of the MetaCube demonstration database. Compare the information in the **Definition** frame with the icons on the Logical Object Map for the Product dimension element. Also, compare the fields in the **Attribute Table** frame with the database objects displayed in the Physical Object Map to the right.

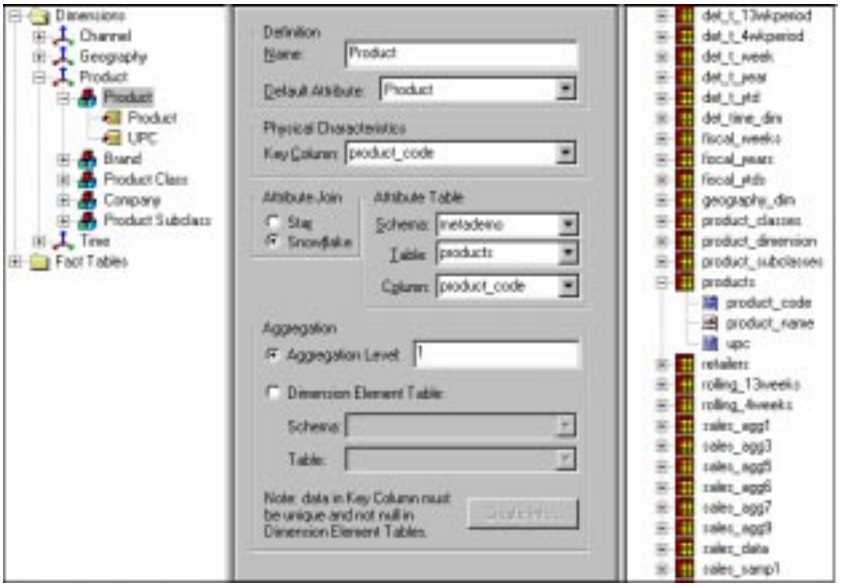


Figure 5-2
Dimension Element Editing Pane

The Dimension Element editing pane contains the following fields and buttons and has the following metadata defined for this example:

- **Name:** the name of the dimension element is **Product**.
- **Default Attribute: Product** is the default attribute that displays when a user drills up or down to a hierarchy level that branches. You cannot complete the Default Attribute field until you have defined the attributes for the dimension element.
- **Key Column: product_code** is the column used to join aggregate tables to the dimension table or, in a snowflake schema, to join the attribute table to the dimension table.

Important: *If you are using dimension element tables, the* ***Key Column*** *is used as the primary key for the dimension element table. You must ensure that unique values are stored in the column named in the* ***Key Column*** *field.*

- **Attribute Join:** the underlying metamodel is a **Snowflake** schema, so you must also complete the fields in the **Attribute Table** frame.
- **Schema:** the schema owning this table is **metademo** (**metademo** owns all the tables for the MetaCube demonstration database). This field is only active when a snowflake schema is specified.
- **Table:** the attribute table name is **products**. This field is only active when a snowflake schema is specified.
- **Column:** the column used to join the attribute table to the dimension table is **product_code**. This field is only active when a snowflake schema is specified.
- **Aggregation Level:** if you specify that you are using aggregation levels, you must also enter the aggregation level value in the text box to the right. The aggregation level is **1**.

  Specifying an **Aggregation Level** is only required when implementing dimension tables with an aggregate level column and aggregate rows (that is, in a non-DET implementation). This column is not used when implementing DETs. When implementing DETs, the dimension table contains no aggregate level column, so this field is left blank.
- **Dimension Element Table:** if you specify that you are using dimension element tables, you must also complete the **Schema** and **Table** fields. You can then click **Create Info** which brings up the Dimension Element Create Information dialog box described in "Creating Dimension Element Tables" on page 5-10.

  Although it is not recommended, Warehouse Manager allows you to structure a dimension so that some dimension element information is stored in the dimension table and some is stored in separate DETs. In this case, the dimension table would have some aggregate rows and an aggregate level column for the non-DET storage of dimension hierarchy information. The **Agg Level Column** field must be used to specify the name of the aggregate level column.

Objects included in a dimension element definition must correspond to the actual structure of the physical database. That is, table names must belong to the designated schema / owner, and column names must exist and be correct for that table. Although you can enter incorrect information in the editing pane fields, Warehouse Manager's Verify feature identifies this error and rejects the object definition. For more information about Warehouse Manager's Verify Feature, see "The Verify Feature" on page 5-40.

Creating Dimension Element Tables

Clicking the **Create Info** button in the Dimension Element editing pane brings up the Dimension Element Create Information dialog box, where you can specify the SQL that populates the dimension element table. Click **Generate SQ**L to have Warehouse Manager automatically specify the SQL. The unique values are selected from the dimension table, as shown in Figure 5-3.

Figure 5-3
Dimension Element Create Information Dialog Box

After specifying the metadata for dimension element tables, using Agent Administrator, submit a Full Aggregate job to create and populate the DETs. Refer to "Submitting Jobs" on page 6-11.

The Hierarchy Editor

The Hierarchy editor is a graphical representation of the hierarchical relationships among dimension elements. Using the editor to define dimensional hierarchies establishes the mechanism by which the MetaCube analysis engine accesses aggregate tables and determines the most efficient access paths to data. Hierarchies also support the MetaCube Explorer drill-down and drill-up functionality.

The Hierarchy Editor is accessed from the Dimension editing pane, described on page 5-5.

Each of the boxes in the hierarchy editor represents a different dimension element, and the arrows between the boxes point from lower to higher levels, forming one or more consolidation paths.

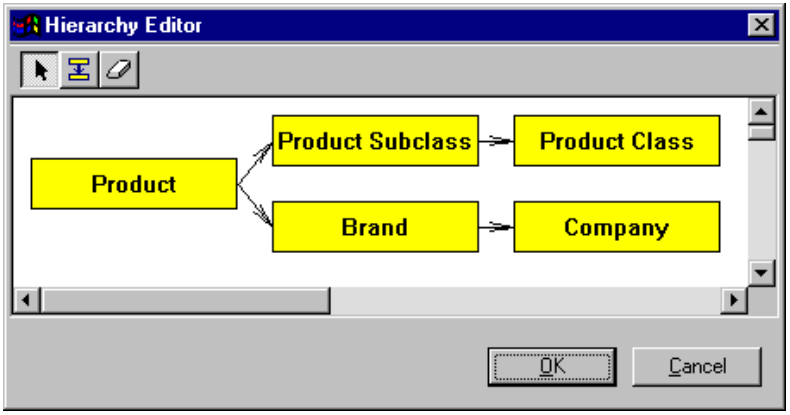


Figure 5-4
Hierarchy Editor Dialog Box

Figure 5-4 shows the hierarchy for the **Product** dimension in the MetaCube demonstration database. **Product**, the base element, branches along two paths: to **Product Subclass** and **Product Class** along one branch and to **Brand** and **Company** in the other.

The organization of boxes in the Hierarchy Editor screen must accurately reflect the hierarchical relationships established by the logical design of your data warehouse and reflected in the physical database tables. The Hierarchy editor is a graphical way to specify this metadata; it is not used to make changes on the underlying physical tables.

When defining a dimension hierarchy, observe these rules:

- Only one element may be the lowest or base element in a hierarchy; remaining elements must connect directly or indirectly to the base element.
- Each element must have at least one relationship incorporating it into the overall hierarchy.
- Elements may not be joined in a circular pattern.

Warehouse Manager will not verify hierarchies that do not satisfy these criteria.

Complex Hierarchies

Dimension hierarchies may be defined as simple, with a single consolidation path, or as complex (branching), with multiple consolidation paths. When defining a complex hierarchy, the rules stated above must be followed.

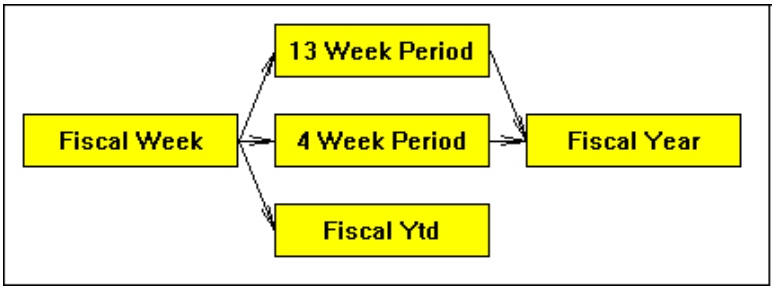


Figure 5-5
Time Dimension in the Hierarchy Editor

In the **Time** dimension of the MetaCube demonstration database, shown in Figure 5-5, the base element, **Fiscal Week**, consolidates along two different paths to **Fiscal Year** and along a third path to **Fiscal Ytd**. On the first consolidation path, **Fiscal Week** rolls up to **4 Week Period**, and on the second consolidation path, **Fiscal Week** rolls up to **13 Week Period**. **4 Week Period**, **13 Week Period**, and **Fiscal Ytd** are on different consolidation paths because neither of the weekly time intervals fits evenly into the other nor into the **Fiscal Ytd**. However, both **4 Week Period** and **13 Week Period** evenly consolidate into the 52-week **Fiscal Year**, and for this reason, their consolidation paths can converge there.

Complex hierarchies in the metadata allow the MetaCube analysis engine to guide users through dimensions. If a MetaCube user selects an attribute associated with **Fiscal Ytd** and drills down, MetaCube, on the basis of the metadata defined above, automatically returns **Fiscal Week** level detail. On the other hand, if the user drills down from the **Fiscal Year** level, MetaCube asks the user to choose between **4 Week Period** or **13 Week Period**, since the hierarchy forks at this point.

Attributes

MetaCube users formulate queries using attribute names. Therefore, the names you specify for attributes should be meaningful to those users. When specifying the metadata for attributes, you assign not only the name of the attribute, but also other features of the Explorer user interface, such as balloon help, prototype data for sample reports, and whether attribute names are available to Explorer users to incorporate into queries and into filters.

You cannot define an attribute until you have defined the physical characteristics of both the dimension and the dimension element with which that attribute is associated.

The Attribute Editing Pane

Figure 5-6 shows the completed editing pane for the **Brand** attribute of the MetaCube Demonstration Database. Compare the fields in the editing pane with the Physical Object Map to the right.

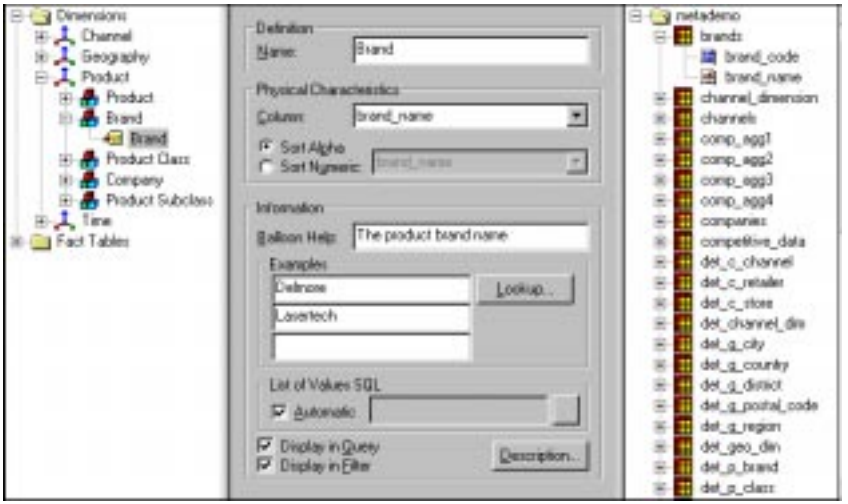


Figure 5-6 *Attribute Editing Pane*

The Attribute editing pane contains the following fields and buttons and has the following metadata defined for this example:

- **Name:** the name of the attribute is **Brand**. This name appears in the MetaCube query applications.
- **Column:** the column storing the data for this attribute is **brand_name**.
- **Sort Alpha:** the attribute values in the **brand_name** column will sort in alphabetical order as entered in the database.
- **Sort Numeric:** if this button is activated, the values in the **Column** you specify will sort numerically. You may instead specify another column as the numeric sort column in the drop-down list to the right of this button.
- **Balloon Help:** the balloon help that displays in Explorer for this attribute is **The product brand name**.

- **Examples:** the values for the Explorer example report are **Delmore** and **Lasertech**. Click **Lookup** to search the underlying database table for the values you wish to use.

Tip: *You cannot use the* ***Lookup...*** *button to select example text for the Examples fields unless the dimension hierarchy has been defined.*

- **List of Values SQL:** when set to **Automatic**, MetaCube automatically generates SQL for use by Explorer in retrieving a list of the unique values for this attribute; this list displays in Explorer when users formulate a query filter.

  You may instead uncheck the **Automatic** checkbox and click the **...** button. This displays the List of Values SQL dialog box, where you can create a SQL statement yourself to select the attribute values from a custom table that contains only the unique attribute values.

- **Display in Query:** this attribute is available to Explorer users for queries. This attribute will only appear in the Explorer Ad Hoc page if this checkbox is checked.
- **Display in Filter:** this attribute is available to Explorer users for filters. This attribute will only appear in the Explorer Filter Element Definition dialog box if this checkbox is checked.

When MetaCube verifies this metadata, it checks that the column name is correct for the attribute table you specified in the Dimension Element editing pane. For more information about Warehouse Manager's Verify Feature, see "The Verify Feature" on page 5-40.

Dimension User Interface Editor

The Dimension User Interface editor, accessed from the Dimension editing pane, allows you to define the appearance of the MetaCube Explorer Ad Hoc tab page. The Dimension User Interface editor allows you to define:

- the icon that represents the dimension.
- the order in which the attributes appear in the DSS System hierarchy

.

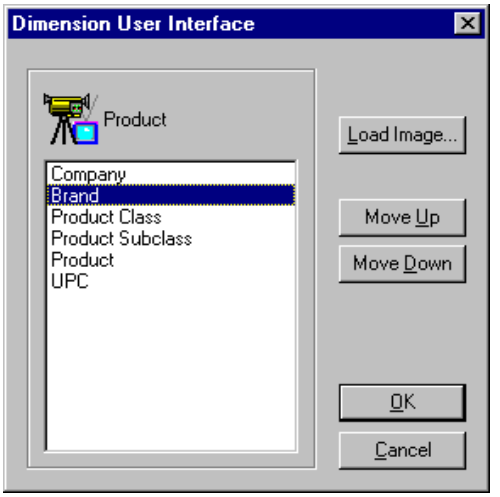


Figure 5-7
Dimension User Interface Editor

Figure 5-7 shows the **Product** dimension of the MetaCube demonstration database in the Dimension User Interface editor. Displayed above the list box is the name of the dimension and the icon that represents it on the Explorer Ad Hoc tab page. The list box displays the names of the dimension attributes in the order they appear in the Explorer Ad Hoc tab page.

It can be helpful to Explorer users to see the hierarchy of attributes within a dimension, so you might order the list to reflect the underlying hierarchy. Rearranging names in this list alters the Explorer display, only, and has no affect on the underlying database or the dimension element hierarchy.

Fact Tables

The metadata in the **Fact Tables** folder describes a fact table and its associated measures, aggregate tables, and sample tables. For each fact table in the DSS System, you can choose, from the library of all defined dimensions, the ones to incorporate into a given data source. Since all data sources are independent, a dimension can be associated with more than one fact table.

Fact tables consist of the following columns:

- Columns that store numeric data (measures) at the most detailed level
- Columns that join the fact table to base dimension elements in the dimension tables

As an example, assume a **Sales** data source contains a single measure—**Units Sold**—and two dimensions—**Geography** and **Product**. The fact table would contain three columns:

- The **units_sold** data column
- Two columns storing codes, such as **store_code** and **product_code**, to join to each of two dimension

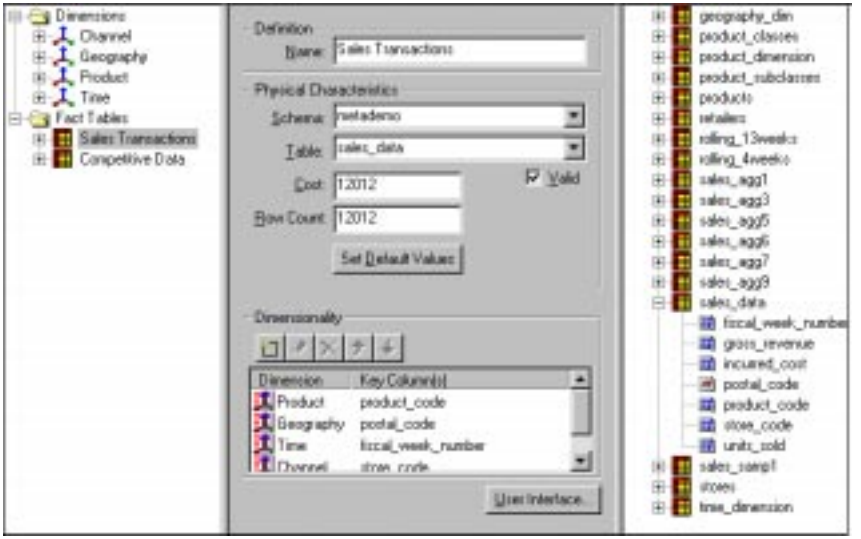
For each business transaction stored in this data source, the **units_sold** column stores the number of items sold, the **store_code** column stores a code for the store where the items were sold, and the **product_code** column stores a code for the particular product. The codes join to the base dimension element (and its associated attributes) in the two dimension tables. These joins also provide access from the fact table to higher level dimension elements and attributes in the dimension tables.

The **Fact Tables** folder contains every fact table included the DSS System. You cannot define a fact table until you have defined the physical characteristics of dimensions it joins to.

The Fact Table Editing Pane

Figure 5-8 shows the completed editing pane for the **Sales Transactions** fact table of the MetaCube demonstration database. Compare the fields in the editing pane with the Physical Object Map to the right.

Figure 5-8
Fact Table Editing Pane



The Fact Table editing pane contains the following fields and buttons and has the following metadata defined for this example:

- **Name:** the name of the fact table is **Sales Transaction**s. This name appears in the MetaCube query applications as a data source name.
- **Schema:** the schema owning this table is **metademo** (**metademo** owns all the tables for the MetaCube demonstration database).
- **Table:** the name of the underlying database table is **sales_data**.
- **Cost:** the performance cost of accessing the table is **12012**.
- **Row Count:** The number of physical rows in the table is **12012**.

- **Set Default Values:** click to automatically enter the row count of the table into the **Cost** and **Row Count** fields.
- **Valid:** the fact table is marked **Valid**. This opens the fact table for querying from MetaCube Explorer and other client applications.
- **Dimensionality:** the names and key columns for all dimensions associated with the Sales Transaction fact table are listed in this frame.

The Fact Table User Interface Editor

The Fact Table User Interface editor allows you to define the appearance of the MetaCube Explorer Ad Hoc page. The Fact Table User Interface editor allows you to define:

- the icon that represents the fact table.
- the order in which the measures appear in the DSS System hierarchy.
- whether measures are displayed before dimensions.

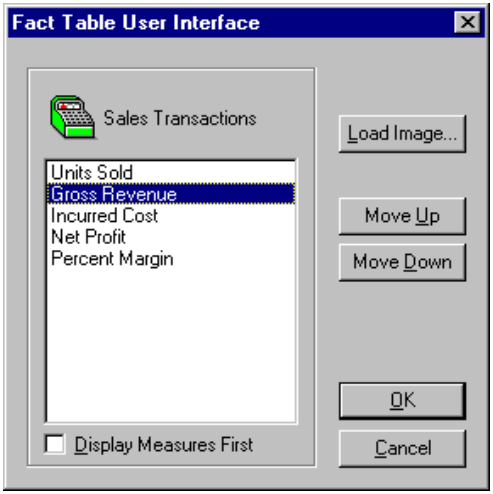


Figure 5-9
Fact Table User Interface Editor

Figure 5-9 shows the **Sales Transactions** fact table from the MetaCube demonstration database in the Fact Table User Interface editor. Displayed above the list box is the name of the fact table and the icon that represents it on the Explorer Ad Hoc page. The list box displays the names of the fact table measures in the order they appear in the Explorer Ad Hoc tab page.

Measures

Numeric data tracked in a fact table are referred to as *measures*. In a data warehouse, measure data are stored in the central fact table and in aggregate tables, which are summarized versions of the fact table. Every fact table must have at least one measure.

There are two types of measures:

- Stored measures—data stored in the columns of the fact table
- Calculated measures—calculated by the MetaCube analysis engine at the time a query is executed, according to a formula that you specify

In the MetaCube demonstration database, **Gross Revenue**, **Incurred Cost**, and **Units Sold** are stored measures. **Net Profit** and **Percent Margin** are calculated measures.

For example, **Percent Margin** is a calculated measure that is not stored in the fact table. By storing in the metadata a formula for calculating percent margin, the MetaCube analysis engine can calculate this measure when the query calls for it. The formula used to return a value for **Percent Margin** is based on **Gross Revenue** and **Incurred Cost**, two stored measures in the fact table.

Calculation of data is required to obtain correct results in summarized reports. For example, if a percent margin value were stored in the fact table for each row, and then these values were added together for a summarized report, the answer would be incorrect. Therefore, MetaCube calculates the percent margin using the summarized totals in the report for its calculation.

The Measure Editing Pane

You use the Measure editing pane to specify the metadata for all measures, stored or calculated, in the DSS System.

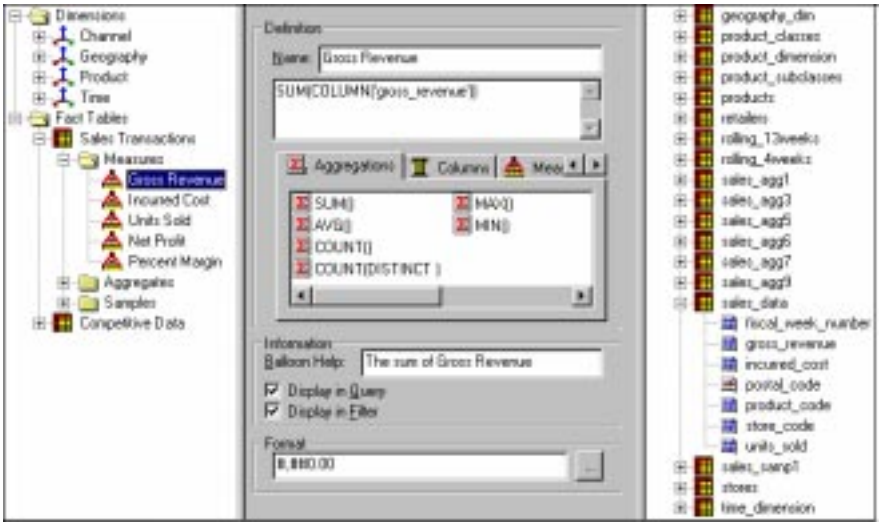


Figure 5-10 *Measure Editing Pane*

The Measures editing pane contains the following fields, tabbed pages and buttons and has the following metadata defined for this example:

- **Name:** the name of the measure is **Gross Revenue**.
- **Aggregations:** use this tab to incorporate an aggregate function into the definition for a stored measure. The **Aggregations** tab displays all the aggregate functions supported by MetaCube; these include SUM( ), AVG( ), COUNT( ), COUNT(DISTINCT), MAX( ), and MIN( ). An aggregate function is required in the definition of a stored measure. If you do not use one of the other aggregate functions, the SUM( ) function is required.
- **Columns:** use this tab to incorporate the name of a database column into your definition of stored measure. The **Columns** tab displays all the columns in the fact table.

- **Measures:** use this page to incorporate a previously defined measure into a formula. The **Measures** page displays all defined measures, both stored and calculated, except the currently highlighted measure.
- **Operators:** use this page to incorporate the arithmetic operators into a formula for a calculated measure. The **Operators** page displays the arithmetic operators: + - / and *, as well as ( ) to indicate precedent.
- **Constraints:** use this page to define a measure constraint. Previously defined constraints are displayed on this page, as well.
- **Balloon Help**: the balloon help that displays in Explorer for this measure is **The sum of Gross Revenue**.
- **Display in Query:** this measure is available to Explorer users for queries. This measure will only appear in the Explorer Ad Hoc page if this checkbox is checked.
- **Display in Filter:** this measure is available to Explorer users for filters. This measure will only appear in the Explorer Filter Element Definition dialog box if this checkbox is checked.
- **Format:** the format for the display of numeric values for this measure is **#,##0.00**. Click the **...** button to display the Format dialog box. MetaCube users can change the display of numeric data, if they wish; however, the format you define here is the default. The formatting options available in Warehouse Manager are the same as those used in Microsoft Excel. For more information on the formatting options, refer to Warehouse Manager online help.

The primary working area of the Measure editing pane is the set of tabs and the text box that displays definitions of stored and calculated measures. Use the scrolling arrows to display the entire set of tabs. They type of measure you are defining determines which tabbed pages you use. Although you can specify a measure by typing its definition into the text box, it is easier and more accurate to use the Warehouse Manager interface to build the definition automatically by clicking or dragging. Sometimes, it may be necessary to combine manual entry with clicking or dragging.

Stored Measures

When you specify a stored measure, you create metadata to describe the data columns contained in the fact table. Use the **Aggregations** and **Columns** pages to build a stored measure.

Calculated Measures

When you specify a calculated measure, you create metadata to describe the calculation the MetaCube analysis engine will perform when the query is run. The calculation is based on previously defined stored measures or other calculated measures. Use the **Measures** and **Operators** pages to build a calculated measure.

You may add constant values to the formula for a calculated measure. Constants must be typed into the formula by clicking the appropriate location in the formula and entering the value.

Measure Constraints

As with any filter definition, three components make up the definition of a constraint:

- An **Operand** is either a measure or an expression derived from one or more measures.
- An **Operator** is a comparison operator, such as = or <>.
- A **Value** is a constant to which the operand is compared.

Tip: *The SQL generated for a measure constraint contains a HAVING clause to constrain the measure.*

Syntax for Measure Definitions

Measure definitions may be entered manually in the text box of the Measure editing pane, or you may use the mouse to click the tabs and select the item needed to create the definition.

The table below provides some examples of measure definitions. Use these examples as prototypes for the measure definition you wish to create.

Measure Type	Syntax Examples
Stored	SUM(COLUMN('GROSS_REVENUE'))
	SUM(COLUMN('INCURRED_COST'))
	COUNT(DISTINCT COLUMN('PRODUCT_CODE'))
Calculated	FACT('Gross Revenue') - FACT('Incurred Cost')
	1-(FACT('Incurred Cost') / FACT('Gross Revenue'))

Aggregates

A data warehouse can hold a significant amount of data in its fact table, and, as you continue to add data to the warehouse, the fact table can become large enough to negatively impact performance. Processing time for queries on large tables is directly related to the number of rows in the table. The larger the number of rows, the longer the processing time.

MetaCube enables you to preprocess data by calculating summary values from the fact table and storing this summarized data in aggregate tables. Incorporating aggregate tables into the data warehouse allows the MetaCube analysis engine to return reports quickly since retrieving data from aggregate tables reduces the number of rows a query must process to retrieve and calculate report data.

Aggregate tables summarize data according to the levels of the dimension hierarchy above the base level. By recognizing the hierarchical relationships among dimension elements, the MetaCube query optimizer routes a given query to any aggregate table that stores data at the same or lower level in the hierarchy than the attributes contained in the query. An aggregate table that summarizes product sales by month, for example, can not be used to process queries requesting weekly sales figures; however, that aggregate table can be used to process queries for quarterly or yearly sales.

Therefore, queries requesting summarized data access aggregate tables; queries requesting base level data access the fact table. The MetaCube analysis engine determines what table to access to process any given query.

The metadata for an aggregate table specifies columns that store measures. Since aggregate tables typically include all measure columns contained in the fact table, they contain stored measures. As with fact tables, calculated measures should not be stored in aggregate tables. The MetaCube analysis engine processes calculated measures at the time the query is run, based on the formulas in the metadata.

Tip: *Warehouse Optimizer can be used to help you decide which aggregates most benefit your DSS System needs. For information on Warehouse Optimizer, refer to Chapter 7, "Warehouse Optimizer," and Warehouse Optimizer online help.*

Aggregate Tables

Aggregate tables can be defined in the metadata before they actually exist as physical tables. As part of the specification of aggregate tables, Warehouse Manager generates the SQL statements necessary to build the tables you define. Using MetaCube Agent Administrator, the MetaCube Agent, Aggregator, actually builds aggregate tables using the automatically generated SQL.

When you have completely defined aggregates, you must build the tables for your data warehouse. Building aggregate tables is a separate step and is not accomplished by simply registering the metadata using Warehouse Manager. Once an aggregate table exists in your data warehouse, mark it as **Valid**. This allows the MetaCube analysis engine to use it for queries.

For more information on creating physical aggregate tables, refer to "Automatically Generating Physical Aggregate Tables" on page 5-34.

Aggregate Groups

When you specify aggregate table definitions in the metadata (and subsequently build them), you can specify that the data be summarized either by dimension element or by attribute. As components of an aggregate table definition, the set of dimension elements and attributes associated with an aggregate table is called an *aggregate group*.

Aggregating on Dimension Elements

Aggregating on dimension elements allows MetaCube to process queries for each of that dimension element's attributes. Aggregating on the **Brand** dimension element in the MetaCube demonstration database, for example, summarizes product data by **brand_code**. If a user queries on **Brand Name**, MetaCube can use the **Brand Code** aggregate table to join to the dimension table. With this technique, one aggregate can be used to handle queries for all attributes of a dimension element. Moreover, because this aggregate joins to the entire dimension table, it can also be used to return results for higher-level dimension elements and their related attributes. For these reasons, aggregating on dimension elements is recommended.

Aggregating on Attributes

In certain cases, it is useful to aggregate on attributes. For queries on a single attribute, attribute aggregates deliver better performance than dimension element aggregates. Building aggregates on frequently requested attributes may justify the use of aggregating on attributes. The chief disadvantage of using attribute aggregates is that the table can be used only to return results for the single attribute contained in the table. That is, it cannot be used to roll up to higher levels in the dimension hierarchy.

The Aggregate Table Editing Pane

Figure 5-11 shows the completed editing pane for the **Sales Agg3** aggregate of the MetaCube demonstration database. Compare the fields in the editing pane with the Physical Object Map to the right

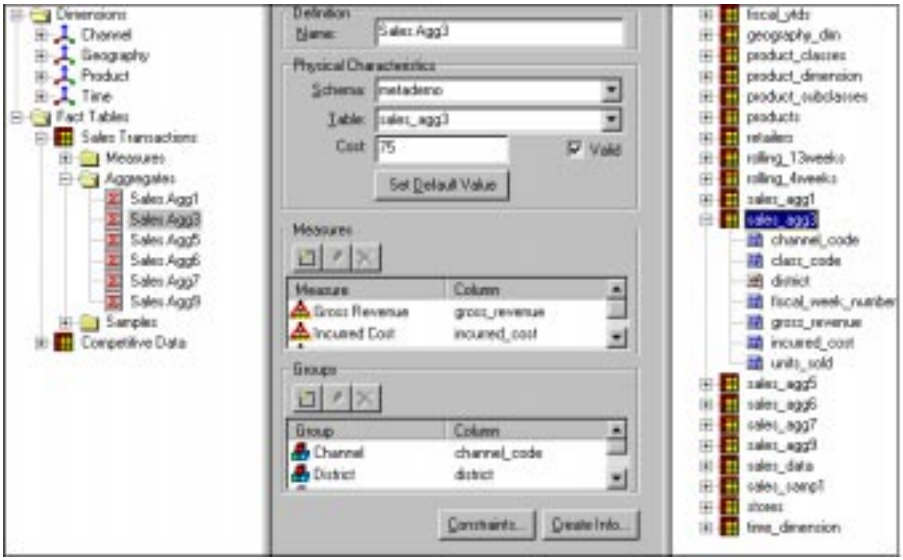


Figure 5-11
Aggregate Table Editing Pane

The Aggregate Table editing pane contains the following fields and buttons and has the following metadata defined for this example:

- **Name:** the name of the aggregate is **Sales Agg3**.
- **Schema:** the schema owning the table is **metademo** (**metademo** owns all the tables for the MetaCube demonstration database).
- **Table:** the underlying database table is **sales_agg3**.
- **Cost:** the performance cost of querying this table is **75**.
- **Set Default Value:** click to automatically enter the row count of the table into the **Cost** field.
- **Valid:** the aggregate table is marked **Valid**. This opens the aggregate table for querying from MetaCube Explorer and other client applications.

- **Measures:** the measures included in the aggregate table are **Gross Revenue**, **Incurred Cost**, and **Units Sold**.
- **Groups:** the group of dimension element and attribute tables join to the aggregate with the following columns:
  - The **Channel** dimension joins at the **channel_code** column, or **Channel** dimension element, thereby losing **Retailer** and **Store** detail.
  - The **Geography** dimension joins at the **district** column, or **District** dimension element, thereby losing **City** and **Postal Code** detail.
  - The **Product** dimension joins at the **class_code** column, or **Product Class** dimension element (the highest element), thereby losing all other levels of detail.
  - The **Time** dimension joins at the **fiscal_week_number** column, or **Fiscal Week** dimension element (the base element), thereby making all time detail available to this aggregate table.

Tip: *To figure out what detail will be lost with a specific table join, look at the hierarchy of the dimension.*

Aggregate Constraints

The *aggregate constraint* (or filter) allows you to create smaller aggregate tables by filtering out data that will not be needed. For example, suppose your data warehouse contained a **Region** dimension with two attributes—**Eastern** and **Western**; however, many more queries request information about the **Eastern** region than about the **Western** region. Using an Aggregate filter, you could create an aggregate table in which all data for the **Western** region is excluded. The resulting aggregate table would provide summarized data for the **Eastern** region only. By doing this, you can reduce by roughly one half the number of rows processed for a query requesting data only for the **Eastern** region.

Defining an aggregate constraint is a two-step process:

1. Select the dimension that contains the attributes on which you wish to filter.
2. Define the filter itself.

After you choose the dimension you want to filter the selected aggregate on, define the *filter elements*—components of a filter. Filter elements are defined in the Filter Element Definition dialog box. In this dialog box, you build a comparison expression, or filter definition, that causes Warehouse Manager to limit the range of data retrieved for populating the aggregate table. Only data for which the comparison expression evaluates to true is retrieved into the table.

Tip: *If you are specifying an aggregate constraint for the time dimension, you cannot specify a relative time filter. To obtain correct results, you must specify an absolute time filter for an aggregate table. In the* ***Time*** *frame, the* ***Relative*** *button is disabled.*

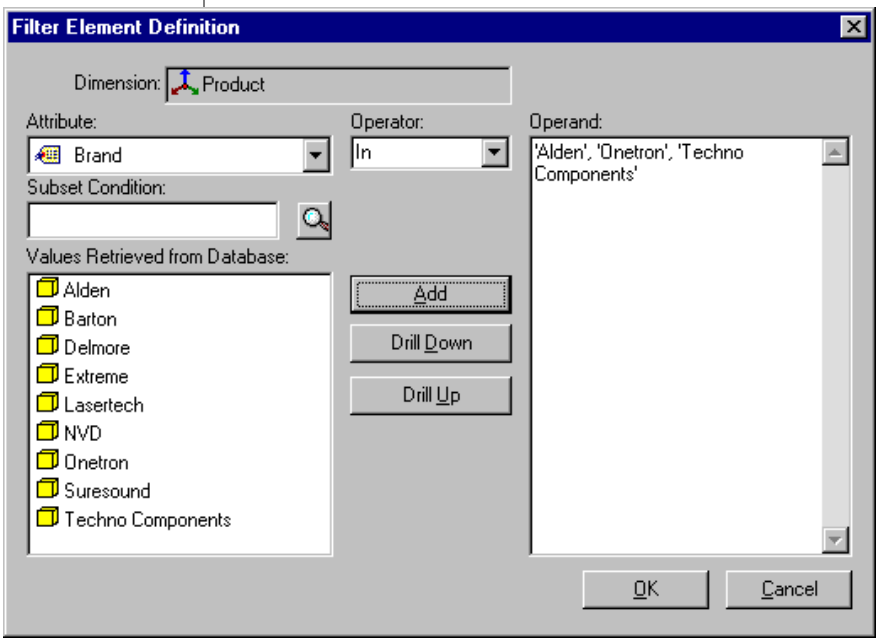


Figure 5-12
Filter Element Definition Dialog Box

The features of the Filter Element Definition dialog box are described below:

- **Dimension:** displays the name of the dimension for which you are building the filter.
- **Attribute:** displays a list of available attributes you can use for the comparison expression. Only attributes within the dimension for which you are building the filter appear in this box.

Important: *The attribute you use as the basis for an aggregate filter must be specified as a member of the aggregate group.*

- **Subset Condition:** enter search criteria to retrieve a subset of the attribute values.

  If the list of attribute values exceeds 200, you may need to subset the list. Subsetting a value list is discussed in the section "Working with a Long Value List" on page 5-32.

- **Search button:** use to retrieve all possible attribute values that correspond to the **Subset Condition** you enter. If the **Subset Condition** text box is left blank, the Search button retrieves all attribute values.
- **Values Retrieved from Database:** displays the list of all possible values for the attribute displayed in the **Attribute** text box.
- **Operator:** use to select the operator for the comparison expression. The comparison operators that you can use are shown in the table below:

Operator	Meaning
=	Equal; the default and most commonly used operator
< >	Not equal
>	Greater than; most useful for numeric comparisons*
>=	Greater than or equal to; most useful for numeric comparisons*
<	Less than; most useful for numeric comparisons*
<=	Less than or equal to; most useful for numeric comparisons*
In	Within the specified list; used for both numeric and character string comparisons
Not In	Not within the specified list; used for both numeric and character string comparisons
Like	Used for pattern matching; the wildcard is % and can be used to replace zero or more characters in the search pattern

(1 of 2)

Operator	Meaning
Not Like	Used for pattern matching; the wildcard is % and can be used to replace zero or more characters in the search pattern
Is Null	Is empty
Is Not Null	Is not empty

(2 of 2)

* The operators >, <, >=, and <= can also apply alphabetical parameters to string values. For example, the *Alden* brand is less than the *Delmore* brand, since A precedes D in the alphabet.

- **Operand:** displays the attribute value that completes the filter's comparison expression. You can double-click a value or drag a value from the **Values Retrieved from Database** list into this text box. You can also select multiple values from the list to drag simultaneously.
- **Add:** use to add attribute values to the comparison expression. Click one or more values in the **Values Retrieved from Database** list, then click the **Add** button.
- **Drill Down:** use to display attribute values in the next level down in the dimension hierarchy. Drilling down automatically changes the name of the attribute on which you are filtering, thereby changing the comparison expression.
- **Drill Up:** use to display attribute values in the next level up in the dimension hierarchy. Drilling up automatically changes the name of the attribute on which you are filtering, thereby changing the comparison expression.

Tip: *When drilling up or down, you may be required to specify the attribute to which you wish to drill. This happens when hierarchies branch (that is, they follow more than one path between levels). For more information on drilling up and down, refer to* *"Multiple Drill Paths" on page 5-32*.

Working with a Long Value List

If an attribute's list of values is longer than 200 items, you may receive the caution shown in Figure 5-13.

Figure 5-13
Long Value List Error Message

As this caution explains, the value list is too long to display all at once, so a partial list has been retrieved. If the value or values you wish to select are contained in this list, use the truncated list. However, you may wish to select other values not in the initial list retrieved by Warehouse Manager.

To divide the attribute value list into subsets, use the text box to enter the beginning letter or letters of the values from which you wish to select. For example, entering `m` and clicking the **Search** button displays a list of all character string attribute values starting with the letter m. Entering `mo` displays a list of all character string attribute values starting with the letters mo. The search is case-sensitive, so you must enter capital letters to locate values that use upper case. You can include SQL wildcard syntax in this field; for example, entering `%o%` retrieves all the values containing the letter o.

Multiple Drill Paths

When a hierarchy branches along two or more paths, you may need to clarify which path Warehouse Manager should follow when drilling up or down.

For example, in the MetaCube demonstration database, the **Product** dimension contains a branching hierarchy, as shown in Figure 5-14. The **Product** attribute—the base attribute in the hierarchy—is the point where the hierarchy branches, following two paths upward through the dimension

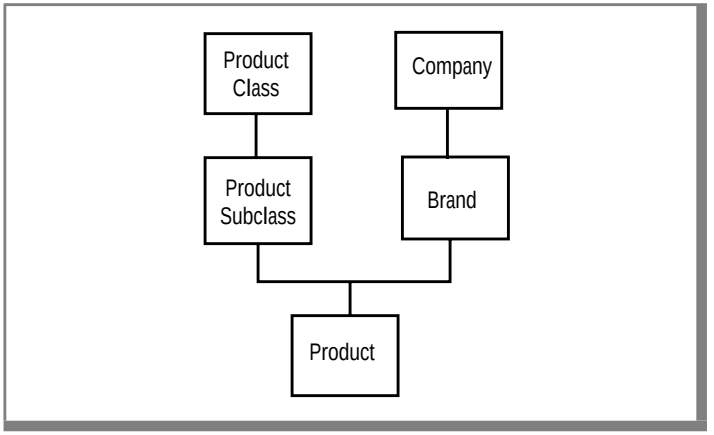


Figure 5-14 *Branching Hierarchy for the Product Dimension in the Demonstration Database*

When you drill up on an attribute where a hierarchy branches, you need to tell Warehouse Manager which branch to follow.

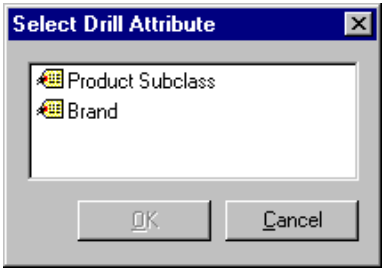


Figure 5-15 *Select Drill Attribute Dialog Box*

This dialog box allows you to select the attribute you wish to drill up to in the **Product** dimension. The two attributes are in different branches.

Time dimensions typically have multiple paths—the MetaCube demonstration database illustrates this. The **Time** dimension has three consolidation paths that branch starting with the **Fiscal Week** attribute. Drilling up from **Fiscal Week** allows you to choose to consolidate weeks in one of the following ways:

- **4 Week Periods**, and from there to **Fiscal Year**

- **13 Week Periods**, and from there to **Fiscal Year**
- **Fiscal YTD** (year-to-date) period

When you drill up or down, you move along the consolidation path one level at a time. It is not possible to skip levels.

Drilling Down to Multiple Attributes at the Same Level

Your data warehouse may be designed with multiple attributes at a single level within a dimension. If so, for any dimension where multiple attributes occur at some level in the hierarchy, you specified a default attribute for that dimension element. When drilling to a level in a dimension where there are multiple attributes, MetaCube automatically retrieves values for the default attribute you specified in the metadata.

In the demonstration database for example, in the **Product** dimension, the **Product** and **UPC** attributes both describe individual products at the lowest level of the hierarchy. The **Product** attribute is the product name; the **UPC** attribute is the Universal Product Code (UPC) for each product.

The default attribute for the product level of detail is **Product**. Therefore, if you drill down in the **Product** dimension, Warehouse Manager automatically displays **Product** attribute values, not **UPC** values.

Automatically Generating Physical Aggregate Tables

You can generate SQL statements that create a physical aggregate table using the Aggregate Create Information dialog box. This dialog box has three tabbed pages.

Create Statement Page

On the basis of your specification of an aggregate table, Warehouse Manager generates an aggregate definition consisting of SQL statements that, when executed, extract data from the fact table that is used to populate the aggregate table. You can copy this SQL to a SQL development environment to manually build the aggregate table, if you wish. If you automatically build aggregate tables, the SQL is used by the MetaCube Aggregator, one of the MetaCube Agents, to actually build the tables.

When you click **Generate SQL**, Warehouse Manager automatically generates the SQL for the aggregate table and displays it in the **Create Table Statement** text box. Alternatively, you can enter the SQL statements manually.

After creating the SQL statement you can optionally append database-specific options, such as storage parameters. Text entered in this text box is appended to the CREATE TABLE SQL statement.

Indexes Page

Use the **Indexes** page to define, optionally, one or more indexes for optimizing access to the aggregate table. If you want to index the aggregate table and will automatically generate aggregate tables using MetaCube Aggregator, enter indexing statements here.

Grants Page

Use the **Grants** page to define, optionally, the privileges required to access the aggregate table. By default, MetaCube assigns the same permissions to aggregate tables that you assigned to the fact table.

Grant user permissions here if you want to grant user permissions other than those assigned to the fact table in your data warehouse and you plan to automatically generate aggregate tables using MetaCube Aggregator.

Sample Tables

When queries are run against very large fact tables, performance can be adversely affected. However, for reports with low-level detail, aggregate tables, which can improve data retrieval time, cannot be used. The MetaCube sampling feature provides a mechanism whereby Explorer users can query randomly generated subsets of the fact table to get statistically valid reports. The sampling feature is known to Explorer users as Estimating Results.

Important: *When a MetaCube data warehouse resides in an Informix Dynamic Server with Advanced Decision Support and Extended Parallel Option, the MetaCube analysis engine takes advantage of this server's ability to retrieve sampled data and, therefore, does not make use of separate sample tables.*

The advantage of using sample tables is that queries run much faster, yet still produce highly reliable reports. Explorer users can do the following tasks:

- Choose when to run reports against sample tables
- Determine the level of confidence they require for a given report
- View, in the resulting report, the range of error for all data in the report

You can define (and subsequently build) one or more sample tables for the fact table in every data source of your data warehouse. When you define a sample table, you specify, among other things, the number of rows it should contain.

You can use MetaCube Agent Administrator to automatically generate sample tables. The MetaCube Sampler agent generates the sample tables. After you have specified the metadata for the sample tables you wish to create, MetaCube Sampler generates statistically valid tables containing the number of rows you have specified.

When you have completely defined the sample table and it exists as a physical table in your data warehouse, mark the sample table as **Valid**. This opens the sample table for queries from MetaCube Explorer and other client applications.

Before marking a sample table **Valid**, you must build the tables in your data warehouse. Building sample tables is a separate step and is not accomplished by simply registering the metadata using Warehouse Manager.

The Sample Table Editing Pane

Figure 5-16 shows the completed editing pane for the **Sales Samp1** sample table of the MetaCube demonstration database.

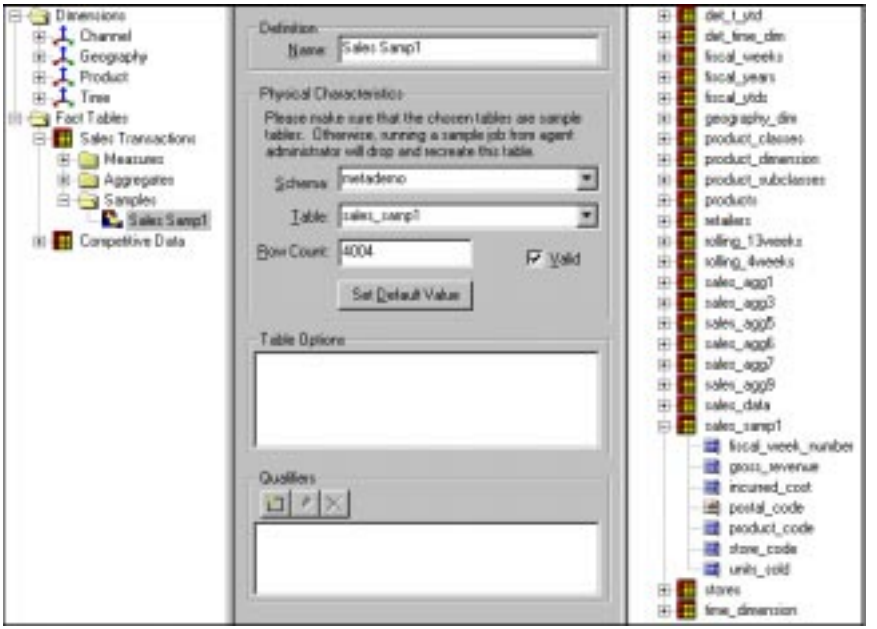


Figure 5-16 *Sample Table Editing Pane*

The Sample Table editing pane contains the following fields and buttons and has the following metadata defined for this example:

- **Name:** the name of the sample table is **Sales Samp1**.
- **Schema:** the schema owning the table is **metademo** (**metademo** owns all the tables for the MetaCube demonstration database).
- **Table:** the underlying database table name is **sales_samp1**.
- **Row Count:** the number of rows included in this table is **4004**.
- **Set Default Value:** click to automatically enter the row count of the table into the **Row Count** field.
- **Valid:** the sample table is marked **Valid**. This opens the sample table for querying from MetaCube Explorer and other client applications.

- **Table Options:** there are no database-specific table creation options, such as storage parameters, for this table.
- **Qualifiers:** there are no indexing or user permission SQL statements appended to this table.

When the MetaCube Sampler builds the sample table, it uses this metadata. To learn about using Agent Administrator and the MetaCube Sampler, refer to Chapter 6, "Managing MetaCube Agents."

Warehouse Manager Assistants

You can quickly specify the name of one or more new metadata objects and the location of the underlying relational database tables or columns using Warehouse Manager assistants. The assistants define physical characteristics for multiple metadata objects simultaneously. You complete the definition in the editing pane.

Depending on which assistant you use, you select the tables or columns to associate with the new metadata objects.

- Select the underlying tables for the following assistants:
  - Dimension Assistant
  - Fact Table Assistant
  - Aggregate Assistant
  - Sample Assistant
- Select the underlying columns for the following assistants:
  - Dimension Element Assistant
  - Attribute Assistant
  - Measure Assistant

Available Tables or **Available Columns** lists all of the underlying tables or columns in the database. Once you specify the columns or tables, new icons representing them appear in the Logical Object Map, identified by names the assistant generates for them.

The assistant automatically generates names for the new metadata objects based on the names of the tables or columns. In the **Naming conventions** frame, you can change the names in the following ways:

- Check **Drop Table Prefix** or **Drop Column Prefix** to remove the first word of the table or column name and its connecting underscore character.
- Check **Drop Table Suffix** or **Drop Column Suffix** to remove the final word of the table or column name, beginning with the connecting underscore.
- If a table or column name contains a suffix and a prefix, check both options to remove both the first and final words of the table or column name and the connecting underscores.

  If both options are checked, but the table or column name contains only two words, the final word of the table or column name is removed.

If you do not check either option, the metadata object will have the same name as the underlying table or column.

The following table shows what each assistant defines for the new metadata object and any tasks you must complete after creation of the metadata object by the assistant.

Assistant	What the Assistant Defines	Further Action by You
Dimension Assistant	■ Name ■ Schema ■ Table	Complete the remaining fields in the Dimension editing pane.
Dimension Element Assistant	■ Name ■ Key Column	Complete the remaining fields in the Dimension Element editing pane.
Attribute Assistant	■ Name ■ Column ■ Sort Column ■ Balloon Help	Complete the remaining fields in the Attribute editing pane.

(1 of 2)

Assistant	What the Assistant Defines	Further Action by You
Fact Table Assistant	■ Name ■ Schema ■ Table ■ Cost (1,000,000,000 default) ■ Row Count (1,000,000,000 default)	Complete the remaining fields in the Fact Table editing pane. When you have completely defined the data source, return to the Fact Table editing pane, and mark the fact table **Valid**. This opens the fact table for queries.
Measure Assistant	■ Name ■ SUM(COLUMN('*specified_column*')) as the Stored measure definition	Complete the remaining fields in the Measure editing pane.
Aggregate Assistant	■ Name ■ Schema ■ Table ■ Cost (1,000,000,000 default)	Complete the remaining fields in the Aggregate editing pane. Since this table exists in the database, mark the aggregate **Valid**. This opens the aggregate table for queries.
Sample Assistant	■ Name ■ Schema ■ Table ■ Cost (1,000,000,000 default)	Complete the remaining fields in the Sample editing pane. Since this table exists in the database, mark the sample **Valid**. This opens the sample table for queries.

(2 of 2)

Important: *You cannot use the Aggregate Assistant or Sample Assistant to define the metadata for aggregate tables or sample tables if the physical tables do not exist in the database.*

The Verify Feature

Important: *You should always verify your DSS System to confirm that there are no errors before committing changes to the database.*

Once you have defined metadata, MetaCube Warehouse Manager allows you to verify the metadata. You can verify an individual metadata object and all objects belonging to it or verify an entire DSS System.

Important: *Verification is done on metadata where tables have been checked* ***Valid****, meaning they are open for querying from MetaCube query applications. Therefore, to assure that metadata is verified for fact tables, aggregate tables, and sample tables, check the* ***Valid*** *checkbox. If the underlying database tables do not yet exist, clear the* ***Valid*** *checkbox after verification.*

When Warehouse Manager scans the metadata, it verifies that the following conditions are met:

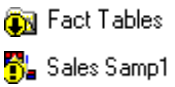
- All dimensions have exactly one base dimension element.
- All schemas, tables, and columns reference objects that actually exist in the relational database.
- All mandatory fields for objects contain valid values.
- All dimension elements have exactly one default attribute.
- All dimension hierarchies are constructed according to the rules for hierarchies and include all of the dimension elements.

There are two ways to use the verify feature:

- You can verify an object and any objects below it in the tree with the **Verify** command in the **Object** menu.
- You can verify an entire DSS System with the **Verify DSS System** command in the **File** menu.

If you are making many changes or additions to the DSS System you might want to verify the metadata occasionally to assure that it does not contain errors.

After you have completed the verify operation, Warehouse Manager flags invalid objects in the Logical Object Map. Objects that contain invalid objects are also flagged. Following are two samples of invalid object flags:



Indicates that the **Fact Tables** folder contains an invalid object.

Indicates that **Sales Samp1** has an invalid table definition.

You can learn what problems occurred during verification with the **Show Verify Errors** command in the **Object** menu.

The Verify Errors dialog box displays all of the problems that the verify process encountered for a particular object. The error messages that you are likely to encounter, along with explanations of what they mean, are included in the table below.

Message	Explanation
Dimension does not have a base dimension element.	There can be only one base dimension element for each dimension. Use the Hierarchy editor to verify the dimension hierarchy.
Dimension has multiple base dimension elements.	There can be only one base dimension element for each dimension. Use the Hierarchy editor to verify the dimension hierarchy.
Dimension element is not reachable via roll-ups from the base element.	Every dimension element must be part of the dimension hierarchy. Refer to the hierarchy rules in "The Hierarchy Editor" on page 5-11.
Dimension element does not have a default attribute.	You must specify a default attribute for each dimension element.
Dimension element has multiple default attributes.	You must specify exactly one default attribute for each dimension element.
Invalid schema specified for property.	You have specified a schema name that does not exist in the database to which you are connected.
Invalid table specified for property.	You have provided a table name that does not exist in the specified schema in the database to which you are connected.
Invalid column specified for property.	You have provided a column name that does not exist in the currently specified table in the database to which you are connected.
Measure definition is invalid.	You have defined a measure with an expression that could not be understood. Refer to "Syntax for Measure Definitions" on page 5-23.

(1 of 2)

Missing mandatory property.	You have not filled out all of the required fields for this object.
Property is out of the acceptable range with value.	Some fields have only certain ranges of valid values. You have entered a value out of the acceptable range.
There is a cycle in the dimension element roll-up hierarchy.	You have created a dimension hierarchy that loops back on itself. See the rules in the section "Dimension User Interface Editor" on page 5-15.

(2 of 2)

Chapter

6

Managing MetaCube Agents

In This Chapter

This chapter introduces the following MetaCube products:

- MetaCube Scheduler, a server-side scheduling mechanism that controls the operation of the various MetaCube Agent processes
- MetaCube Agent Administrator, a tool for managing MetaCube Scheduler and the agent processes Scheduler initiates
- MetaCube Agents

About MetaCube Scheduler

MetaCube Scheduler allows users to submit jobs to the server. It evaluates each job's priority, the privileges of the user who submitted the job, and any dependencies or conditions that must be satisfied before the job can run. It then spawns the appropriate MetaCube process to execute the job on the server.

MetaCube Scheduler offers more sophisticated features than an operating system scheduler, such as UNIX cron. The following are examples of MetaCube Scheduler features:

- User- and job-level priorities
- Ability to designate the times and days of the week at which a particular user may run jobs
- Multiprocessor support
- Support for multiple jobs, with dependencies between jobs, as well as dependencies on external conditions
- Ability to submit and monitor jobs from either the client or the server

MetaCube Scheduler serves as the parent process to all of the MetaCube Agent processes. These processes include jobs spawned to execute SQL statements, QueryBack functions, full and incremental aggregation functions, and even operating system commands.

Once Scheduler has been started, all MetaCube Agents functions can be managed from the graphical Windows environment of MetaCube Agent Administrator. Scheduler itself, however, must be started at the server.

Starting Scheduler

To start the Scheduler, log on to the server operating system under which you installed MetaCube Agents software. At the operating system prompt, type:

- `sc run` for UNIX.
- `metasc run` for Windows NT.

MetaCube Scheduler will prompt you for a login and database name. After entering this information, you will be returned to the operating system prompt.

Important: *Your database login must have DBA privileges.*

Skipping the Prompts

To preclude Scheduler from prompting you for a login and database name, you can create a resource file, named **.scrc**, to store the appropriate values for the login and database name, which Scheduler passes to the database. The **.scrc** file is documented in *MetaCube Installation and Configuration Guide* for your server platform.

In the **.scrc** file, enter a line using the following format:

```
loginID#database_name
```

where *loginID* is your database login and *database_name* is the name of the database where the metadata and data warehouse files are stored.

Checking the Status of Scheduler

To check the status of Scheduler, log on to the server operating system under which you installed MetaCube Agents software. At the operating system prompt, type:

- `sc status` for UNIX.
- `metasc status` for Windows NT.

This produces status information similar to the following text:

```
Run_State is set to              Normal Operation
max_jobs is                     5
sleeptime is                    30
number of jobs running:         0
number of jobs pending:         0
```

This status information is actually stored in a database table. In the event of system failure or other abnormal occurrence, this information may not be precisely synchronized with actual operating system events.

Stopping the Scheduler

To stop Scheduler, log on to the server operating system under which you installed MetaCube Agents software. At the operating system prompt, type:

- `sc stop` for UNIX.
- `metasc stop` for Windows NT.

Scheduler stops, but any processes already spawned by Scheduler will complete. The operating system displays the following message and you are returned to the operating system command line:

```
MetaCube Scheduler Stopped...
Current Jobs will finish.
```

This action records in the database a command to stop Scheduler. The process does not necessarily stop immediately. The next time Scheduler wakes from its sleep cycle, it checks for the stop command. When it finds the command, Scheduler stops.

Important: *You can stop the process using operating system commands to kill the process, but this is not recommended, as it may leave the database in an indeterminate state.*

With the MetaCube Agents Control dialog box in MetaCube Agent Administrator, you can view the status of Scheduler and you can stop Scheduler. You cannot, however, use that dialog box to start Scheduler. For information on the MetaCube Agents Control dialog box, see "Other Administrative Tasks" on page 6-14

About MetaCube Agent Administrator

MetaCube Agent Administrator is a graphical software system that allows you to administer all the features and functions of MetaCube Scheduler as well as the various Agents that it spawns. With Agent Administrator, you can manage the job queue, submit new jobs, and perform other administrative duties that a MetaCube installation may require.

Agent Administrator online help system provides step-by-step procedures for all the tasks in this section.

Logging in to MetaCube Agent Administrator

After opening Agent Administrator, a login dialog box prompts you for a user name and password. Logging in connects Agent Administrator to a database.

If you have not previously defined a database connection for Agent Administrator, you must do so before attempting to log in. Click **Cancel** on the MetaCube Login dialog box, then choose **Options** on the **Tools** menu. The Preferences dialog box displays, where you can define database connections. See Appendix D or Agent Administrator online help for more information on configuring a database connection.

Managing the Job Queue

The job queue is a list of all jobs submitted to Scheduler for server-side processing, regardless of the application from which jobs were submitted. Database tables store information for each job, including the commands associated with the job, its priority, and the identity of the user who submitted the job.

How MetaCube Scheduler Processes Jobs in the Queue

MetaCube Scheduler processes the jobs stored in the queue based on their target start time, their priority, and the maximum number of jobs that can be run concurrently (a value set using Agent Administrator). The priority assigned to jobs determines the order in which concurrent jobs run. Scheduler runs jobs until either the maximum number of concurrent jobs is met or no more jobs require processing.

Periodically Scheduler polls to detect changes to the queue, such as the submission or deletion of a job. The rate at which Scheduler performs this polling is a value set using Agent Administrator. The Agent Administrator application does not perform any automatic polling of the job queue, so you must refresh the queue yourself to retrieve current information about jobs.

Information Provided by the Job Queue

In Agent Administrator, all jobs in the queue, whether pending, executing, or complete, appear in the Job Queue dialog box.

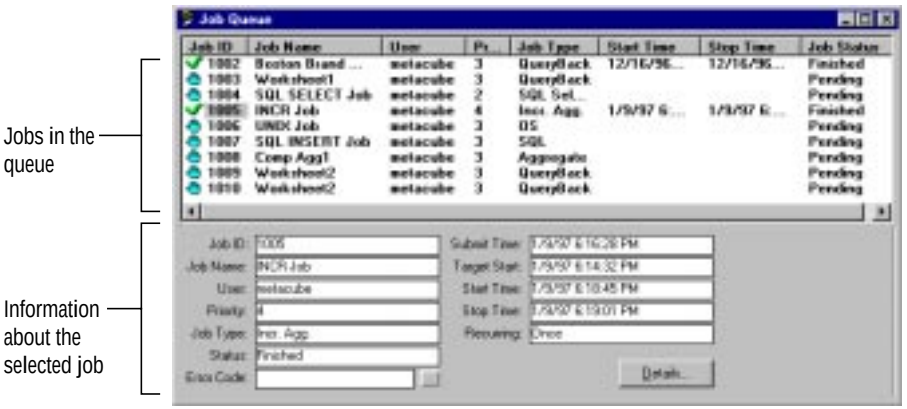


Figure 6-1
Job Queue Dialog Box

Each entry in the Job Queue dialog box provides the following information for a job:

- A unique identification number
- An icon indicating whether the job is pending, running, or complete and whether the job returned error messages. The status of each entry is also reported in the **Job Status** column (**Pending** for jobs waiting to execute, **Running** for jobs currently executing, **Deleting** for jobs that are being deleted, or **Finished** for jobs that have executed).
- The name assigned to the job
- The user who submitted the job
- The priority assigned to the job
- The type of job
- The time Scheduler spawned a process to execute the job (blank if the job is pending)
- The time the job completed (blank if the job is pending or running)

The bottom portion of the Job Queue dialog box provides information in textual form about a specific job. Clicking a job ID in the job list selects the entry for that job and obtains the most recent information for that job from the database. The bottom portion of the screen provides all of the same job information provided in the job list above, but it also delivers the following additional information:

- Recurring status (does the job run once or repeatedly)
- Any error code returned with the job and a button that displays information about the error
- A **Details** button that displays additional information about the job (the nature of this detailed information varies from job to job)

Displaying the Contents of the Queue

Using Agent Administrator, you can perform any of the following tasks related to the display of information in the job queue:

- Display the Job Queue dialog box
- Refresh the job queue status information to display the most recent job data
- Display the most recent information for a particular job
- Display details about a job
- Filter the queue

  You can define one or more filters that instruct Agent Administrator to display or not display certain types of jobs. When specifying a filter, you choose a property, such as job type, a comparison operator, such as equals, and a value to which the property should be compared. For example, you might create a filter that displays only job types equal to SQL Select jobs.
- Sort the queue

  By clicking on the header for any job list element, such as the header for the job name or the job ID, you can sort the queue in ascending or descending order for that element.

Modifying a Job in the Queue

While using Agent Administrator, you can modify the contents of the job queue in the following ways:

- Delete a job

  Deleting a job from the queue prompts Scheduler to delete from the database any temporary tables that may have been created to store the job's result set. A job can be removed from the queue by either the user who originally submitted the job or the data warehouse administrator.

  If the job is running when you delete it, the job queue will list the job as **Deleting**. When the job terminates with an error or finishes normally, the job will be deleted from the job table. If the job terminates abnormally, you can change the job status to **Finished** and delete it from the job queue.

Tip: *Removing an Aggregate, Incremental Aggregate, or Sample job does not remove the tables created by those jobs.*

- Change a job's dependencies

  Agent Administrator allows you to impose a set of dependencies that must be satisfied before a job or job set executes. Each time Scheduler polls the database for new jobs, these dependencies are checked. You can stipulate that the execution of a job depends on the *completion* of another job or on the *successful completion* of a job. You can also stipulate that the execution of a job depends on the satisfaction of an arbitrary condition, as evaluated by an SQL statement. SQL dependencies should return an integer value, such as SELECT COUNT (*). A returned value of 0 indicates that the condition *has not been satisfied*. Values returned other than 0 indicate the condition *has been satisfied*.

- Change a job's priority

  Agent Administrator allows you to assign a priority between 1 (the lowest) and 5 (the highest) to a job. To obtain an overall priority for a job, MetaCube Scheduler sums the job's priority with the priority assigned to the user who submitted the job. (User priorities are assigned using MetaCube Secure Warehouse.) Scheduler uses this overall priority to determine the order in which multiple jobs will run.

- Change a job's time and frequency of execution

  Agent Administrator and the MetaCube query applications allow scheduling of a job for execution at a specific date and time. If nothing is specified, a job is scheduled to execute right away. You can also schedule a job to run on a recurring basis, such as weekly. By default, jobs run only once.

- Change a job's status

  Jobs can have three possible states: **Pending** jobs are waiting to run, **Running** jobs are currently executing, and **Finished** jobs have completed their execution. You may want to change a job's status if it has finished and you want to run it again (change the status to **Pending**), or if a job terminates abnormally, you may need to change the status to **Finished**.

Submitting Jobs

Agent Administrator allows you to submit the types of jobs described in the following table. For all of the jobs listed below, you can define the job's dependencies, priority, and time and frequency of execution.

Job	Description
SQL Non-Select	Executes data modifications using SQL statements you specify, such as UPDATE, INSERT, and DELETE. An SQL Non-Select job can also perform database maintenance tasks on tables and indexes using statements such as CREATE, ALTER, and DROP.
SQL Select	Runs SQL Select statements you specify, returning results stored in a table in the database.
Operating System	Runs a specific operating system command. These may be shell scripts or custom executables. You specify the command, whether it is a local or remote job, the host, the user ID, and the password.
Full Aggregate	Creates new aggregate tables and dimension element tables* based on descriptions registered in MetaCube metadata.
Incremental Aggregate	Creates incremental versions of existing aggregate tables based on changes to a fact table. You specify the metamodel schema, the DSS System, the fact table, whether to use serial or parallel processing. You can also specify Informix table options.
Post Incremental Aggregates**	Incorporates incremental versions of aggregate tables into existing aggregate tables, inserting or replacing rows as necessary. This job also incorporates an incremental fact table into the existing fact table.
Sample	Creates new sample tables based on descriptions registered in MetaCube metadata. You specify the DSS System, and select the registered sample tables from a list.
Web Publisher***	Creates an HTML-formatted file that holds the results of a query. Refer to "Specifying a MetaCube Web Publisher Job" on page 6-21 for instructions on submitting a Web Publisher job.

(1 of 2)

Job	Description
Alert	Sends an email message when a specified SQL statement returns one or more rows. You specify the email recipient(s), message to send, and an SQL Select statement.
Job Set	Submits a collection of jobs that can consist of any of the types of jobs listed in this table, including another Job Set.

(2 of 2)

Important: ** You can use the Full Aggregate agent to create only DETs. Refer to* *"Manual Generation of DETs" on page 6-12*.

*** The Incremental Aggregate agent adds a* ***mc_upd_flag*** *column to the incremental aggregate tables. If you run a Post Incremental Aggregate job without first running an Incremental Aggregate job, you will receive an error indicating that the* ***mc_upd_flag*** *column is missing.*

**** You can use Agent Administrator to submit a Web Publisher job only when all MetaCube components are installed on the same Windows NT machine. Otherwise, you must submit the job through DOS.* *"Specifying a MetaCube Web Publisher Job" on page 6-21* *provides instructions on submitting a Web Publisher job through DOS.*

QueryBack jobs can be submitted from MetaCube Explorer or MetaCube for Excel. These jobs display in the job queue.

Manual Generation of DETs

A set of command line options for the MetaCube Full Aggregate agent executable (**aggexec**) enables the building of dimension element tables only. You can run **aggexec** from the command line in UNIX to build dimension element tables for a UNIX database. On a Windows NT PC, you can build dimension element tables for a Windows NT database or, using ODBC, you can build dimension element tables for a UNIX database.

The **aggexec** command takes several arguments to specify exactly the dimension element tables to create. The following table lists the command line arguments.

Important: *The* `-M`, `-A`, *and* `-E` *options are mutually exclusive; you may use only one of them at a time on the command line. You may need to run* ***aggexec*** *several times to create the desired set of dimension element tables. Alternatively, create a shell script to run several* ***aggexec*** *commands in a single batch.*

Argument	Meaning
`-d`	Required. ■ On the UNIX command line, the name of the database. ■ On a Windows NT PC, the name of the ODBC Data Source used to connect to the database. The database may reside on a Windows NT or a UNIX system.
`-D`	Required. DSS System name.
`-U`	Required for Windows NT. Database login ID; not used when running **aggexec** on a UNIX command line.
`-P`	Required for Windows NT. User password for database login ID; not used when running **aggexec** on a UNIX command line.
`-M`	Required. Dimension name as defined in metadata; cannot be used with the -A or -E option. Builds all DETs specified in the metadata for that dimension.
`-A`	Required. Physical aggregate table name; cannot be used with the -M or -E option. Builds all DETs required for that aggregate table.
`-E`	Required. Dimension element name; cannot be used with the -M or -A option. Builds only the specified DET.

The following examples illustrate the use of the **aggexec** command.

The following command, issued on a UNIX command line, builds the **DET_Retailer** dimension element table in the MetaCube Demo DSS System stored in a database named **demo402**:

```
aggexec -d demo402 -D "MetaCube Demo" -E DET_Retailer
```

The following command, issued on a UNIX command line, builds all dimension element tables required for the aggregate table named **Sales Agg1** in the MetaCube Demo DSS System stored in a database named **demo402**:

```
aggexec -d demo402 -D "MetaCube Demo" -A "Sales Agg1"
```

The following command, issued on a Windows NT PC, builds all dimension element tables required for the **Channel** dimension. The user **metacube** logs on to the **demo40** database using the password **mcpasswd** via an ODBC Data Source named **demo402**.

```
aggexec -d demo402 -D "Demo 40" -U metacube -P mcpasswd -M Channel
```

Words that contain spaces must be surrounded by double quotes.

Before using **aggexec** from the command line to build DETs, you must have completed entering the metadata specifications for them in Warehouse Manager.

Important: *If the name of the dimension table and the name of the dimension element table are the same in the metadata, no change occurs. No dimension element table is created by* ***aggexec****.*

Using **aggexec** from the command line with the options for creating dimension element tables creates only dimension element tables; no aggregate tables are created.

Other Administrative Tasks

Although managing the job queue and submitting jobs are the primary tasks you will perform using Agent Administrator, you can also perform some other important administrative tasks:

- Define settings for MetaCube Scheduler

  Agent Administrator allows you to control MetaCube Scheduler by specifying the following:

  - The number of jobs that can run concurrently
  - The frequency with which Scheduler checks for changes to the queue
  - The mode of operation for Scheduler (normal, normal but accepting no new jobs, or stopped)
  - The amount of information recorded at the server when jobs are processed—refer to "MetaCube Agents Log Files" on page 6-15.

- Define parameters

  Agent Administrator allows you to define parameters, which are renamed SQL statements that return one value. Parameters allow users of MetaCube applications to include information in a filter that may vary from situation to situation. For example, a parameter might specify a period of time, such as weekly or monthly.

  Appendix A provides a complete description of parameters.

- Define system messages

  Agent Administrator allows you to define system messages, which are generally used to inform users about recent changes to the database. These messages are accessible through MetaCube Explorer, Web Explorer, MetaCube for Excel, and the MetaCube application programming interface.

MetaCube Agents Log Files

Using the MetaCube Agents Control dialog box, you can set the following logging levels for MetaCube Agents:

- No Logging—provides no log files
- Minimal Logging—provides a log of all error messages and some runtime functions
- Intermediate Logging—provides a log of all error messages and all runtime functions
- Full Logging—provides a log of all error messages, all SQL statements, and all runtime functions

Once you set the logging level you must restart MetaCube Scheduler to apply the change. A separate **.log** file is created for each MetaCube Agent process. The **.log** files are located in the MetaCube directory.

The following is an example of an error message displayed in a log file created with full logging:

```
1997 06 19 16 50 24 Error Code: 3. Message: ODBC[37000] -FM#:
19- DB Error: -201 - [INTERSOLV][ODBC Informix
driver][Informix]A syntax error has occurred.
```

The log text represents the following information:

- The numbers at the beginning of the message represent the date and time the error occurred. This error occurred on June 19th, 1997 at 16:50:24.
- `Error Code: 3` represents the MetaCube error. This is an ODBC error.
- `Message: ODBC[37000]` represents the type of ODBC error. This is a syntax error. Refer to ODBC documentation for more information.
- `FN#: 19` represents the ODBC function occurring at the time of the error. This error occurred during a Prepare Statement. Refer to ODBC documentation or Microsoft documentation for more information.
- `DB Error: -201` represents the database error that occurred. This is a syntax error. Refer to your database documentation for more information.

Tip: *If you are using an Informix database, use the* ***Find Error*** *utility, available with the Informix administrative tools, to find the database error.*

About MetaCube Web Publisher

MetaCube Web Publisher is a utility for creating Hypertext Mark-Up Language (HTML) files that hold the query results of a query stored in the database or a completed QueryBack job. These files can be viewed using a Web browser. This allows users who do not have MetaCube software installed on their PCs to view reports generated through queries to the data warehouse using a local intranet Web server. MetaCube Web Publisher can also create a table of contents HTML page containing links to all reports generated in HTML format. Thus, from a single page, users can easily locate and view query results.

Configuring Systems for Web Publisher

MetaCube Web Publisher can be executed in one of the following ways:

- As a job type, submitted from Agent Administrator on a Windows NT PC
- As a command-line utility, executed at the DOS prompt

When creating HTML-formatted MetaCube reports, the following MetaCube components are needed:

- MetaCube analysis engine
- MetaCube Web Publisher utility
- MetaCube Agent Administrator and MetaCube Agents

In addition, a Web server for your intranet must be installed on the network for users to access and view the HTML pages produced by Web Publisher.

The method used to create HTML pages depends on the platform on which the components listed above are installed.

Configurations Using MetaCube Agents for Windows NT

When the MetaCube components are all installed on a Windows NT platform, you can use MetaCube Agent Administrator to specify jobs to generate HTML reports.

Important: *All MetaCube components, including the MetaCube Agents for Windows NT, must be installed and running in the same directory on the Windows NT PC. If you used the defaults in the MetaCube installation programs, these components are all installed in the MetaCube directory.*

When you use the MetaCube Agent Administrator to submit Web Publisher jobs, MetaCube Scheduler and MetaCube Agents for Windows NT must be running for the job to complete. To verify that MetaCube Agents for Windows NT are functioning properly, view the **sc.log** file in the MetaCube directory.

When specifying the output file name for the Web Publisher job, you can place it directly into the directory used by the Web server by specifying the full pathname to that directory. If you do not specify a full pathname for the output, the HTML file is written into the MetaCube directory. If the Web server is running on a machine other than where MetaCube is installed, you may need to transfer the output file to the correct location so it can be read by the Web server.

To schedule a recurring Web Publisher job, use the Windows NT Scheduler to run the job periodically.

Configurations Using MetaCube Agents for UNIX

If the MetaCube Agents are running on a UNIX platform, you cannot use Agent Administrator to specify a Web Publisher job. Instead, use the command line utility, **metapubl**, from a DOS prompt or in a DOS batch file. When you use the **metapubl** command, the query is submitted directly to the database without using the MetaCube Agents. Refer to

If the Web server is running on a Windows NT platform, the output HTML file can be placed directly into the Web server's directory by indicating the full pathname for the file. If the Web server is running on the UNIX platform, the HTML output file must be made available by moving it to the appropriate location.

MetaCube Web Publisher Output Files

When you submit a query through MetaCube Web Publisher, the results are displayed as either a text-based report or a chart. When you request a report, Web Publisher formats returned query data into a single HTML file. MetaCube Web Publisher provides several chart formats for the display of query results:

- Bar
- 3D Bar
- Line
- Area
- Percent Bar
- Stacked Bar

Bar and 3D Bar charts are convenient for displaying the difference between two values in a given range or over a period of time. To create meaningful bar charts, measures should be formatted in columns (the default for measures) and attributes in row.

Line charts typically graph changes or variations over time. They also are used to visually track the relationship between any two variables. Separate lines represent each column of data in the query results. Attributes should be oriented by row and display along the horizontal *x*-axis, and measures should be in column orientation, displayed along the vertical *y*-axis.

Area charts are similar to line charts, except that they display multiple attributes combined into a single area. Attributes should be oriented by row for this chart.

Percent Bar charts display attribute data organized in columns. The percent bars are charted along the horizontal *x*-axis, and measures display along the vertical *y*-axis. To produce a percent bar chart, one attribute must be in row orientation, and at least one attribute must be in column orientation.

The Stacked Bar chart is useful for displaying results of queries that contain multiple measures, where the values are based on the same unit of measure: for example, dollars. To produce this chart, format attributes in row orientation.

Important: *All charts require at least one attribute to be formatted in row orientation. If there is no attribute in row orientation, Web Publisher does not produce a chart at all.*

When you request a chart as HTML output for the results of a query, Web Publisher creates two file types:

- An HTML file
- One or more GIF-formatted files for the actual chart graphic

Both file types are required to display the chart in a Web browser.

Linking HTML Pages

MetaCube Web Publisher contains a feature that allows you to link the HTML pages you generate. This feature is typically used to create a table of contents page that lists MetaCube query results that can be viewed using a Web browser. The table of contents page is produced at the same time that the report or chart is generated and contains:

- a link to the report or chart.
- the date and time the query that produced the HTML report or chart was executed.

A table of contents page might look like this:

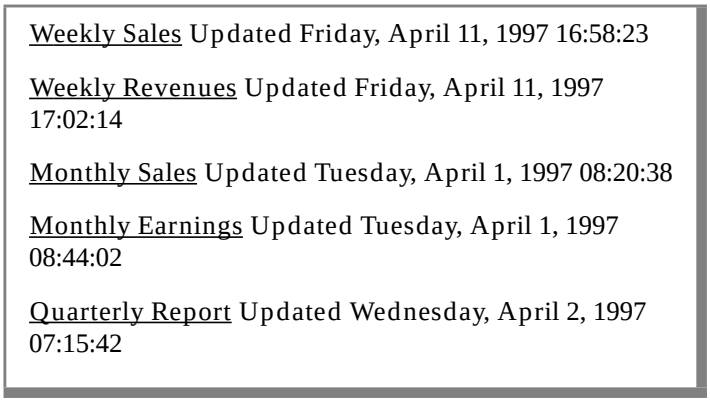


Figure 6-2 *Web Publisher Table of Contents Page*

Each underlined report title is the name of the saved query used to retrieve the data and is a link to the report or chart that displays the data.

When you specify a table of contents page, Web Publisher creates the table of contents page if it does not exist. To create a table of contents page as shown in Figure 6-2, you specify the same table of contents file name for several Web Publisher query jobs. When the specified table of contents file already exists, Web Publisher appends to it the link to the query results HTML page, thus allowing you to create the list of links that makes up the table of contents.

Important: *The table of contents file must reside in the same directory as the query output files.*

Another use of the Web Publisher linking feature is to link reports and charts together. To do this, you would generate the first HTML page, assigning it a name but not assigning a table of contents filename. Then you would generate a second HTML page, specifying as the table of contents filename the filename you assigned to the first HTML page. When Web Publisher generates the second report, it places a link to the second report into the first report. You can continue the process to link together as many reports or charts as you wish.

To create table of contents pages or link reports and charts together:

- For configurations using MetaCube Agents for Windows NT (using Agent Administrator), refer to "To specify optional header, footer, and table of contents files" on page 6-21.
- For configurations using MetaCube Agents for UNIX (using a DOS prompt), refer to the table on page 6-24.

Specifying a MetaCube Web Publisher Job

There are two ways to specify a Web Publisher job:

- Using MetaCube Agent Administrator
- Using the DOS command line utility syntax

Specifying a MetaCube Web Publisher Job Using Agent Administrator

To specify and submit a Web Publisher job, Agent Administrator must be configured and connected to the database where the data warehouse is stored. In addition, the MetaCube Agents for Windows NT must be running on the same Windows NT PC and be configured to connect to the same database. The software for Agent Administrator, MetaCube Agents for Windows NT, and the MetaCube analysis engine must all be installed in the same directory on the Windows NT PC. Refer to Agent Administrator online help for instructions on submitting a Web Publisher job.

You can specify a header and footer file to be automatically added the HTML page generated by MetaCube Web Publisher. The format of the file may be plain text or HTML.

To specify optional header, footer, and table of contents files

1. To include a header in the query output file, specify the name of the file that contains the header text. Clicking the **...** button to the right of the **Header File** box allows you to select the file by clicking its icon.
2. To include a footer in the query output file, specify the name of the file that contains the footer text. Clicking the **...** button to the right of the **Footer File** box allows you to select the file by clicking its icon.
3. To generate a link between the HTML query output page and either a table of contents page or another query output page, specify the name of the file to which the link will be added. Clicking the **...** button to the right of the **TOC File** box allows you to select the file by clicking its icon. If you are specifying a new table of contents page, type its name, using the file name extension **.htm**.
4. To complete the Web Publisher job, click the **Submit** button.

Specifying a MetaCube Web Publisher Job Using Agent Administrator on a Remote PC

Although you can specify a Web Publisher job on a PC other than the Windows NT platform where MetaCube Agents are also installed and running, you cannot submit the job through Agent Administrator. The MetaCube Web Publisher Job dialog box does not contain the **Depends**, **Schedule**, or **Submit** buttons.

Instead, the MetaCube Web Publisher Job dialog box contains a **View Command** button. After following the steps above to specify a Web Publisher job (with the exception of dependency and scheduling information), you can view the DOS command that has been created as a result of your specifications. You can then create HTML pages by issuing this command at the DOS prompt on your PC.

Specifying a MetaCube Web Publisher Job Using DOS

At the DOS prompt, you can issue a command that will cause MetaCube to generate HTML pages using the Web Publisher software that resides on a Windows NT PC. Executing the DOS command does not involve using MetaCube Agent Administrator or MetaCube Agents. It performs the query directly and returns HTML-formatted pages.

The following table lists the components of the command line call to the MetaCube Web Publisher: that is, the **metapubl** utility.

Component	Required/Optional	Description
metapubl	Required	The MetaCube Web Publisher utility; begins every DOS command.
/user=	Optional	User login ID; if not provided, MetaCube uses information on the most recently accessed database as contained in the **metacube.ini** file.
/pass=	Required	User password; used to log on to the database.
/conn=	Optional (not recommended)	The ODBC data source for connecting to the database; if not provided, MetaCube uses the most recently accessed database as contained in the **metacube.ini** file. If provided, and it differs from the entry in **metacube.ini**, the **metacube.ini** file is rewritten with the value specified here.
/dss=	Optional	The DSS System name; if not provided, MetaCube uses information on the most recently accessed database as contained in the **metacube.ini** file.
/schema=	Optional	Provides a metaschema prefix; defaults to `metacube.` (that is, *metacube* followed by a period).
/outp=	Required	Name of the HTML file to produce.
/head=	Optional	Name of the HTML or text file to add to the beginning of the HTML page generated. The filename must end with the extension **.htm**.
/foot=	Optional	Name of the HTML or text file to add to the end of the HTML page generated.
/query= or /jobid=	Required	Name of the query or the QueryBack job ID number that will produce the results to be formatted into an HTML page.

(1 of 2)

Component	Required/Optional	Description
/type=	Optional	Type of output to produce; defaults to report. The argument can equal one of the following: ■ report ■ bar ■ 3dbar (a three-dimensional bar chart) ■ Percent ■ line ■ area ■ stacked
/toc=	Optional	Name of the table of contents file to be created or appended or the file to which this HTML page should be linked.
/log[=]	Optional	Turns on logging, which tracks the progress of creating an HTML page using Web Publisher. The default name of the log file is **metapubl.log**. To use another name for the log file, use the optional = sign and provide the filename.
/debug	Optional	Turns on debug mode, which tracks detailed information on the progress of creating an HTML page using Web Publisher. The name of the log file is always **metapubl.log**.

(2 of 2)

The sample command below illustrates the use of the required components:

```
metapubl /pass=informix /outp=mo_sales.htm "/query=Monthly Sales"
```

If an argument contains any embedded spaces, such as "`/query=Monthly Sales`", the entire argument must be enclosed in double quotes. Optionally, any argument can be enclosed in double quotes.

To specify the full pathname for a query that is not located in the **ROOT** directory, use commas to separate subdirectory names, as in the following example:

```
"/query=root,myfolder,query_folder,query_name"
```

To put the output file in a location other than the current directory, type the full pathname, as in the following example:

```
"/outp=c:\root\myfolder\output_folder\output_filename"
```

Chapter

7

Warehouse Optimizer

In This Chapter

This chapter introduces MetaCube Warehouse Optimizer and describes the functions it performs.

For step-by-step instructions explaining how to use Warehouse Optimizer, refer to Warehouse Optimizer online help system.

About MetaCube Warehouse Optimizer

MetaCube Warehouse Optimizer is a graphical tool designed to help data warehouse administrators decide how best to use aggregation to improve query performance.

When processing queries, the MetaCube analysis engine calculates the most efficient path for retrieving data. Many queries request summary data. Normally, queries requesting such summary-level data require the database to perform the SUM function while processing the query. You can improve processing of these queries requesting summarized data by creating aggregate tables, which are special tables containing summaries of more detailed data. The MetaCube analysis engine automatically accesses aggregate tables if they exist and improves processing time.

In the typical data warehouse, where query performance is critical, you should create some aggregate tables. When developing aggregate tables, your objective is to create summary tables that provide the greatest performance benefit at an acceptable cost in terms of additional storage and maintenance. Data density and usage patterns are the primary factors in determining the optimal aggregates.

MetaCube Warehouse Optimizer can analyze data density and usage patterns and recommend aggregate tables. To perform this analysis, Warehouse Optimizer queries your data model to determine the actual density of data along different combinations of dimensions and incorporates into its analysis a weighted audit of users' queries. To help you make an informed decision about aggregate use, Warehouse Optimizer can display information on the aggregates it recommends. Once you choose aggregates, Warehouse Optimizer can create their metadata descriptions.

Over time, use of a data warehouse is the best indicator of which aggregate tables to create and maintain. Before the data warehouse is in use, however, you need to make some assumptions about the queries users will execute most frequently. Later, you can update your aggregates to reflect the needs of your users more accurately.

Optimizing aggregate tables can be time-consuming, but updates are rarely needed and once the optimization is complete, users can greatly improve their query performance.

Important: *Since Warehouse Optimizer is very processor and memory intensive, at any given time, only one instance of Warehouse Optimizer can run against the same database or connect to the same MetaCube analysis engine.*

For more information on aggregates, refer to "Aggregates" on page 5-24.

Before You Begin

Before you begin using Warehouse Optimizer, ensure its accurate and efficient performance by doing the following tasks.

Setting Fact Table Costs Using Warehouse Manager

Warehouse Optimizer bases its aggregate recommendations on query costs obtained from the MetaCube analysis engine and relative values Warehouse Optimizer assigns to possible aggregates. The MetaCube analysis engine obtains query costs from the fact table costs set in Warehouse Manager. Although fact table costs can be set to a relative value for other MetaCube products, for Warehouse Optimizer to produce accurate recommendations, a fact table's cost must be set to its actual row count.

Configuring Memory for Warehouse Optimizer

The aggregate recommendation stage of Warehouse Optimizer is a memory-intensive process. Warehouse Optimizer reads all possible combinations of dimension elements into memory. Each combination represents a potential user query. Warehouse Optimizer requires approximately 18 to 20 MB of memory, regardless of how many combinations it processes. Warehouse Optimizer also requires at least 10 KB of memory for storage. (Warehouse Optimizer stores a minimum of 100 combinations in memory.)

Each combination of dimension elements requires roughly 100 bytes of memory, varying slightly with the number of dimensions and combinations included in your analysis. Warehouse Optimizer calculates the number of combinations it can read into memory based on the amount of memory you allocate in the **metacube.ini** file. (Using the **metacube.ini** file to allocate memory is described in Appendix B.) If no value is set for the Warehouse Optimizer memory parameter in the `[General]` section, Warehouse Optimizer reads 100,000 combinations into memory at a time by default.

If you do not allocate enough memory for all combinations to be read into memory at once, Warehouse Optimizer creates a cache file, called **_whocach_cc.mwo**. This file is located in the same directory as the Warehouse Optimizer analysis file and uses only 40 bytes of disk space per combination. Warehouse Optimizer swaps combinations into memory from this cache file. For example, if you allocate 10 KB of memory, and you are processing 1000 combinations, approximately 100 combinations are swapped into memory at a time, while the rest are stored in the cache file until needed. Once the recommendation stage is complete, Warehouse Optimizer cleans up and removes the cache file. If the application terminates abnormally, the cache file is cleaned the next time you initiate the recommendation stage. With less memory swapping, Warehouse Optimizer recommends aggregates faster. When configuring memory, consider the trade-offs among the following factors:

- Memory swapping directly affects processing time—the more memory swapping necessary, the longer processing takes.
- The size of your analysis—if you have a small amount of data to analyze, you can allocate less memory without affecting processing time by memory swapping.
- The memory available on your computer—this may determine how much memory you can allocate for Warehouse Optimizer.

Creating a Sample Table

You may want to create a sample table on which to run your analysis, thereby reducing query time and speeding up computation of storage costs. If the number of rows in your fact table is in the millions, you should use a sample table. To create a sample table, first use MetaCube Warehouse Manager to create a metadata description for the sample table, then use MetaCube Agent Administrator to submit a Sample job. Informix recommends that any sample table you create for purposes of aggregate recommendation should be at least twice as large as your largest dimension table.

"Sample Tables" on page 5-35 describes sample tables, and Warehouse Manager online help gives step-by-step instructions on how to create metadata for sample tables. Chapter 6, "Managing MetaCube Agents," describes the use of MetaCube Agent Administrator, and online help for Agent Administrator describes how to submit a Sample ob.

The Warehouse Optimizer User Interface

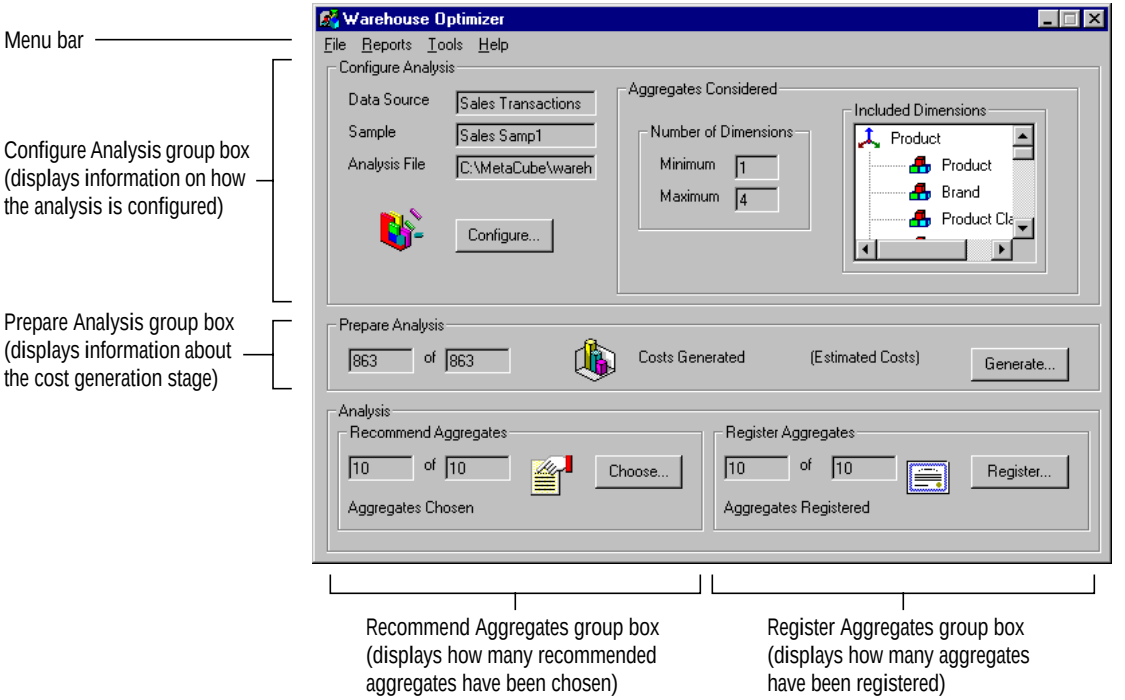
After connecting to the database, using Warehouse Optimizer consists of four main stages:

- Configuring the analysis
- Calculating the storage and query cost for combinations of dimension elements
- Recommending aggregates
- Registering aggregates (only users authorized in MetaCube Secure Warehouse can perform this stage)

You may perform the entire process at one time, or you may perform different stages at different times. Warehouse Optimizer stores all the results of its analysis in a local analysis file, which you can reopen at any time to continue, complete, or change the analysis.

You must complete one stage of analysis before moving on to the next, except when recommending aggregates. In that case you may move on to registering aggregates after only one aggregate has been recommended.

Figure 7-1
Warehouse Optimizer Main Window



Configuring the Analysis

There are several considerations when configuring your analysis:

- Choosing a data source
- Selecting the analysis type
- Setting the number of dimensions in aggregates (applies to partial analysis only)

Choosing a Data Source

You can run Warehouse Optimizer against a fact table or a sample table, if one exists. Running your analysis against a sample table can decrease processing time without significant impact on the accuracy of the results. For more information on using sample tables in Warehouse Optimizer refer to "Creating a Sample Table" on page 7-6.

Selecting the Analysis Type

MetaCube Warehouse Optimizer can identify candidates for aggregation by operating in two modes:

- **Full analysis** considers all possible combinations of the dimension elements that are referenced in the fact table's metadata.
- **Partial analysis** allows you to select specific dimension elements to exclude from consideration and to choose the maximum and minimum number of dimensions to include in each candidate aggregate.

Typically, you should run a full analysis. Partial analysis may be useful when a fact table contains very detailed data for a given dimension element that users rarely query at that level. In such a case, you might simplify the aggregate analysis by excluding these seldom-used dimension elements.

You might also use partial analysis when you know users query only at a particular level or set of levels. For example, if users often query by the dimension element *Day* along with any of the *Product* dimension elements, you might restrict analysis to only combinations that include these elements. These combinations are the only ones considered as potential aggregate combinations, known as *candidate aggregates*.

Setting the Number of Dimensions in Candidate Aggregates

When performing a partial analysis, Informix suggests you also set the maximum and minimum number of dimensions included in each candidate aggregate. Aggregates with too many dimensions are frequently too large to provide a performance benefit, and aggregates with too few dimensions are rarely useful.

When you eliminate dimensions or dimension elements for a partial analysis, they will not be considered in candidate aggregates (saving time and resources during the storage cost calculation and aggregate recommendation stages). However, this does not reduce the number of combinations that Warehouse Optimizer generates during analysis. Although dimension elements can be eliminated from consideration in candidate aggregates, their elimination does not exclude them from being included in queries a user might submit. To calculate the average cost of a user query against the fact table and any existing aggregate tables, Warehouse Optimizer generates *all* possible query combinations and their associated query costs. Then Warehouse Optimizer compares the average query cost for the existing data warehouse with the average query cost for the data warehouse assuming that the candidate aggregates are created. Based on this comparison, Warehouse Optimizer recommends aggregates that yield the lowest average query cost.

If your DSS System consists of a large number of dimensions and dimension elements that result in more than 500,000 combinations, Informix recommends that you create another DSS System consisting of a subset of the original DSS System and run Warehouse Optimizer on the subset. The subset DSS System should include the dimensions and dimension elements that are accessed frequently in user queries.

Once the analysis has been configured, you need not rerun this stage unless you want to change the configuration or the metadata in your database has changed.

Generating Costs

Tip: *To calculate the combination totals referred to in this section, giving you an idea of storage cost processing time, refer to* *"How to Calculate Combination Totals" on page 7-13*.

In this stage of analysis, Warehouse Optimizer determines the following factors:

- The cost of each possible query against the existing fact table and any existing aggregate tables
- The cost of storing each candidate aggregate table

Reducing Cost Generation Time

Although generating costs can be a lengthy process, you can use any of the following strategies to reduce computation time:

- Estimate costs (discussed below)
- Run a partial analysis, eliminating some dimensions or dimension elements to reduce storage cost computation time
- Run your analysis against a sample table, reducing the number of rows to process in your queries and therefore reducing query time

Should You Estimate Costs?

If you choose to estimate storage costs, Warehouse Optimizer:

- submits queries only for the candidate aggregates made of base combinations.
- estimates storage costs for all remaining combinations.

If you have more than 200 base combinations and a fact table with more than 100,000 rows, generating actual costs for the base combinations can take a day or more. However, Warehouse Optimizer estimates the remaining storage costs at a rate of 100,000 combinations per hour.

If you do not estimate storage costs, Warehouse Optimizer submits queries for all candidate aggregate combinations. The time needed for cost computation varies, depending on the load on the database server, the size of the dimension tables, and the computational speed and power of your system. However, to give you an idea of the time involved, if the fact table is large (for example, over 100,000 rows) and the number of combinations is also large (more than 10,000), the cost-computation stage can take days.

Informix recommends that you estimate costs in the following situations:

- When using a sample table—estimated storage costs for combinations with more than two elements are generally more accurate than those costs computed by submitting queries against the sample tables
- If your analysis includes more than 10,000 combinations

- If you use Version 8.21 UC1 of Informix Dynamic Server with Advanced Decision Support and Extended Parallel Option, and you use Informix-CLI Version 2.5 TD2.

How Warehouse Optimizer Generates Costs

Eliminating combinations in partial analysis removes them from consideration as possible aggregates but does not eliminate them from being possible queries that a user might submit. Consequently, Warehouse Optimizer computes the query cost for all combinations but computes the storage cost for candidate aggregates only.

Warehouse Optimizer quickly computes query cost, determining whether the query can be executed against an existing aggregate or the original fact table. The cost is the total number of rows processed for the query.

The time needed to compute storage costs can vary greatly depending on the number of combinations of candidate aggregates for which Warehouse Optimizer computes *actual* storage costs. To determine the actual storage cost of candidate aggregates, Warehouse Optimizer submits a GROUP BY query corresponding to each candidate aggregate table. Warehouse Optimizer creates a view, called **tempview**, to store the results while counting the number of rows returned.

Because the computation of storage costs can be a lengthy process and Warehouse Optimizer must remain connected to the database throughout, failures can occur during this stage. During processing, any failure in executing a query to the database returns the following error:

```
Runtime Error "3146" ODBC call failed
```

Listed below are some possible causes for this error:

- Improper specification of the metadata might cause references to invalid or nonexistent tables in the SQL statement generated by the MetaCube analysis engine in creating the aggregate views. Use MetaCube Warehouse Manager to verify the accuracy of the metadata for the fact table, dimensions, and sample table for the data source you are using.
- Connection to the database server failed. If this occurs, rerun cost generation. Warehouse Optimizer continues running where it left off when the error occurred.

- The computer where the database server is located was down or inaccessible for any reason.
- More than one Warehouse Optimizer was running against the same data source. Problems that might occur include:
  - Warehouse Optimizer creates each view with the same name, **tempview**, for each storage cost computation. Therefore, another Warehouse Optimizer running on the same data source cannot create this view.
  - One Warehouse Optimizer process might drop the tempview created by a second Warehouse Optimizer process—the first process creates the view and the second process deletes the view.
- If an earlier Warehouse Optimizer process failed before dropping the view it created, orphan processes maintaining a lock on the view may be left on the database server. Consequently, an attempt to drop the view fails. Using the operating system, kill any leftover processes before starting the cost-generation stage.
- Insufficient temporary space in the database to run the query associated with creating each view.
- Insufficient shared memory configured on the system running the database server.

  For more information on configuring shared memory, refer to your database's administration documentation.
- Incorrect permissions to create or drop a view or to execute a query against the selected data source. Make sure that the Warehouse Optimizer user has owner permissions.

Tip: *If Warehouse Optimizer seems to be taking too long to process a single aggregate combination on fact and dimension tables that are not large, you can stop and restart. Warehouse Optimizer resumes analysis where it left off. Lengthy processing might be the result of a query waiting for resources to be released in the database. However, if dimension tables are large (even though the fact table is small), the joins required to run the query might take some time to complete.*

Once costs are generated, you need not rerun this stage unless you change the analysis configuration or the data density of your database has changed. If you change the analysis configuration, Warehouse Optimizer automatically updates storage costs. You can rerun Warehouse Optimizer to regenerate aggregate recommendations at any time, such as after MetaCube has collected user query audit data.

How to Calculate Combination Totals

There are two combination totals referred to in this chapter—the number of *base combinations*—all combinations consisting of one or two elements—and the total number of combinations. These numbers directly affect the processing time for an analysis. As you read this chapter, you may want to use the following equations to calculate the combination totals, giving you an idea of processing time and helping you decide on your analysis strategy.

To calculate the number of base combinations

1. Add the total number of dimension elements in each dimension together.

   Use the following description of a DSS System as an example:

   - Dimension A has 4 dimension elements.
   - Dimension B has 2 dimension elements.
   - Dimension C has 3 dimension elements.

   Using this example you get the following calculation for step 1:

   4 + 2 + 3 = 9

2. Multiply the total number of dimension elements contained in a dimension by the total number of dimension elements contained in any other dimension. Do this for all unique two-dimension combinations and add the totals together.

   The calculation based on the above sample would be:

   (4 x 2) + (4 x 3) + (2 x 3) = 26

3. Add the total from step 1 and the total from step 2:

   9 + 26 = 35

   In the example DSS System there are 35 base combinations.

To calculate the total number of combinations in a DSS System

1. Add one to the total number of dimension elements in each dimension.

   Using the same example DSS System above, you get:

   4 + 1 = 5

   2 + 1 = 3

   3 + 1 = 4

2. Multiply these totals together.

   Using the totals from step 1:

   5 x 3 x 4 = 60

3. Subtract 1 from the total.

   Using the total from step 2:

   60 - 1 = 59

   In the example DSS System there are a total of 59 combinations.

Recommending Aggregates

Recommending aggregates consists of the following tasks:

- Analyzing query audit data, if you choose to use this information
- Reading the combinations generated in previous stages into memory. If not enough memory is configured to hold all combinations, then the combinations are first read onto disk, then swapped into memory a portion at a time.
- Choosing the best aggregates

You must decide two things for aggregate recommendation:

- Whether to use audit data
- How many aggregates to recommend

Audit Data

Audit data is information about user queries, such as which data is requested, who is requesting it, how long the query takes to process, how many rows are retrieved, and other similar information. You can view query audit data in a columnar text format by running the **Query Audit Detail** report. This data, if available, can be very useful in recommending aggregates that are specific to the needs of your users.

Important: *Using MetaCube Secure Warehouse, you must specify for each user whether you want MetaCube to collect audit data.*

If you specify that you want to use audit data, you must indicate the weight that audit data has in the analysis process. The weight you enter reflects your confidence in the audit data. Audit weight ranges from one to ten, with each number representing a percentage value. For example, setting audit weight to 3 means that you think the audit data accounts for 30% of users' queries. If the audit data accurately reflects full use of the data warehouse by all users, you should specify 10, for 100% weighting of the audit data.

How Many Aggregates to Recommend

The main concern in deciding how many aggregates you want Warehouse Optimizer to recommend is time. The time required to analyze audit data is proportional to the amount of audit information available. As a guideline, assume that audit data containing 2,000 items for 6,000 combinations takes about ten minutes.

The time required for reading combinations into memory or the cache file is approximately five to ten minutes for one million combinations. Swapping combinations from the cache file into memory will slow the next stage, choosing aggregates.

The time required to choose aggregates is proportional to (mn^2) where:

- m is the number of aggregates to recommend.
- n is the number of combinations included as candidate aggregates.

Recommending 10 aggregates from 50,000 combinations takes about 14 hours of processing time if you have configured enough memory for all combinations to be read directly into the memory.

Viewing Recommended Aggregate Details

You can view detailed information on the recommended aggregates to help you choose the aggregates you want to register. You can view this information in the following forms:

- Aggregate Recommendations Report
- Cost Graph
- Aggregate Information in the Register Aggregates dialog box

Register the aggregates that will give you the most improvement in query performance with the least storage cost. In other words, register the aggregates up to the point of diminishing returns.

Aggregate Recommendations Report

After Warehouse Optimizer completes its recommendations, you can view a report that contains aggregate recommendations. The report lists the following information about each recommended aggregate.

Item	Meaning
Rank	The rank of the aggregate table in order of overall query cost improvement
Agg Size (Rows)	The number of rows in the aggregate table
Avg Query Cost	The average cost to run a query if the aggregate table and all higher ranking recommended aggregate tables are created
Dimension Elements	The dimension elements included in the aggregate table

Cost Graph

The Cost Graph displays the aggregate recommendation data in chart format. The graph shows:

- the average cost to run a query.
- the cumulative storage costs (in number of rows).

The Cost Graph is useful for presenting query cost and storage cost information, and quickly finding the point of diminishing returns using a graphical representation. Using the chart and the report generated by Warehouse Optimizer, you can determine the proper balance of storage required for a set of recommended aggregates and the improvement in query performance that the aggregates can provide for your data warehouse.

Registering Aggregates

Once you have found the point of diminishing returns, you can decide how many aggregates to register and change the names of the aggregates to be more meaningful, if desired.

Only users who have been granted permission to the data warehouse administrative tools in MetaCube Secure Warehouse (secure users, in other words) can register aggregates. If you are not a secure user, save the analysis file and contact a secure user, who can register aggregates using Warehouse Optimizer or MetaCube Warehouse Manager.

In the registering aggregates stage, use the information provided about recommended aggregates to help you decide which of the aggregates you want to register in metadata. Pay close attention to the query and storage costs displayed in the Aggregate Information group box. The query cost of each aggregate is the average cost of a query, assuming that the selected aggregate and all higher ranking aggregates are built. The storage cost is the cumulative cost of storing the selected aggregate and all higher ranking aggregates.

When registering aggregates, you can specify how many of the highest ranking recommended aggregates you want to register. You cannot randomly select aggregates from a list of recommended aggregates, since the query and storage costs listed for each aggregate are based on the assumption that all higher ranking aggregates are built. For example, if Warehouse Optimizer recommends five aggregates, you can choose to register two, and Warehouse Optimizer will register the top two. You cannot pick particular aggregates from that list of five and register them.

Registering recommended aggregates does not take long, because the only processing necessary is the creation of metadata for the aggregate.

Once you register aggregates, Warehouse Optimizer indicates which aggregates have been registered in the current session. You may reselect aggregates from the current recommendations at any time. If you choose to register more aggregates than were previously registered, only those aggregates that have not been registered before will be registered. If you want to remove registered aggregates from the metadata, use Warehouse Manager.

Once you register the aggregates in metadata, you must use MetaCube Agent Administrator to run the jobs that create aggregate tables.

Verifying Optimum Performance of Existing Aggregate Tables

If you are running Warehouse Optimizer to verify that your existing aggregate tables produce the greatest performance benefits, you must uncheck the **Valid** check box using MetaCube Warehouse Manager for each aggregate. If you do not do this, Warehouse Optimizer recommends new aggregates based on how the data warehouse performs when it uses the existing aggregates.

Once you have removed the **Valid** mark from all aggregate tables, complete all stages of Warehouse Optimizer. Then, using Warehouse Manager, compare the metadata for the aggregates by Warehouse Optimizer to the metadata for your existing aggregate tables. If you Warehouse Optimizer recommended aggregates that did not already exist and that you want to use, create the physical aggregate tables using Agent Administrator. Delete the metadata for the aggregate tables you do not want.

Optimizer Performance Log

Warehouse Optimizer can create a performance log, which you can use to view information on the number of combinations processed and the processing time necessary for the cost generation and aggregate recommendation stages. The performance log, called **who.log**, is located in the MetaCube installation directory.

The performance log tracks the following information:

Stage	Data Tracked
Generating costs	If estimating costs: ■ Number of combinations for which actual costs were generated—number of base combinations ■ Number of combinations for which estimated costs were generated ■ Amount of time to generate actual costs ■ Amount of time to generate estimated costs
	If not estimating costs: ■ Number of combinations for which costs were generated ■ Amount of time needed to generate costs
Recommending aggregates	■ Total number of combinations processed ■ Number of combinations stored in memory at a time ■ Start and finish time for each recommendation

Important: *You can turn on and off performance logging in the* ***metacube.ini*** *file, as described in Appendix B. For step-by-step instructions, refer to Warehouse Optimizer online help.*

Updating Recommendations

You will want to update the recommendations made by Warehouse Optimizer from time to time. For instance, after collecting user query audit data, rerun Warehouse Optimizer to update the recommendations based on the data stored in the audit tables. You can update recommendations without reconfiguring the analysis or regenerating costs, unless you select a different data source or new aggregate tables have been created. Updating your aggregate table recommendations with Warehouse Optimizer is an important step in keeping your data warehouse running as efficiently as possible.

Appendix

A

Understanding Parameters

A parameter is defined by a name and an SQL statement. When a parameter is encountered in a query specification, the SQL statement is executed to return a result of a single value. That value is then substituted for the parameter before the query itself is executed.

Parameters are used in filters applied to queries. For example, using the MetaCube demonstration database, a filter might be defined as follows:

```
City In <<Enter a city>>
```

When applied to a query, this filter:

- causes the Query Parameters dialog box to display, so that the user can type in the name of the city on which to filter.
- limits data returned from the data warehouse to only data for the city entered by the user in the Query Parameters dialog box.

Suppose, for example, the user types `San Francisco` in the Query Parameters dialog box. In that case, the <<Enter a City>> parameter causes the following SQL to be generated by the MetaCube analysis engine:

```
WHERE metademo.geography_dim.city In ('San
Francisco')
```

This WHERE statement limits the data returned by the query to data for San Francisco only.

The relative time filters provided by MetaCube Explorer, Web Explorer, and MetaCube for Excel are actually parameter names. When a MetaCube relative time filter is applied to a query, the MetaCube analysis engine generates a WHERE clause that returns a code for the time period defined by the filter. For example, in the MetaCube demonstration database, the **Current Week** time filter, is defined as:

```
Fiscal Week In (<<Current Period>>)
```

When applied to a query that retrieves data from the **Sales Transactions** fact table, the query generates the following WHERE clause:

```
WHERE metademo.sales_data.fiscal_week_number In (65)
```

The number *65* is the result of executing the Fiscal Week parameter's SQL statement; that statement returns the code that indicates, in the time dimension table, the row that contains the date of the current fiscal week. When the query, constrained by the Fiscal Week relative time filter, executes, only data for the current fiscal week is returned. The user is not prompted for relative time parameters.

Parameters in QueryBack Jobs

It is common for jobs to be submitted well before they are actually executed by MetaCube Scheduler. This happens most frequently when a job is set to recur automatically. For example, a job may be submitted in January that will execute twelve times throughout the year. In such cases, any parameter values in the query must be dynamically substituted at runtime.

For example, suppose a MetaCube user submits a QueryBack job that generates a sales report every week. The query may contain a relative time filter that defines the current week. To produce the report, the MetaCube QueryBack agent must look up the current week information, adjust the SQL required to generate the report, and then perform the query. A parameter is the query definition's dynamic component, changing each time the QueryBack process executes the query.

When submitting queries to MetaCube QueryBack, users can:

- specify a single future date and time at which QueryBack should execute the query.
- request that QueryBack process the query at recurring intervals in the future, such as every day or every month.

When submitting a relative time query at a future date or at recurring intervals, QueryBack must be able to update the query to retrieve the most recent data.

To support this QueryBack functionality, you must, using MetaCube Agent Administrator, specify the definition of a parameter for every dimension element of the *Time* dimension. This allows MetaCube QueryBack to determine the most recent time period for that dimension element when the query is run.

The definition of a parameter is an SQL statement that returns one and only one value. In the case of the relative time parameter, the value returned is a code that flags the current (or most recent) time period relevant to the query. For example, in the MetaCube demonstration database, the SQL statement that returns the current fiscal week code is:

```
SELECT fiscal_week_number
FROM metademo.time_dimension
WHERE current_period = "Y"
```

The result of executing this SQL statement is the return of the code value 65, which, in the demonstration database, flags the most recent (or current) week in the database. Refer to "Relative Time" on page 2-27 for a discussion of relative time and the current period flag.

If you examine the time dimension table for the MetaCube demonstration database, you will notice that only one row is flagged in the **current_period** column with a Y; the dates in that row are the most current in the database. MetaCube QueryBack substitutes the relevant date for the current query and executes it, retrieving data for that time period.

The general syntax for SQL statements to retrieve the most recent time periods is:

```
SELECT dimension_element_column
FROM time_dimension_table
WHERE current_period_column = "Y"
```

Using this statement as a model, you can create the required SQL statements for every time dimension element in your own data warehouse.

User-Defined Parameters

You may use MetaCube Agent Administrator to predefine parameters that Explorer, Web Explorer, or MetaCube for Excel users can then incorporate into parameterized filters in the same way that they incorporate relative time filters. Predefining parameters for users:

- makes it easier for them to create their own parameterized filters.
- assures that queries containing parameters can be submitted to QueryBack and execute successfully.

As an illustration, you might define a parameter named *First Brand*. The following SQL executed for this parameter, retrieves data only for the first brand listed in the **Brands** table:

```
SELECT brand_name FROM brands WHERE brand_code IN (SELECT
MIN(brand_code) from brands)
```

An Explorer or MetaCube for Excel user could define a filter, using this parameter:

```
Brand = <<First Brand>>
```

This usage of a user-defined parameter produces no Query Parameters dialog box for user input when the query is executed. However, the query will retrieve only information for the single brand—the first brand in the **Brands** table. In addition, because the parameter is defined through Agent Administrator, any query using this parameter will execute successfully using QueryBack.

The parameters you define using MetaCube Agent Administrator can be used for a wide variety of tasks to ultimately retrieve the value used for filtering a MetaCube query. Preparing useful parameters for MetaCube users can increase their ability to retrieve meaningful reports from the data warehouse.

Parameters in SQL Jobs

MetaCube supports parameters in SQL jobs as well as QueryBack jobs. To submit an SQL job that is dynamically altered at runtime, include the parameter name enclosed in double angle brackets

```
<< >>
```

For example, suppose you wanted to run an UPDATE statement every week. The UPDATE statement changes the label on a report to reflect the name of the current week, as the week is defined in the MetaCube system. Assuming that **fiscal_week** is a parameter name, the SQL statement can then be submitted with the embedded parameter for fiscal week:

```
UPDATE report_labels SET btitle =
   SELECT week_description FROM time_dimension
   WHERE week_code = <<fiscal_week>>
```

When the SQL statement executes, a numeric value is substituted for <<fiscal_week>>.

Appendix

B

The metacube.ini File

The **metacube.ini** initialization file, located in the Windows directory of any PC running MetaCube applications except MetaCube Web Explorer, stores some MetaCube preferences and configuration information. You can modify these values by editing the file using a text editor such as Windows NotePad.

The following table explains the entries in the **metacube.ini** file. The parameters listed in **metacube.ini** are a subset of all MetaCube preferences. Most preferences in MetaCube applications are set within the application software, and they should not be modified using **metacube.ini**.

The **metacube.ini** file is divided into sections, with each section heading denoted by square brackets, such as [Engine] or [Agents]. Within each section, parameters can appear in any order. Parameter names are not case sensitive.

[Section] Parameter	Description
[General]	
Configuration	The currently active configuration by which MetaCube connects to the Informix database.
PassThroughDriver	The full pathname to the database's ODBC driver; used by SQL Optimizer.
OptimizerMemory	Allocates memory (in kilobytes) for use during the aggregate recommendation stage of MetaCube Warehouse Optimizer.
LogOptimizerPerf	Indicates whether the performance log for MetaCube Warehouse Optimizer is turned off or on (0=off; 1=on).
InstallDir	The directory in which MetaCube software is installed.
Config1 Config2 Config3 etc.	Other saved configurations. All saved configuration names appear as a choice in the Preferences dialog box for all MetaCube applications.
[*config_name*]	Each configuration listed in the [General] section has its own section of the configuration file. The section name is the configuration name.
IncrSchema	This value is provided when you configure the MetaCube Agent Administrator. Indicates the prefix to be used when MetaCube Aggregator creates incremental aggregate files.
Login	The last user ID used to login.
MetamodelSchema	Identifies the owner/schema of the metadata tables.
DSSSystemName	The name of the DSS System accessed at login.
ConnectDatabase	The database vendor (1=Oracle;2=MS Access; 5=Informix database).

(1 of 3)

ConnectString	The name of the ODBC Data Source.
PDQPriority	Informix databases only. When PDQ Priority is enabled, the number indicating the priority for PDQ queries.
DataSkip	Informix databases only. Indicates whether to use the Informix Data Skip feature for queries. (0=do not use Data Skip; 1=use Data Skip; 2=use Informix server default)
[Engine]	
EnabledExtensions	Names of all Explorer snap-in modules currently enabled.
DisabledExtensions	Names of snap-in modules previously enabled, now disabled.
[Agents]	
CenterTo	Indicates whether dialog boxes center in relation to the monitor or the Agent Administrator main window.
AutoUpdateOnSubmit	Indicates that Agent Administrator updates the job queue automatically when a new job is submitted.
DependOn	Indicates that the Dependencies dialog box displays all jobs or only jobs that are pending (`AllJob` or `Pending`).
WaitTime	Indicates the maximum time to wait when a lock occurs. Default value is 300 seconds.
[Excel]	
UseSampling	Indicates that Sampling is being used.
Accuracy	A numeric value between 1 and 100 indicating the sample table that the MetaCube analysis engine should access to process a query. A higher accuracy value indicates a larger sample table should be accessed if one is available.

(2 of 3)

SlowQueryWarning	Indicates the threshold value above which the Slow Query Warning is displayed for queries.
MaxTotalFetches	The maximum number of rows that MetaCube will retrieve for a single SQL statement.
[Fonts]	
Name	Name of screen font for variable text in Agent Administrator.
Size	Size of the screen font for variable text in Agent Administrator.
Bold	Indicates whether variable text in Agent Administrator is bold (`False`=not bold; `True`=bold).
Italic	Indicates whether variable text in Agent Administrator is italic (`False`=not italic; `True`=italic).
Underline	Indicates whether variable text in Agent Administrator is underlined (`False`=no underline; `True`=underline).

(3 of 3)

Appendix

C

The mcrexec Utility

Mcrexec is a utility that enables a Windows 95, Windows 98 or Windows NT user to remotely run UNIX OS commands on any UNIX server in the network. Use this utility when the MetaCube Agents are on a Windows NT machine running against a UNIX database server.

Mcrexec logs in to the UNIX server with the username and password you supply, then passes the command string to the server.

To run mcrexec, use the DOS prompt to go to the directory containing **mcrexec.exe**. If you used the defaults during installation, the directory is **MetaCube**.

Enter the following command:

```
mcrexec -host hostname -u username -pwd password
    -port port -c command -h -epc
```

Where the command line flags are defined as follows:

- -host defines the remote host on which the UNIX command is to be run.
- -u defines the username that is to be used to log onto the remote host.
- -pwd defines the password for the username.
- -port is an optional argument that can be used to specify the port number on the remote host to which this utility should connect itself.

  By default MCrexec connects to port 512.
- -c defines the command to be run on the remote machine.

- -h prints out the usage for MCrexec.
- -epc is used when mcrexec is called through the Scheduler. For the argument following this flag, enter the encrypted password and command strings.

  Mcrexec internally decodes the string and assigns proper values to the corresponding arguments.

Appendix

D

Setting Configuration Parameters

This appendix explains the options available for configuring a database connection.

Use the Preferences feature to set default values for:

- ODBC data source connection information.
- Informix database information.

MetaCube saves preferences in the metacube.ini file, stored in your Windows directory. This file also stores connection information and controls features that do not appear as fields in the **Preferences** dialog box. For example, the metacube.ini file stores a default login name, so that after you have connected once to a particular data source, you need not enter these values at the login dialog.

Configurations Tab

This tab contains information about connecting from MetaCube applications to the database that houses the data warehouse. The box on the left side of the tab displays the names of existing connection configurations. Clicking a configuration name displays its parameters in the fields on the right.

Connection configurations may be created in any MetaCube application and they are not unique to that application. All named connection configurations, regardless of the application currently in use, appear in the **Configurations** tab and can be used for connecting to a database. Likewise, you may make changes to an existing configuration from any of the MetaCube applications.

The fields in the **Configurations** tab are:

- **Name:** Use to name this set of configuration parameters; it can be any name, although matching this name with the name of the ODBC data source to which you are connecting may provide logical, meaningful names for configurations.
- **Schema:** Use to specify the schema / owner of the MetaCube metadata tables in the database. For an Informix database, which supports table owners, this value indicates the name of the user who owns the metadata tables, followed by a dot (.). In ANSI-standard SQL, the dot separates the name of the table owner from the table name.

  For databases that do not support table owners, such as Microsoft Access, this value is a prefix common to all the MetaCube metadata table names. The prefix is not followed by a dot. Depending on your company's naming conventions, this value may end with an underscore character. In the MetaCube demonstration database, a Microsoft Access database, the MetaCube schema owns all metadata tables and the value for this field is "metacube_".
- **DSS System:** Use, optionally, to indicate the name of the DSS System to access upon connection with the database. To connect to the MetaCube demonstration database, enter "MetaCube Demo" in this field. Use the **...** button to display the **Choose DSS System** dialog box that lists all available DSS Systems. This field may be left blank.
- **Database Type:** This value should be either Informix Online or Access, the two database types to which MetaCube connects. The database type for the MetaCube demonstration database is Microsoft Access. Use the arrow button to display the available database types.
- **Data Source:** Use to specify the name of the ODBC User Data Source defined using the ODBC administrator. If the name here and the ODBC User Data Source name do not match, no connection can be made. The ODBC data source name is case sensitive. Use the arrow button to display names of all existing ODBC User Data Source configurations. For information on configuring ODBC User Data Sources, refer to the *MetaCube Installation and Configuration Guide*.

- **DSS Cache File:** Obsolete; do not use.

  When a MetaCube application is opened, the MetaCube analysis engine compares the name and date of the local storage file with the name of the file specified in the configuration and the date of the latest update to the data warehouse. If there is no mismatch, the local storage file is used; if there is a mismatch for either value, the MetaCube analysis engine refreshes the local storage file. If you frequently use two or more configurations, having uniquely-named local storage files for each configuration avoids the process of refreshing data each time you switch configurations.

- **Database Parameters:** If you select Informix Dynamic Server as the Database Type, the following configuration options become available.

  - **PDQ Priority**: The default value for this parameter is set by your data warehouse administrator and, if you have been given permission, you may change the setting. This parameter sets queries submitted by Warehouse Optimizer as PDQ queries. The Informix PDQ (Parallel Database Query) feature allows the database to manage resources when processing both OLTP and decision support transactions. The default value for PDQ Priority is -1; this specifies to use the database server default for processing Warehouse Optimizer queries. If you enable PDQ processing, you can set the priority value to a value between 1 (LOW) and 100 (HIGH). (If the number is zero (0), PDQ processing is OFF.) For information on PDQ query processing, refer to the Informix database server documentation.

  - **Data Skip**: The default value for this parameter is set by your data warehouse administrator and you may change it, if you wish. This parameter sets the Informix Data Skip feature that allows a query to complete even when data stored on unavailable fragments cannot be retrieved. The effect of turning on Data Skip is to allow all SELECT statements to complete; all unavailable fragments are skipped when a query is processed. However, operations that write to the database do not complete.

    For information on data skipping, refer to the Informix database server documentation.

Appendix

E

Notices

IBM may not offer the products, services, or features discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

> © (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. (enter the year or years). All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

AIX; DB2; DB2 Universal Database; Distributed Relational Database Architecture; NUMA-Q; OS/2, OS/390, and OS/400; IBM Informix®; C-ISAM®; Foundation.2000™; IBM Informix® 4GL; IBM Informix® DataBlade® Module; Client SDK™; Cloudscape™; Cloudsync™; IBM Informix® Connect; IBM Informix® Driver for JDBC; Dynamic Connect™; IBM Informix® Dynamic Scalable Architecture™ (DSA); IBM Informix® Dynamic Server™; IBM Informix® Enterprise Gateway Manager (Enterprise Gateway Manager); IBM Informix® Extended Parallel Server™; i.Financial Services™; J/Foundation™; MaxConnect™; Object Translator™; Red Brick Decision Server™; IBM Informix® SE; IBM Informix® SQL; InformiXML™; RedBack®; SystemBuilder™; U2™; UniData®; UniVerse®; wintegrate® are trademarks or registered trademarks of International Business Machines Corporation.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Windows, Windows NT, Excel, Visual C++, Visual Basic, Visual Studio and Visual Basic for Applications are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Tidestone™ is a trademark and Formula One® is a licensed trademark of Tidestone Technologies, Inc. in the United States and/or other countries.

Other company, product, and service names used in this publication may be trademarks or service marks of others.

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

Symbols