

# Data Warehouses and Data Mining

## Chapter Outline

- Introduction, 431
- Two-Minute Chapter, 432
- Indexes, 433
  - Binary Search*, 434
  - Pointers and Indexes*, 435
  - Creating Indexes*, 436
  - Problems with Indexes*, 437
- Data Warehouses and Online Analytical Processing, 437
  - Data Warehouse Goals*, 438
  - Data Warehouse Issues*, 439
- Data Extraction, Transformation, and Transportation, 441
- OLAP Concepts, 443
- OLAP Database Design, 445
  - Snowflake Design*, 446
  - Star Design*, 447
- OLAP Data Analysis, 448
  - Cube Browsers*, 448
  - OLAP in SQL*, 450
  - SQL Analytic Functions*, 455
  - SQL OLAP Windows Partition*, 456
- Data Mining and Business Intelligence, 458
  - Data Configuration*, 459
  - Classification*, 460
  - Association Rules/Market Basket Analysis*, 463
  - Cluster Analysis*, 467
  - Geographic Analysis*, 469
- Summary, 472
- Key Terms, 473
- Review Questions, 473
- Exercises, 474
- Additional Reading, 477

## What You Will Learn in This Chapter

- What is the difference between transaction processing and analysis?
- How do indexes improve performance for retrievals and joins?
- Is there another way to make query processing more efficient?
- How is OLAP different from queries?
- How are OLAP databases designed?
- What tools are used to examine OLAP data?
- What tools exist to search for patterns and correlations in the data?

## A Developer's View

**Miranda:** Faster. Faster. Come on, run faster!

**Ariel:** What? Are you training for a marathon?

**Miranda:** No. It's just these queries they want me to write are taking forever to run. They worked OK when I tested them with small amounts of data. But now, I don't know.

**Ariel:** Maybe you just need a faster computer?

**Miranda:** No, I think I need a different system. These queries are retrieving data, but it is data from many different tables. And these managers

want all of these strange subtotals.

**Ariel:** Wow! There are a lot of totals. How do you expect anyone to read those? I think I see four different levels of totals and that's on one page!

**Miranda:** Yes, and that's only part of what the managers want. I'm happy they are using the system, but I don't see how they can make any sense out of these reports. I think I might need a separate system to reorganize this data and create these reports for the managers. Then they want to do some type of statistical analysis as well!

### Getting Started

Most companies have data and databases. What managers need are tools to organize and analyze the data. Relational databases are good for handling transactions data, but it can be difficult to retrieve and analyze huge amounts of data quickly. So you create a data warehouse with a structure designed to retrieve data quickly. Add cube browsers to explore the data. Add statistical tools to analyze the data, and managers can make better decisions.

## Introduction

**What is the difference between transaction processing and analysis?** Relational database systems were designed to store large amounts of data efficiently. In particular, they are very good at quickly storing and retrieving basic transaction data. Look at the common Sale and SaleItem tables, and you will see data stored compactly. For example, the Pet Store SaleItem table has only four columns and they all contain simple numbers. An individual sale can be recorded or retrieved quickly. Each sale uses a different row, which separates transactions. Each new sale or item purchased can be entered into a new row without affecting any of the other rows or sales. However, this structure causes problems for other types of queries. Queries that involve multiple tables use joins that can require the DBMS to match data values from millions of rows. Think about the number of joins and subtotals required when someone asks the DBMS to analyze the data by computing subtotals on several different factors (such as employee, region, product category, and month). Computing breaks and subtotals across many factors, multiple tables, and millions of rows of data can cause performance problems even on fast hardware.

Vendors of database systems attempted to solve some of these problems by creating indexes on the tables. The indexes make it substantially faster for the DBMS to find specific rows of data within a table, and particularly to improve join performance. An index is a sorted list of the key data that can be searched quickly. However, there is a trade-off: Adding indexes to a table speeds retrieval queries but slows down data updates and transactions because the indexes continually have to be rebuilt. This conflict has led to focusing the existing relational systems for **online transaction processing (OLTP)**; whereas different storage and retrieval systems are used for **online analytical processing (OLAP)**. Data from the OLTP is extracted and cleaned, and then it is placed in a data warehouse. The data warehouse is heavily indexed and optimized for data retrieval and analysis. Additional procedures and routines are available to analyze the data, support interactive exploration by managers, and statistically search it for meaningful correlations and information. This chapter looks at the basic concepts to explain why the different approaches are needed. The data mining section defines some of the basic statistical tools available. The specific details of defining indexes and how to use the tools are covered in the workbooks because the details depend heavily on the specific DBMS. If you want more detailed explanations of the statistics and tools, check out the Data Mining book (<http://www.JerryPost.com/Books/DMBook>). It uses SQL Server and some open-source tools to examine common data mining applications.

## Two-Minute Chapter

---

Relational databases are designed to efficiently store and protect transaction data. Splitting data into separate tables makes it faster and safer to add new rows. But, retrieving the data requires joining multiple tables on primary keys, which can be slow. Managers today need to analyze data—which can require retrieving and summarizing huge numbers of rows. Most systems add indexes to speed retrieval of data. An index is a separate sorted list of data with pointers into the actual tables. Adding indexes reduces retrieval time—which is good for data retrieval and analysis; but bad for data storage because every index requires time to update when data is added or changed. In many cases the best answer is to keep the relational database for transactions but create a new data warehouse that holds copies of the data optimized for data retrieval and analysis. In many cases, extracting and cleaning data from multiple sources is the hardest part of building a data warehouse. The steps need to be automated so data can be extracted on a regular basis.

A data warehouse is often organized in a star design with a fact table at the center, connected to dimension tables that contain attributes of interest to managers. Multi-dimensional cube browsers are useful for enabling managers to browse through data. The cubes display multiple levels of subtotals and managers can interactively select which dimensions and levels to display. Data is often organized in hierarchies (such as time) which managers can roll-up to view totals or drill down into for details. Be cautious when defining computed values—sums are usually fine but averages or computations requiring multiplication can be tricky because you need to specify whether multiplications should be computed first (use a query) or last (in the cube browser). The Microsoft PivotTable is an interactive cube browser that is easy to use and runs inside Excel and can also create interactive charts.

The SQL standard includes modifications to the GROUP BY clauses to display grand totals (super-aggregate totals). The WITH ROLLUP and WITH CUBE op-

tions combined with the GROUPING function are useful to compute these additional totals within SQL. SQL also includes the RANK and DENSE\_RANK functions to assign sequential numbers to sorted data. For example, employees could be ranked by their total sales value for the month. RANK and DENSE\_RANK treat ties differently, where DENSE\_RANK does not skip values so you would get values such as 1, 2, 3, 3 instead of 1, 2, 2, 4.

The SQL PARTITION command is used to create a “window” to examine a moving subset of data. It is particularly useful for computations such as moving averages such as averaging the three most recent data rows or for computing running subtotals. The LAG and LEAD functions provide access to data in rows behind or ahead of the current rows.

Reports are used to display data for common operations and transactions. Queries are used for ad hoc questions, and transaction programming. OLAP functions are used for aggregates, comparisons, and drill-down operations. Data Mining tools are used for deeper analysis and statistical techniques to identify unknown relationships. Common methodologies include classification, association rules, cluster analysis, and geographic analysis. Classification tools include decision trees, Bayesian analysis, and neural networks which attempt to identify how dimensions influence fact measure variables. The classic application of association rules is market basket analysis to see which items are commonly purchased together. Cluster analysis is used to identify categories or groups of items such as grouping customers who have similar features. Geographic analysis is useful for data based on location and often uses mapping systems to display layers of data.

## Indexes

**How do indexes improve performance for retrievals and joins?** Although tables are often pictured as simple lists of rows and columns, a DBMS cannot simply store all data in sequential files. Sequential files take too long to search and

**Figure 9.1**

Find an item in a sequential table. Even if you know the primary key value, the system has to start at the first row and continue until it finds the desired match. On average, with  $N$  total rows, it takes  $N/2$  row retrievals to find a particular item.

ID	LastName	FirstName	DateHired
1	Reeves	Keith	1/29/2013
2	Gibson	Bill	3/31/2013
3	Reasoner	Katy	2/17/2013
4	Hopkins	Alan	2/8/2013
5	James	Leisha	1/6/2013
6	Eaton	Anissa	8/23/2013
7	Farris	Dustin	3/28/2013
8	Carpenter	Carlos	12/29/2013
9	O'Connor	Jessica	7/23/2013
10	Shields	Howard	7/13/2013

require huge operations to insert new rows of data. Examine the short table of employees in Figure 9.1, and consider the steps involved to find the row where EmployeeID is 7. The DBMS would have to read each row sequentially and check the ID until it found the proper match. In this case, it would have to read 7 rows. If there are  $N$  total rows, on average, it takes  $N/2$  rows to find a match. If there are a million rows, a typical search would require reading 500,000 rows! Clearly, this method is not going to work for large datasets. The situation is even worse for inserting new rows of data—if you want to keep the list sorted. The system would have to read each row of data until it found the location for the new row, then continue reading every other row and copy it down by one row. Deletions are actually easy because the DBMS does not really remove the data. It simply marks a row as deleted. Later, the database can be reorganized or packed to remove these marked spaces.

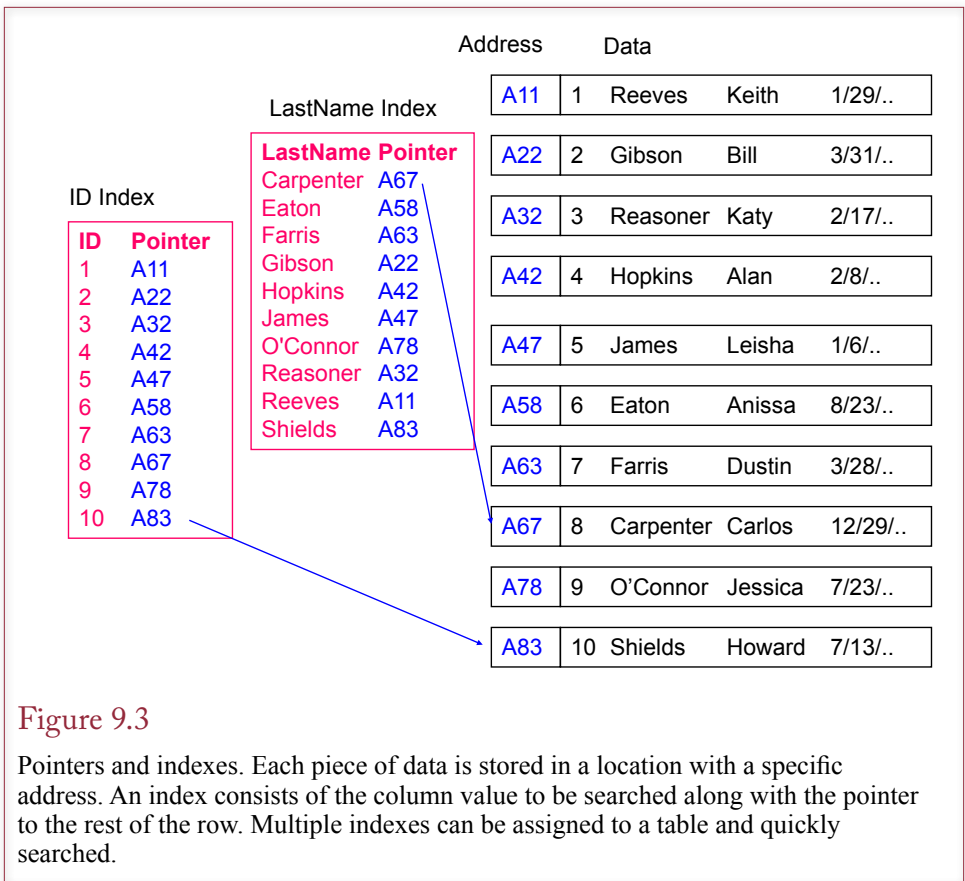
### Binary Search

Looking at the data, it is clear that the DBMS is not taking advantage of all of the information. In particular, if the data rows are sorted, a substantially faster search method can be used to find items. Figure 9.2 shows how to take advantage of the sorting. Think of the process as searching through a paper dictionary or a phone book. Instead of starting at the first page and checking each entry, you would open the book in the middle, then decide whether to search the first half or the second half of the book depending on what name you find in the middle. Finding the middle entry of *Goetz*, you know that *Jones* falls in the latter half of the data. With that one retrieval, you instantly cut your search in half. Following the same process, you would divide the remaining entries in half and search only the appropriate section. In the example, only 4 attempts are needed to find the entry for *Jones*. This **binary search** process continues to divide the remaining data in half until the desired row is found. In general, with  $N$  total rows, a binary search will find the desired row in a maximum of  $m = \log_2(N)$  attempts. Another way to

**Figure 9.2**

Binary search. To find the entry for Jones, divide the list in half. Jones falls below that value (Goetz), so divide the second part in half again. Jones falls above Kalida. Continue dividing the remaining sections in half until you find the matching row.

1 ↓	Adams Brown Cadiz Dorfmann Eaton Farris Goetz
3 ↓	Hanson Inez
4	Jones
2 ↑	Kalida Lomax Miranda Norman



understand this formula is to realize that because you cut the list in half each time, you are looking for  $m$ , where  $2^m = N$ . Now consider a table with a million rows of data. What is the maximum number of rows you have to read to find an entry? The value for  $m$  is 20, which is considerably better than the average of 500,000 for the sequential approach!

### Pointers and Indexes

A binary search is a relatively good way to search data tables that are sorted, so it makes sense when you want to search by primary key, which is common for table joins. But what if the primary key is a numeric CustomerID and you want to search by LastName instead? How can the table be sorted in multiple ways? The answer lies with indexes and pointers.

Data is not actually stored in physical tables. It is usually broken into pieces and stored within a special file. When it is stored, each piece (perhaps an entire row) is placed at an open location and given an address. The address is a **pointer** that tells the operating system exactly where the piece of data is stored. It might be as simple as an offset number that specifies the number of bytes from the start of the file. Figure 9.3 shows that indexes can be created using the column to be searched (ID or LastName) along with the address pointer. The indexes are independent and have been sorted so they can be accessed quickly. As soon as the appropriate entry is found, the address pointer is passed to the operating system

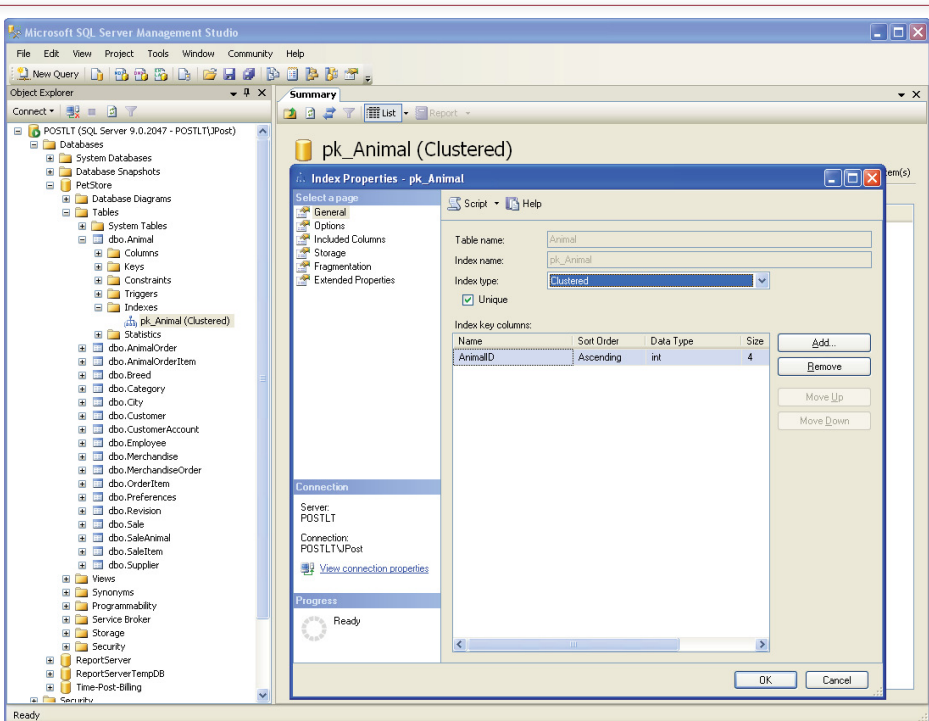


Figure 9.4

Index for primary key. SQL Server automatically generates and maintains indexes for primary key columns. Higher-end systems, such as SQL Server, provide several options to optimize the storage and use of indexes. These options can be used to improve the performance of your queries and applications.

and the associated data is immediately retrieved. In practice, even the indexes are not stored sequentially. They are generally stored in pieces as B-trees. B-trees can be searched at least as quickly as can be done with a binary search, and they make it relatively easy to insert and delete key values. B-trees are explained in Chapter 12, but you do not need to know the details to understand the benefits of indexes. You can create indexes in SQL using the CREATE INDEX command. Bear in mind that the reason for indexes is to substantially reduce the time it takes for the DBMS to find (or match) a particular row of data.

### Creating Indexes

Most systems automatically create indexes for primary key columns, because these are typically used in JOIN statements. Figure 9.4 shows an example of an index created on the AnimalID primary key column of the Animal table in the pet store case. The example is from Microsoft SQL Server, but other systems, such as Oracle, are similar. Microsoft Access also generates primary key indexes automatically, but it has fewer options.

Most DBMSs have a graphical interface tool to create and edit indexes. However, it is relatively easy to use the SQL CREATE INDEX command. Figure 9.5 shows the basic format. The command is straightforward, since you just list the table name and the columns you want in the index. The example shows a composite

```
CREATE INDEX ix_Animal_Category_Breed  
ON Animal (Category, Breed)
```

Figure 9.5

SQL CREATE INDEX command. The basic syntax is straightforward. Give the index a unique name, then specify the table and columns to be used. Most systems support additional options to control the details such as storage and type of index.

index that uses multiple columns. Most systems support additional keywords that control the various options, such as whether the index is unique, how and where it should be stored, or how to handle columns with long data types. In this example, you could include options to specify that the index contains unique values, or to store the index on a separate disk drive partition. These options are different for each DBMS, so you will have to read the documentation carefully to decide which ones you need. Alternatively, most systems provide a query optimizer that will automatically suggest indexes for you to add, with the desired attributes.

### Problems with Indexes

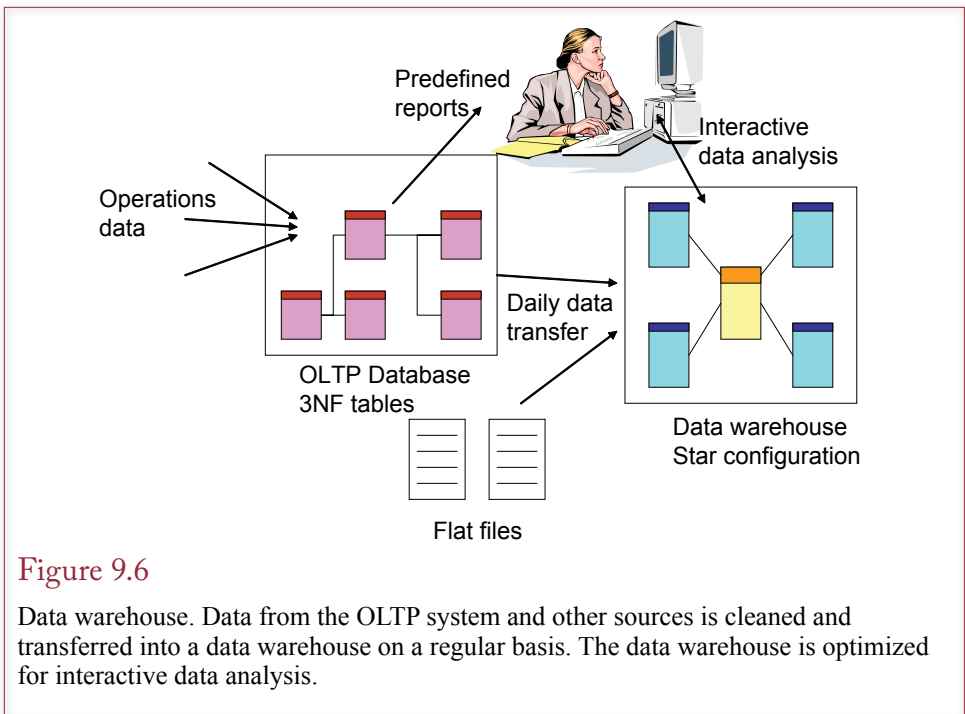
Consider a table in which 10 indexes (columns) are defined. When a new row of data is added to the table, every index has to be modified. At a minimum, the database has to insert a new row into each of the 10 indexes. In most cases, it will also have to reorganize each index and probably update the statistics tables for the indexes. An index substantially improves the ability to search a data table. But for every index you create, the DBMS will slow down every time new data is entered or modified because the indexes have to be rebuilt. So your big decision is which columns to index.

Adding more indexes speeds up data retrieval but slows down data entry and data updates. This conflict is the heart of the problem between data analysis and transaction processing. Transaction processing—collecting the data—needs to be fast to efficiently store and protect the data. On the other hand, data analysis focuses on retrieving existing data and often needs to retrieve huge amounts of data quickly. Building multiple indexes and duplicating data are two ways to vastly improve data retrieval speed, but at the cost of interfering with collecting the data. A common solution is to create a data warehouse—which holds a copy of the data used just for data analysis.

## Data Warehouses and Online Analytical Processing

**Is there another way to make query processing more efficient?** Ultimately, the trade-offs with indexes can be insurmountable. To perform complex searches, you need many indexes on every table. But too many indexes slow down the transaction processing. Additionally, a typical organization has data stored in several different databases and sometimes other files. Obviously, the transaction systems need priority—without them, the business cannot operate. But managers increasingly need to perform complex analyses of data. The solution: Keep the transaction systems and create a new database for managers to perform online analytical processing.





Increasingly, managers want more than the traditional reports that are produced by OLTP systems. Managers want the ability to interactively examine the data. They do not always know what questions to ask or what they are looking for. They need the ability to quickly look at different views of the data. These types of queries can involve huge amounts of data and require joins across multiple tables. Fortunately, the access is almost always read-only—very little data is altered—and read-only queries can be several times faster than updateable queries. Managers also want the ability to statistically analyze the data, and these tools generally need to know something about the layout and structure of the data. The answer is to put a copy of the data into a new fixed structure.

### Data Warehouse Goals

Many organizations have chosen to avoid these conflicts by creating a second copy of the database. A **data warehouse** holds a copy of the transaction data in a special database that is dedicated to answering managerial queries. Data may come from various sources, but all of it has been cleaned so that it is consistent and meets referential integrity constraints. Data can be stored in pre-joined format, resulting in duplication of data. But, since this data is not updated directly, and since storage space is relatively inexpensive, the duplication is well worth the increased performance. A second option is to build multiple indexes on every table. Again, since the data is not being continually updated, the indexes are rarely updated. In both cases, special functions and query controls are included to rapidly create different views of the data. Generally, data is transferred from the transaction system once or twice a day and moved in bulk to the data warehouse.

The basic concepts of a data warehouse are shown in Figure 9.6. The transaction databases continually collect data and produce basic reports, such as inventory and sales reports. The data warehouse represents a separate collection of the

data. Although it might use the same DBMS, it requires new tables. On a regular basis, data is extracted from the transaction databases and from other files. This data is checked to make sure it is consistent; for example, all of the key values must match for referential integrity. Then it is added to the data warehouse, which usually does not store data in normalized tables. Instead, it has special structures like the star configuration. In these cases, data is often duplicated. For example, the same city and state combination may show up in thousands of data records.

Online analytical processing is usually related to data warehouses, but technically, you can build OLAP systems on transaction databases without using the intermediate data warehouse. A bigger challenge is that each vendor offers different technology and different implementations. In general terms, OLAP consists of a set of tools to browse the data and to analyze and compare data in the database.

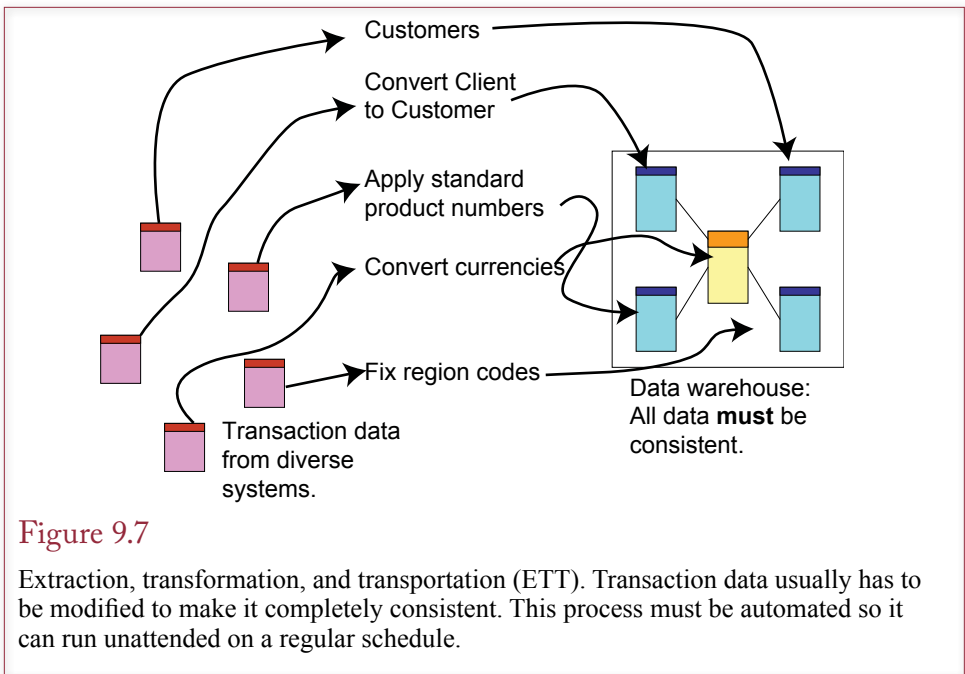
Managers are also learning to use statistical tools to perform more formal analyses of data. **Data mining** or **business intelligence (BI)** tools use automated or directed statistical methods to search the data for patterns and relationships. The statistical tools include regression, discriminant analysis, pattern recognition (e.g., neural networks), and database segmentation (e.g., clusters, k-means, mixture modeling, and deviation analysis). These tools generally require substantial computing power and extremely high-speed data retrieval. Even with current high-speed systems, many of the techniques would need days or weeks to analyze some of the large datasets that exist. The point is that if users want to work on this type of analysis, the databases will have to be configured and tuned to their specific needs.

## Data Warehouse Issues

Despite advances in database management systems and improvements in computer hardware, some queries take too long to run. Additionally, many companies have data stored in different databases with different names and formats, or even data stored in older files. The purpose of a data warehouse is to create a system that collects this data at regular intervals, cleans it up to make it consistent, and stores it in one location. A second primary goal of a data warehouse is to improve the performance of OLAP queries. In most cases, performance is improved by denormalizing the data. Joining tables is often the most time-consuming portion of a query, so new data structures are created that perform all of the joins ahead of time and store redundant data into fewer tables.

Three main challenges exist in creating a data warehouse: (1) Setting up a transfer system that collects and cleans the data, (2) Designing the storage structure to obtain the best query performance when handling millions or billions of rows of data, and (3) Creating data analysis tools to statistically analyze the data. Most companies choose to purchase data mining software for the third step. Few organizations have programmers with experience writing detailed statistical analysis procedures, and several companies sell prepackaged tools that can be configured to search data for patterns. The second issue—OLAP design—is discussed in the next section.

Cleaning and transferring data is often the most difficult part of establishing a data warehouse. Figure 9.7 shows the process known as **extraction, transformation, and transportation (ETT)**. You will quickly find that most companies have many different databases, with different table and column names, and different formats for the same type of data. For instance, one database might have a column `Customers.LastName` declared at 20 characters, and a second database uses `Cli-`



ents.LName set at 15 characters. The process of extracting data from these sources needs to be automated as much as possible; it is too hard and too expensive to try to clean data by hand. Remember that the data has to be transferred on a regular basis—at least daily. So you often have to write complex queries to merge data from different sources. In this small example, you would probably import one table (e.g., Customers), and then run a NOT IN query to get the list of names that are in the Clients table but not in the Customers table. These new names would then be added to the data warehouse. Some of the DBMS vendors have created import tools that will help you automate these data comparisons, but ultimately, most companies end up writing custom code to handle this complex process. For instance, Microsoft uses SQL Server Integration Services (SSIS). A key element in the process is to extract the data from the OLTP systems without interfering with the ongoing operations. Specialized tools and queries utilizing parallel processing on multiple-processor machines are often used in this step, but the details depend on the DBMS, the hardware, and the database configuration.

One method that can sometimes be used to reduce the data volume is to extract and transfer only data that has been changed since the last transfer. However, this process requires that the OLTP system track the date and time of all changes. Many older systems do not record this information for all elements. For instance, a sales database has to record the date and time of a sale, but it probably does not record the date and time that a customer address was changed.

Transforming the data often involves replacing Null values, converting text to numbers, or retrieving a value from a joined table and updating a value in the base table. All of these operations can be handled by SQL statements, and you will have to create modules that can be executed on a regular basis to extract the data, clean the data, and insert it into the new database.

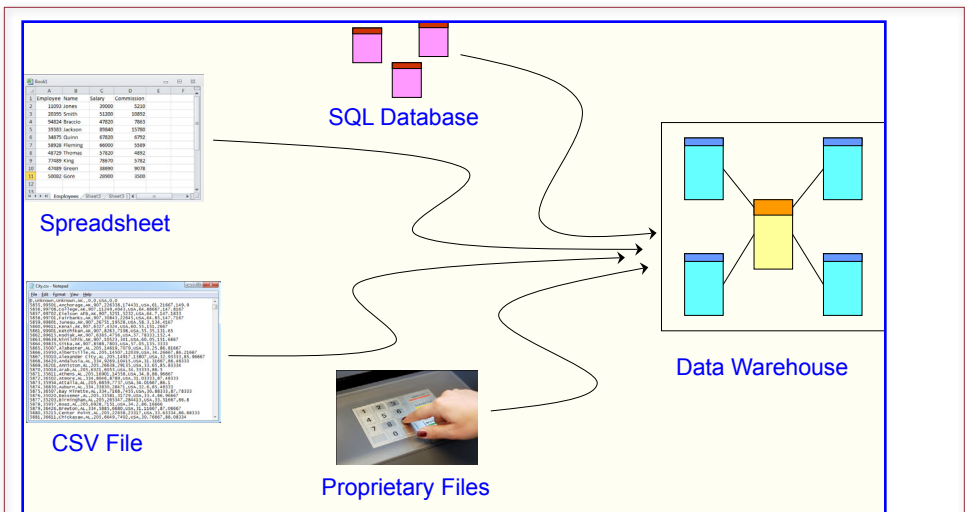


Figure 9.8

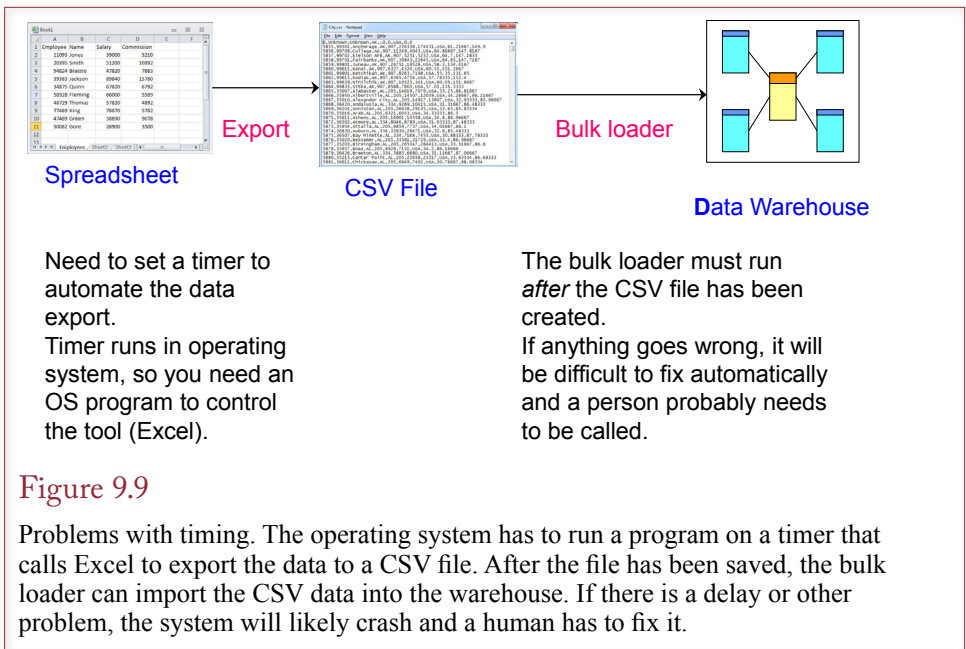
Data sources. The ETL process has to be automated so data can be extracted and loaded automatically every day. SQL sources are generally easy because tables can be linked and used directly. CSV files are relatively standard and can be handled with data loaders. Spreadsheets and other proprietary files can cause problems.

## Data Extraction, Transformation, and Transportation

**How is data loaded into the data warehouse?** One of the most difficult tasks in creating a data warehouse is setting up the extraction, transformation, and transportation (ETT) or loading (ETL) of data. Basically, you need to find all of the sources of data, find a way to extract it from its existing format, transform the data so it is internally consistent with every other piece of data, and load it into the data warehouse. More importantly, you have to create programs and tools so the entire process is automated. The ETL processes need to run on a timed basis (such as once a day). They need to run automatically, with no human intervention. As a database developer, it will be your responsibility to create and tests programs to handle these tasks. In large projects, developing all of the tools can easily take several months.

Figure 9.8 indicates the importance of focusing on the main sources of data: SQL databases, CSV files, spreadsheets and proprietary files. SQL databases are the easiest to handle. Most of the major DBMSs can be configured to connect to “external” databases. Hence, you can create connections from the data warehouse to the other databases. Once linked, you can write SQL statements to compare, transform, and copy data from the linked table into the warehouse tables. Another standard file type is the **comma separated values (CSV)** file. Data is stored sequentially in rows. The columns are separated by commas; although most tools enable you to change the delimiter to something else. For example, you might want to use tabs (ASCII character 9) in case the text data in a column happens to contain commas. In general, the bulk loaders make it relatively easy to import CSV data files.

Excel spreadsheets and other proprietary formats can be more challenging. In both cases, you might have to use the original tool (e.g., Excel) to save the data



to a CSV file then call the data warehouse bulk loader to import the CSV file. The problem with this approach is that it is more difficult to automate. As shown in Figure 9.9, you need to write a program in the operating system that uses a timer to start Excel and call an Excel macro to export the data to a CSV file. After the file has been saved, the program calls the bulk loader to import the CSV data into the warehouse. Then the warehouse programs can run to extract and transform the data. You need to know how to use several different programming tools to even create this process. It will be difficult to write a program to automatically catch all errors and fix them on the fly. More likely, if something goes wrong, the program will crash and you will be called to fix the problem. And those calls always come at 3 AM.

Any system that relies on multiple steps across different machines, operating systems, and software will have to be modified almost any time one of the components changes. For example, when Microsoft updates Excel or Windows, the programs will have to be tested and probably modified. The goal here is not to scare you (well, maybe a little); but to help you understand some of the challenges to developing ETL programs—and why they take so long to create and test.

One of the goals in building an ETL system is to get the data into a SQL data source as early as possible. Once the data is in relational tables, you can use the full power of SQL to compare and transform the data. You will make heavy use of SQL commands of the form: `INSERT INTO warehouse_table (...) SELECT ... FROM linked_table`.

Remember that the `SELECT` statement can transform the data as it extracts it. Also, you can use `NOT IN` or `LEFT JOIN` clauses to choose only data that is missing or is not already in a second source table. If data needs several processing steps, you might have to write stored procedures or functions to perform more complex calculations. SQL Server has some useful tricks for creating temporary tables within functions and procedures. It is always best to stick with standard

SQL commands, but sometimes you need to rely on the more complex programming tools available.

The main step is to extract and transform the data so that it is internally consistent. Missing (Null) values are sometimes acceptable, but unmatched data is not. For instance, you cannot have a CustomerID in Sales data that lacks a related key in the Customer table. If the data all comes from a relational DBMS with referential integrity constraints, this problem is minimized. When the data sources include multiple databases, spreadsheets, and CSV files, all of the referential integrity constraints have to be built and tested as the data is loaded.

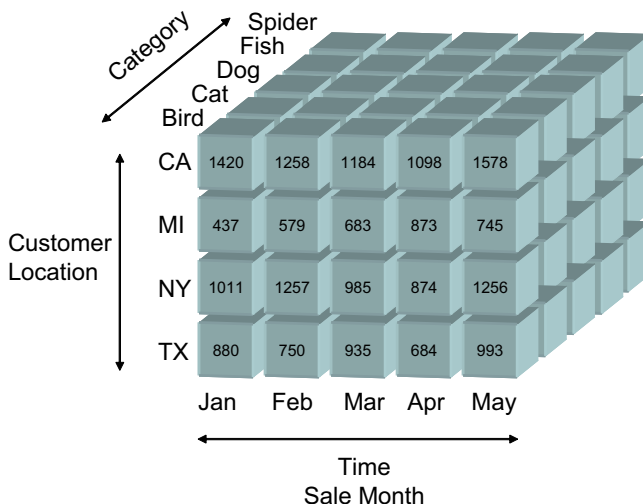
Once the data is consistent, the data warehouse has tools to define fact and dimension attributes. Most warehouses also support renaming attributes, adding descriptions, and assigning formats so that the attributes are easier to understand. For instance, if the database files use abbreviations such as CID or EID, you can assign the more descriptive titles CustomerID and EmployeeID.

## OLAP Concepts

**How is OLAP different from queries?** Probably the most important goal of OLAP is to make the data accessible to managers. They should be able to browse through the data without having to write queries. The concept of the multidimensional cube shown in Figure 9.10 turns out to be a useful approach for many problems. The cube contains data about a specific **fact** (such as sales), and the **dimensions** (sides) represent factors that are potentially interesting to the managers. You could write queries to retrieve all of the data. In fact, the cube is probably defined by a query. However, managers do not want to write queries, and no one wants to assume that managers are going to write accurate queries. Consequently, managers use specific tools to examine the data interactively. For instance, Microsoft provides the PivotTable browser for use on the desktop. It can connect to any

**Figure 9.10**

Multidimensional cube. The fact element is sales. The dimensions are location, time, and category. Managers are interested in various combinations of the dimensions, and can use a cube browser to look at various subtotals.



common data source and enables managers to interactively see sections of the data and subtotals. Other vendors (and Microsoft) provide additional browsing tools.

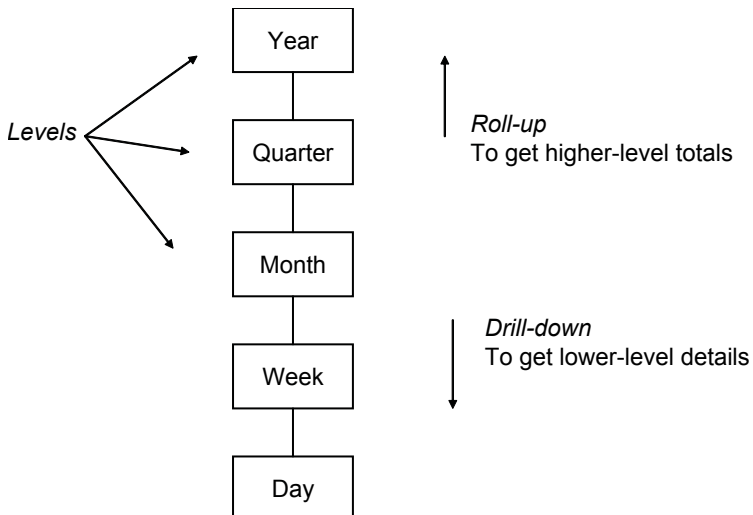
To illustrate the process, consider a simple example from the Pet Store database. Managers are interested in adoptions of animals. In particular, they want to look at adoptions by date, by the category (cat, dog, etc.), and by the location of the customer (state). The attribute they want to measure is the sale price, but you can also create more complex facts, such as price times quantity for merchandise sales. Figure 9.10 shows how this small query could be pictured as a three-dimensional cube. The OLAP tools enable managers to examine any question that involves the dimensions of the cube. For instance, they can quickly examine totals by state, city, month, or category. They can look at subtotals for the different categories or details within individual states. Currently, the front face of the cube shows sales subtotals by state and month. The cube browser makes it easy to rotate the cube to display a different face—such as sales by category over time, or category by state. Users can also examine just one slice of the cube, such as sales by location and category for a specific month. All of these options are performed without asking the manager to write SQL. The desktop tools support drag-and-drop operations to choose the dimensions to be compared.

The OLAP tools also support the ability to look at tools or to change and look at details. Managers might want to start with high-level subtotals and **drill down** to see the details. For instance, a manager might be looking at total sales by month and spot a drop in a particular month. He or she can drill down to see the details of sales by category or location within that month. The opposite of drill-down is to **roll up** the data into totals or averages. Instead of looking at detail sales for a given state, the manager might want to see the totals for an entire month.

A **data hierarchy** is another common element in OLAP. Many dimensions have an explicit hierarchy of values. For instance, Figure 9.11 shows the common

**Figure 9.11**

Drill down and Roll up. In a given dimension, drill down provides more detail. Roll up aggregates the values from subcategories.



hierarchy for dates. A given event (e.g., sale) occurs on a specific date, but that date is defined by the year, quarter, month, or week in which it occurred. Managers might want to examine data at any level within the hierarchy, or drill down or roll up as they are looking at one level. Several standard hierarchies exist in business data—such as dates and locations—and most systems know how to generate these levels automatically. However, the tools also enable you to create custom hierarchies for specific types of data.

Once the OLAP database is defined, users need tools to analyze the data. A cube browser is important because it enables users to look through the data and follow interesting observations. Statistical tools fall into the category of data mining or business intelligence. Vendors provide several versions of tools, some are more automated than others. The goal in all cases is to identify potentially interesting patterns.

## OLAP Database Design

---

**How are OLAP databases designed?** Database design for OLAP is different from traditional database design. Some of the concepts are similar, but ultimately, most OLAP tools store the data in cube structures instead of relational tables. Additionally, OLAP design hides table joins from the end user. The manager sees only the cube. Consequently, the heart of OLAP design is to identify: (1) Facts to be measured, (2) Dimensions to be evaluated, and (3) Data hierarchies. The remaining design issues consist of choosing the best way to organize these elements.

Facts are relatively easy to identify. In a business context, a fact is often a dollar value, but you can also include counts of items, such as the number of items sold. In any case, you can simply ask the managers what items need to be measured. All facts must be **measures**—that is, they must be numeric values. For that reason, you cannot include categorical data (such as “small,” “medium,” or “large”). Some systems support multiple facts within a single cube, but they should be related. For instance, you might include the count of the number of items sold with the value of the items sold. Be careful to identify these values with distinctive and accurate names so users clearly identify the correct role of the data.

You need to be careful when the fact is a computed value. The problem is that you need to control the computational order. What happens if you build the cube using the original SaleItem table? Then you could only use Quantity and SalePrice as measures. It would be tempting to create a calculated measure:  $\text{Amount2} = \text{Quantity} * \text{SalePrice}$ . However, this approach can lead to incorrect results. It is critical that you understand the difference between these two approaches. The correct method is to build a query for any computation that needs to be done on a line-by-line basis ( $\text{Price} * \text{Quantity}$  is a common example). If you wait and build it in the OLAP design cube as a calculated measure, then the cube will (1) slice the data, (2) subtotal any measures separately (Price and Quantity), then (3) perform your calculations:  $\text{Sum}(\text{Price}) * \text{Sum}(\text{Quantity})$ . So your calculations will be performed on data that has already been totaled. Figure 9.12 shows the difference with a small example. When you use a query for the fact table to compute the multiplication, the columns are multiplied first and then summed, giving the correct total of \$23.00. If you use the original table as the fact table and specify the computation as the cube’s calculated measure, the cube first adds the quantity and price columns and then performs the multiplication, giving the incorrect result of \$45.00. The solution is detailed line-by-line computations in a query and to use that query as the fact table.



Quantity	Price	Quantity*Price
3	5.00	15.00
2	4.00	8.00
<b>5</b>	<b>9.00</b>	<b>45.00 or 23.00</b>

Figure 9.12

Order of computations. Multiplications should be performed in a query that is used for the fact table to get the correct total of \$23.00. Computing it in the cube calculation causes sums to be computed first and then multiplied to give the incorrect value of \$45.00.

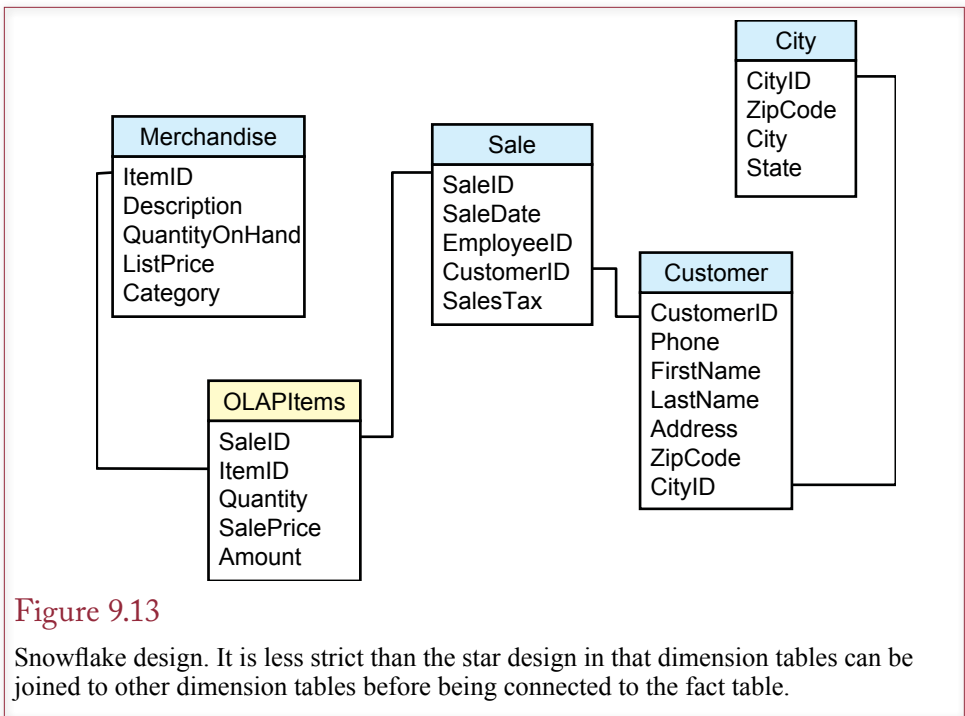
The second step is to choose the attributes or dimensions that form the sides of the cube. The dimensions come from columns for which the users want to compute subtotals. In one sense, an OLAP cube is like a SELECT statement with multiple GROUP BY statements. Any item that would appear in the GROUP BY clause becomes a dimension. The user gets to dynamically choose which dimensions to include at any time. The catch is that you have to find all of the tables that contain the desired dimensions and be sure they are linked to the desired facts.

The third step is to identify and generate all of the desired hierarchies within the dimensions. Some dimensions (e.g., dates) have well-known hierarchies. In other cases, you will have to talk with users to identify the desired levels and create the hierarchies manually. Each OLAP tool has a different method for defining hierarchies, so the actual steps are not covered here.

### Snowflake Design

Once you have identified the facts and dimensions needed for a cube, you can construct the cube within the OLAP tool. Although the details vary, two general models are commonly used to store the data: the snowflake and star designs. The **snowflake design** is similar to a traditional relational design, so it is easy to understand. However, it might not be the most efficient design. Both designs begin with the fact table to define the desired measures. The difference lies in how the dimension data is stored and accessed. With the snowflake design, the system uses predefined joins to connect any tables. As shown in Figure 9.13, you can connect tables through other tables. For instance, City connects through Customer, which connects to the Sale table. The data remains in the original normalized tables, and all columns in the tables are available to be used as dimensions.

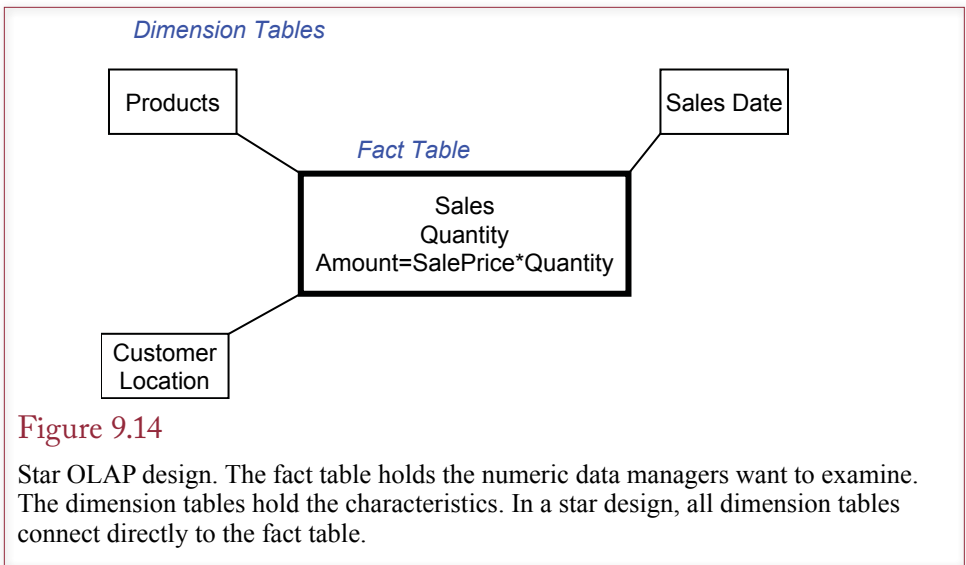
The difficulty with the snowflake design is that the OLAP browser needs to process the joins, which can require considerable computational power and time. Systems that use this approach rely heavily on indexes to reduce the access times. Often, the data is moved out of transaction tables, into a read-only set of tables. Since the data is rarely updated, the system can create a huge number of indexes without worrying about needing to update them because insertions and deletions are not supported. When data is transferred from the OLTP system, the indexes are removed, the data loaded, and the indexes are rebuilt at one time. Some tools also add internal pointers within the data, essentially integrating the indexes into the data for even faster performance.



## Star Design

The **star design** focuses on speeding up data retrieval, essentially by removing joins. It accomplishes this task by denormalizing the data. Essentially, it saves duplicate data. In the standard sales example, the customer data would be entered for every sale. If you could scroll through the raw data, you would see the customer location repeated for every sale. Obviously, this approach requires more storage space. However, remember that insert, update, and delete are not supported on the individual items. Consequently, the problems discussed in Chapters 2 and 3 that are caused by non-normalized data are avoided. Figure 9.14 shows the star design for the sample sales problem. Once you understand the users' goals, the star design is relatively easy to create. You simply identify the fact measures and the dimensions. The system then copies all of the needed data to place the dimensions close to the fact measures. If you add enough dimension tables, you will see the reason for the star name. The fact table sits in the center and is connected to the dimension tables through rays. On the other hand, the snowflake design begins the same way, but you can add tables that connect through other dimensions instead of directly to the fact table. This extended pattern with multiple levels leads to a snowflake appearance.

Which design is better? This question is beyond the scope of this book, because it is a difficult question to answer. In fact, vendors continue to argue over the benefits and weaknesses of each method. They both work best for non-transaction data that is bulk-updated on a regular basis. Both require the storage of additional information (either indexes or duplicate data). In the end, performance depends on multiple factors. If you are thinking about buying a new system, you need to test your specific data with various systems and decide which approach works best in your situation. In terms of configuration, it is easiest to think in terms of the star design. Identify the facts and connect the dimensions directly to the fact table.



## OLAP Data Analysis

**What tools are used to examine OLAP data?** Beyond transaction processing, managers collect data to assist in making decisions. Many levels and types of decisions exist in business, so many different tools exist, with new ones created every year. Two general categories of tools exist: (1) Cube browsers, and (2) Statistical tools used for data mining. This section focuses on the cube browsers, and the following section summarizes some of the common data mining tools.

The most common form of cube browsers are interactive tools that make it easy for managers to examine subtotals, select subsets of the data, and drill down to see detailed data. Many vendors provide these interactive cube browsers, but several common features exist. Once you understand the overall structure, you can learn the details of a specific tool. The SQL standard of 2003 introduced SQL extensions to support retrieval and analysis of OLAP data. Additional standardization work concentrates on **multidimensional expressions (MDX)**, or the more recent mdXML. Although they are not interactive, SQL or MDX make it easier to write code that can be executed to retrieve or analyze data.

### Cube Browsers

Vendors who provide OLAP tools generally include a cube browser to support interactive browsing by decision makers. All of the major DBMS vendors provide similar tools, but the construction and browsing techniques vary considerably. Also note that the business intelligence tools might require separate development tools and additional licenses (fees) to deploy to users. If you are using a DBMS that does not have an integrated BI system or cube browser, you can generally use Microsoft Office to build a PivotTable on the desktop that connects to your back-end database.

Figure 9.15 shows a sample cube for the Pet Store created with the Microsoft Business Intelligence Development Studio. The cube data was generated by creating a view to define the measures and dimensions related to sales of merchandise items. The Value fact was created as  $\text{SalePrice} * \text{Quantity}$ , and the SaleDate was

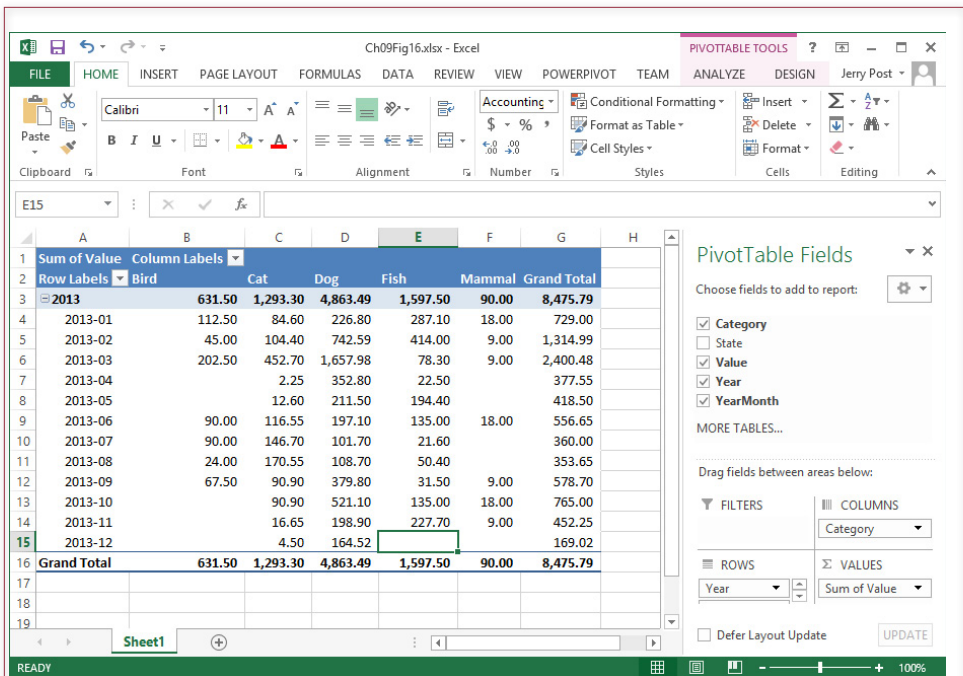


Figure 9.15

An OLAP cube browser. The time (SaleDate) dimension is shown in the table of data along with the merchandise Category. Users can change the display simply by dragging the dimensions on or off the grid. They can also add filter fields such as the State dimension. The year-month-date hierarchy enables users to drill down or roll up data.

extended into a year-month-date time hierarchy. Once the cube is defined, managers can browse the cube without needing to know anything about the underlying structures.

Browsing the cube is as simple as dragging the desired dimensions and facts onto the display grid. Users can experiment at will, because the dimensions can always be interchanged or removed. Subtotals are automatically generated for hierarchies and the user can click the designated buttons to drill down or roll up the totals. Users can place dimensions on the page to use as filters. To show a different subset of data, the user simply opens the desired dimension (filter, row, or column) and selects the desired attributes. In the example, you could open the Sale filter and select only a couple of states to immediately see the Category and SaleMonth values for the chosen states. Starting with Visual Studio 2010 (and later 2012) the cube browser displays all dimensions in rows and does not support column headings. To see a more tabular approach, use a PivotTable.

Microsoft Office contains the PivotTable and PivotChart utilities that can run inside of Excel, or even deliver interactive Web pages. A PivotTable is the interactive cube browser. A PivotChart uses the same principles to display dynamic charts. The primary advantage of charts is the ability to visualize the data—particularly trends over time on line charts and correlations using scatter charts.

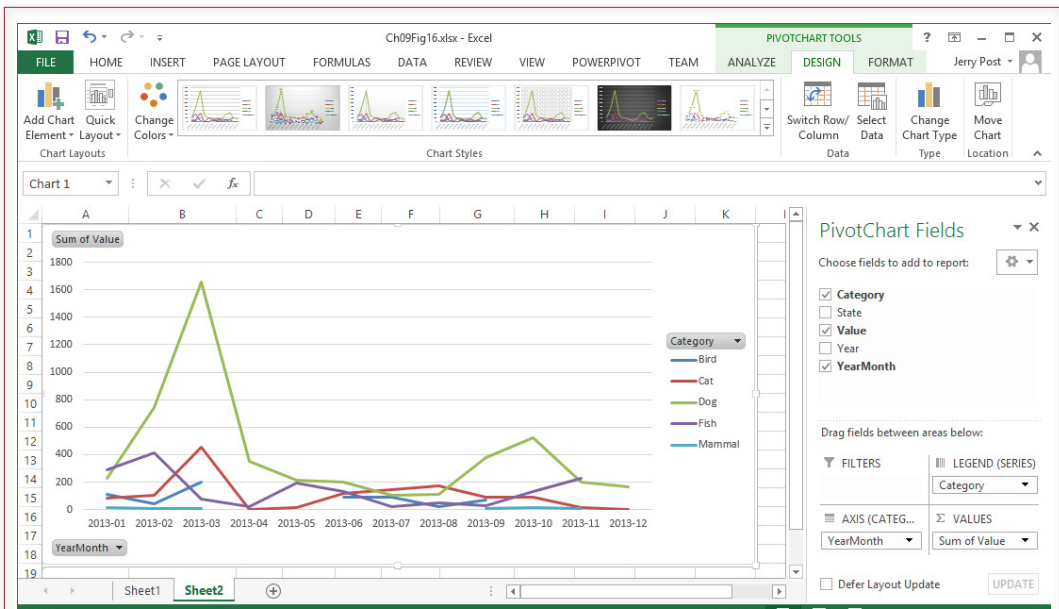


Figure 9.16

Microsoft PivotChart. Pivot tools make it easy for managers to examine cube data from any perspective, to select subsets of the data, to perform calculations, and to create charts.

The process of creating PivotTables and PivotCharts is similar. Although you could use Microsoft Query to collect and refine the data, it is usually easier to save a view in the original database that retrieves the desired data. In particular, you should create any needed calculations in the query. Microsoft Excel has menu options to help you create PivotTables and PivotCharts, so it is relatively easy to create and to use the resulting objects. Figure 9.16 shows a PivotChart based on the merchandise sale data. The operation of the PivotChart is similar to the cube browsers. Once the chart is built, managers can drag the dimensions around to create a new chart.

## OLAP in SQL

Think about the concepts of the OLAP cube for a couple of minutes, and you will recognize that it is a method of examining the results of multiple GROUP BY statements. The cube browsers simply make it easier to display the results and interactively explore the relationships. Interactivity is nice but sometimes you need a programmatic approach to a problem. Perhaps you need a formal report, or to transfer data, or to automate a statistical analysis. The SQL 99 standard added some features that provide OLAP-type results within SQL. Several vendors have integrated these new commands, although the syntax might be slightly different for each vendor.

In the Pet Store example, what happens if you use a GROUP BY statement with two columns? Figure 9.17 shows the partial results of a Pet Store query that contains a GROUP BY computation with two columns (animal category and month sold). Notice that it provides a subtotal for each category element for each month.

```

SELECT Category, Month(SaleDate) As SaleMonth,
       Sum(SalePrice*Quantity) As Amount
FROM Sale INNER JOIN SaleItem
       ON Sale.SaleID=SaleItem.SaleID
       INNER JOIN Merchandise ON
       Merchandise.ItemID=SaleItem.ItemID
GROUP BY Category, Month(SaleDate)

```

Category	Month	Amount
Bird	1	135.00
Bird	2	45.00
⋮	⋮	⋮
Cat	1	396.00
Cat	2	113.85
⋮	⋮	⋮

Figure 9.17

SELECT query with two GROUP BY columns. You get subtotals for each animal category for each month. You do not see totals across an entire category (Birds for all months), and you do not get the overall total.

Figure 9.18

ROLLUP option. Adding the ROLLUP option to the GROUP BY statement generates the super-aggregate totals. In this case, the query provides totals for each Category element and the overall total. Notice that the corresponding Month is a null value.

```

SELECT Category, Month(SaleDate) As SaleMonth,
       Sum(SalePrice*Quantity) As Amount
FROM Sale INNER JOIN SaleItem
       ON Sale.SaleID=SaleItem.SaleID
       INNER JOIN Merchandise ON
       SaleItem.ItemID=Merchandise.ItemID
GROUP BY Category, Month(SaleDate) WITH ROLLUP;

```

*Oracle syntax:*

GROUP BY ROLLUP (Category, TO\_CHAR(SaleDate, 'mm')

Category	Month	Amount
Bird	1	135.00
Bird	2	45.00
⋮	⋮	⋮
<b>Bird</b>	<b>(null)</b>	<b>607.50</b>
Cat	1	396.00
Cat	2	113.85
⋮	⋮	⋮
<b>Cat</b>	<b>(null)</b>	<b>1293.30</b>
⋮	⋮	⋮
<b>(null)</b>	<b>(null)</b>	<b>8451.79</b>

```

SELECT Category, Month(SaleDate) As SaleMonth,
       Sum(SalePrice*Quantity) As Amount,
       GROUPING (Category) AS Gc,
       GROUPING (Month(SaleDate)) AS Gm
FROM Sale INNER JOIN SaleItem
       ON Sale.SaleID=SaleItem.SaleID
       INNER JOIN Merchandise ON
       SaleItem.ItemID=Merchandise.ItemID
GROUP BY Category, Month(SaleDate) WITH ROLLUP

```

Category	Month	Amount	Gc	Gm
Bird	1	135.00	0	0
Bird	2	45.00	0	0
⋮	⋮	⋮	⋮	⋮
<b>Bird</b>	<b>(null)</b>	<b>607.50</b>	0	<b>1</b>
Cat	1	396.00	0	0
Cat	2	113.85	0	0
⋮	⋮	⋮	⋮	⋮
<b>Cat</b>	<b>(null)</b>	<b>1293.30</b>	0	<b>1</b>
⋮	⋮	⋮	⋮	⋮
<b>(null)</b>	<b>(null)</b>	<b>8451.79</b>	<b>1</b>	<b>1</b>

Figure 9.19

GROUPING function. The GROUPING function returns a value of one when the row displayed is a super-aggregate for the selected column parameter.

Assuming all animal types were sold in all months, you would see 12 values for birds, 12 for cats, 12 for dogs, and so on. What you do not get are **super-aggregate** totals, or totals for an entire category or across all rows. For instance, what is the total value of bird merchandise sold for the entire year?

### ROLLUP

You could get these super-aggregate totals by using additional SELECT statements. However, SQL 99 added the ROLLUP option specifically to compute super-aggregate totals. Figure 9.18 shows the results for the Pet Store query. The total across all months is calculated for each element in the Category column. This total is displayed with a null value for the Month column. At the bottom, the overall total is displayed with two null values. Of course, the super-aggregate totals are not normally printed in bold, so they can be hard to spot. A bigger question is, What happens if there is a missing (null) value for some months? In the case of a missing date for a sale of bird items, the display would contain two similar lines (Bird, null, 32.00). One of the lines would be the total sales of bird products for months with missing dates. The second total would be the super-aggregate total across all months. But how do you know which is which? It is possible to scrutinize the numbers with totals and realize that the larger total should be the super-aggregate value. But with other functions, such as Average, there might not be any way to tell. Notice that the Oracle syntax is slightly different from the SQL Server syntax. The Oracle version is slightly closer to the standard (which does not require the parentheses), but you should understand both versions.

```

SELECT Category, Month(SaleDate) As SaleMonth,
       Sum(SalePrice*Quantity) As Amount,
       GROUPING (Category) AS Gc,
       GROUPING (Month(SaleDate)) AS Gm
FROM Sale INNER JOIN SaleItem
       ON Sale.SaleID=SaleItem.SaleID
   INNER JOIN Animal ON
       SaleItem.ItemID=Merchandise.ItemID
GROUP BY Category, Month(SaleDate) WITH CUBE

```

Category	Month	Amount	Gc	Gm
Bird	1	135.00	0	0
Bird	2	45.00	0	0
⋮	⋮	⋮	⋮	⋮
<b>Bird</b>	<b>(null)</b>	<b>1358.82</b>	0	<b>1</b>
Cat	1	45.00	0	0
Cat	2	113.85	0	0
⋮	⋮	⋮	⋮	⋮
<b>Cat</b>	<b>(null)</b>	<b>1293.30</b>	0	<b>1</b>
⋮	⋮	⋮	⋮	⋮
<b>(null)</b>	<b>(null)</b>	<b>8451.79</b>	<b>1</b>	<b>1</b>
<b>(null)</b>	1	<b>1358.82</b>	<b>1</b>	0
<b>(null)</b>	2	<b>1508.94</b>	<b>1</b>	0
⋮	⋮	⋮	⋮	⋮
<b>(null)</b>	12	<b>164.70</b>	0	0

Figure 9.20

CUBE option. The CUBE option computes super-aggregate values for all columns in the GROUP BY statement. The rows near the bottom with the Gm indicator value of 1 are the totals by month for all categories of products.

To help identify the super-aggregate lines, the SQL standard introduced the GROUPING function. As shown in Figure 9.19, the function usually returns a value of 0. When the row displayed is a super-aggregate computation, it displays a value of 1. In the example, the totals across months for each category produce a value of one for the GROUPING(Category) function. The overall total contains values of one in both indicator columns. This function could also be used in other computations or even in WHERE conditions. For instance, you might want to perform a computation with the super-aggregate totals.

### CUBE

Looking at the results, it is clear that the ROLLUP option does not provide all of the information a manager might want. Notice that the super-aggregate totals only apply to the Category column in the examples. There are no corresponding totals for the Month column, which would represent sales of all categories for a given month. Of course, you could obtain those totals if you rewrite the query and reverse the order of the Category and Month columns in the GROUP BY clause.

The CUBE option provides the solution. The CUBE option is similar to ROLLUP, but it computes and displays the super-aggregates for all GROUP BY columns. In Figure 9.20, notice that the only change to the SQL was replacing the



```

SELECT Category, Month(SaleDate) As SaleMonth,
       Sum(SalePrice*Quantity) As Amount
FROM Sale INNER JOIN SaleItem
       ON Sale.SaleID=SaleItem.SaleID
       INNER JOIN Merchandise ON
       SaleItem.ItemID=Merchandise.ItemID
GROUP BY Category, Month(SaleDate) WITH CUBE
HAVING GROUPING(Category)=1 Or GROUPING(Month(SaleDate))=1

```

Category	Month	Amount
Bird	(null)	607.50
Cat	(null)	1293.30
:	:	:
(null)	(null)	8451.79
(null)	1	1358.82
(null)	2	1508.94
:	:	:
(null)	12	164.70

Figure 9.21

GROUPING SETS to hide detail. The GROUPING function can be used to hide the details so users can focus on the super-aggregate totals.

ROLLUP keyword with CUBE. The result still includes the super-aggregate totals across months for each category. These totals have a value of 1 for the Gc indicator column. But, the query also produces the super-aggregate totals for each month across all categories of products. The values for the three months are displayed near the bottom of the results. Notice the null value under Category, and the Gm column value of 1 indicating that it is the super-aggregate total for the month. Again, the Oracle syntax is slightly different, where the key phrase becomes: GROUP BY CUBE (Category, TO\_CHAR(SaleDate, 'mm')).

Because of these additional totals, you will most likely use the CUBE option more often than ROLLUP. However, if you add several columns to the GROUP BY statement, you could get so many subtotals that you might prefer to use ROLLUP to simplify the display. Ultimately, the decision comes down to what the users need to see, or which values you need in additional computations. Remember that you cannot rely on the null value to identify super aggregates. You must use the GROUPING (e.g., Gc and Gm) function instead.

The SQL standard provides additional options, including the ability to create CUBES or ROLLUPS based on the combined value from multiple columns. The standard calls for a GROUPING SETS function to hide the detail subtotals and only display the super-aggregate totals. However, this function is not supported by all systems, and it is actually easier to use the GROUPING function directly. As shown in Figure 9.21, the SQL is straightforward by adding the conditions to a HAVING statement.

Although the ROLLUP and CUBE options bring new features to SQL, the results can be difficult to read. In terms of OLAP value, you would not want to show the results to managers or expect them to be able to use these tools interactively. On the other hand, they could be useful for feeding data into a procedure that you write which needs to perform more advanced computations or transfer the data to a spreadsheet.

## SQL Analytic Functions

The SQL-99 standard added some mathematical functions that are useful for common OLAP analyses. For example, the statistical functions of standard deviation (STDDEV\_POP and STDDEV\_SAMP), variance (VAR\_POP and VAR\_SAMP), covariance (COVAR\_POP and COVAR\_SAMP), correlation (CORR), and linear regression (REGR\_SLOPE, etc.) are now part of the standard. Because most database systems already had proprietary versions of these functions, the impact is not that great, but it will help if vendors adopt the standard names for the functions.

Two of the more interesting new functions are **RANK** and **DENSE\_RANK**. These functions assign numbers to the sorted results that indicate the ranking of the data. A new table (SampleSales) was created to illustrate the functions. The sample data was created specifically to illustrate the difference between the two functions. You could create a view in the Pet Store database and run the same query, but the results will be different. Figure 9.22 shows the query and the results. First, note that the syntax is somewhat complicated. The reason for the complexity is because these two functions are designed to work with partitions—which are explained in the next section. The query in this example uses all of the rows of data, but you still need the OVER clause to specify the correct sort order.

Look closely at the results, and you will see the difference between the RANK and DENSE\_RANK functions. If you have ever worked with ranked data (such as sports or election results), you know the basic problem: How do you handle ties? Both functions give tied values the same rank (2 in this case). But the RANK function keeps counting the number of entries and assigns a rank to the next non-tied value that includes all of the entries above it. In this example, White receives a rank of 4 because three people have higher sales. The DENSE\_RANK function counts the ranks instead of the rows. Consequently, White receives a dense rank of 3 because it is the next ranking value. You can choose whichever function you need for a particular problem. The syntax is the same, but you have to remember the difference between the two.

**Figure 9.22**

RANK functions. The sort order for the rank function is specified separately. Ties are given the same rank. RANK skips values that would have been assigned to tie values. DENSE\_RANK does not skip values.

```
SELECT Employee, SalesValue,
       RANK() OVER (ORDER BY SalesValue DESC) AS Rank,
       DENSE_RANK() OVER (ORDER BY SalesValue DESC) AS Dense
FROM SampleSales
ORDER BY SalesValue DESC, Employee;
```

Employee	SalesValue	Rank	Dense
Jones	18000	1	1
Black	16000	2	2
Smith	16000	2	2
White	14000	4	3

```

CREATE VIEW qryMonthlyMerchandise AS
SELECT Category,
       TO_CHAR(SaleDate, 'yyyy-mm') As SaleMonth,
       sum(SalePrice*Quantity) As MonthAmount
FROM Sale INNER JOIN SaleItem ON Sale.SaleID=SaleItem.SaleID
       INNER JOIN Merchandise ON Merchandise.ItemID=SaleItem.ItemID
GROUP BY Category, TO_CHAR(SaleDate, 'yyyy-mm')

SELECT Category, SaleMonth, MonthAmount, AVG(MonthAmount)
       OVER (PARTITION BY Category
            ORDER BY SaleMonth ASC ROWS 2 PRECEDING)
AS MA
FROM qryMonthlyMerchandise
ORDER BY Category, SaleMonth;

```

Category	SaleMonth	MonthAmount	MA
Bird	2013-01	135	135
Bird	2013-02	45	90
Bird	2013-03	202.5	127.5
Bird	2013-06	67.5	105
⋮	⋮	⋮	⋮
Cat	2013-01	396	396
Cat	2013-02	113.85	254.925
Cat	2013-03	443.7	317.85
Cat	2013-04	2.25	186.6
⋮	⋮	⋮	⋮

Figure 9.23

SQL-99 OLAP PARTITION versus GROUP BY. The window PARTITION statement enables you to display aggregate data (average) along with the detail rows. The GROUP BY statement only provides the summarized data. Also note the use of the PRECEDING statement in the partition to calculate across previous rows of data. This version is based on the Oracle syntax.

## SQL OLAP Windows Partition

The SQL-99 standard defines some a useful extension for OLAP that should make certain types of queries substantially easier in SQL. The standard introduced the concept of **partitions** or data windows. A partition is similar to a GROUP BY clause because you specify columns whose values are used to define the partitions. But partitions offer additional options and enable you to display detail and aggregate data at the same time. Figure 9.23 demonstrates an advanced capability using the Oracle syntax, which is almost identical to the standard. SQL Server 2012 supports the same syntax but the date conversion (TO\_CHAR) has to be replaced with Year(SaleDate)\*100+Month(SaleDate).

The main query needs a little explanation. Its goal is to compute a moving average over time within each Category. A moving average computes the average of a specified number of rows, then moves to the next row and slides the window to the next rows. The PARTITION BY Category command specifies that the computations are to be performed for each separate value of the Category variable and reset when a new Category value is found. The ROWS 2 PRECEDING command

```
-- Create a view to get the simple monthly merchandise totals
CREATE VIEW qryMonthlyTotal AS
SELECT SaleMonth, Sum(MonthAmount) As Value
FROM qryMonthlyMerchandise
GROUP BY SaleMonth;

SELECT SaleMonth, Value,
       SUM(Value) OVER (ORDER BY SaleMonth) AS running_sum,
       SUM(Value) OVER (ORDER BY SaleMonth RANGE
                       BETWEEN UNBOUNDED PRECEDING
                       AND CURRENT ROW) AS running_sum2,
       SUM (Value) OVER (ORDER BY SaleMonth RANGE
                       BETWEEN CURRENT ROW
                       AND UNBOUNDED FOLLOWING) AS remaining_sum
FROM qryMonthlyTotal
ORDER BY SaleMonth;
```

Month	Value	Sum1	Sum2	Remain
2013-01	1358.82	1358.82	1358.82	8451.79
2013-02	1508.94	2867.76	2867.76	7092.97
2013-03	2362.68	5230.44	5230.44	5584.03
2013-04	377.55	5607.99	5607.99	3221.35
2013-05	418.50	6026.49	6026.49	2843.80
2013-06	522.45	6548.94	6548.94	2425.30
2013-07	168.30	6717.24	6717.24	1902.85
2013-08	162.70	6879.94	6879.94	1734.55
2013-09	288.90	7168.84	7168.84	1571.85
2013-10	666.00	7834.84	7834.84	1282.95
2013-11	452.25	8287.09	8287.09	616.95
2013-12	164.70	8451.79	8451.79	164.70

Figure 9.24

OVER and RANGE functions. The first SUM function computes the total from the beginning to through the current row. The second SUM function does the same thing more explicitly. The third SUM function totals the values from the current row through the remaining rows in the query.

specifies that three rows are to be included in the computation: the current row and the two rows before it. If fewer than three rows exist, the DBMS uses only the values that do exist.

One of the strengths of the OVER statement is that you can specify different partitions within the same SELECT statement. The standard also supports relatively powerful options to specify a variety of ranges of rows. It is used to perform calculations relative to the current row, so you can compute differences and averages backward and forward. Figure 9.24 shows some commonly used options for the RANGE function. The entire query computes three values of totals. The first SUM command totals the values in the rows from the beginning of the query through the current row. The second SUM column does the same thing, but more explicitly states the beginning and ending rows. The third SUM column computes the total from the current row through the last row of the query. In the second and

```

--LAG or LEAD (Column, # rows, default)
SELECT SaleMonth, Value,
       LAG(Value, 1, 0) OVER (ORDER BY SaleMonth) AS Prior_
Month,
       LEAD(Value,1,0) OVER (ORDER BY SaleMonth) AS Next_
Month
FROM qryMonthlyTotal
ORDER BY SaleMonth

```

SaleMonth	MonthAmount	Prior_Month	Next_Month
2013-01	1358.82	0	1508.94
2013-02	1508.94	1358.82	2362.68
2013-03	2362.68	1508.94	377.55
⋮	⋮	⋮	⋮
2013-12	164.70	452.25	0

Figure 9.25

LAG and LEAD functions. As inline functions, they easily return a value from a prior or following line. You can specify how many lines to go backward or forward.

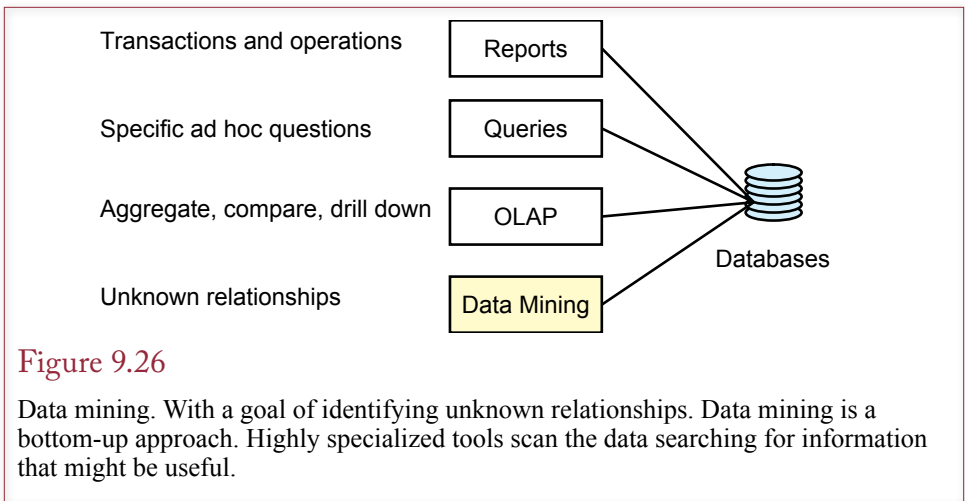
third examples, notice the use of the UNBOUNDED keyword to specify the start or end row. You could have replaced those with specific numbers if you wanted to compute only the totals for a specified number of preceding or following rows.

Most database systems also make it easy to use LAG and LEAD functions. These functions are designed to be used as inline functions that refer backward or forward to a specified number of rows. For example, the LAG function refers to values on previous rows. Figure 9.25 shows the basic syntax for Oracle and SQL Server with the result of a one-period lag and one-period lead. The power of the functions is that it is also easy to use the lag or lead variables in additional calculations. These functions are not part of the official SQL standard, so there are still some differences among the vendors. For example, you might not be able to specify the default value, which is useful for the first (or last) few rows that do not have defined values. But because most systems support the functions, and because they are so useful, they are worth studying. As of SQL Server 2012, the PARTITION, LAG, and LEAD functions are available in SQL Server with the same syntax.

## Data Mining and Business Intelligence

**What tools exist to search for patterns and correlations in the data?** The goal of data mining is to discover unknown relationships that can be used to make better decisions. Figure 9.26 summarizes the various methods available to retrieve data from the database. Reports are predefined and generated as part of the transaction system. Queries are used to answer ad hoc questions, but require knowledge of SQL or a query builder. The OLAP cube

The topics in this section are more advanced and might be saved for a second course. The examples require installation of the Business Intelligence tools. In-depth details and explanations are provide in the separate Data Mining textbook. But this section provides an introduction to the basic concepts and goals.



**Figure 9.26**

Data mining. With a goal of identifying unknown relationships. Data mining is a bottom-up approach. Highly specialized tools scan the data searching for information that might be useful.

browser enables managers to retrieve data interactively, but primarily focuses on subtotals. Data mining is different in that the tools use statistical comparisons to search for patterns, and many of the tools are relatively autonomous. Managers have to select the appropriate tool and interpret the results, but the goal of data mining is to run with minimal input.

A few tools require more input and specification by model builders. Most of the technologies are exploratory, in the sense that you are searching for unknown relationships as opposed to trying to confirm a suspected one. Some of the routines are derived from statistical analysis; others are highly detailed and created for specialized tasks. This section presents an overview of some of the more popular technologies. Detailed statistical and programming issues are not covered here, but can be found in specialized textbooks.

Figure 9.27 lists some of the common data mining categories. Occasionally, a DBMS vendor will include a few of the technologies with the base system. However, most vendors sell business intelligence tools as add-on products. Many other tools are available from specialized data mining companies. In either case, you generally require the services of a modeler to help build the proper models and interpret the results. Data classification and market basket analysis are two common methods of analyzing data in business because they are useful for many types of problems. Geographic systems are powerful solutions to specific questions. Web site analysis through time-series evaluation of logs is increasingly popular. New technologies and new methodologies that can evaluate ever-larger datasets are being developed continually.

### Data Configuration

Configuring data is one of the most critical tasks of analysis, and the task a database developer is most likely to focus on. The categories in this section are organized in terms of how the data needs to be organized—which is related to the ultimate task. For example, several classification tools exist, but they all rely on the same data structure.

Data for analysis can come from a data warehouse or from relational tables. Standalone tools, such as the open-source tools, often require that data be stored in CSV files. Most data warehouses and relational DBMSs can export data into

Classification/prediction/regression  
Association rules/market basket analysis  
Clustering  
    Data points  
    Hierarchies  
Neural networks  
Deviation detection  
Sequential analysis  
    Time series events  
    Website analysis  
Spatial/geographic analysis  
Textual analysis

Figure 9.27

Data mining techniques. Classification and market basket analysis are popular technologies in business. New technologies and new methods of estimating relationships are still being developed.

CSV files. If nothing else, a SQL SELECT statement can be used to extract the data—complete with commas—and then the output can be redirected or saved to a text file.

### Classification

As shown in Figure 9.28, many business problems can benefit from **classification analysis**. Several tools have been developed to estimate relationships that can predict an outcome. Statistical methods like regression are readily available. However, the two drawbacks to statistical methods are that they tend to assume linear relationships exist, and the estimates are based on averages—but often the most important hidden relationships are too small to be identified by averages. For example, you might be searching for new customers that can be encouraged to return and make more purchases. Since they are new, you might not have enough average data to create a statistically important effect.

Problems that can be evaluated by classification analysis have an outcome that is affected by a set of indicator attributes. The basic objective is to estimate the strength of the effect of each indicator variable and its influence on the outcome. For instance, a bank would have historical data on borrower attributes such as job stability, credit history, and income. The data mining system could estimate the

Figure 9.28

Classification examples. Many common business problems can benefit from classification analysis. Each problem has an outcome and the goal is to classify elements into the outcome choices based on a set of attributes.

Which borrowers/loans are most likely to be successful?  
Which customers are most likely to want a new item?  
Which companies are likely to file bankruptcy?  
Which workers are likely to quit in the next six months?  
Which startup companies are likely to succeed?  
Which tax returns are fraudulent?

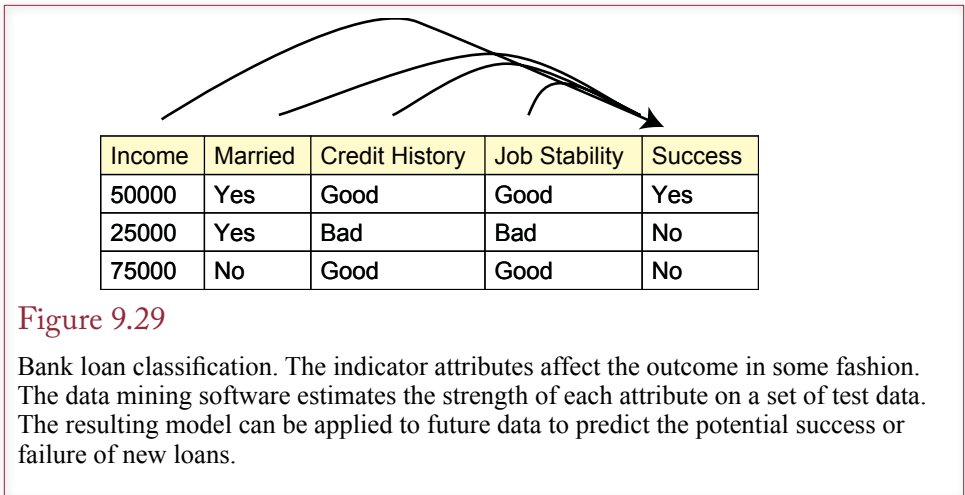


Figure 9.29

Bank loan classification. The indicator attributes affect the outcome in some fashion. The data mining software estimates the strength of each attribute on a set of test data. The resulting model can be applied to future data to predict the potential success or failure of new loans.

effect of each of these variables on the ultimate outcome (paying off the loan or defaulting). These weights could be applied to future customer data to help determine whether to grant a loan, or to affect the interest rate to charge.

Figure 9.29 shows a tiny sample of data for the lending situation. Note that the data might be categorical (Yes/No) or continuous (e.g., Income). Some classification tools can work with either type of data, but some require you to convert to categorical data. For example, the income data could be converted to bins, such as low: 0-30,000; medium: 30,000-70,000; high: 70,000-120,000, and wealthy: above 120,000. Of course, then you face the new data mining question of where to draw the lines to separate the categories. Some tools provide techniques to help make this decision as well.

Common classification tools include: various regression methods, Bayesian analysis, decision trees (particularly for hierarchical data), genetic algorithms, and neural networks. Of the group, neural networks typically require the least supervision, whereas advanced regression techniques rely on the skills of an experienced modeler. The key issue with any classification analysis is to determine how accurately the model can predict both existing and new cases. All of the techniques have strengths and weaknesses that you need to evaluate before choose a tool for a specific problem. Most require a solid knowledge of fundamental statistics to interpret the results.

### *Data for Classification*

Data for classification problems is typically stored similar to a relational table. Each row holds one instance of data and the columns represent the attributes. At least one attribute (column) is the predicted or dependent column, but that decision is made by the analyst or modeler. This data is usually easy to generate because a SQL SELECT command can commonly be used to choose the columns and rows and to perform simple computations.

### *Example*

Most data mining tools require a considerable amount of data to work effectively. In most business transaction applications, you will have plenty of data. However, the sample Pet Store database is intentionally kept small to make it easier to handle. The Rolling Thunder Bicycle company database has considerably more data



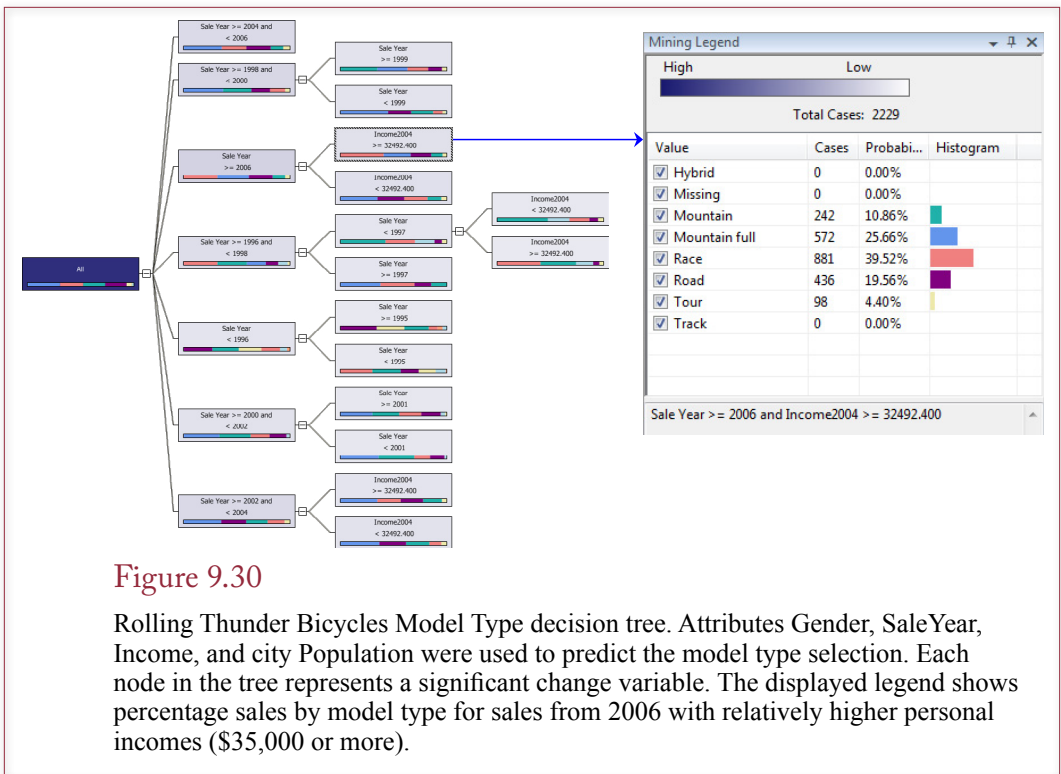


Figure 9.30

Rolling Thunder Bicycles Model Type decision tree. Attributes Gender, SaleYear, Income, and city Population were used to predict the model type selection. Each node in the tree represents a significant change variable. The displayed legend shows percentage sales by model type for sales from 2006 with relatively higher personal incomes (\$35,000 or more).

and works better for data mining illustrations. As a classification example, consider a basic goal of examining sales by model type. The goal is to see what factors affect the choice of model type, so the detail data consists of each individual bicycle sale. The fact attribute is the ModelType column. The company collects only minimal demographic data—something the managers might want to add in the future. However, Gender is available, the SaleYear is available in case purchases changed over time. Additionally, the City table contains two values from the Census Bureau: Income and Population which represent average customers within a city. A straightforward query generates the data in the correct format.

The data can be analyzed with standalone tools or a model can be built within the SQL Server Business Intelligence Studio. Every tool uses slightly different techniques and algorithms, so the results can vary slightly depending on the tool selected. The results shown in Figure 9.30 come from SQL Server's Decision Tree model. This tool examines the data to identify significant change points which are then marked as nodes in the tree. The selected node represents sales from 2006 on from cities with per capita income of about \$35,000 and more. The legend for that node shows the percentage breakdown of model type sales for that group. The marketing managers can compare the values to the nearest node (incomes less than \$35,000) to see the different purchase patterns.

Many other classification tools can be used. They all examine the impact of the selected attributes on the target fact variable. However, the outputs are slightly different and can provide different perspectives on the data. Some tools, such as regression, require pure numeric data; others can use categorical data.

## Association Rules/Market Basket Analysis

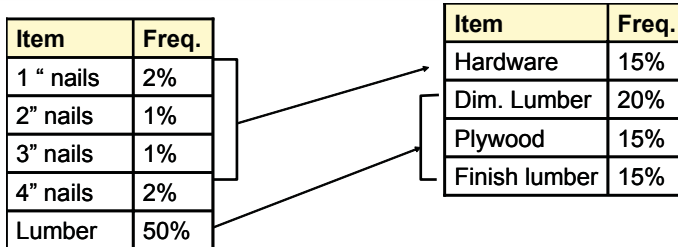
**Market basket analysis** is the tool that is credited with driving the acceptance of data mining. Originally, the techniques were applied to analyzing consumer purchases at convenience stores, hence the term market basket. The more generic term of **association rules** indicates that the methodology can be used for other situations. The basic question these systems answer is, What items are customers likely to buy together? Or, in terms of rules, Does the existence of A imply the existence of B? In the classic example, a convenience store discovered that shoppers who purchase diapers often purchase beer at the same time—particularly on Thursday and Friday nights. The importance of this piece of information is that managers can use it to increase sales. For instance, you might consider placing the two items close to each other in the store to encourage even more customers to purchase both items. Likewise, manufacturers might use similar knowledge to cross-sell items by providing coupons or product descriptions in the packaging of the related items.

Market basket analysis requires that you have a set of transaction data that contains a list of all items purchased by one person. Today, this data is readily available from supermarkets and large chains that use bar-code scanners. Most companies sell this data to specialized firms that resell it to other companies. The analysis software then scans the data and compares each item against the others to see if any patterns exist. In the process, the software computes three numbers that you use to evaluate the strength of the potential relationship or rule. The definitions are easier to understand with pairs of items, but they also apply to multiple items. The **support** for a rule is measured by the percent of transactions that contain both items. Statistically, the probability is denoted as  $P(A \cap B)$  (the probability of A and B occurring together) and computed by counting the number of transactions with both items and dividing by the total number of transactions. Similar numbers can be computed for A and B alone, or the percentage of times each individual item has been purchased. Higher values of support indicate that both items are frequently purchased together—but the number does not tell us that one causes the other. The **confidence** of the rule (A implies B) is measured by the percentage of transactions with item A that also contain item B. Statistically, it is the probability that B is in the basket, given that A has already been chosen, denoted  $P(B|A)$ . By statistical definitions,  $P(B|A) = P(A \cap B) / P(A)$ , so it is relatively easy to compute. Again, higher values of confidence tend to indicate that purchases of item A lead to purchases of item B. The third statistic reported by most data mining tools is lift. **Lift** is the potential gain attributed to the rule, compared to purchases without the rule. If the value is greater than 1, the lift is positive. Conceptually,

### Figure 9.31

Evaluating a market basket association. Support is the percentage of both items being purchased in one transaction. Confidence is the probability of purchasing beer (B) given that diapers (D) are purchased. Lift is the contribution of the effect to sales and should be greater than 1.

Support:	$P(B \cap D) = .6$	$P(D) = .7$	$P(B) = .5$
Confidence:	$P(B D) = P(B \cap D)/P(D) = 0.857$		
Lift:	$P(B D)/P(B) = 1.714$		



Item	Freq.
1 " nails	2%
2" nails	1%
3" nails	1%
4" nails	2%
Lumber	50%

Item	Freq.
Hardware	15%
Dim. Lumber	20%
Plywood	15%
Finish lumber	15%

Figure 9.32

Balanced frequencies. Items that are rarely purchased will lead to false rules. The solution is to define the items so that they balance. In this case, combine nails into a hardware category and split lumber into smaller categories.

it indicates the gain in sales resulting from the association. Statistically, it can be computed as  $P(A \cap B) / (P(A) * P(B))$  or as  $P(B|A)/P(B)$ .

Figure 9.31 shows how the numbers are computed for the diapers and beer example. The numbers are fictional but representative of the situation. Notice that the lift is substantially higher than 1 (1.714), indicating that the association strongly contributes to sales of beer. Data mining software computes all of these numbers for essentially all pairs of items. If there are many items, the process can take quite a while to run. Also, multiple items could be considered in the analysis: Does the purchase of sheets and pillowcases lead to the sale of more towels? However, combining too many dimensions leads to huge computational issues, so most analyses are done with a limited set of comparisons.

Working with market basket analysis, you will quickly encounter several problems. First, items with a small number of purchases can result in misleading values. If an item is purchased only once or twice, then almost anything else purchased with it will seem to be related. Consequently, you will have to examine the data and change groupings to ensure that most items are purchased with approximately the same frequency. Figure 9.32 shows a hypothetical situation at a hardware store that sells a lot of lumber but only a limited number of nails and screws. To prevent spurious rules, the answer is to combine the nails and screws into a broader hardware category, and split the lumber transactions into more detailed definitions. How do you know if problems exist? You can use additional queries to quickly count the number of sales of each item. The newer OLAP functions also make it easy to compute the percentages if the raw count numbers are hard to read.

The other problems that you can encounter with market basket analysis include the fact that some rules identified will be obvious to anyone in the industry. For example, a fast food chain would undoubtedly see a relationship between burgers and fries. A tricky problem arises when the system returns rules that do not make sense or cannot be explained. For example, a hardware chain found that sales of toilet rings were closely tied to the opening of new stores. Even if this correlation is true, what do you do with it?

#### *Data for Association Analysis*

Data for association analysis generally comes from transaction systems—particularly sales data. The catch is that analysis systems use two different methods for

SaleID	ItemID	Description	Category
4	36	Leash	Dog
4	1	Dog Kennel-Small	Dog
6	20	Wood Shavings/Bedding	Mammal
6	21	Bird Cage-Medium	Bird
7	40	Litter Box-Covered	Cat
7	19	Cat Litter-10 pound	Cat
7	5	Cat Bed-Small	Cat
8	16	Dog Food-Can-Premium	Dog
8	36	Leash	Dog
8	11	Dog Food-Dry-50 pound	Dog

Transaction data. It is easy to extract with SQL on the SaleItem and Merchandise tables.

36, 1
20, 21
40, 19, 5
16, 36, 11

Basket data. Converting from the SaleItem table to this format requires a cursor program that builds each row as a string for each SaleID.

**Figure 9.33**

Two data formats for association analysis. Coming from a relational database, the top format is the easiest to create. The second format requires programming. It could be based on Category instead, but not both ItemID and category at the same time.

arranging data. You need to read the tool's documentation carefully to determine the correct layout. Figure 9.33 shows the two common layouts, labeled *transaction* and *basket*. The transaction format mirrors the relational database approach. Essentially you just need data from the SaleItem table, and perhaps the Merchandise table if the manager wants to analyze data by category instead of ItemID. The data can be retrieved easily using a standard SELECT statement.

Unfortunately, some of the early tools created for association analysis were written with the requirement that each basket be specified as one line of text, with the item values separated by commas. Each row represents one basket and the rows are variable length. There is no easy way to convert relational data into this basket format. It can be done, but it requires writing programming code that uses a cursor to track through each row of data in the SaleItem table. The ItemID values are collected and built into a new string that appends a comma and the new ItemID for each row. When the SaleID switches, the new string row is written to the file. The code is straightforward, but eventually you will need a generic program that can be applied to any table or query because you will tire of rewriting the code every time it is needed.

Pr...	Importance	Rule
1.000	0.261	Track, Hybrid -> Road
0.857	0.220	Track, Tour -> Road
0.783	0.184	Track, Mountain -> Road
0.778	0.164	Track, Race -> Road
0.763	0.175	Hybrid, Mountain full -> Mountain
0.729	0.161	Hybrid, Road -> Mountain
0.718	0.159	Road, Race -> Mountain full
0.718	0.151	Hybrid, Tour -> Mountain
0.713	0.151	Tour, Mountain full -> Race
0.709	0.173	Tour, Mountain full -> Road
0.694	0.135	Road, Mountain -> Mountain full
0.689	0.137	Track -> Road
0.685	0.135	Hybrid, Race -> Mountain
0.674	0.150	Tour, Mountain -> Road
0.667	0.110	Track, Tour -> Mountain
0.667	0.517	Track, Hybrid -> Tour
0.667	0.099	Track, Race -> Mountain
0.659	0.113	Tour, Mountain -> Race
0.658	0.130	Tour, Road -> Mountain
0.652	0.134	Tour, Race -> Road
0.652	0.144	Road, Mountain full -> Race
0.651	0.095	Mountain, Race -> Mountain full
0.651	0.173	Mountain, Mountain full -> Road
0.651	0.124	Tour, Race -> Mountain
0.650	0.177	Race, Mountain full -> Road

Figure 9.34

Association rules for bicycles purchased by each customer. Same customer, possibly different times. Microsoft's probability and importance calculations are non-traditional but the interpretation is the same. Rules with high probability and high importance are likely to repeat.

### Example

It can be fun to experiment with market basket analysis. In some cases, it is obvious which items are purchased together (burgers and fries), in other cases the results are surprising. It is the surprising results that are the most useful. Still, association analysis can generate hundreds or even thousands of rules. It takes time and some experience to read through the rules and find the ones that can be useful. Rolling Thunder Bicycles has a couple of possibilities for using association analysis. It might be tempting to look at the traditional market basket and see which items were purchased at the same time. However, remember that almost all bicycles are built using groups of components. A group specifies all of the default components, so a market basket analysis would simply identify all of the components within a group. But, we already know those values, so there is no surprise. If customers routinely override the defaults and selected their own components, the results would be more interesting.

Instead, Figure 9.34 shows the results of examining model type purchases by customer. Essentially, CustomerID is the market basket and model types are the items purchased. Customers can buy multiple bicycles, perhaps at different times.

The question being asked is whether there is a pattern in purchases of model types. Consider the first rule which has a high probability (1.00) and relatively high importance (0.261). It says the customers who purchased a track and a hybrid bicycle also purchased a road bike. Consequently, future customers who purchased the first two model types should be contacted to suggest that they might also want to buy a road bike. Alternatively, the company could offer discounts on hybrid or track bikes which might then increase the sales of road bikes—which would not be discounted. Only some of the rules are shown in the figure. The challenge with association rules is to find the ones that are strong, important, and meaningful.

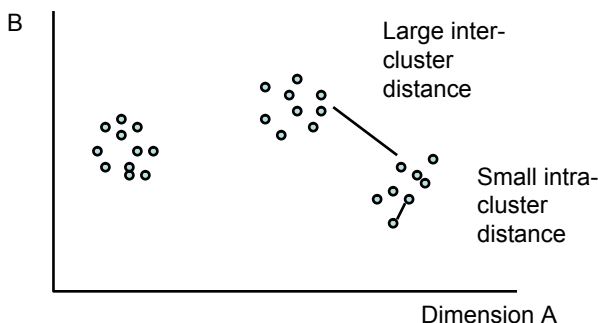
## Cluster Analysis

**Cluster analysis** is used to identify groupings of data—data points that tend to be related to each other. It can be used to identify groups of people, for example, to categorize customers. If you know that customers fall within certain groupings, you can use the information about a few customers to help sell additional products to the others in the group. Most likely, customers in the same group will want similar products. For instance, a bookstore can use the purchases of some items to categorize a customer and then identify books that similar customers bought and suggest them to the other shoppers. Likewise, you could use cluster analysis to categorize the skills of employees that work in various departments and use that information when hiring new workers.

As shown in Figure 9.35, clusters are relatively easy to see in two dimensions. The objective of the software is to identify the data points that are close to each other (small intra-cluster distance), yet further away from other points (larger inter-cluster distance). Unfortunately, most datasets do not exhibit clustering as strongly as shown in this example. But cluster analysis is a useful data exploration technique because it can reveal patterns that you might not see with other tools. However, keep in mind that datasets with a large number of observations (rows) and many dimensions are extremely difficult to cluster. Even with relatively modern computers, it can take hours or days to evaluate large, complex problems. So start cautiously and try to build clusters using smaller samples and a limited number of dimensions.

**Figure 9.35**

Cluster analysis. The goal is to find data points that are grouped close to each other and farther from other groups. Larger datasets with multiple dimensions are difficult and time-consuming to evaluate.



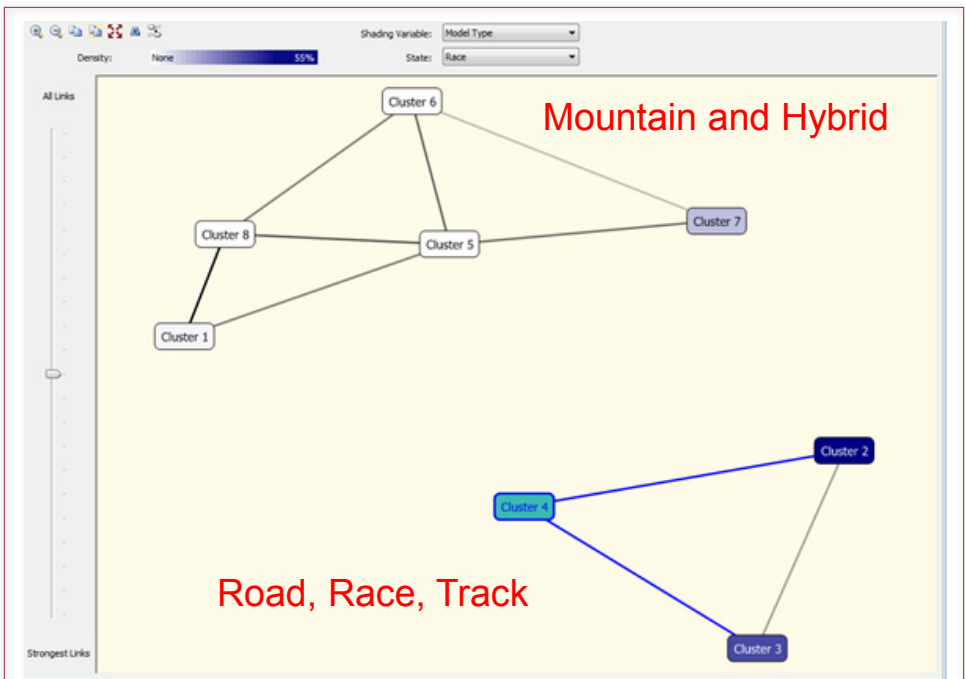


Figure 9.36

Cluster based on bicycle attributes. This chart focuses on model type. Based on the shading, the two main clusters are split by road/race bikes versus mountain/hybrid types. Details within the two groups are based on sale price and order year. Bike size also plays a role in differentiating the clusters.

### Data for Cluster Analysis

Data for cluster analysis is straightforward because it is similar to relational tables. Each column defines values for a chosen attribute. Each row represents one instance of the data. If the query results contain two attributes (columns), then each row represents one point on the two-dimensional chart. This data is easily retrieved using a standard SELECT query.

The one catch with cluster analysis is that some versions will not run if the dataset is too large. Too large is defined both in terms of the number of dimensions (columns) and the number of observations (rows). The specific limits depend on the algorithm used by the tool and the processing speed of the computer. This problem is similar to the issue of dimensionality in association analysis. You might need to reduce the number of dimensions, or combine items into aggregates. For example, it might make sense to examine sales of categories instead of individual items. Ultimately, this decision must be made by the manager or statistical analyst. However, it is often wise to start with smaller problems using aggregated data. Once these work, you can begin disaggregating the data and looking at larger problems.

### Example

Rolling Thunder Bicycle Company presents several opportunities for cluster analysis. Aligning with the other examples, this example builds cluster based on

model type. It is possible to include multiple attributes for each point so the data examines each bicycle in detail: Construction type (which is a proxy for material used), order year, sale price, bike size, and time to build. The attributes selected depend on the goals of the analysis. You might want to start with a smaller number of attributes—partly because including too many dimensions makes the model more difficult and time-consuming to estimate.

Figure 9.36 shows one version of the cluster results. Note the two large groupings—these are largely determined by model type. The top grouping consists of mountain and hybrid bikes. The lower group consists of road and race bikes. The tools provide additional charts to enable you to determine the differences between the clusters within the groups. These charts are not shown here, but they indicate that the details are determined by order year, sale price, and bike size. If managers want to examine these effects in more detail, it would make sense to run additional cluster analyses focusing on two or three of these attributes at one time. Ultimately, managers will want to see results from a variety of different models. For example, clusters might generate some intuition about the data, which could then be analyzed with classification tools.

### Geographic Analysis

**Geographic information systems (GIS)** display data in relation to its location. The systems are generally classified as visualization systems. They are useful for displaying geographical relationships and showing people how data is influenced by location. Few systems have true data mining capabilities for scanning the data to find patterns. Nonetheless, they are an important tool in analyzing data. Some relationships are much easier to understand if you see them on a map. Figure 9.37

Figure 9.37

Geographic analysis. This basic map shows sales by state. As shown by the key, darker colors represent larger sales. Additional data, such as income, could be shown as overlays or compared in charts.





shows a simple map of sales by western states. Additional data could be displayed with more colors or charts could be placed on each state.

Larger DBMS vendors have begun incorporating spatial and GIS systems into their offerings. You can also purchase standalone systems from other vendors. Beyond drawing maps, a true GIS has several methods for displaying data on the map. Basic techniques include shading and overlays, often used to display sales by region. Overlays show multiple items on different levels, making it easier to see how several items relate to each other as well as to location. For instance, marketers might compare sales, income, and population by geographic region.

In addition to the software, you need two important components for geographic systems. First, you need map data. Generally, this data is sold with the analysis system, but detailed data is sometimes sold as an add-on option. Highly detailed data down to individual street level is available for the United States (and much of Europe), but it is a large database. Second, you need to **geocode** your data and probably buy additional demographic data that is already geocoded. Essentially, you need to collect and store some type of geographical tag for your data. At a basic level, you probably already know country and state. But you might also want to add a region code, or a city code, or perhaps even latitude and longitude. If all of your sales are through individual stores, it is relatively easy to get the geographic position of each store from maps or GPS systems. An interesting possible option in the future arises from the increasing use of cell phones. Because of federal emergency regulations (e-911), cell phones are required to have positioning systems. Eventually, it is conceivable that this information will be provided to businesses, so your transaction systems can record exact locations of salespeople, and possibly even of customers. Please keep in mind the serious privacy issues these technologies create, but as you build new databases, you should think about incorporating geocode information into the data capture tables. Once the data has been collected, the GIS makes it easy to display relationships.

### *Data for Geographic Analysis*

Most GIS tools are standalone tools. For example, Microsoft's MapPoint is integrated into Excel. On a larger scale, ESRI's ArcGIS is definitely a standalone (or Web based) tool. Similarly, Google Earth is largely Web based. Most of these

#### **Figure 9.38**

Common geographic identifiers. At least one of these attributes must be coded into the data to use a GIS.

- State
- Country
- Region (custom defined)
- Latitude, Longitude
- Address
- City
- County
- ZIP Code
- Census Tract
- Standard Metropolitan Statistical Area

tools can extract data from a DBMS and use it in their displays and analyses, so data preparation can still be handled within the DBMS.

GIS data is typically stored in a relational format. Each column represents a single attribute and each row contains values for one location. The critical point is that at least one column must contain a geographic identifier. For example, a query might compute sales by state, so one column contains the state code. Each tool supports different types of geographic codes, but Figure 9.38 shows the types of geographic identifiers support by most systems. Some of the items in the list are defined within the U.S. only; however, there are often similar values in other nations. For instance, Postal Code is an international version of the U.S. ZIP Code.

At least in the U.S., some national data is already coded geographically. In particular, data collected by the Census Bureau is tagged by several identifiers such as City, State, ZIP Code, Census Tract, and Standard Metropolitan Statistical Area (SMSA) or large city region. Some tools include access to common Census data, but much of the data is available for free download from the Census Bureau Web site. This data is useful for comparisons or overlays with your business data. In particular, economic models suggest that it is useful to compare average income to sales.

### Example

GIS systems are different from most other data mining tools. You need a specific tool to be able to plot data geographically, and these tools are almost always standalone tools. Consequently, you generally export the data from the database.

Figure 9.39

Sales of bicycles by state in 2009. The legend is hidden but states colored in darker green represent higher sales based on dollar value.



A relatively inexpensive tool is Microsoft's Map Point software. You also might be able to use online tools such as Google Earth—but even some of those carry fees if you want to add your own data. Some applications can be handled online, such as placing stick pins or drawing routes in Google Maps or Microsoft Maps. Shading regions or states based on sales data has usually more difficult with the online tools.

As a small example of Microsoft Map Point, it is straightforward to write a query to retrieve sales value by state for 2009. Running the query, the results can be copied and pasted into an Excel spreadsheet. Once Map Point is installed, it can be run as an add-in. The tool automatically picks up the state codes (although it does not recognize PR for Puerto Rico). Figure 9.39 shows the data plotted using darker colors for higher sales. It is also straightforward to insert push pins, sized dots, or data charts.

## Summary

---

Large databases are optimized for transactions processing—to handle day-to-day operations efficiently, data is stored in normalized tables. But most managers need to join several tables to retrieve and understand the data. Indexes speed joins and data retrieval, but slow down transactions. This dichotomy means that it is often better to create a separate data warehouse to use for data analysis. Data can be extracted and cleaned from transaction systems, and placed into star or snowflake designs enabling managers to focus on the dimensions that surround a particular fact.

OLAP cubes are a powerful tool to enable managers to quickly sift through data and examine subtotals from a variety of perspectives. Without writing intense SQL queries, managers can compare values across product categories, time, and even across multiple dimensions simultaneously. OLAP cube browsers also contain easy methods to filter the data to specific rows or cube sections.

Many statistical data mining tools have been developed to help managers analyze data. They often require training and specialized knowledge by the workers, but can be powerful tools to understand relationships among the data. Classification and clustering algorithms help break the data into groups. Comparing the various groups makes it possible to better understand customers and expand the market. Association or market basket rules are popular with stores that sell a large variety of items. Identifying items that are purchased together makes it possible to suggest products to other customers. It can also lead to insights in store layout and customer psychology. Geographic systems are useful for any problem involving location. Specialized tools and demographic data are available to see the geographic relationships that exist.

### A Developer's View

Miranda saw that some business questions are difficult to answer, even with SQL. When managers are not exactly sure what they are looking for, you need to consider the OLAP and data mining approaches. Providing an OLAP cube is a good first step because it makes it easy for managers to see subtotals and slice the data to whatever level they want. More sophisticated statistical data mining tools are available, but generally require additional training and knowledgeable users. Just remember that performance often requires moving OLAP data into a separate data warehouse.

## Key Terms

---

association rules	geographic information systems (GIS)
binary search	lift
business intelligence (BI)	market basket analysis
classification analysis	measures
cluster analysis	multidimensional expressions (MDX)
comma separated values (CSV)	online analytical processing (OLAP)
confidence	online transaction processing (OLTP)
data hierarchy	partition, SQL
data mining	pointer
data warehouse	RANK
DENSE_RANK	roll up
dimensions	snowflake design
drill down	star design
extraction, transformation, and transportation (ETT)	super-aggregate
fact table	support
geocode	

## Review Questions

---

- ✓ 1. Why are indexes so important in relational databases?
2. Given the power of a relational DBMS, why might a company still need a data warehouse?
3. What main problems are encountered in setting up a data warehouse?
- ✓ 4. How are OLAP queries different from traditional SQL queries?
5. What is an OLAP cube?
6. What are hierarchical dimensions and how do they relate to roll up and drill down operations?
- ✓ 7. What basic analytical functions are defined in SQL?
8. What is the goal of data mining?
9. What are the main categories of data mining tools?
10. How is data organized in a data warehouse?

## Exercises

---

1. Find at least two commercial OLAP tools and compare the features.
2. Find a commercial data mining tool and outline the steps needed to extract and transform data from a typical DBMS so it is usable by the system.
3. Find a commercial data mining tool and outline the steps needed to perform a market basket analysis.
4. Assume you have two separate sets (tables) of customer data. You need to merge the two sets and eliminate the duplicates. The two tables use different ID/key values. Describe any problems you expect to encounter and how you might resolve them.
5. This question requires some tricky SQL. Assume you have a query (AnnualSales) with columns for SaleYear and Sales. Write the plain SQL (without the LAG function) to compute the difference in sales (current year value – prior year).

Most of the following questions require an OLAP cube processor. You should have access to SQL, an OLAP browser within the DBMS, or a PivotTable. For the data mining tools, if you do not have access to specialized software, you can use Excel for simple analyses.



### Sally's Pet Store

6. Create a cube to browse merchandise sales by date, state, employee, and item category.
7. Create a cube to browse animal adoptions by time, category, breed, gender, and registration.
8. If you are using SQL Server or Oracle, write the grouping and cube query to compute sales by employee by month, similar to the query in Figure 9.21.
9. Create a cube to browse purchases of merchandise from suppliers based on time, employee, and location. As facts, include the value of the purchase, the shipping cost, and the delay between order and receipt.
10. If you have access to market basket software, evaluate the sales tables to see if any associations exist. If you do not have the software, set up the query to retrieve the data.
11. Using SQL Server or Oracle, create a view to compute sales by month (YearMonth). Use the Lag function to compute the percentage change from the previous month.
12. Using SQL Server or Oracle, create a view as in the previous question that computes the sales by month. Then use the AVG and OVER functions to compute the three-month moving average.
13. Using SQL Server or Oracle, create a view that computes the total merchandise purchases by month. Then create a query that displays the month, total, and running total to date.

14. Identify at least two specific data mining tools that would be useful for this company and explain what data would be used and how they might be used to improve sales or operations.
15. Using monthly sales of merchandise, forecast sales for the next three months.
16. Is there a geographic pattern to sales? Do some states or regions have more sales?
17. Compute the total sales by employee for the year and list them in descending order with the computed ranking, similar to Figure 9.22.
18. Use the bulk load or import facilities of your DBMS to load several new items into the Merchandise table. File: MerchandiseNew.csv. Hint: Import the CSV file into a new table and use an INSERT statement to move the data into the Merchandise table.
19. Import the CSV file NewCustomers.csv into the database as a new, temporary table. Add the customers to the Customer table but be careful. Some of the “new” customers already exist in the Customer table—do not add duplicate values.
20. Import two CSV files (NewSales.csv and NewSaleItems.csv). The Sale file has a SaleID and a CustomerID. The CustomerID is valid, but the SaleID values are temporary and cannot be used in the main database. The NewSaleItems file has the matching SaleID and an ItemID. The ItemID is valid, and the SaleID matches the temporary value in the matching NewSales file. Import both files into the database, insert the new sales into the main Sales table, generating a new SaleID value. Assign that new SaleID value to insert the NewSaleItems entries into the SaleItem table.




### Rolling Thunder Bicycles

21. Create an OLAP cube to evaluate sales (value and quantity) by model type, state, time, and sales employee.
22. Create an OLAP cube to evaluate production time (ShipDate – OrderDate) by order date (time), model type, month, and employee who assembled the frame.
23. Create an OLAP cube to evaluate purchases of components by time, manufacturer, road or mountain bike, and component category.
24. Run a regression analysis to determine how sales by city by year are affected by population and income.
25. Using monthly sales by model type, forecast sales for the next six months.
26. Write a query to retrieve the data to perform a market basket analysis of component sales—to test which components were installed on the same bike.
27. Create an OLAP cube to evaluate sales (quantity) by paint type, letter style, and model type.
28. Create a query that computes total sales by year. Create another query that displays those annual values and computes the percentage change from year to year. Hint: Define a new column as  $\text{PriorYear} = \text{Year}-1$  and use it in a join.

29. Using SQL Server or Oracle, create query that computes total sales by Year and Model Type and compute and show only the super-aggregate totals for model type and year.
30. Using SQL Server or Oracle, create a query that displays Year, Month (year/month), Sales, and year-to-date sales using the SQL Analytic functions. Hint: The syntax is slightly easier if you first create a view to compute MonthlySales (Year, YearMonth, Sales).
31. Using SQL Server or Oracle, if it does not already exist, create a View that computes total sales by YearMonth. Using the SQL Analytic functions create a query to compute a 3-month moving average by model type. Hint: Leave out the Hybrid and Track model types because of their limited sales.
32. If you have access to a GIS such as Microsoft MapPoint, write the query and import the data to display a map similar to Figure 9.39 showing sales of Race bikes in 2012.



### Corner Med

33. Use association software or computations to see if some diagnoses commonly arise together.
34. Assume the ICD10 conversion does not exist. Use the crosswalk tables to identify the matching values for the existing ICD9 codes in the VisitDiagnoses table. Comment on any problems you find.
35. Using categorization software, such as regression, neural network, or decision tree, try to identify features of patients that spend the most money.
36. Create an OLAP cube to explore physician data in terms of patients and procedures. Managers want to focus on revenue and patients visited per day, week, and month.
-  37. Forecast the number of patients expected for a specific month. Hint: Use simple regression unless you have access to a time series analyzer.
38. Using SQL Server or Oracle SQL Analytic functions, show the monthly revenue generated by procedures by each of the physicians, along with the super-aggregate totals. Including the Grouping values.
39. Using SQL Server or Oracle SQL Analytic functions, count the number of patient visits per day for the month of March, and show the running total for the month.
40. Create a view that computes Revenue by month. Using either the Lag function or a JOIN by Year - 1, compute the percentage change in revenue by month.

## Web Site References

<a href="http://www.oracle.com/technology/tech/bi/index.html">http://www.oracle.com/technology/tech/bi/index.html</a>	Oracle business intelligence tools
<a href="http://www.microsoft.com/en-us/bi/default.aspx">http://www.microsoft.com/en-us/bi/default.aspx</a>	Microsoft SQL Server analysis tools
<a href="http://www-03.ibm.com/software/products/us/en/category/SWQ20">http://www-03.ibm.com/software/products/us/en/category/SWQ20</a>	IBM DB2 business intelligence tools
<a href="http://publib.boulder.ibm.com/infocenter/rbhelp/v6r3/index.jsp?topic=%2Fcom.ibm.redbrick.doc6.3%2Fsqlrg%2Fsqlrg36.htm">http://publib.boulder.ibm.com/infocenter/rbhelp/v6r3/index.jsp?topic=%2Fcom.ibm.redbrick.doc6.3%2Fsqlrg%2Fsqlrg36.htm</a>	SQL 99 OLAP standards and example.

## Additional Reading

- Apte, C., B. Liu, E. Pednault, and P. Smyth, Business applications of data mining, *Communications of the ACM*, 45(8) August 2002, 49-53. [Some examples of data mining, also part of a special issue on data mining.]
- Golfarelli, M, and S. Rizzi, A Methodological Framework for Data Warehouse Design, *Proceedings of the first ACM international workshop on Data warehousing and OLAP*, 1998, ACM Press, 3-9. [Relatively formal definition of facts, dimensions, and hierarchies.]
- Han, J. and M. Kamber, *Data Mining: Concepts and Techniques*, San Francisco: Morgan Kaufmann/Academic Press, 2001. [A general introduction to data mining techniques.]
- Hastie, T., R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning/2e*, New York: Springer-Verlag, 2009. [A strong foundation book on the statistics and algorithms of data mining including all of the math.]
- Peterson, T., J. Pinkelman, and B. Pfeiff, *Microsoft OLAP Unleashed*, Indianapolis: Sams/Macmillan, 1999. [Details on OLAP queries and data warehouses in SQL Server.]
- Post, Gerald, *Data Mining Applications/2e*, 2012, <http://www.JerryPost.com/Books/DMBook>. [Detailed applications of data mining with SQL Server and some open-source tools.]
- Scott, J. Warehousing over the Web, *Communications of the ACM*, 41(9), September 1998, 64-65. [Brief comments on Comcast using a Web interface for its data warehouse.]