21

OLAP

Here we are 21 chapters in. You've learned all the great and fantastic stuff – now forget most of it.

Up to this point, we've been thinking in terms of transactions – we've been working primarily in the world of **Online Transaction Processing**, or **OLTP**. Now, however, it's time to get dimensional – to start thinking in terms of analysis and **Online Analytical Processing**, or **OLAP**.

In this chapter we will:

- Discuss the differences between the needs of transaction processing versus analysis processing
- Discuss how these differences necessarily lead to substantially different solutions
- Dispel the myth that your OLTP solution can also work as a great OLAP solution
- Define the concept of a data cube, and indicate how they can help provide a solution to the special requirements of an OLAP environment
- Look at the basic new OLAP Services that now come as part of SQL Server 7.0

The Requirements of End Users

As corporations build business applications and store their daily data in back-end databases associated with their applications, the databases swell in size. This swelling eventually has negative impacts on the applications themselves - slowing them down, hampering concurrency, reducing scalability, and even causing them to crash at times.

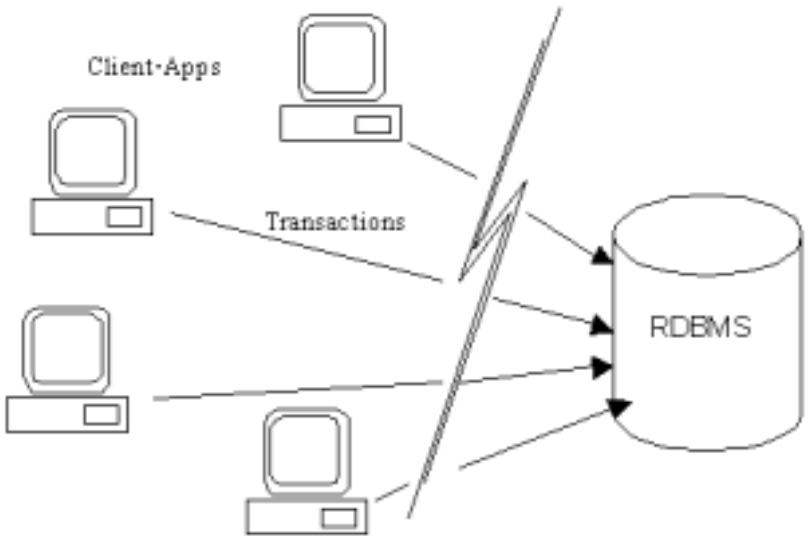
End users may use the data sources differently from one another. Two main categories of users can be distinguished:

- Those who want to access the data sources on a daily basis, retrieving certain records, adding new records, updating, or deleting existing records
- Those who want to make sense of the enormous amounts of data piling in the database, generating reports that will help them come up with the right decisions for the corporation and give it the competitive edge that will make it succeed in the marketplace.

Two separate systems, Online Transaction Processing (OLTP) and Online Analytical Processing (OLAP), help satisfy the different requirements of the two categories of users. The following sections present the characteristics of the two systems. A comparison is then drawn between the two systems to come up with common rules that govern the creation of such systems.

Online Transaction Processing (OLTP)

Just as I mentioned in the previous section, OLTP systems are designed to allow for high concurrency making it possible for many users to access the same data source and conduct the processing they need. As the name implies, these systems allow for transactions to be processed against the database. Transactions mean controlled changes to the data in the tables due to inserts, updates and deletes during the conduction of business processes. The following figure depicts a basic OLTP system:



The figure shows that numerous client applications are accessing the database to get small pieces of information, add new data, update, or delete existing data. The broken line between the client applications and the server symbolizes that such connection can physically take place in many different ways. For instance, client applications could be accessing the database through a transaction monitor, or they could be connected directly through the network.

Examples of OLTP systems include data-entry programs, such as, banking, ticket reservation, online sales applications, etc. No matter what the application is, OLTP systems are usually built with the following objectives in mind:

- Process data generated by transactions
- Maintain a high degree of accuracy by eliminating data redundancy
- Ensure data and information integrity
- Produce timely documents and reports, such as receipts and invoices
- Increase work efficiency
- Improve and enhance offered services to build and maintain customer loyalty

These objectives are usually accomplished by keeping the database in third normal form (or better), eliminating redundancy in the system, and maximizing relationships among the business entities as represented by the tables.

Online Analytical Processing (OLAP)

OLAP systems fall under the broader scope of **Decision Support Systems (DSS)** and **Executive Information Systems (EIS)**. The goal of OLAP systems is analyzing huge amounts of data, generating summaries and aggregations in many different ways to help decision makers find patterns and trends that would allow them to improve the performance of their corporations, gain competitive edge, and optimize business processes.

With OLAP systems, you need to forget about nicely keeping your relational database in the third normal form. You'll need to break this form and de-normalize the database (or flatten it) allowing for some redundancy with improving query performance in mind. This is because data stored in OLAP systems rarely undergoes edits and changes. The data is kept there for query purposes; to generate reports that would help decision makers plan the future of their enterprises. With this established, the database no longer conforms to the relational database definition and conditions; it becomes what is called a **dimensional database**. These databases are used to build

data cubes, which are multi-dimensional representations of the data that facilitate online business analysis and query performance. The dimensions of a cube represent distinct categories for analyzing business data. Such categories include time, geography, or product line among others.

> **SQL Server 7.0's OLAP Manager allows you to build up to 64 dimensions for each data cube.**

Later, we will discuss data cubes and dimensions in more detail. For now, let's just see how these concepts serve DSS and EIS in a more useful manner than OLTP.

OLTP or OLAP?

Now that you have seen the general ideas behind the two systems of OLAP and OLTP, let's consider the following example.

Let's take the banking business for example. During the bank work hours, bank tellers help customers perform their much needed transactions, like depositing funds into their accounts, transferring funds between accounts, and withdrawing funds from these accounts. The customers may also conduct their own transactions using an automatic teller machine (ATM) or phone-based and/or computer-based banking service. In other words, such transactions are not limited to a particular part of the day but can take place around the clock. All of these operations lead to changes in the data stored in the database. These changes could be inserts of new records, updating, or deleting existing records.

OLTP is built to allow these transactions to be made by a large number of users accessing the database concurrently. Databases serving OLTP systems are usually relational and in the third normal form, and their table indexes need to be selected carefully for the right fields. OLTP databases should be built to promote performance, allow high frequency of transactions, and represent tables in a relational fashion with parent-child relationships. Transactions held in such systems include inserts, updates, deletes, and selects.

Let's now look at a different scenario with the banking example. Suppose that the bank managers are conducting future planning. They definitely need to look at current and historical performance data of the bank. If they were to query the database that is used for the OLTP system, they will face big difficulties and cause major problems to other users who are conducting their transactions.

These issues arise because the queries used to build management reports will usually be summary, or aggregation queries. For example, they might want to know the total amount of transactions conducted by all customers in a certain region. Such queries will have to sift through large amounts of data that is fragmented and scattered over many joined tables. For example, an accounting general ledger transaction could be stored in a dozen different tables. The queries will have to pull fields from these joined tables to build the views needed by the management, grouping and performing aggregations as it does so. All of this will impose a large overhead on the database management system (DBMS) slowing down the OLTP applications and the report generation process for the managers as well.

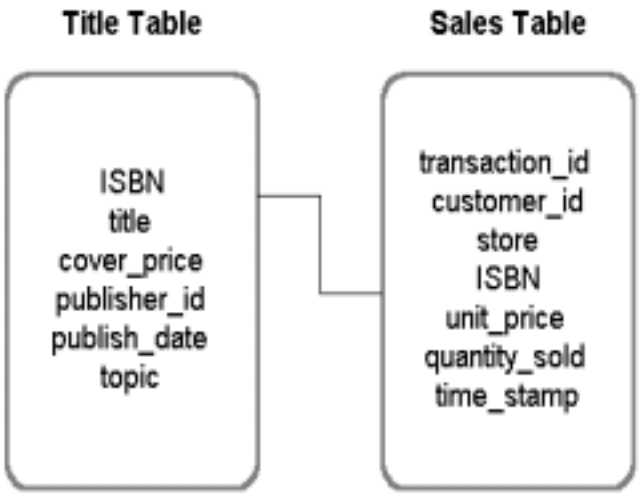
To face these challenges, it was necessary to separate the managers who use existing bank data to build their future outlook and planning, and have them use a different system based on OLAP principals. This means creating two different systems: an OLTP system for transaction processing by bank staff and customers, and an OLAP system to help with the decision-making.

Now we have two different systems, should these systems use the same database with separate tables for each system, or should they use two completely different databases? The answer to this question depends on how much effect one of the systems will have on the performance of the other. It is very likely that the two systems will be used at the same time. This causes performance problems even if the tables are separate. This is because the two systems still share many resources on the database server, and these resources may be depleted quickly with the two systems in use. These two systems are usually optimized differently. If we optimize for OLAP, we may adversely affect the performance of the OLTP system, and vice versa. Therefore, even though it is theoretically possible to tap into the same database, it is a good idea to keep separate databases on separate database servers for the two systems. With this, each system will have its own resources, and optimizing it will not affect the other system.

Querying an OLTP System

Although the relational model allows for great flexibility in defining ways to look at and process the data in the database, we often find that the way data is processed in a business solution is different, especially by decision

makers. Decision makers are not interested in the details of every single transaction recorded in the database; they are interested in looking at the big picture instead. Let's consider a `sales` database for a bookseller, for instance. The database is likely to include a `sales` table, such as the one presented here:



As the OLTP system is used over time, the table grows in size and becomes full of sales data, sometimes for the same customers, buying the same goods at different prices. Thus, the table becomes a good candidate for analysis.

The business analyst may be interested in finding out the effect the price had on the sales of programming-related books. In this case, she would be looking for the results in a table or even graph format similar to the chart below:

Price	**Quantity Sold**	**Revenue**
$20	7,000	140,000
$30	3,500	105,000
$40	2,500	100,000
$50	2,000	100,000

To get such results, the business analyst may ask a SQL developer to write a complex query for her that would extract the data from the OLTP system and put it in the format she wants. Such query may look like the code presented below:

```
SELECT     unit_price AS Price,
           SUM(quantity_sold) AS 'Quantity Sold',
           unit_price * SUM(quantity_sold) AS Revenue
FROM       sales, title
WHERE      title.ISBN = sales.ISBN
      AND  title.topic = 'programming book'
GROUP BY   unit_price;
```

Needless to say, it's very time and resource consuming for the business analyst to have to follow this path whenever she wants to get some summaries and aggregations of data. Yet, this is a simplified version of what could happen in the real world. In production OLTP systems, such a query may be much larger, involving many table joins that would effect the speed at which the results will return and effect the performance of other users of the database.

Dimensional Databases

The solution to the problems inherent with requesting complex queries from OLTP systems is to build a separate database that would represent the business facts more accurately. The structure of this database will not be relational; instead, it will be **dimensional**.

The Fact Table

The central table of a dimensional database is called the **fact table**. Its rows are known as **facts** and they are **measures** of activities.

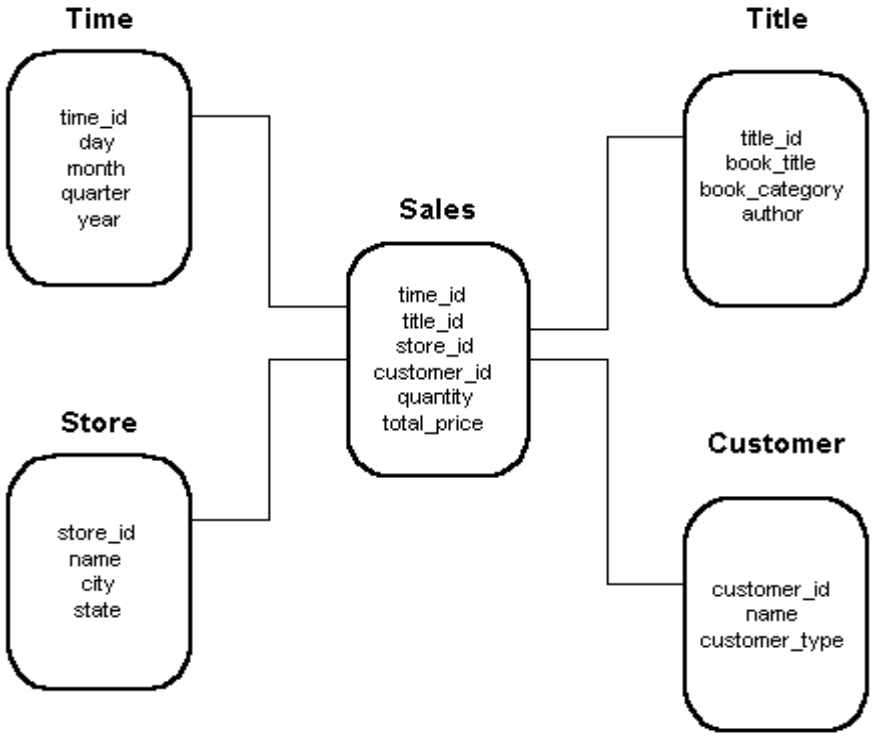
For example, we may build a `sales` fact table to record daily sales transactions of products offered by a given store. The `sales` table includes the facts of the business activities. `quantity` and `total_price` are the measures of facts in the `sales` table, just as they usually are in a typical relational database.

The Dimension Tables

Dimensions help put the facts in context and represent such things as time, product, customer, and location. The dimensions describe the data in the fact table. Continuing with our sales example, it would make sense to have time, store, customer, and product line dimensions.

The fact table, `sales`, captures transactions on a daily level for each store, for all customers, and for all books. This table, therefore, will grow to be very large. To improve the efficiency at which data is retrieved from the database, aggregates are pre-computed at different levels and stored in the database or in an optimized format, as we will see later in the chapter.

The tables linked to the fact table are called **dimension tables** and each represents a dimension. They are used to generate the aggregations from the fact table. For instance, we could find the total monthly sales of all books to all customers by all stores if we were to query the `sales` table grouping by month of the year. Alternatively, we could find the total sales by state at all times, for all customers, and for all books if we queried the `sales` table grouping on state. We can also have aggregations on a combination of the dimensions in the `sales` fact table. For example, we could find the total sales for a particular book category by state on a monthly basis for a certain type of customer by grouping on state and month and adding the appropriate criteria in the `WHERE` clause for the customer and book category.



The Star and Snowflake Schemas

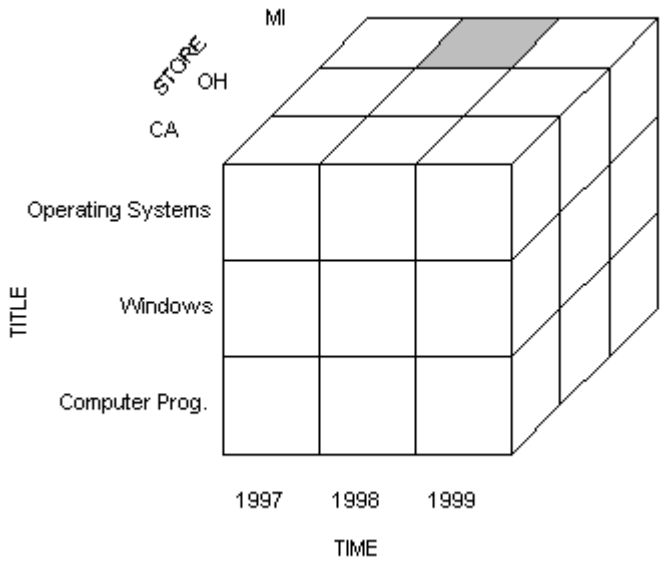
The database schema above, where there is a single fact table with a number of dimension tables linked directly to it, is an example of the **star schema**.

Another schema the dimensional database can follow is the **snowflake schema**. In a snowflake schema, multiple

tables define one or more of the dimensions. What this means is that the snowflake schema is an extension of the star schema, where extra dimension tables are linked, not to the fact table directly, but to dimension tables.

Data Cubes

The example we have just seen is often represented as a **data cube**. The dimensions of the cube represent the dimensions of the fact table. Each cell in the cube represents a fact corresponding to a level of detail for the different dimensions of the cube. Although the graphical representation of the cube can only show three dimensions, a data cube can have up to 64 dimensions when using SQL Server's OLAP services. The following figure shows a representation of a data cube for the `sales` table with the store, title, and time dimensions shown:



If you want to use this cube to find out the total sales for stores in Michigan during 1998 for the Operating Systems book category, you need to look at the shaded cell in the figure, which is the resulting cell from the intersection of those three dimensions. That cell should contain the quantity of books sold and the total price, which are the facts identified in the fact table.

Microsoft's OLAP Services allows you to build your cube from any source of data that has an OLE DB provider. This source can be a relational database in any database management system that has an ODBC driver (such as Oracle or Sybase SQL Server) or has a native OLE DB provider (such as SQL Server, or MS Access). The data source for the cube can also be a dimensional database, text files or LDAP data source.

OLAP Storage Types

Based on the cube data storage method, three options of OLAP are supported by the decision support systems that ship with SQL Server 7.0. These options are:

- Multi-dimensional OLAP (MOLAP)
- Relational OLAP (ROLAP)
- A hybrid of the previous two options (HOLAP)

Each of these options provides certain benefits, depending on the size of your database and how the data will be used.

MOLAP

MOLAP is a high-performance, multi-dimensional data storage format. The data supporting the cubes is stored with this option on the OLAP server as a multi-dimensional database. MOLAP gives the best query performance, because it is specifically optimized for multi-dimensional data queries. Performance gains stem from the fact that

the fact tables are compressed with this option and bitmap indexing is used for them.

Since MOLAP requires that all of the data be copied, converting its format appropriately to fit the multi-dimensional data store, MOLAP is appropriate for small to medium-sized data sets. Copying all of the data for such data sets would not require significant loading time or utilize large amounts of disk space.

ROLAP

Relational OLAP storage keeps the data that feeds the cubes in the original relational tables. A separate set of relational tables is used to store and reference aggregation data in this OLAP system. These tables are not downloaded to the DSS server. The tables that hold the aggregations of the data are called **materialized views**. These tables store data aggregations as defined by the dimensions when the cube is created.

With this option, aggregation tables have fields for each dimension and measure. Each dimension column is indexed. A composite index is also created for all of the dimension fields. Due to its nature, ROLAP is ideal for large databases or legacy data that is infrequently queried.

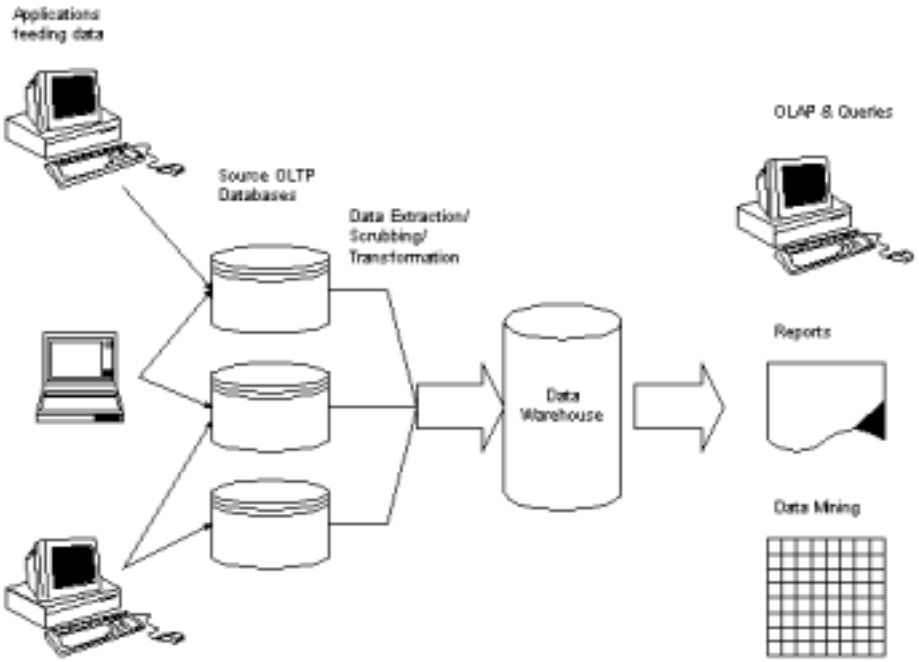
HOLAP

A combination of MOLAP and ROLAP is also supported by the DSS server. This combination is referred to as HOLAP. With HOLAP, the original data is kept in its relational database tables similar to ROLAP. Aggregations of the data are performed and stored in a multi-dimensional format. An advantage of this system is that HOLAP provides connectivity to large data sets in relational tables while taking advantage of the faster performance of the multi-dimensional aggregation storage. A disadvantage of this option is that the amount pf processing between the ROLAP and MOLAP systems may affect its efficiency.

Data Warehouse Concepts

Now we have seen what OLAP and dimensional databases are, let's define what a data warehouse is, and how it can be built in SQL Server 7.0.

A **data warehouse** is a data store that holds the data collected during the company's conduction of business over a long period of time. The data warehouse uses the OLTP systems that collect the data from everyday activities and transactions as its source. The data warehouse concept also includes the processes that scrub (see Data Scrubbing later in the chapter) and transform the data, making it ready for the data warehouse. It also includes the repository of summary tables and statistics, as well as the dimensional database. Finally, it also includes the tools needed by the business analysts to present and use the data. These tools include OLAP tools (such as SQL Server's OLAP Manager), as well as data mining and reporting tools. The following figure depicts the conceptual structure and components of a data warehouse solution:



Data Warehouse Characteristics

A data warehouse is usually built to support decision-making and online analysis (OLAP) because it is designed with the following unique characteristics:

- **Consolidated and consistent data**: In a data warehouse, data is collected from different sources, consolidated and made consistent in many ways, including the use of naming conventions, measurements, physical attributes and semantics. This is important because business analysts accessing the data warehouse and using its data for their decision-making process, have to use consistent standards. For example, date formats may all follow one standard, showing day, month, quarter and year. Data should be stored in the data warehouse in a single, acceptable format. This allows for the referencing, consolidating, and cross-referencing of data from numerous heterogeneous sources, such as legacy data on mainframes, data in spreadsheets, or even data from the Internet, giving the analysts a better understanding of the business.
- **Subject-oriented data**: The data warehouse organizes key business information from OLTP sources so that it is available for business analysis. In the process, it weeds out irrelevant data that might exist in the source data store. The organization takes place based on the subject of the data, separating customer information from product information, which may have been intermingled in the source data store.
- **Historical data**: Unlike OLTP systems, the data warehouse represents historical data. In other words, when you query the data warehouse, you use data that had been collected using the OLTP system in the past. The historical data could be over a long period of time, compared to the OLTP system, which contains current data that accurately describes the system for the most part.
- **Read-only data**: After data has been moved to the data warehouse, you may not be able to change it unless the data was incorrect in the first place. The data in the data warehouse cannot be updated because it represents historical data, which cannot be changed. Deletes, inserts, and updates (other than those involved in the data loading process) are not applicable in a data warehouse. The only operations that occur in a data warehouse once it has been set up, are loading of additional data and querying.

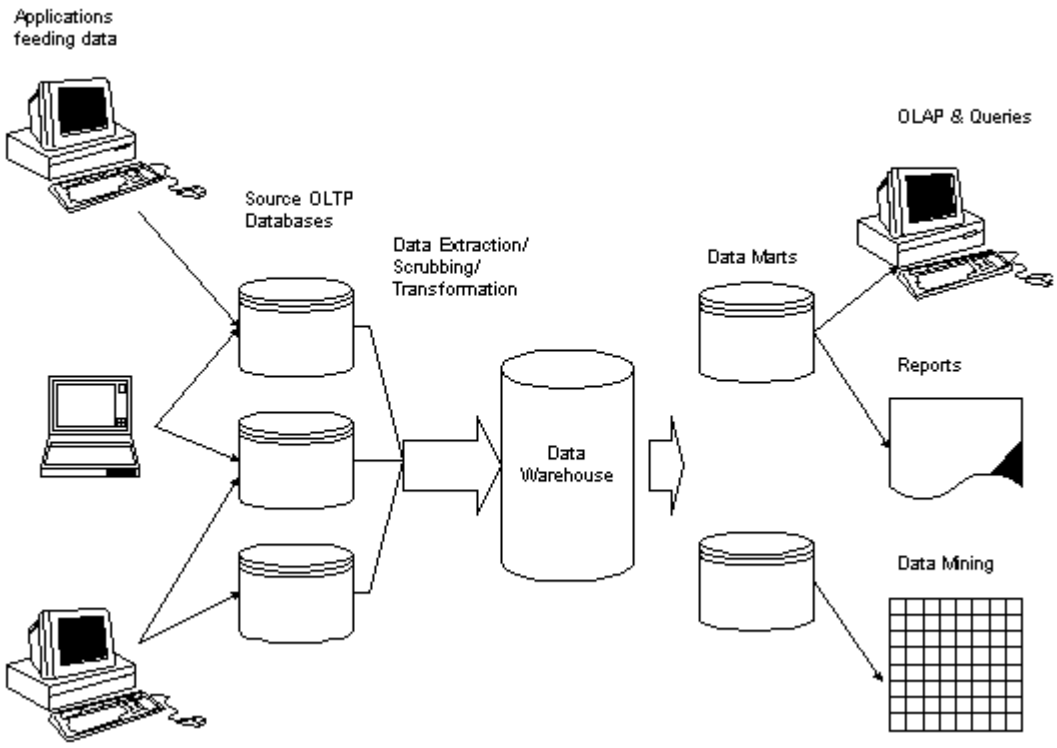
Data Marts

You may find out, after building your data warehouse, that many people in your organization only access certain portions of the data in the data warehouse. For instance, the sales managers may only access data relevant to their departments. Alternatively, they may only access data for the last year. In this case, it would be inefficient to have these people query the whole data warehouse to get their reports. Instead, it would be wise to partition the data warehouse in smaller units, called **data marts**, which are based on their business needs.

In addition, some people in your organization may want to be able to access the data in the data warehouse in

remote areas far from the company buildings. For instance a sales manager may want to access data about products and sales particular to his or her market area while on a sales venture. People such as this would benefit from a data mart, as they would be able to carry a section of the data warehouse on their laptop computers, allowing them to access the data they need at any time.

Of course, with data marts, the data should be kept in synch with the data warehouse at all times. This can be done in a variety of ways, such as using DTS, ActiveX scripting or programs. The following diagram shows the structure of the data warehouse concept with data marts included:



Data Transformation Services

Many organizations need to centralize data to improve decision-making. However, this data can be stored in a large variety of formats in a number of different sources. The row data that exists in these sources has to be reconciled and transformed in many cases before it can be stored in the data warehouse. **Data transformation services** (DTS), which we saw back in Chapter 18, conducts this function providing a means to move data from the source OLTP database to the destination data warehouse while validating, cleaning up, consolidating and transforming the data when needed.

Data Validation

Conducting data validation before the data is extracted from the source OLTP databases and transferred to the destination data warehouse is extremely important. If the data is not valid, the integrity of the business analysis conducted based on it will be in question. For example, if one of the fields is a currency field, and the OLTP data sources exist in multiple countries around the globe, one has to make sure that the data in this currency field is always transferred in the currency of the destination data warehouse. If you transfer 500 French Franks as 500 US dollars, you will be misrepresenting the data in the data warehouse, and any report that includes this value will not be valid.

Another thing you need to pay close attention to when validating the data, is information related to geographical regions; you need to make sure that referenced cities are in the right countries as the `country` field states. Yet another example of validating data is that you need to make sure that the products (books in our example) are represented in a similar manner in all data sources.

Data Scrubbing

Data reconciliation has to take place between multiple sources feeding the same data warehouse. The reconciliation process is referred to as **data scrubbing**. For example, if the book mentioned above is classified in one OLTP database under the category, database, and in another OLTP database under a category called, database systems, aggregations in the data warehouse involving this category will yield inaccurate results, unless the two data sources have been reconciled during the data transformation process.

Data scrubbing can be achieved in different ways. These ways are beyond the scope of this book, but are mentioned here:

- Using DTS import and export wizards to modify data as it is copied from the source to the destination data store
- By writing a Microsoft ActiveX script or program. Such a script or program may use the DTS API to connect to the data source and scrub the data. This method is not the easiest method to perform data scrubbing, but it provides a great deal of flexibility, because it uses the power of the scripting or programming language to access the data, which can often provide a tremendous amount of control and manipulation across heterogeneous data sources.
- DTS Lookup provides the ability to perform queries using one or more named, parameterized query strings. This allows for building custom transformation schemes to retrieve data from locations other than the immediate source or destination row being transformed.

Data Migration

Ideally, when migrating data from OLTP data sources to a data warehouse, data is copied to an intermediate database before it is finally copied to the data warehouse. This intermediate process is necessary to allow for data scrubbing and validation to occur.

Special care should be taken when performing the data migration. The migration process should be performed during periods of low activity at the operational OLTP system to minimize the impact on the users of that system. If the migration is done from multiple data sources that are replicas or participate in replication processes, the migration should happen when all these sources are synchronized to make sure that consistent data is copied from these sources.

> *A commonly deployed strategy is to execute data migration procedures after the nightly database backups occur. This ensures that if a migration procedure crashes the system, the backup was just performed.*

Data Transformation

When you move the data from the source OLTP databases to the destination data warehouse, you may find yourself performing many transformations of existing data to make it more operational and practical when used in the destination warehouse. Below are examples of data transformations you may want to consider when moving data from the OLTP databases to the data warehouse:

- You may break a column into multiple columns, such as dividing a date or timestamp field into its components of day, month, quarter and year.
- You may also find yourself having to calculate new fields based on the values in source fields, such as creating a `total_price` field in the destination data warehouse, which is a result of multiplying the `unit_price` by the `quantity_sold` fields in the source database.
- You may need to merge separate fields into one field, such as merging the `first_name` and `last_name` fields in the source database in one `name` column in the destination data warehouse.
- You may also want to map data from one representation to another, such as translation of code to literal values and converting values from decimal numerical values (1, 2, 3, etc.) to Roman numerals (I, II, III, etc.).

DTS Components

Data transformation services (DTS) is comprised of the import wizard, the export wizard, and COM programming interfaces that allow for the creation of custom import/export and transformation applications. A detailed discussion of these topics was presented in Chapter 18.

Metadata and the Repository

Metadata is, by definition, data about data. In other words, the information about the way storage is structured in the data warehouse, OLAP and DTS services is all kept as metadata, which is stored in the Microsoft Repository. The repository is built to maintain such technical information about the data sources involved with the services mentioned above.

Repository information is stored in the SQL Server `msdb` database. The repository is the preferred means of storing DTS packages in a data warehousing scenario because it is the only method of providing data lineage for packages.

Access to the repository is possible through the interfaces exposed by the OLAP Manager's graphical user interface (GUI). Metadata can also be accessed through the **Decision Support Objects** (**DSO**), and through programs that use interfaces to the repository. However, Microsoft recommends only using the DSO or OLAP Manager's GUI to access the metadata, because the repository is subject to change in next releases, which may render any programs you build to access the repository directly unusable.

> *Note that multiple repositories can exist in a single SQL Server installation. However, DTS supports only a single repository database per server in the Enterprise Manager tree.*

Decision Support Systems

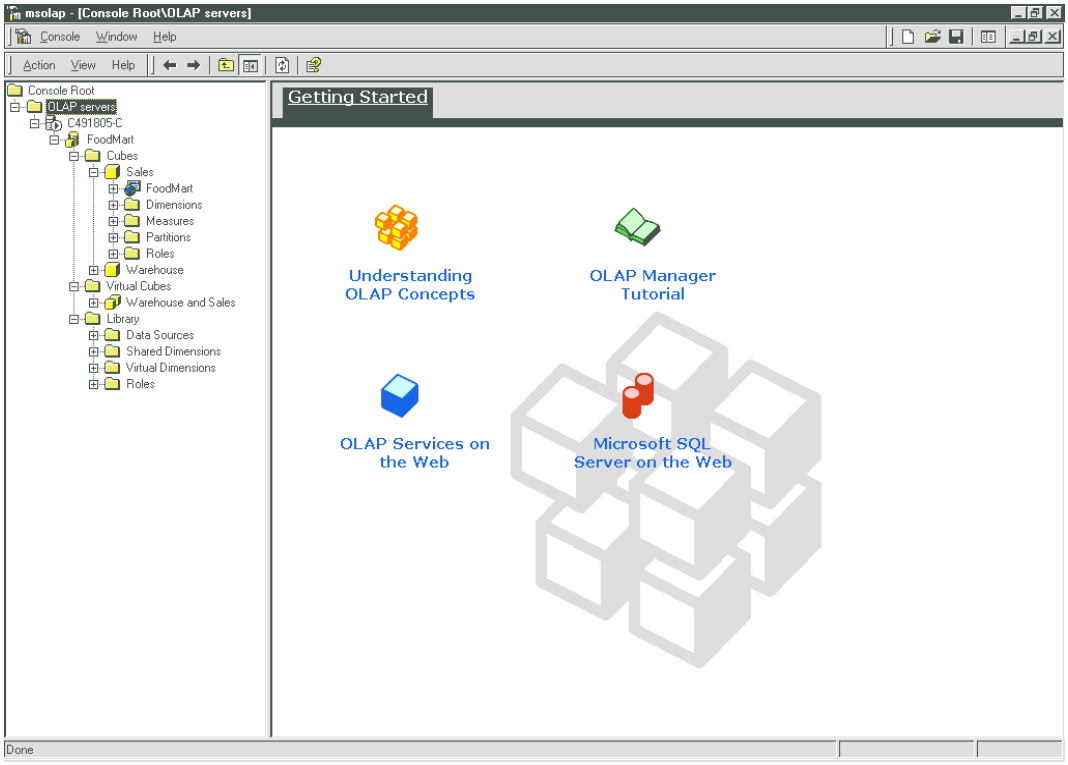
Decision Support Systems is the general name given to tools that provide decision support. Microsoft's suite of decision support tools is called **Decision Support Services** (**DSS**) and consists of the DSS Analysis server and the PivotTable Service. These tools are used in both the storage and user tools to extract and manipulate the data in the data warehouse environment. The DSS Analysis server can utilize heterogeneous data sources across the organization for analysis and querying. DSS Analysis server gives you the option to optimize for query performance, or to optimize for disk storage requirements. Several tools, including English Query and Microsoft Office (especially, MS Excel and MS Access) work with the DSS Analysis server accessing its cubes through the PivotTable Service. This gives the user a great advantage, because she/he is usually familiar with these tools, and can use them efficiently for the data analysis.

> *PivotTable Service was added to SQL Server 7.0 after being tested for quite a long time in Microsoft Excel and Access. This service allows the user to create cross-tab tables on the fly by specifying the columns and rows on which aggregations should be made.*

OLAP Manager

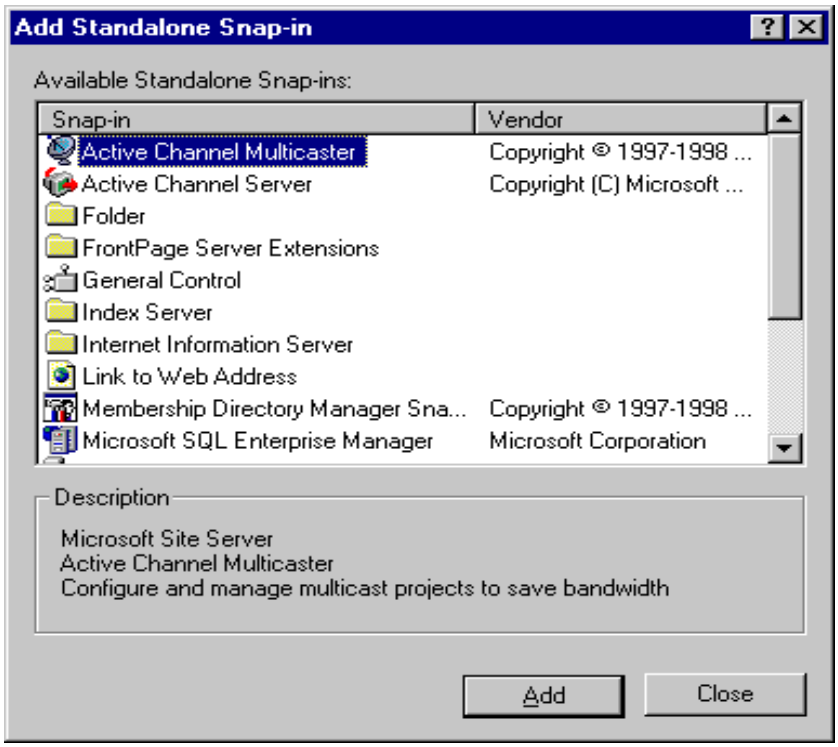
OLAP Manager is installed separately from the SQL Server CD ROM after installing SQL Server. This tool is a graphical user interface that allows the user to build an OLAP solution based on existing data sources. This section will demonstrate how this tool can be used while presenting an OLAP solution example at the same time.

To run OLAP Manager, you need to select the Microsoft SQL Server 7.0 group from the Start | Programs menu, then select OLAP Services | OLAP Manager. You'll see the following screen:



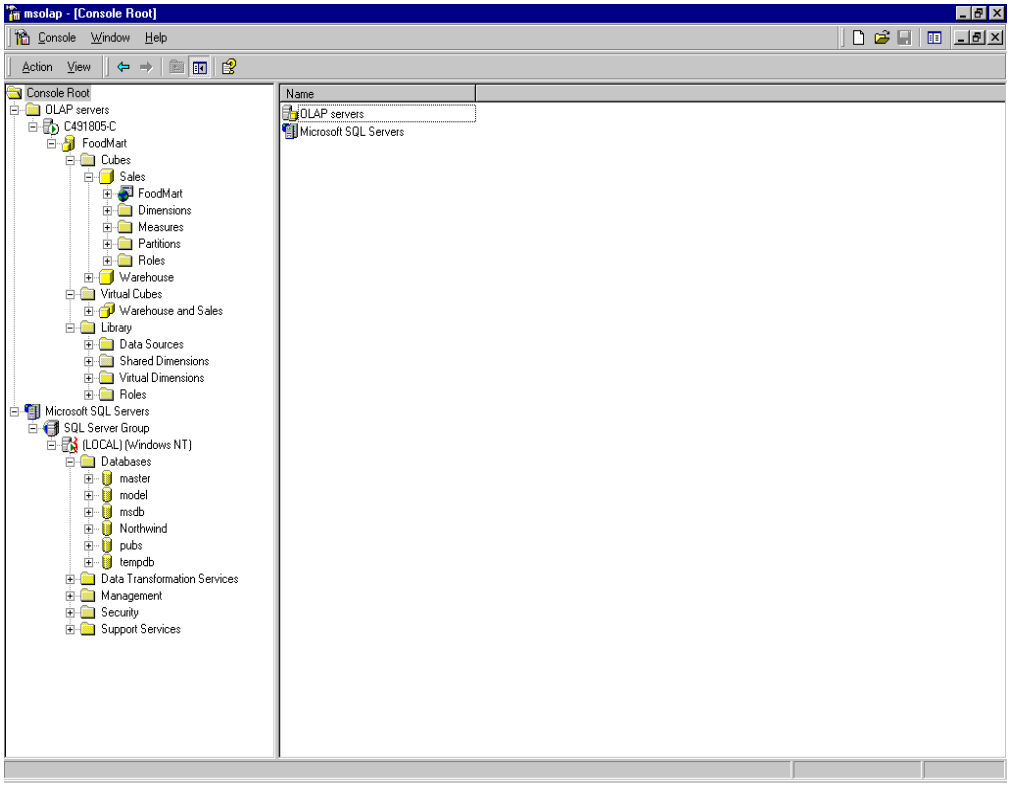
OLAP Manger is a snap-in of the Microsoft Management Console. Like the Enterprise Manager, it consists of two panes. The left pane contains a tree structure that depicts the available OLAP servers, databases, cubes, etc. The right pane contains details of the selected items in the left pane, whenever such details are available.

To add and/or remove existing snap-in items in MMC screen, select Console | Add/Remove Snap-in. Click on the Add button and you'll see the following Add Standalone Snap-in dialog:



Select the Microsoft SQL Enterprise Manager snap-in and click the Add button, then close the dialog. As a

result, you will see a screen similar to this:

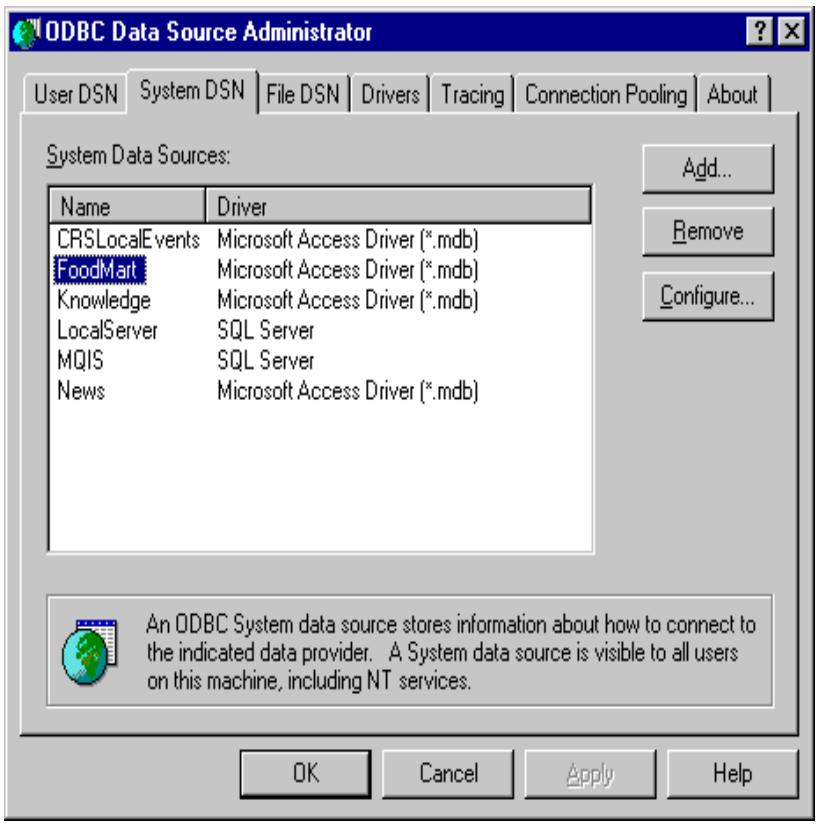


In summary, the OLAP Manager allows you to manage databases, data sources, cubes, dimensions, measures etc.

Creating an OLAP Solution

In this section, we will look at an example OLAP database that ships with SQL Server 7.0 OLAP services. This database is already full of data that can be used to show how the different OLAP and data warehouse tools work together. The database is called `FoodMart` and contains information about the sales and inventory of a national food chain. The database is an Access database that is installed when you install the OLAP server, and its default location is `C:\Program Files\OLAP Services\Samples\FoodMart.mdb`.

An ODBC data source name (DSN) is also created for this database at the time you install the OLAP services. The name of the DSN is also `FoodMart`. To verify that this DSN exists, select the ODBC32 applet from the Control Panel and then click on the System DSN tab, you'll see a screen similar to this:

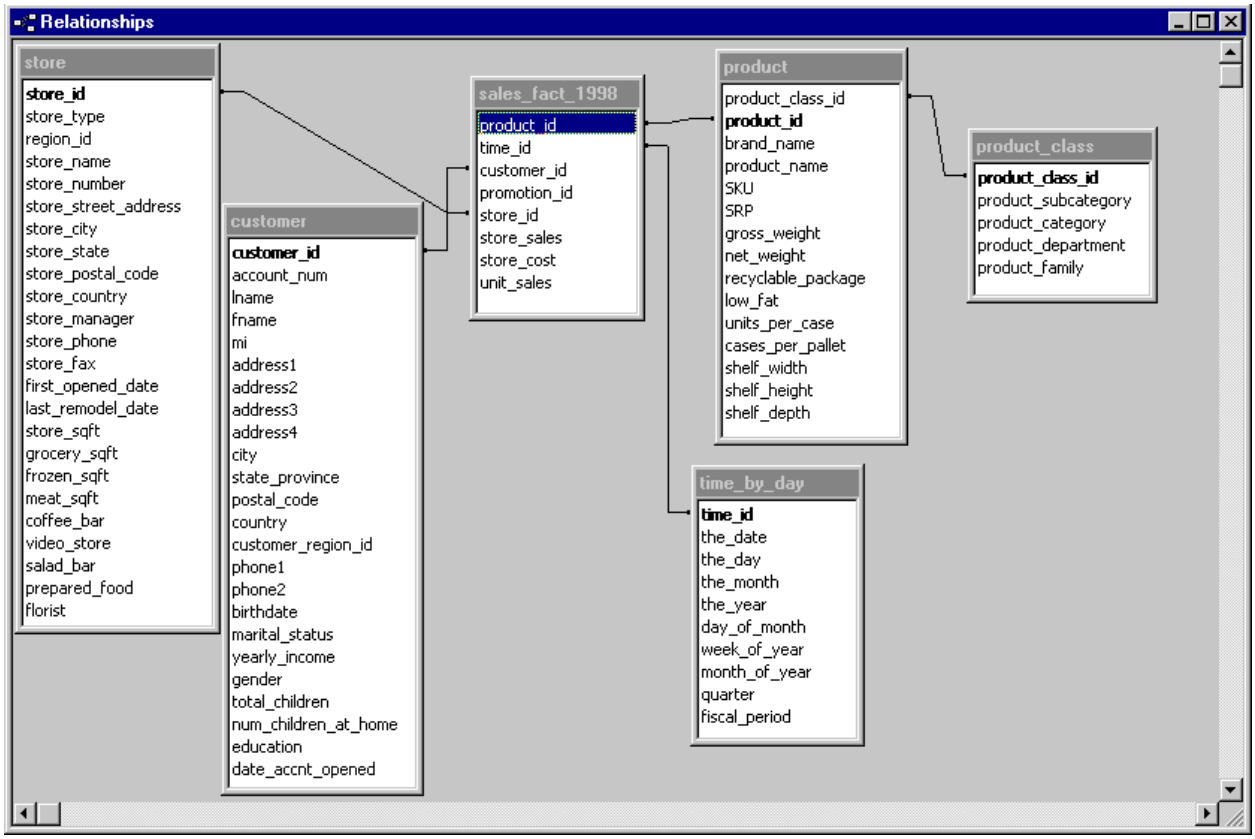


The goal here is to present the sales data for the stores, broken down by region, product, date, and customer for the year 1998.

> *If you want to look at the database in Microsoft Access, you need to close the OLAP Services Manager first, because when it is open, it accesses the* `FoodMart` *database in an exclusive mode.*

Let's now examine the database tables that we will need for our example. We will have a fact table for the year 1998 linked to the dimension tables: `product`, `store`, `time_by_day` and `customer`. The `product` table is also linked to a `product_class` table.

The fact table (`sales_fact_1998`) is the main table with which we need to concern ourselves. This table contains the fields that link it to the dimension tables mentioned above, as well as the measures we need to find eventually. These measures are the `store_sales`, `store_cost` and `unit_sale`. The following screen shot shows the database schema that we will need to use:



The links that the fact table has to the dimension tables allow us to produce the aggregations we want, rolling up the data to any level of these dimensions, as we shall see later in the chapter.

Before we move to the next section, notice that in the `customer` dimension table, there is no field that holds the customer name. Instead, there is a field for first name (`fname`), a field for middle initial (`mi`), and a field for last name (`lname`). It would be better to combine these three fields in one field called `name`. This makes it easier to define the levels of the dimension later on `country`, `state`, `city`, and `name`. To do this, create a new field called `name`, and update its value by running the following query in the `FoodMart` MS Access database:

```
/* To add the new field, run the following query */
ALTER TABLE customer ADD name text(180);
/* To update the name column, run the next query */
UPDATE customer SET customer.name = [fname] & ' ' & [mi] & ' ' & [lname];
```

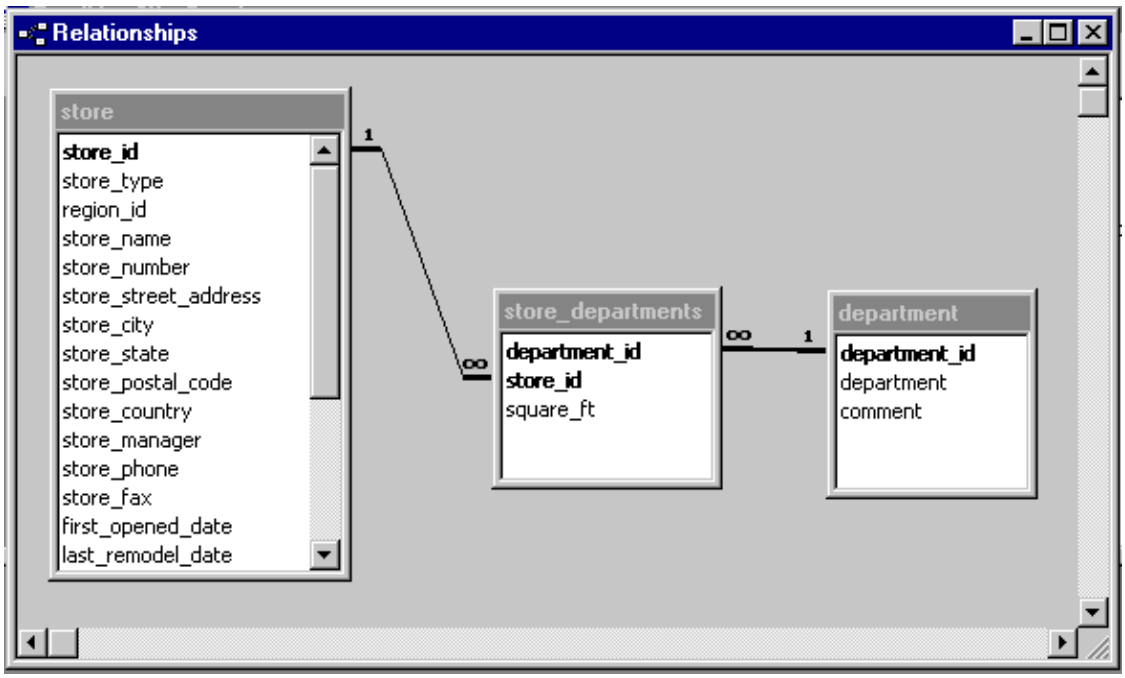
Once you have run the two queries in MS Access, you will see that a new field, called `name`, has been created and populated the way we wanted by concatenating the first name, middle initial, and last name fields.

How Did We Get to this Schema?

You may be wondering how the schema shown above was reached. It is most likely that the original production database was a relational database in the third normal form. This means that it is likely that the `store` table did not exist in this structure. The `store.store_manager` field was probably called `store_manager_id` originally, and linked to a table that held the employee information, maybe called `employee`. In such a table, the information about the employees would be stored, including their titles. When the `store` table linked to the `employee` table, it only needed to have the `employee_id` field as a foreign key pointing to the primary key in the `employee` table.

Also in the same table are the fields, `grocery_sqft`, `frozen_sqft` and `meat_sqft`. These fields were probably in a middle table between the `store` table and another table, perhaps called `departments`. The relationship between the `store` table and the `departments` table is a many-to-many relationship. A middle

table would be used to transform the many-to-many relationship into two one-to-many relationships. The structure of the middle table and its relationships might have looked like this:

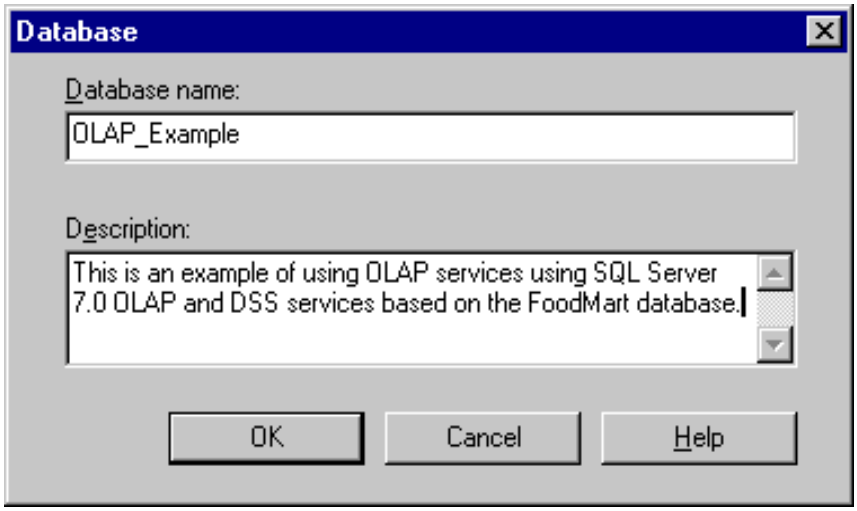


The kind of de-normalization in this particular case is described as de-normalization by flattening the data structures.

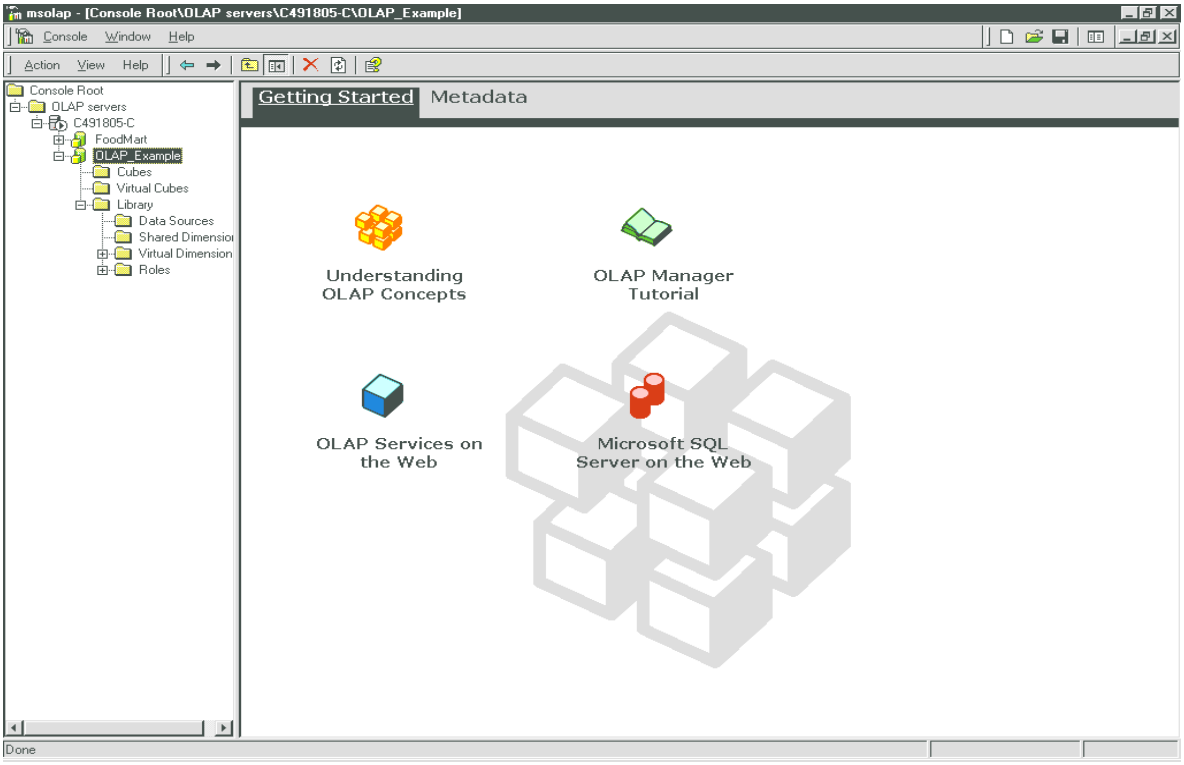
In summary, you can analyze the remaining dimension tables and normalize them to reach a relational database. The resulting database would represent the production database responsible for feeding the data warehouse.

Creating the OLAP Database in the OLAP Manager

To create the database we will be using, right-click the OLAP server name (which happens to be the same name as your machine) in the left pane of the OLAP Manager, and select New Database. Add the following information:

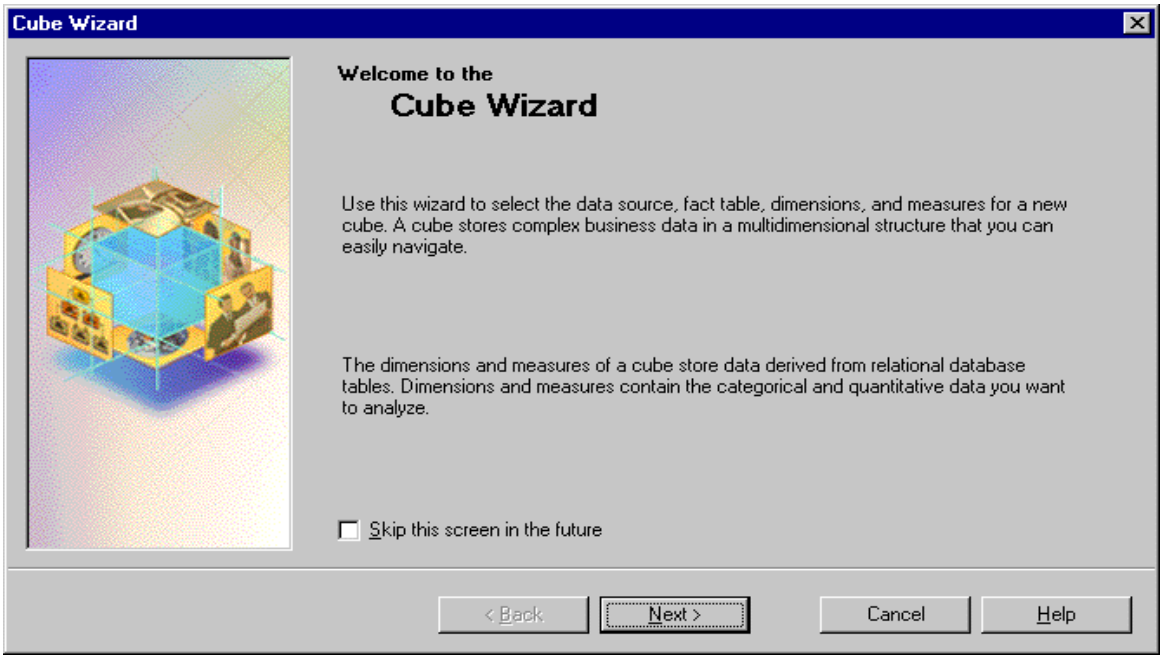


The new database will appear in the tree-view pane of the OLAP Manager. The database will have no cubes yet, no dimensions, no measures, etc. This will become apparent as you expand the tree view for the database by clicking on the + signs that correspond to it and its components:



Adding the Data Sales Cube

To add the first data cube in the database we just created, you need to right-click on the Cubes item in the left pane underneath the OLAP_Example database we have just created, and select New Cube from the context-sensitive menu. You will be presented with a submenu that has the two commands: Wizard and Editor (which is grayed out). Select Wizard, and you will be presented with the first screen of the Cube Wizard:

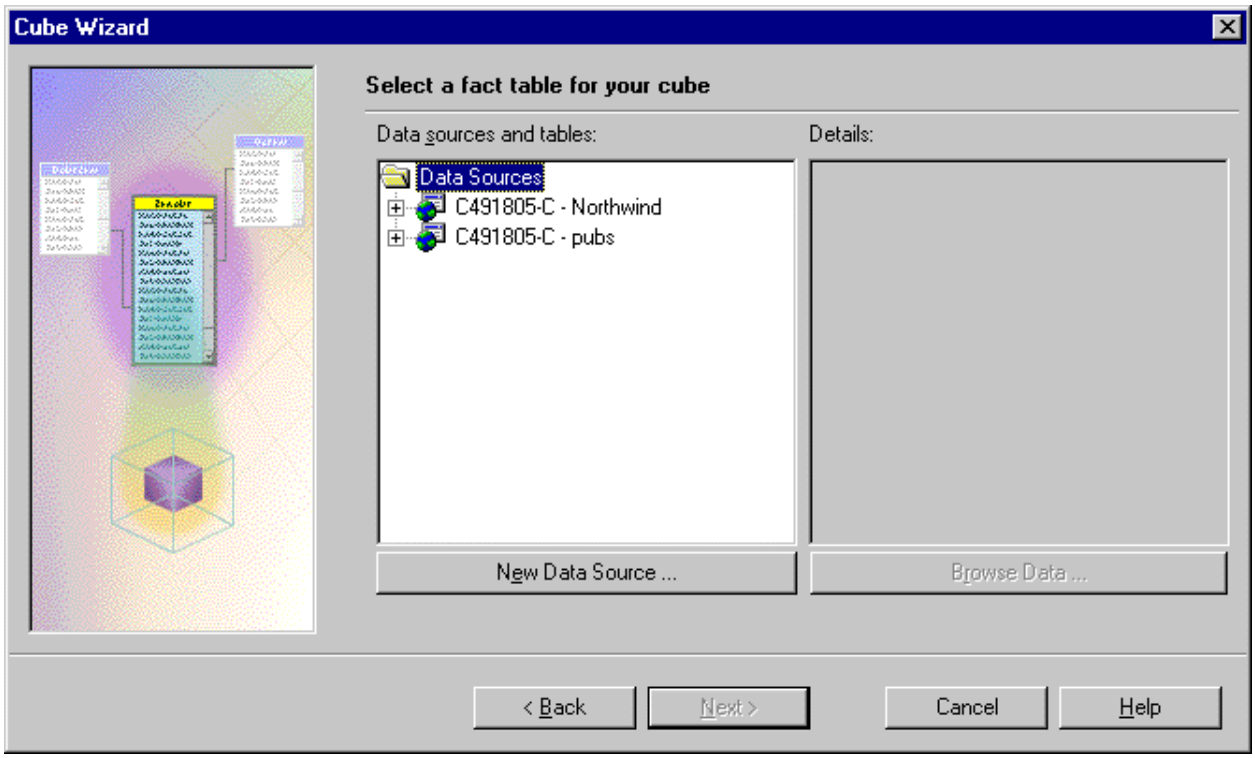


On the first screen of the Cube Wizard, click the Next button. The following screen will allow you to add a new data source to use for building the data cube, or use an existing data source for this purpose.

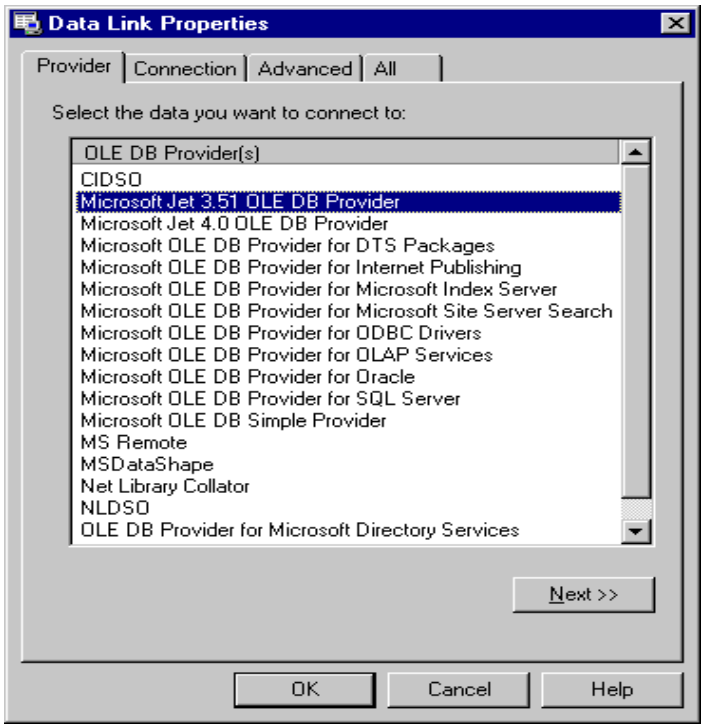
You can also add new data sources outside this wizard by right-clicking the Data Sources item in the left pane of

the OLAP Manager and selecting New Data Source. This will lead you directly to the Data Link Properties screen.

As for our example, click the New Data Source button on the wizard screen:

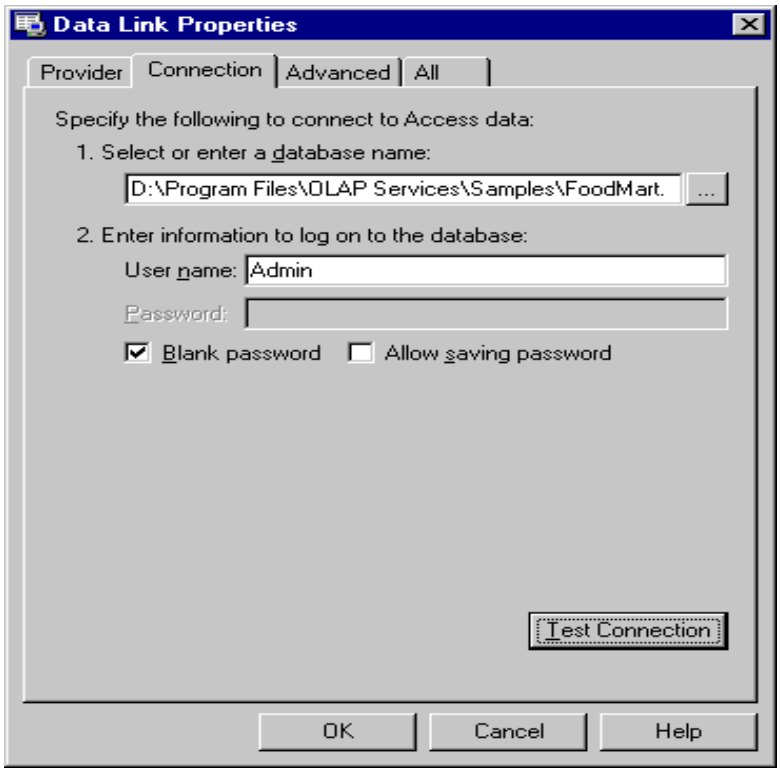


The Data Link Properties dialog will appear, allowing you to setup a data link using an OLE DB provider you have selected from the list. Select the OLE DB provider for Microsoft JET 3.51 or 4.0, and click the Next button:



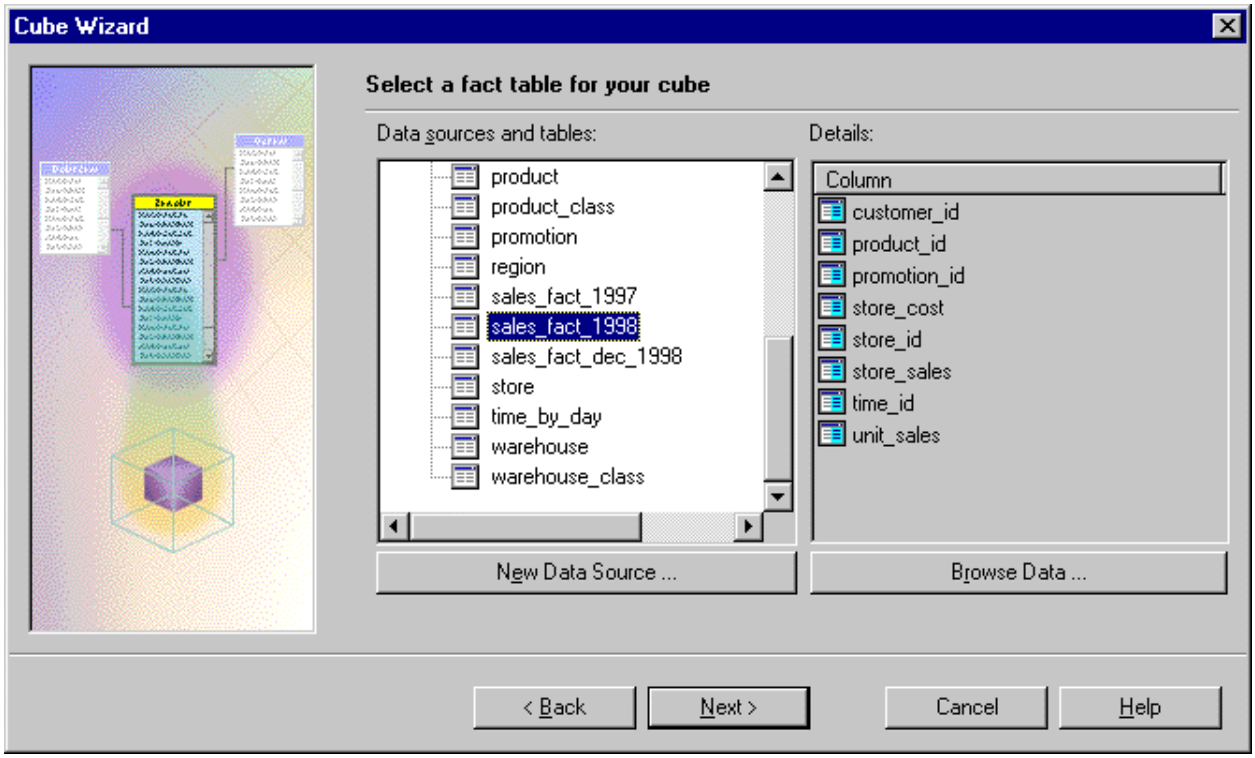
When you do so, you will be presented with the same dialog set to the Connection tab. This tab allows you to

specify the properties of the connection or data link, such as the database name, user name and password. Once you have connected to `FoodMart.mdb`, test the connection by clicking on the Test Connection button. This will ensure that the database server is up and running, that the database name is valid, and that the user name and password are also valid:

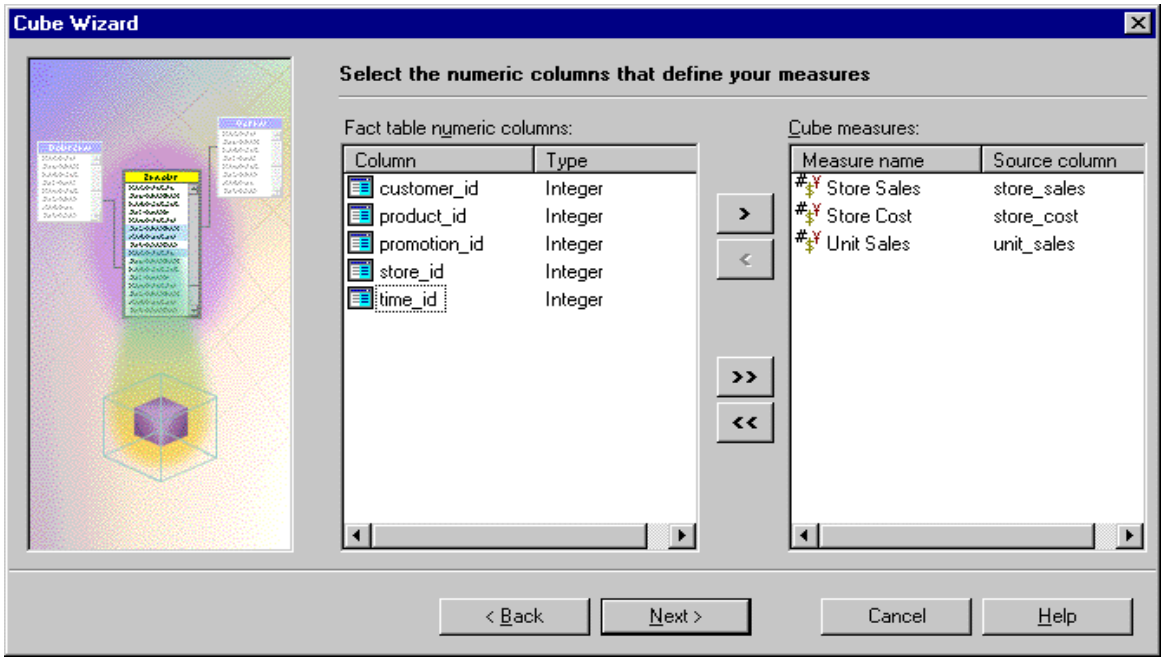


Click OK after testing the connection, and then select the newly added data source on the Cube Wizard screen, then click the + sign next to it to expand it. This will show you all the tables available from the selected database. Select the fact table of your choice, `sales_fact_1998` in our case. As you do so, you will notice that the right side of the wizard screen shows the fields existing in the selected fact table. You will also see a button called Browse Data. Clicking this button will actually bring a spreadsheet showing the data in your fact table. You can navigate through the records at this point. However, let's proceed with the Cube Wizard for now. Click the Next button.

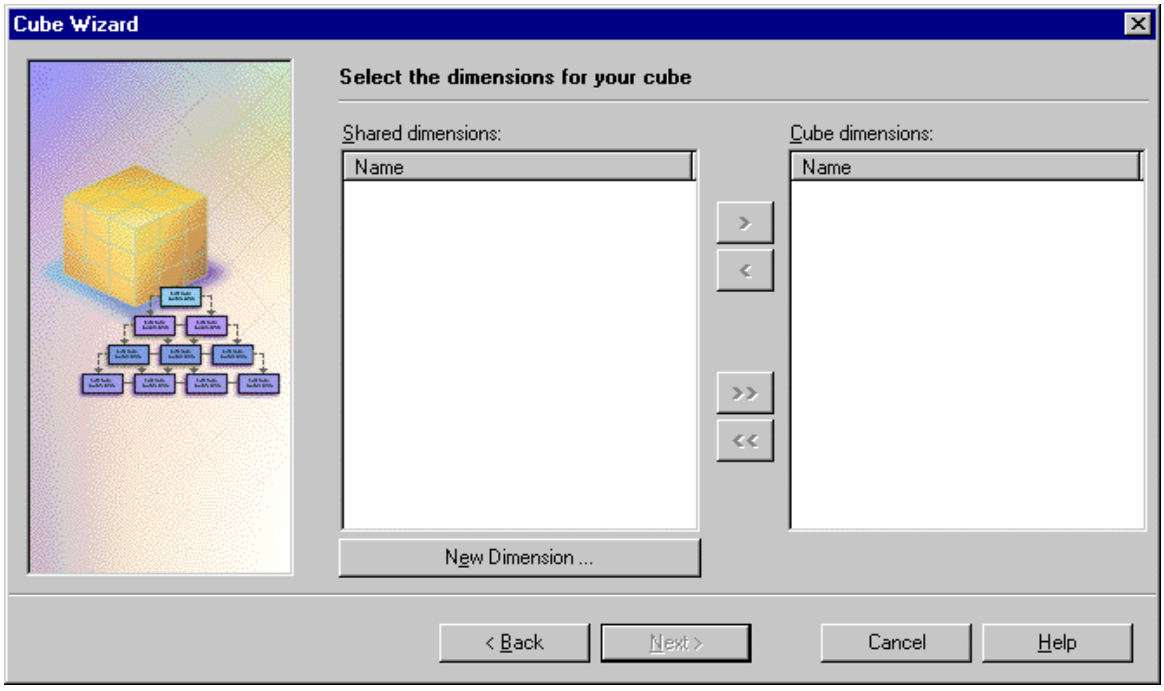
> *Make sure you set your Microsoft Management Console (MMC) so you are in Author Mode. To do this, select Options from the Console menu and make sure you select Author Mode from the Console Mode drop down list on the Console tab.*



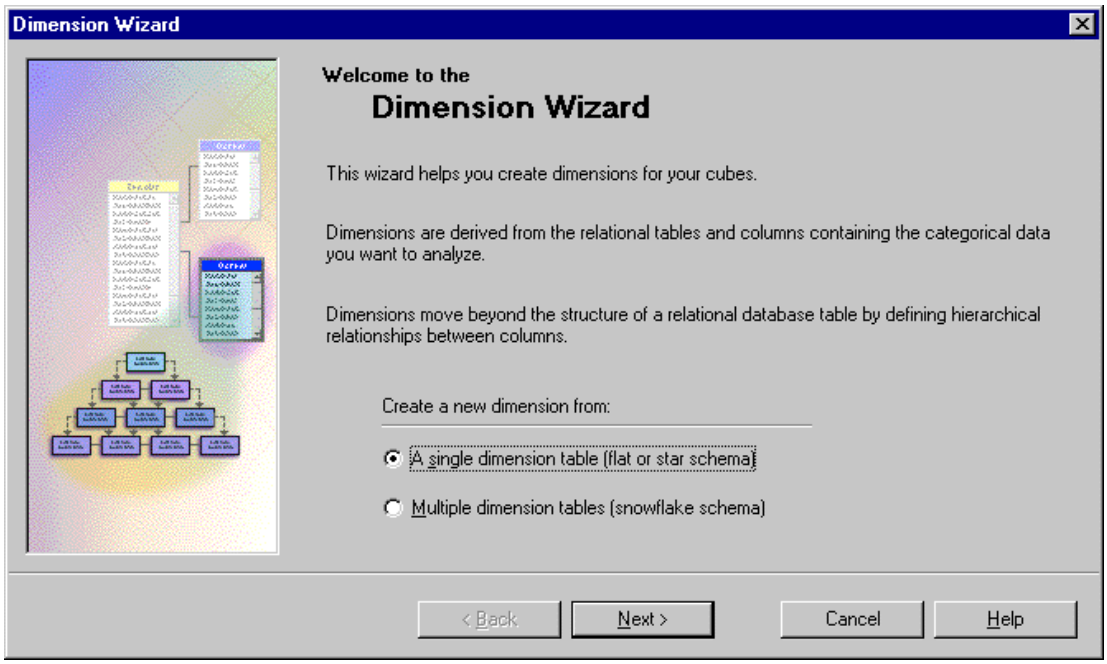
The next screen will allow you to define your measures. On the left-hand side of the screen there is a list of available columns in the fact table. Select the fields store_cost, store_sales and unit_sales individually, double-clicking each one to transfer it to the right-hand side list, the Cube measures list:



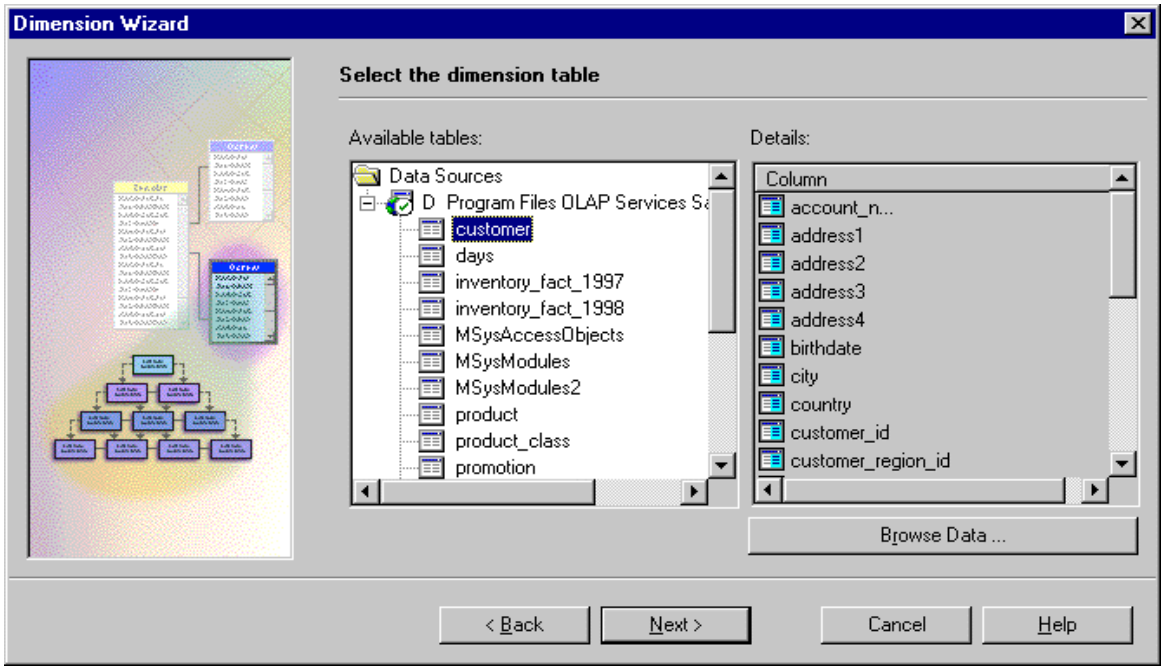
Click the Next button. The next screen will allow you to establish the cube dimensions. When you create dimensions for a cube, you have the option of making the dimension a shared dimension. Since the database has just been created, there are no dimensions defined in it yet, and the list of shared dimensions is empty. Therefore, let's create new dimensions to use for our cube. Click on the New Dimension button to launch the Dimension Wizard:



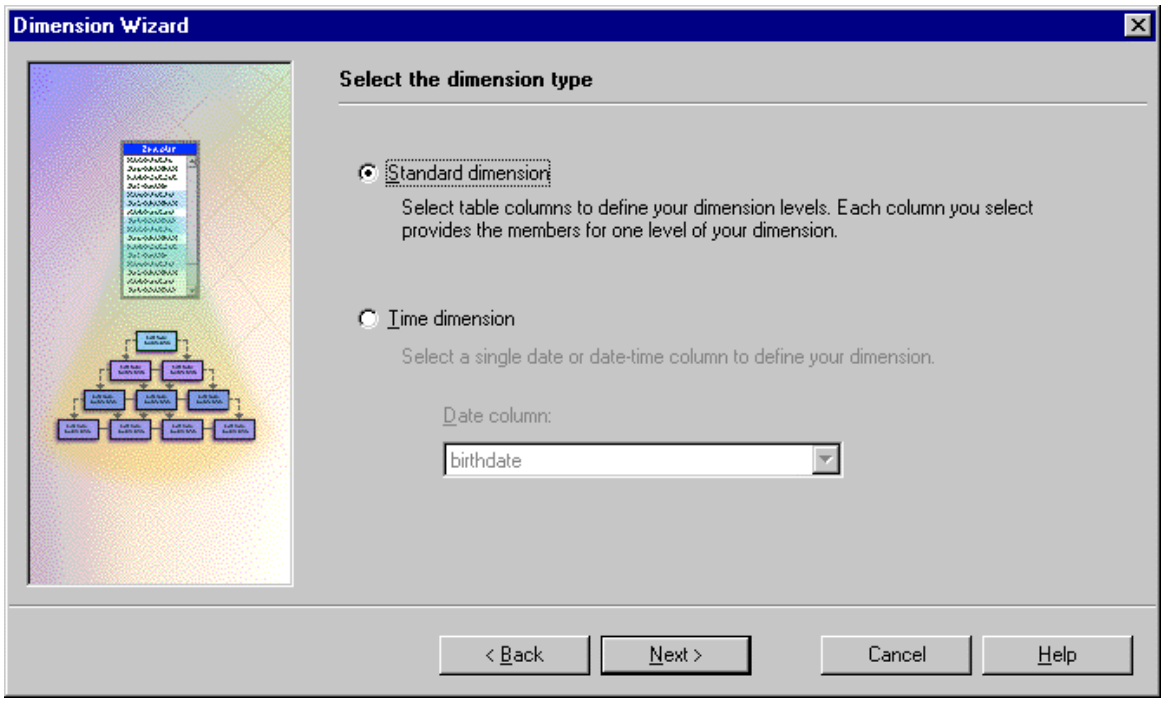
The first screen in the Dimension Wizard, allows you to specify whether the dimension you are creating will be based on a single dimension table, or on multiple tables. The first option corresponds to the star schema for OLAP databases, and the second one corresponds to the snowflake schema. Let's select the first option, A single dimension table, and click the Next button:



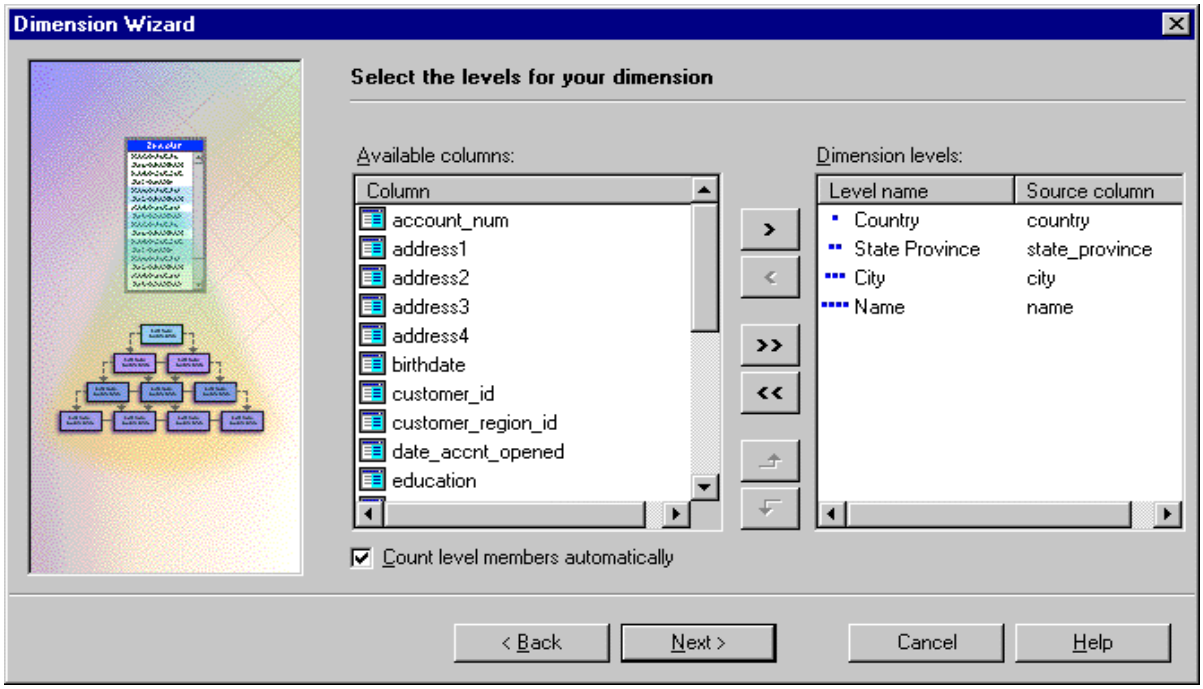
The next screen in the Dimension Wizard, allows you to select the dimension table. A list on the left-hand side shows all the available tables from which you can pick. Once a table is selected, a list on the right-hand side shows the fields in the selected table. Similar to the fact table, a Browse Data button allows you to browse the data in the selected dimension table in a spreadsheet format when you click it. For our example, select the customer table, and click the Next button:



The next screen asks you whether the dimension you selected is a time or a standard dimension. Select Standard dimension and click the Next button:

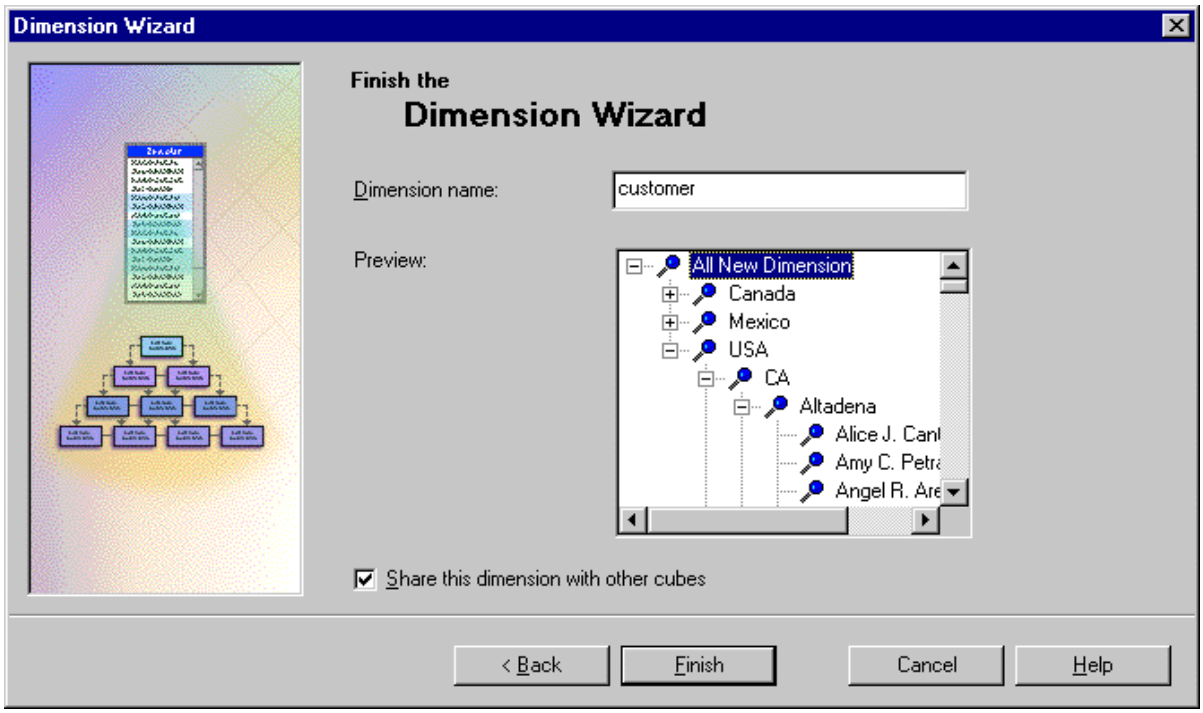


The next screen allows you to select the fields you want to use for your dimension levels. The fields in the customer table are presented on the left-hand list. Select the country, state_province, city and name fields, double-clicking them individually to transfer them to the Dimension levels list on the right-hand side. Note the name field that we added prior to creating our OLAP database and running the Cube Wizard. It makes more sense to select this field for our dimension level than selecting any of the three fields used to create it, namely fname, mi, and lname:

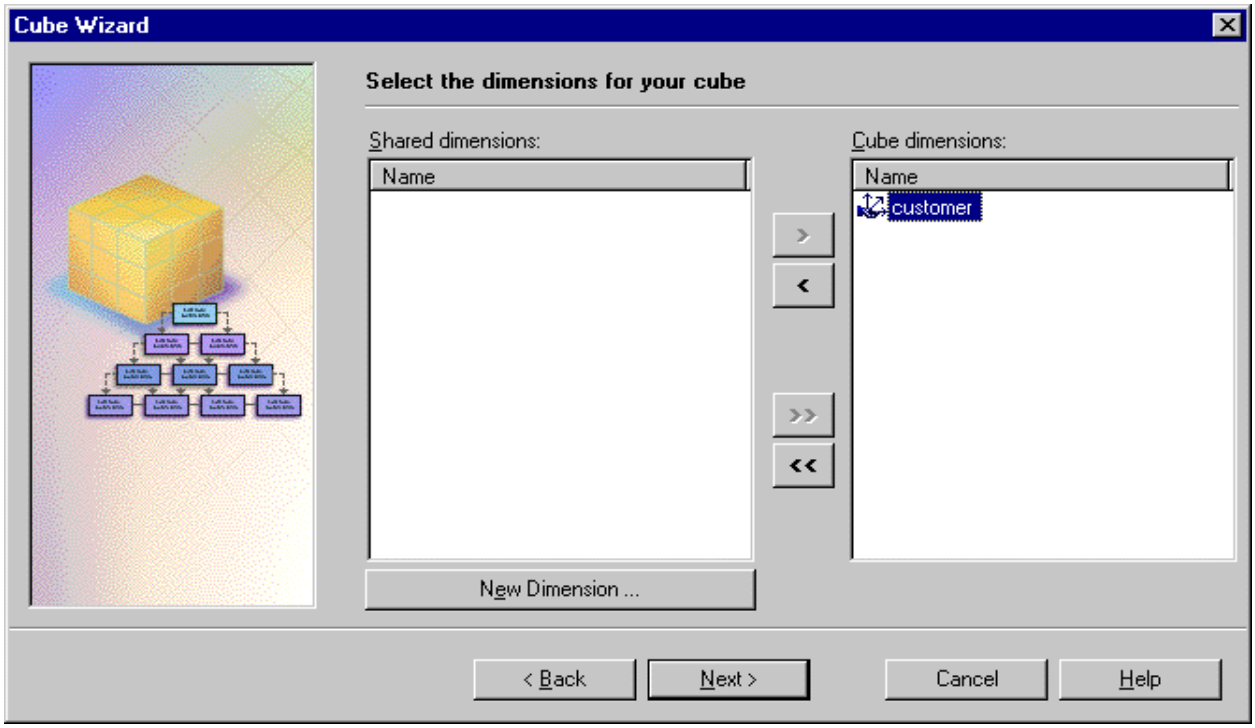


The next screen in the Dimension Wizard lets you select a name for the newly created dimension. Let's call it customer. Notice how a small window within the screen allows you to preview the data in the `customer` table in terms of the levels you selected. If you click the + signs to expand the view, you can clearly see the hierarchy that follows the selected levels for the dimension starting with the country at the top level, followed by the state/province, city, then name at the lowest level. This view shows you that three countries exist in the `customer` dimension table and that California is one of the US states for which there is data in the table. The cities within California that have related customers are shown. When you select one of these cities, you can see a list of the customers whose addresses are in that city.

Please note the checkbox that asks you whether you want to Share this dimension with other cubes in the database. Make sure it is checked, because it is always a good idea to share these dimensions, rather than re-inventing the wheel every time you need to create a similar dimension:

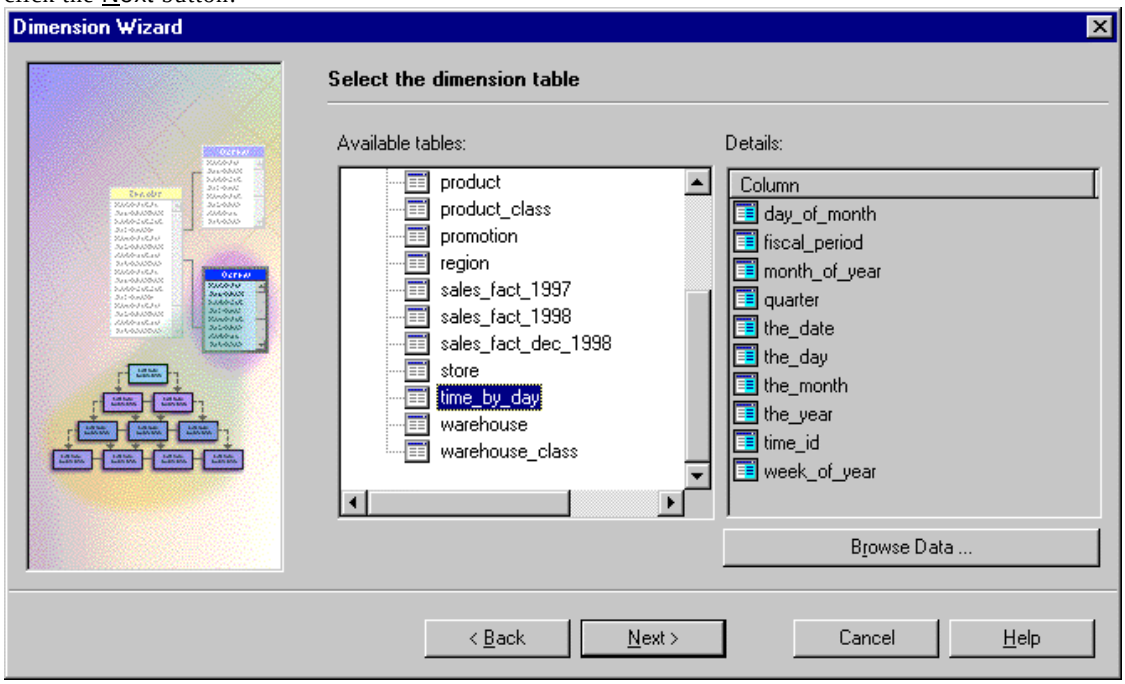


When you click on the Finish button, you are brought back to the Cube Wizard, specifically at the screen where you started the Dimension Wizard, with the new dimension already selected in the right-hand list of dimensions:

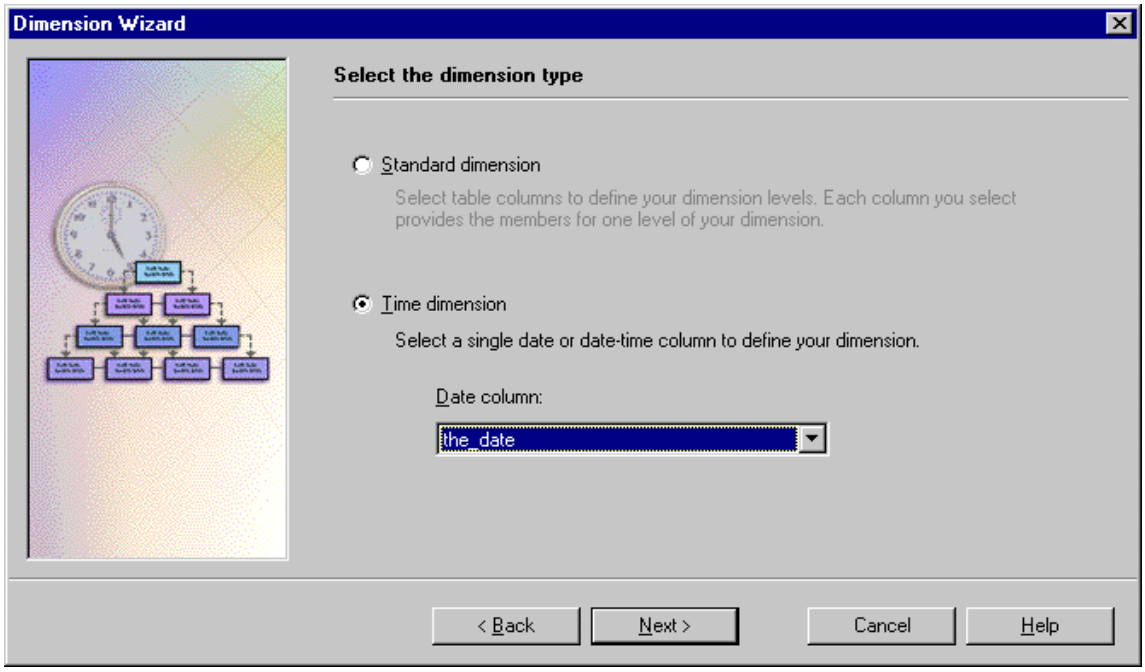


Follow the same steps we have just done to create the customer dimension, to create the store dimension. Make sure you select the following levels for the store dimension: store_country, store_state, store_city, and store_name.

Creating the time dimension is a little different in the Dimension Wizard. From the Select the dimension table screen, select the time_by_day table from the Available tables list to be the table for the `time` dimension, and click the Next button:

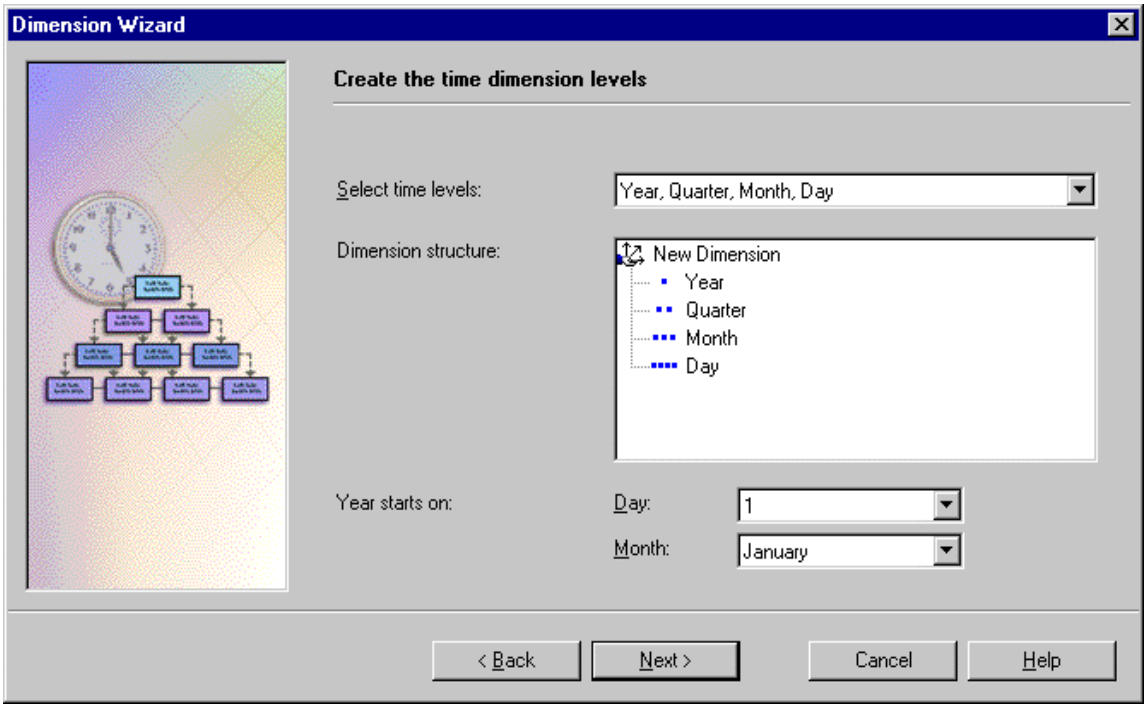


The screen asking you whether this is a time dimension or a standard dimension will pop up. Select the Time dimension radio button, make sure the date-formatted column, the_date is selected in the Date column drop-down list, and click the Next button:



The next screen allows you to define the time dimension levels. From the Select time levels drop-down list, select Year, Quarter, Month, Day. A list below the drop-down list will then show you the dimension structure.

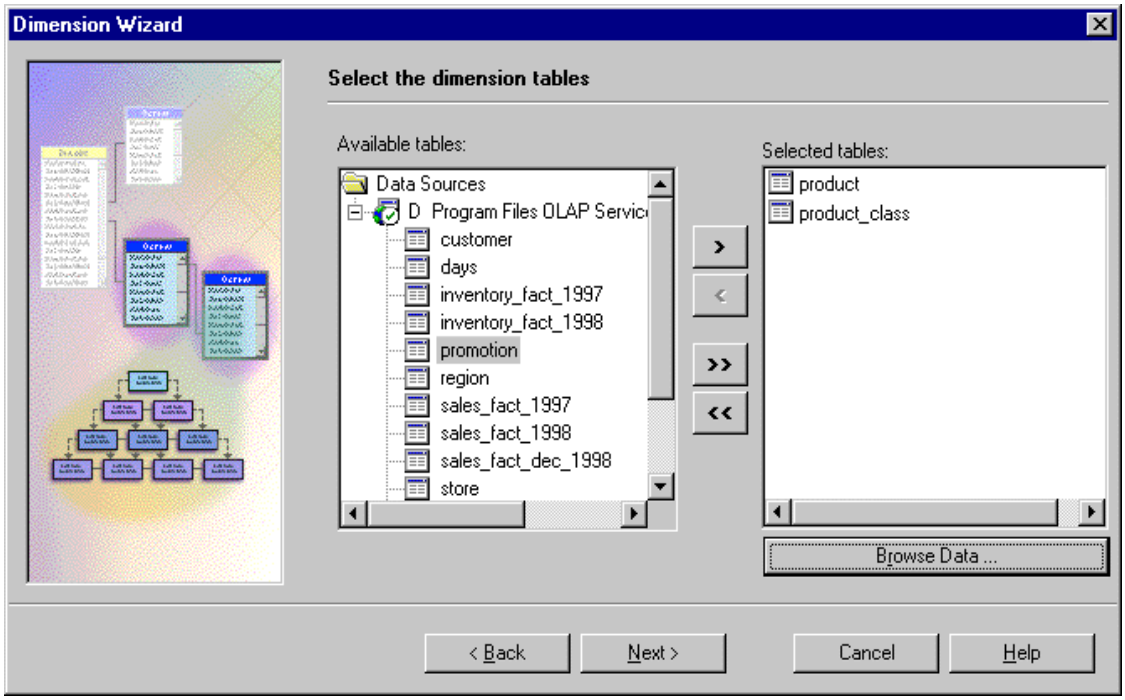
Two drop-down lists called Year starts on allow you to specify the month and day of the start of the fiscal year, in case they are different from the calendar year. This is a great feature. Just imagine how much work this feature will save you compared to having to program this yourself. Let's select the defaults and click the Next button:



Name the dimension as the time dimension, then continue until you get back to the Cube Wizard screen that

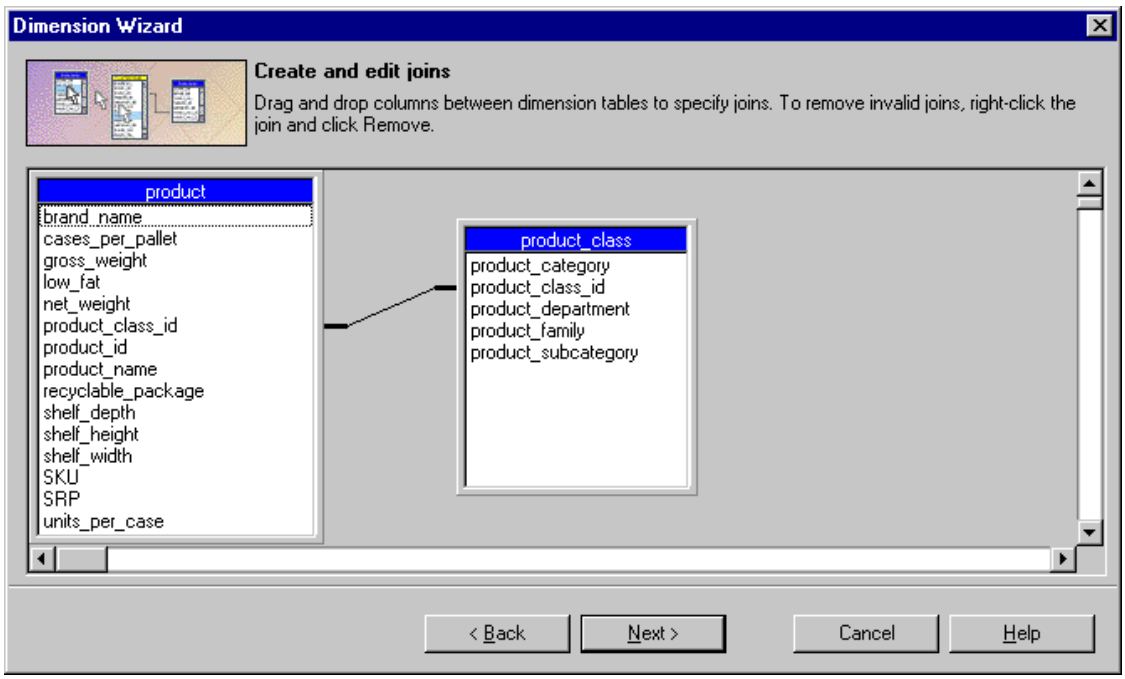
launched the Dimension Wizard. Once you get back to that screen, you should see three dimensions selected in the Cube dimensions list.

Launch the Dimension Wizard one last time to create the product dimension. On the first screen of the Dimension Wizard, select Multiple dimension tables because the levels we will be selecting exist in two tables, `product` and `product_class`. This is a good example of the snowflake schema, because we have the `product_class` dimension indirectly linked to the fact table. Clicking the Next button after you make the selection mentioned above will lead you to the following screen:

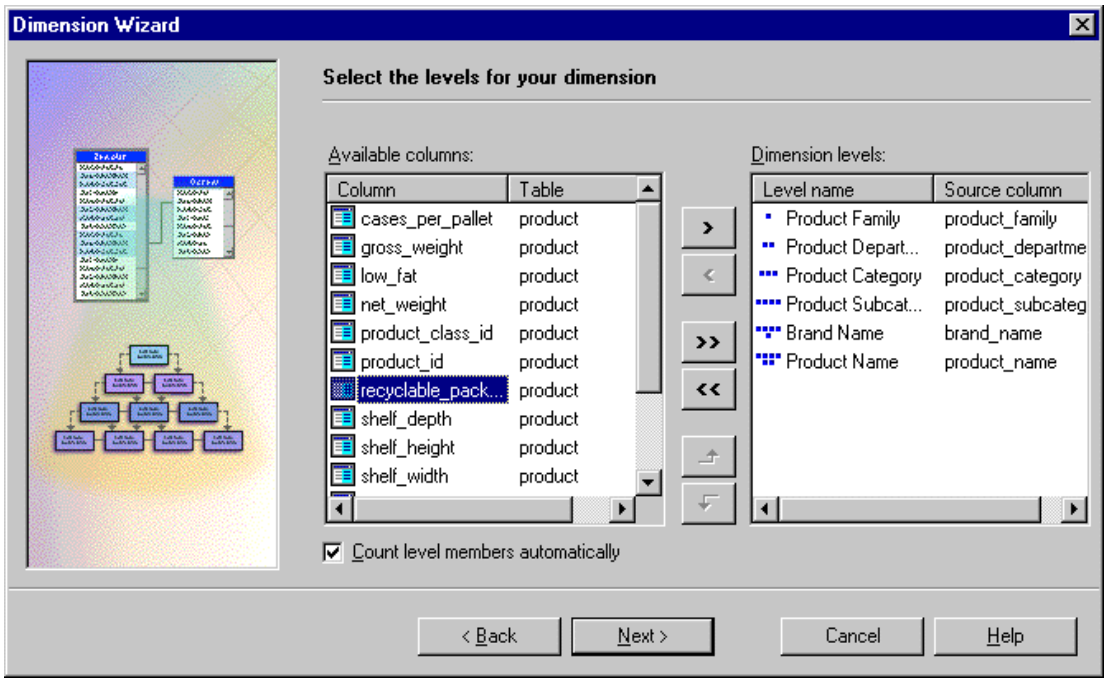


Select the product and product_class tables from the Available tables list, and click on the Next button.

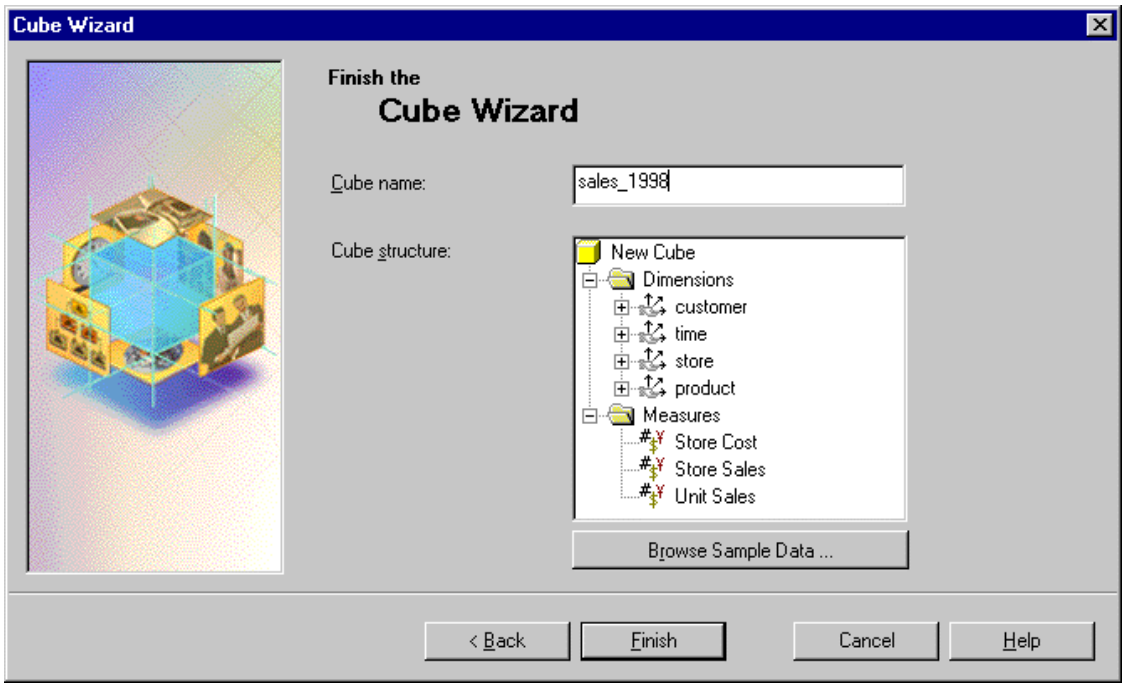
At this point, you will be presented with the following screen. This screen allows you to specify and/or modify the join between the tables selected in the previous step. Since we don't need to make any changes to the join between the product and product_class tables (they're joined on the product_class_id field), let's click the Next button:



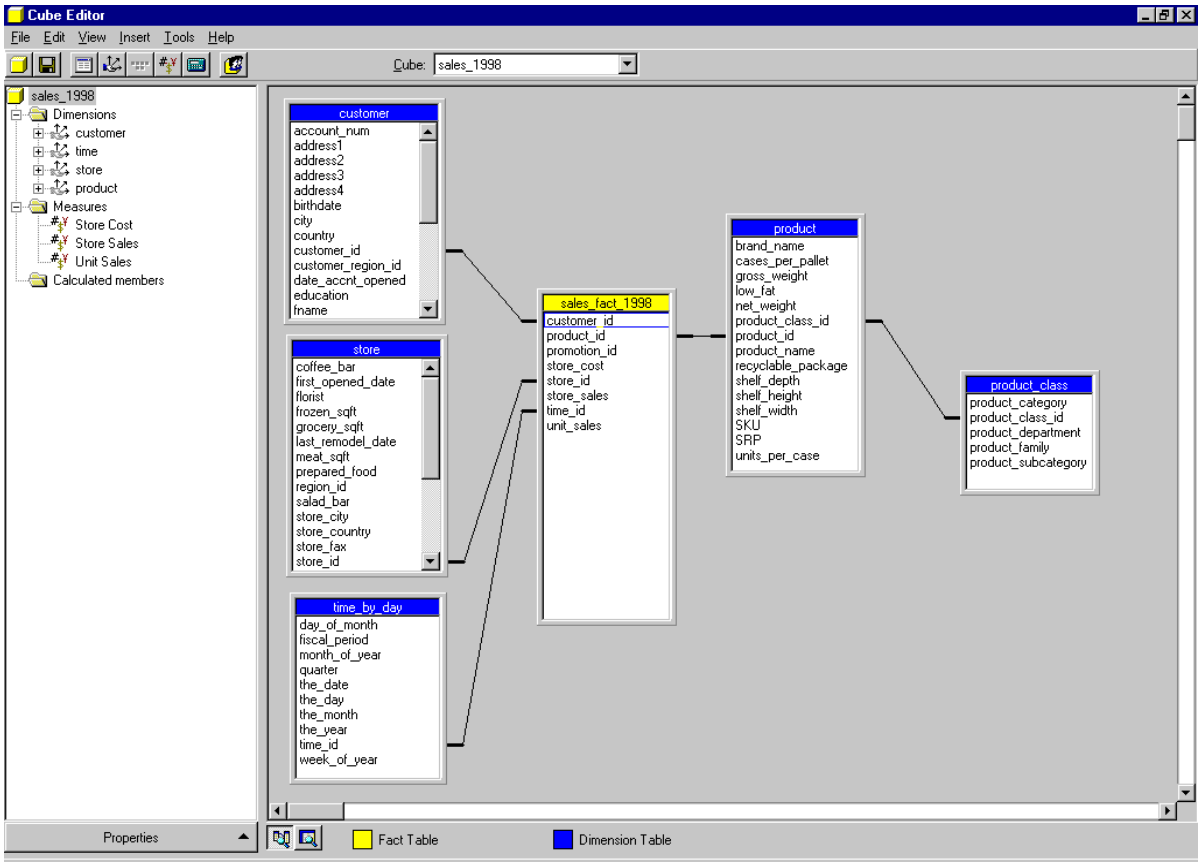
The next screen will allow you to specify the levels of the product dimension. These levels are: Product Family, Product Department, Product Category, Product Subcategory, Brand Name and Product Name. The number of blue dots shown to the left of the selected levels in the selected level list, indicates the ordinal number for the corresponding level. For our example, the first level is the Product Family whereas the fifth level is the Brand Name. Click the Next button to name the dimension as product, and proceed with creating the dimension until you reach the Cube Wizard again:



Clicking the Next button on the Cube Wizard screen will lead you to the screen where you can name your cube. Let's name it sales_1998. Notice the small window that shows you the structure of the newly created cube on this screen. You can expand and collapse the view to make sure you did everything correctly:



When you click the Finish button on the Cube Wizard screen, you get the Cube Editor screen:



As the name implies, this editor can be used for many useful tasks, such as:

- Modifying the cube structure
- Designing the cube storage (we will see how this is done in the next section)
- Validating the cube schema
- Optimizing the cube schema
- Managing the cube roles
- Processing the cube so that we can browse its data

Designing Cube Storage

After the cube has been created, we need to design its storage. Will the cube data be kept in a new dimensional database, stored in the original OLTP database, or in both databases. These three options correspond to whether we want to use MOLAP versus ROLAP storage, or a hybrid of the two, HOLAP.

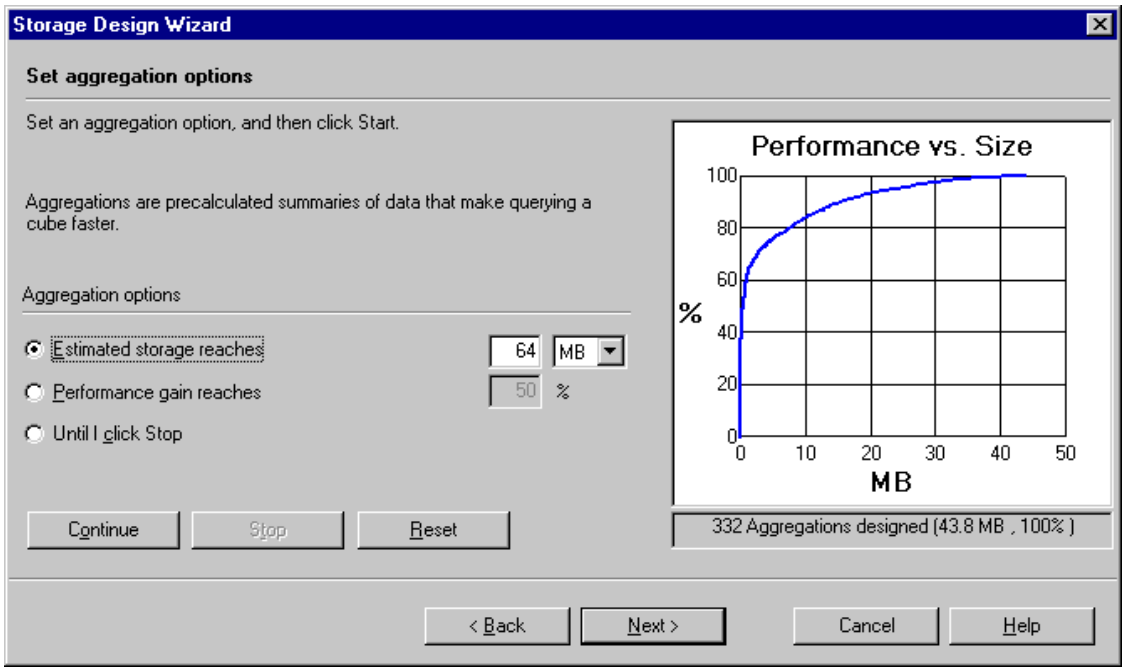
For the purpose of our example, we will be using MOLAP storage to optimize query performance. To design the storage, select the Tools | Design Storage from the Cube Editor menu.

This will start the Storage Design Wizard. Skip the first screen and select the MOLAP option in the next screen. The following screen will allow you to set the aggregation options. These options include:

- Estimated storage reaches x MB: With this option, you can limit the amount of disk space used to store the aggregation results. This option is useful if you have limited disk space, but it may effect query performance if not all possible aggregations are saved on disk. With this option SQL Server's OLAP server will choose which aggregates will be stored.

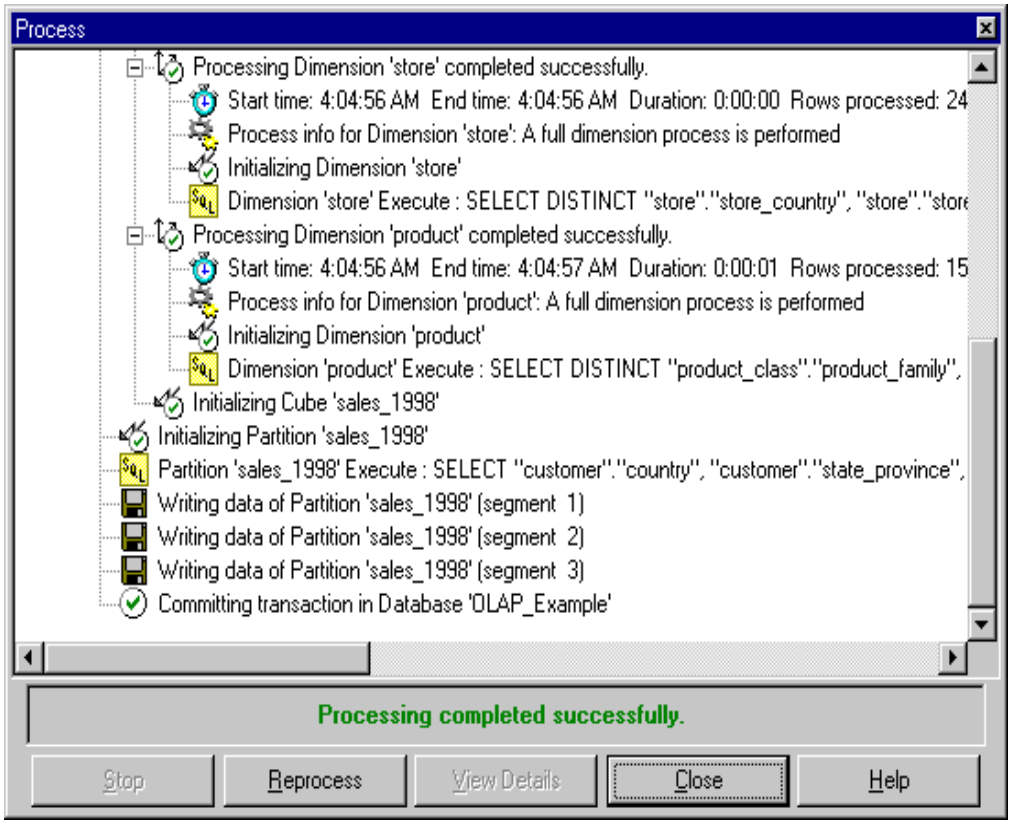
- Performance gain reaches x%: This means, that you are allowing as much disk space as needed to store aggregations, until performance gains due to the aggregation storage reaches x% of the original performance with no storage.
- Until I click Stop: This option allows you to manually set the points which disk storage and performance gains can reach. You do so by watching the Performance vs Size graph and clicking when you see a point that you think would be optimal for you.

Click the Start button and watch the Performance vs. Size graph change, indicating at the bottom, the number of aggregations, storage needed for them, and performance gains. At the end of the run, you will notice that in our case, the maximum number of aggregations is 332. The needed disk space is 43.8 MB. At this point, we will get 100% performance gains. Based on this, let's select the default option and click the Next button:



The next screen will ask you whether you want to process the data, or just save the storage design (and process the aggregations later). Processing the data actually both creates and stores the aggregations for you. Let's choose Process now.

While processing the data, the Wizard will show you a screen that includes information on the progress of the operation:



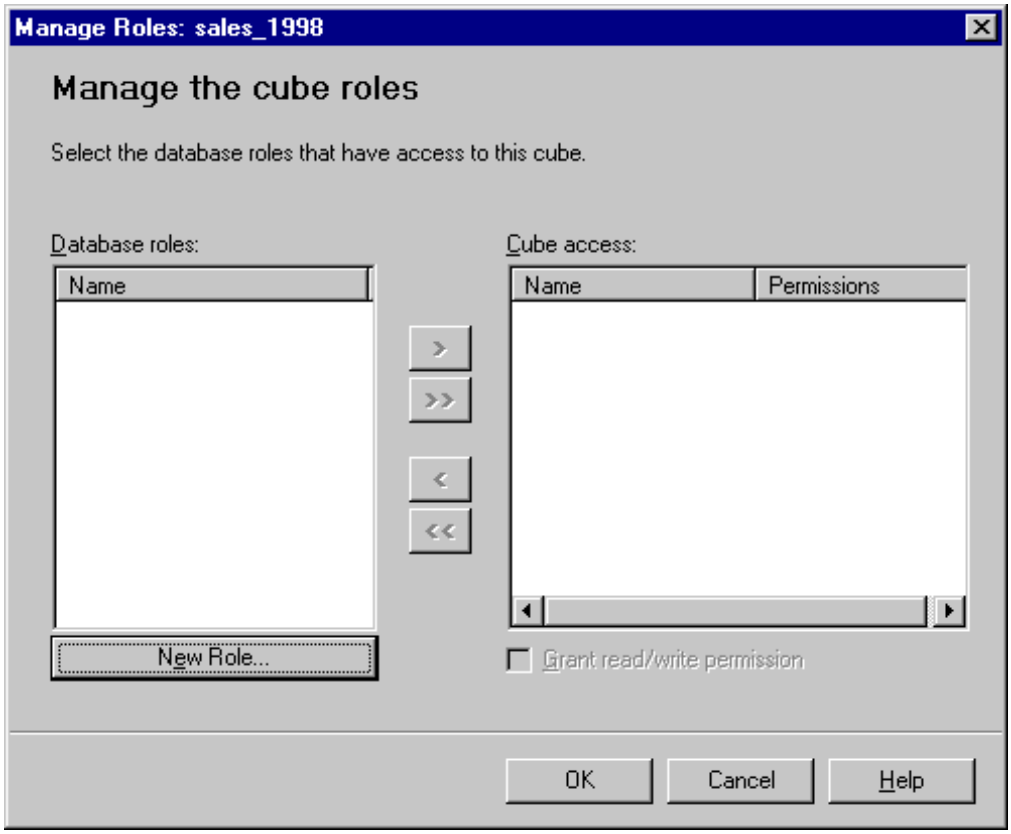
Cube Security

Since the data presented by the cube could be sensitive historical data about the performance of your company, you may want to set up a security scheme as to who can access the cube and use it, and who cannot do so. Cube security is tightly integrated with Windows NT security.

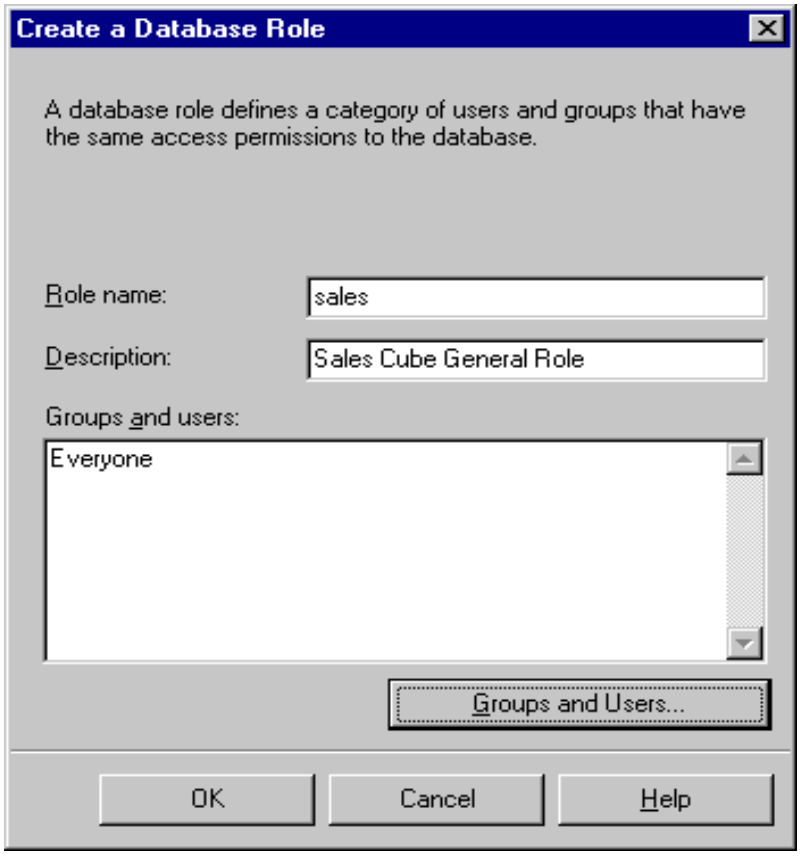
To set up security for the cube you have just created, you need to create a **role**. The role will have certain access privileges to the cube. The role will then be assigned to groups of Windows NT users, setting permissions for them accordingly.

Let's create a new role. To do so, we can either select Tools | Manage Roles from the menu bar of the Cube Editor, or we can close the Cube Editor and add a role to the Roles folder in the OLAP Manager's tree view.

Select Tools | Manage Roles from the Cube Editor's menu bar. This will launch a Manage Role wizard:



Click on the New Role button and you will presented with the dialog shown here:

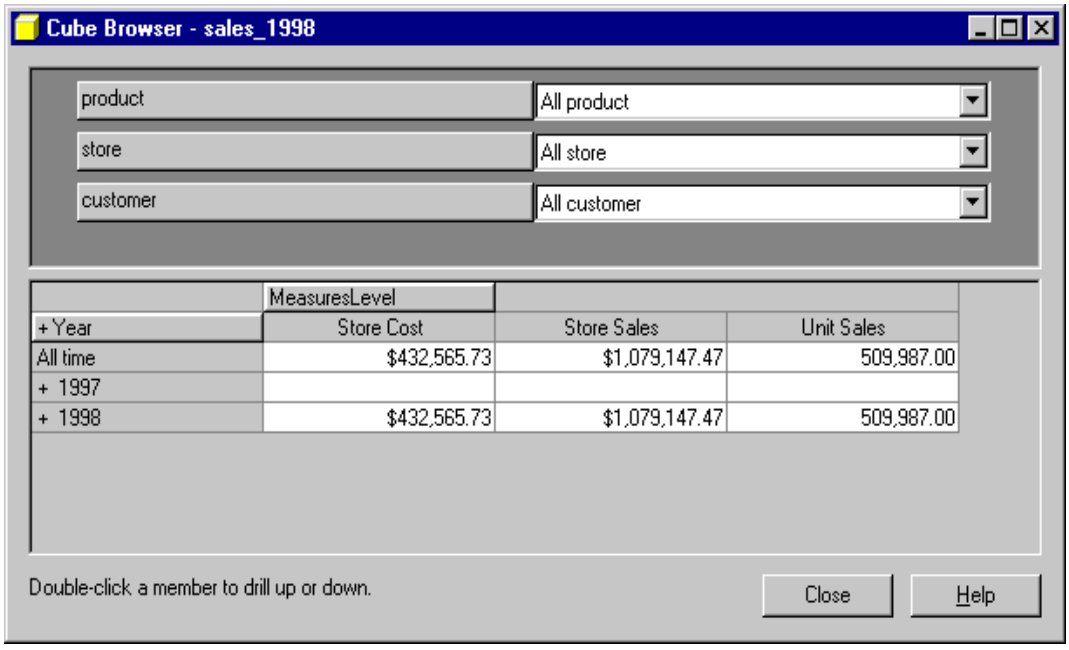


In this screen, you can click the Groups and Users button to add Windows NT users and/or groups. Let's add the Everyone group, then click OK to close the dialog. The new role will then be seen on the Manage Role wizard in the Cube access list with read-only permission. You can grant read/write permissions by checking the option box on the screen. However, for our purposes, we will keep it read-only.

Browsing the Cube Data

Now the `sales_1998` cube has been created, its storage set up, and security taken care of, let's browse the data in it so we can see how the aggregations have been done.

Right-click the sales_1998 cube from the OLAP Manager's tree view menu, and click Browse Data. The following dialog will be shown:

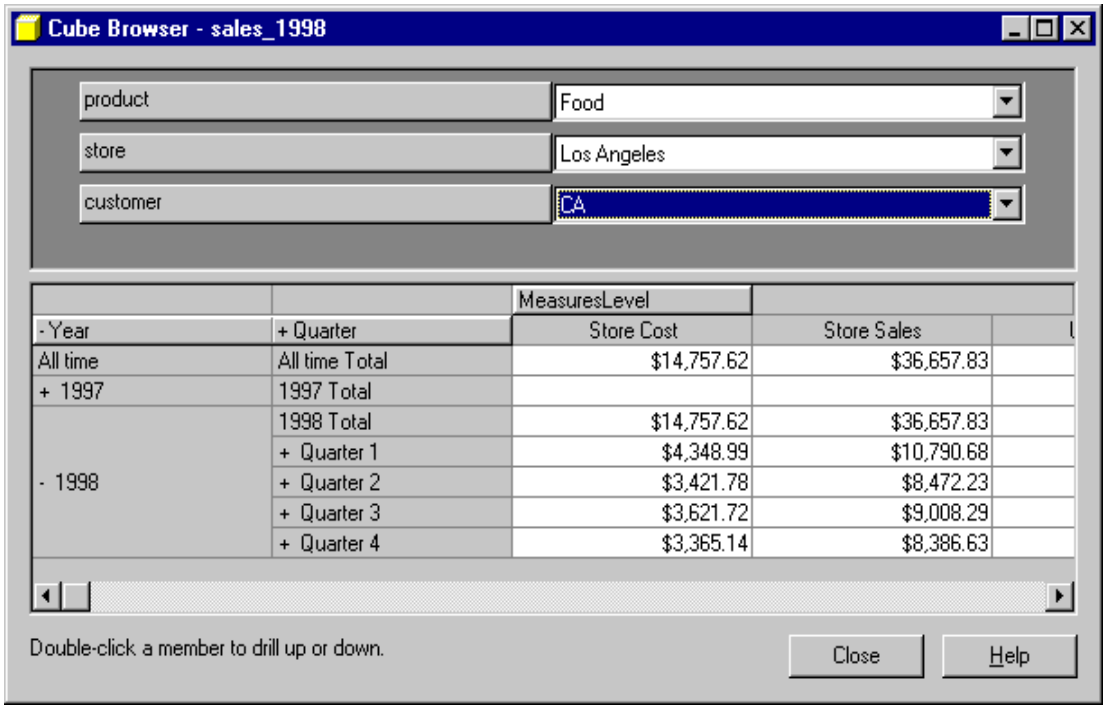


This dialog shows you a spreadsheet with the values of the measure in which we are interested. The left column of the spreadsheet represents one of the dimensions (`time` in this case) and on top, you can see the remaining dimensions created for the `sales_1998` cube. These dimensions are `product`, `store`, and `customer`.

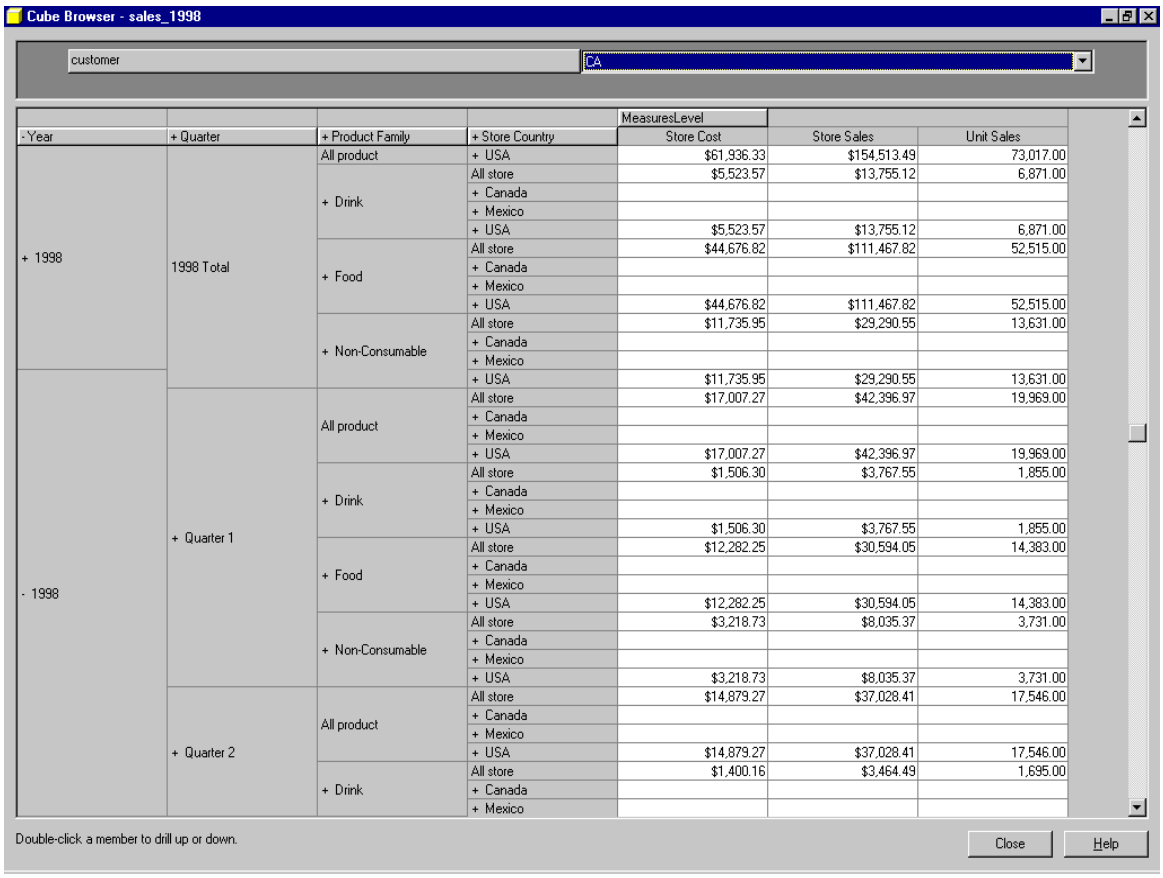
You can drag and drop these dimensions the way you like onto the spreadsheet to show them to the left, and to show how the aggregations are taking place.

Since the fact table in our example stores data for 1998 only, we can see that the measures for 1997 are blank.

To see the measure values by quarter of 1998 for food products, sold in stores in Los Angeles, California to customers, you need to make the selections shown in the drop-down lists and in the time dimension column as shown:



You can also browse data by quarter, by product family, and store country by dragging the product and store dimensions to the spreadsheet and expanding them appropriately, as shown:



Summary

This was an introductory chapter about OLAP services in SQL Server 7.0. OLAP services and data warehousing in SQL Server really deserve much more than one chapter. There are many more topics I wish I could cover. These topics include data mining, using MDX, the SQL language equivalent for data cubes and using programming languages to build custom cube browsers. However, I hope that this chapter will only be a start for you and entice you learn more about this powerful technology and the powerful tools that come with it.

We covered the basics of data warehousing and OLAP at the beginning of the chapter, then I introduced you to the Microsoft SQL Server OLAP Manager and the great wizards and tools that come with it using an example. The example was detailed and probably looked like a tutorial at some instances, but I hope that it will put you on the right track in your endeavor to explore this technology more.

M

O

P

R

S

T

U