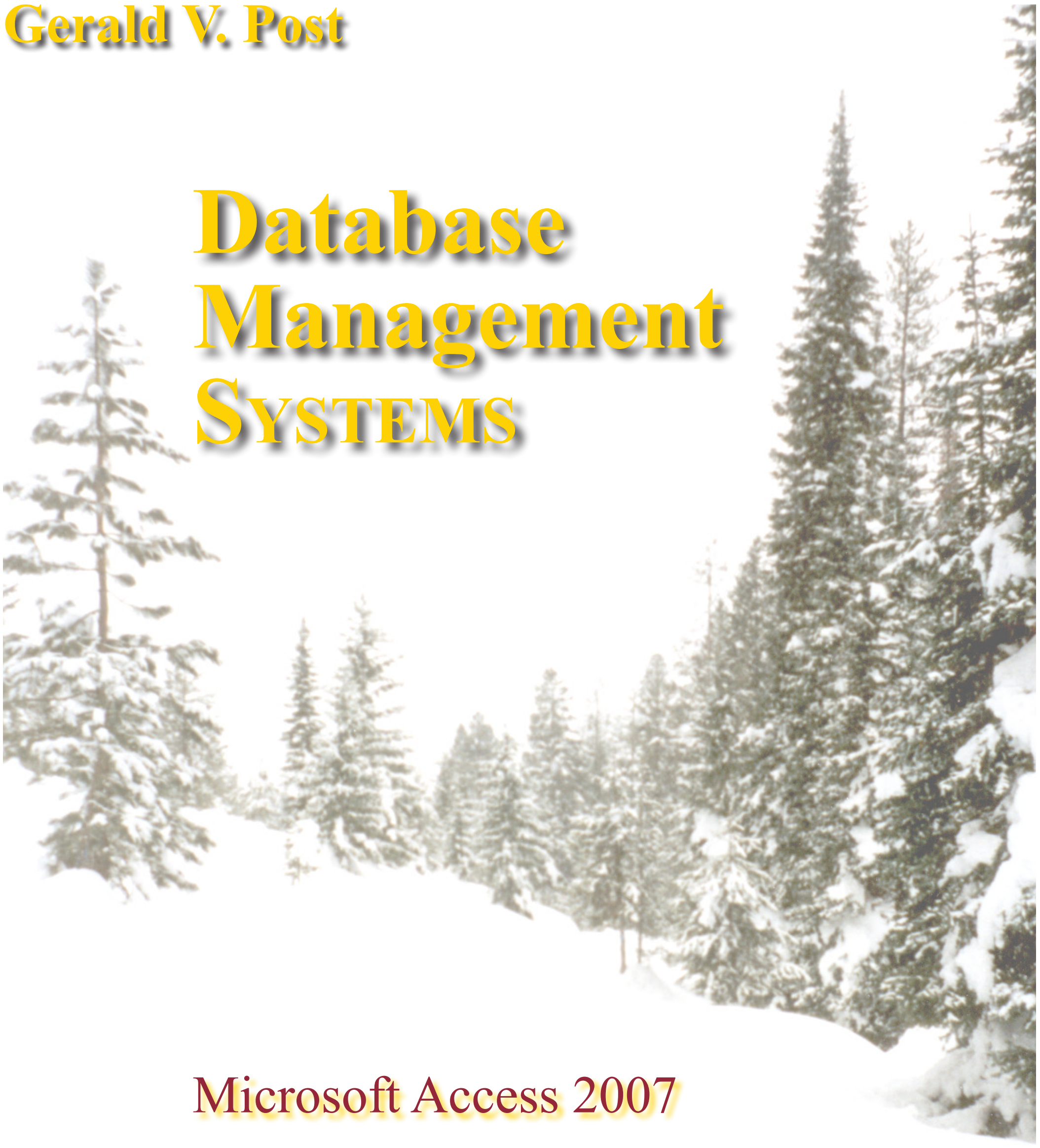
Gerald V. Post

Database Management Systems



Microsoft Access 2007

Designing & Building Business Applications

Fourth Edition

Database Management Systems

Designing and Building Business Applications With **Microsoft Access Office 2007**

Version 4.0.1

Gerald V. Post

University of the Pacific

Database Management Systems
Designing and Building Business Applications
With Microsoft Access

Brief Contents

Contents

Chapter 1

Introduction

Chapter Outline

Objectives

- Identify the main elements of the case.
- Structure the work needed for the case.
- Create a feasibility analysis of the case.
- Create a new database.

Case: All Powder Board and Ski Shop

The ski industry has been through many changes in the 50 years since Bill Shimek founded the ski shop that is now run by his grandson. One of the biggest changes is reflected in the prominence of "Board" in the shop name. Snowboards revolutionized the industry in several respects. They revived youth interest in the sport, brought new designs to equipment and resorts, and increased sales dramatically. On the other hand, the increased changes in ski and snowboard equipment make it more difficult for shops to stock the hundreds of options and combinations that enthusiasts might want. Shops have become larger, forcing small firms out of business. Even large ski shops have had to identify their customers and forecast customer demands carefully to make sure the high-demand equipment is in stock. Tracking sales, trends, and buyer needs has become critical to survival.

Another factor in the industry is that the firms increasingly rely on rentals. Partly because of the rapid changes in the industry, many people prefer to rent equipment so they can avoid having to buy new boards and skis every year. Consequently, the shop buys several relatively standard boards and skis every year and rents them out. At the end of the year, the used equipment is sold at a discount to make room for next year's models.

Inventory

Monitoring inventory is a first critical step in the process of providing the selection demanded by customers. Figure 1.1 shows some of the detailed information needed, as well as the diversity of equipment available. Note that because of the variety of uses, many different types of snowboards and skis exist. Figure 1.1 also shows the importance of the skill categories. Manufacturers produce special boards and skis for each of these categories. Of course, it would be impossible to stock all of the required sizes for rental purposes. Rental boards and skis tend to be as generic as possible. Even for sales, some sizes of the high-end skis and boards have to be special ordered.

Within a category, manufacturers tend to sell boards and skis targeted for different levels of skiers—from beginner to intermediate to expert (Type I, Type II,

Figure 1.1

Inventory

Snowboards	Manufacturer	Mfg ID	Size	Description	Graphics	List Price	QOH
Freestyle							
Pipe							
Standard							
Extreme							

Skis	Manufacturer	Mfg ID	Size	Description	Graphics	List Price	QOH
Cross country-skate							
Cross country-trad.							
Telemark							
Jumping							
Freestyle							
Downhill/race							

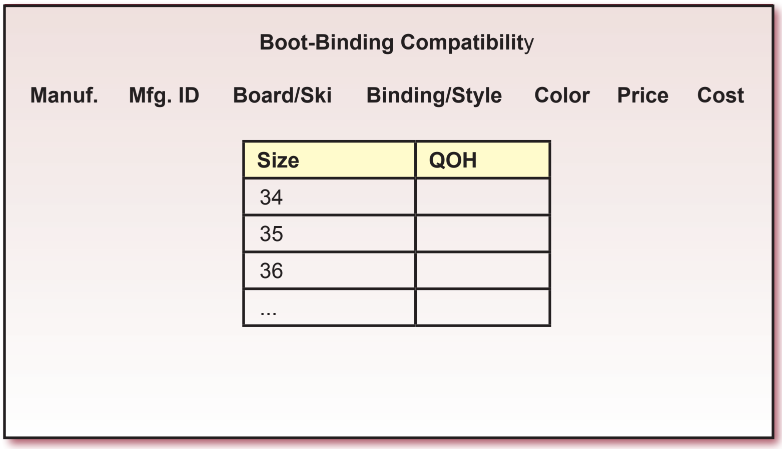
and Type III skier). Even within the type classifications, All Powder salespeople evaluate customers on the basis of their aggressiveness on the slope. Because of the size of snowboards, along with the youthful image of the sport, manufacturers place a high value on the graphics (images and colors) displayed on both sides of the boards. Customers have often been known to choose a board because of the graphics. Some of this emphasis has filtered over to skis as well.

Listing the sizes of boards and skis is somewhat tricky, and definitely presents a challenge to keeping adequate inventory. The length of the ski or board is a critical number, but the customer's choice is also based on several other ski measurements. Snowboards revolutionized board and ski design by adding a narrower waist to aid in turning. This concept migrated to most varieties of skis as well, so customers often want to know the waist width, sideout depth, and effective edge length of skis. Generally, boards and skis with narrower waists are targeted for more advanced skiers. Additionally, the construction of the board or ski, in terms of materials and thickness, significantly affects its flexibility and handling. Customers generally want to feel the ski to evaluate and compare its flexibility, but measures of stance location (for boards) and the rider weight range provide some prediction of the handling characteristics. Most skis and boards are also designed for a particular riding weight. With cross-country skis it is particularly important to get the proper length for the weight of the skier.

Bindings and Boots

Bindings and boots represent another common problem for All Powder and other ski shops. Each ski and each board can technically be fitted with several types of bindings. Each binding type generally requires a matching style of boot and some of the boots can work only with some bindings. For example, snowboards can use clincher, strap, or plate bindings. Cross-country skis can use pin, strap, or rod bindings. Most modern skis use the rod binding, but customers sometimes want boots that fit the older pin bindings. Downhill, freestyle, and slalom skis use similar bindings. Because they are the most popular, the store usually stocks several models—focusing on skill levels.

Figure 1.2



Boot-Binding Compatibility

Manuf. **Mfg. ID** **Board/Ski** **Binding/Style** **Color** **Price** **Cost**

Size	QOH
34	
35	
36	
...	

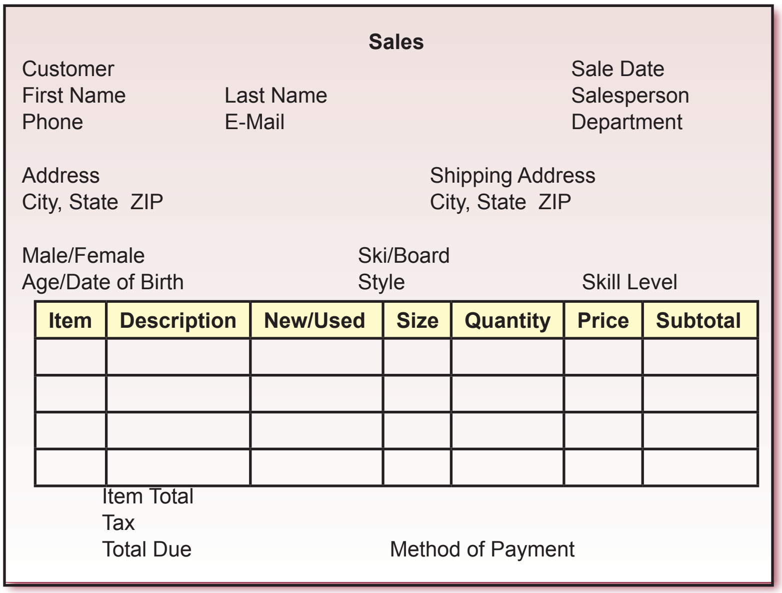
Figure 1.2 shows an example of the card system that All Powder uses to help salespeople select bindings and boots. Currently, the salespeople are supposed to change the quantity on hand whenever a boot or binding is sold. Of course, the cards are rarely kept up-to-date and the salespeople often have to go search the physical inventory to see if a size needed by a customer is in stock. Note that boots and bindings are specifically matched, and a boot for one purpose can rarely be used for a different application. For example, it would not be possible to use a cross-country boot in a downhill binding. The binding is usually listed as a type (rod, step-in, telemark/cable, etc.). On the other hand, it is possible to mount bindings on different types of skis. For instance, you could mount a telemark binding to a downhill ski. Some of the combinations should be avoided, but this knowledge will not be needed in the database.

Sales

The sales form shown in Figure 1.3 is fairly standard. All of the hard work in terms of configuration was done by the salesperson. In some cases, the salesperson might ask the customer to initial some items that might present compatibility issues to make sure the customer is aware of the potential problems. The description generally includes the manufacturer's name and style. The SKU (stock keeping unit) is a special number created within the store to code each item.

Returns are usually accepted on most items as long as they have not been used outside (e.g., scratched or worn boots cannot be returned). It is important for salespeople to identify the type of boarding/skiing and the customer's skill level. This information is used to send customers mailings about special sales. The owner also has started thinking about keeping customer sizes in a database. This information would be particularly helpful in clearing out the previous year's inventory of special sizes (very small or very large), because it would help pinpoint custom-

Figure 1.3



Sales

Customer
First Name Last Name
Phone E-Mail

Sale Date
Salesperson
Department

Address
City, State ZIP

Shipping Address
City, State ZIP

Male/Female
Age/Date of Birth

Ski/Board
Style

Skill Level

Item	Description	New/Used	Size	Quantity	Price	Subtotal

Item Total
Tax
Total Due

Method of Payment

ers who could use those special sizes. The catch is that the owner is concerned about privacy issues and fears that customers may not want to have their sizes on file at the store. However, if a customer has already purchased items in a specific category and size, that data will be available. The difficulty emerges when salespeople ask customers for their sizes when they are not purchasing these products. For instance, it may appear rude to ask a customer who came in to buy ski wax for his or her jacket size.

The store evaluates salespeople on the level of sales they make, so it is important to track sales by each employee. Of course, the database should contain additional information about each employee, such as phone number, address, and his or her primary department assignment. Of course, clerks rarely write down the department names properly, so it makes sense to have a separate lookup table for the department names.

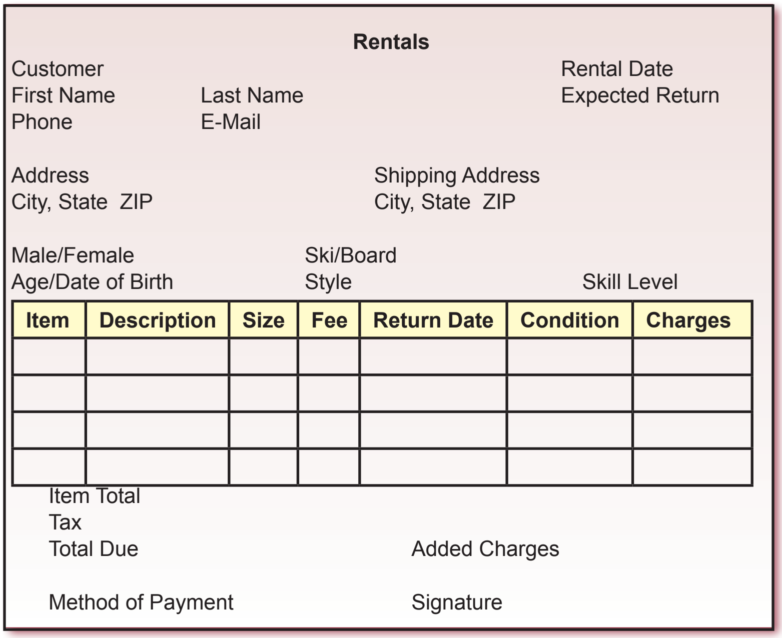
Also, note that some of the best customers participate in several styles, even crossing between using skis and boards. A customer who is an expert at downhill skiing might be a beginner with snowboards.

Rentals

The form to handle rentals is similar to the sales form. But notice in Figure 1.4 that columns have been added for return date, condition, and additional charges. The additional charges are imposed if an item is returned late or if it is returned damaged. Additionally, customers are required to sign the form to indicate their agreement with the skill level, rental conditions, and the release printed on the back of the form. Katy, the current manager, has talked about capturing the signatures digitally and storing them online, but it is not a high priority.

Observe that the current form requires that each rented item be checked off separately when it is returned. Although the store clerks often complain about having

Figure 1.4



Rentals

Customer | Rental Date
First Name | Last Name | Expected Return
Phone | E-Mail

Address | Shipping Address
City, State ZIP | City, State ZIP

Male/Female | Ski/Board
Age/Date of Birth | Style | Skill Level

Item	Description	Size	Fee	Return Date	Condition	Charges

Item Total
Tax
Total Due | Added Charges

Method of Payment | Signature

to mark each row separately, the store managers have determined that about 20 percent of the time, a customer forgets to return an item and has to bring it back later.

Renting ski equipment also raises the issue of reservations. On some holidays, all of the equipment is rented out before 10:00 A.M.. Some long-term customers have said that they would like to be able to reserve equipment. Currently, the rental managers will sometimes set aside equipment if a valuable repeat customer calls in advance. This process works reasonably well, but the managers have talked about creating a system that is available to everyone. One of the drawbacks is that they are concerned that the general public might reserve items and then never show up, leaving equipment idle that could be rented to someone else.

Lab Exercise

The first step in any project is to identify some basic elements of the system. What are the goals? What is the scope? What tools will be needed? What are the benefits? What are the expected costs? How much development time will be needed? All of these questions are difficult to answer, and rarely do the answers have a single value. Instead, you need to create a project plan. The plan will include a feasibility statement that describes the basic costs and potential benefits. As a real-world project, you would also include a list of developers and a statement of expected fees, so the owners can evaluate the decision to hire you.

Project Outline

As a first step in developing the project plan, you need to summarize the overall project. This summary should contain a brief description of the project, its goals, and initial lists of primary forms and reports. Ultimately, this summary will also include the scope and anticipated budget for the project.

Activity: Review the Case and Research the Industry

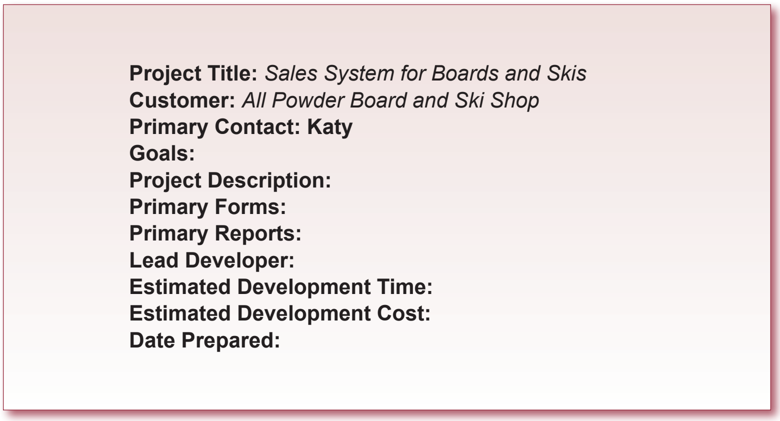
For the purposes of this lab, you will prepare a project proposal for developing the sales system needed by the All Powder Board and Ski Shop. The rental component will be left for another ex-

Action

Find information about skis and snowboards on the Internet.

If necessary, install and upgrade the DBMS.

Figure 1.5



Project Title: *Sales System for Boards and Skis*
Customer: *All Powder Board and Ski Shop*
Primary Contact: Katy
Goals:
Project Description:
Primary Forms:
Primary Reports:
Lead Developer:
Estimated Development Time:
Estimated Development Cost:
Date Prepared:

ercise. You should begin by reviewing the description of the company. You should also use the Internet to check out some of the manufacturers and some of the competitors. You need to be sure that you understand the key factors in the industry. Figure 1.5 provides a possible structure for your summary. You should review the case and enter the basic information requested.

Project Plan

The project plan consists of a detailed breakdown of the steps needed to create the final system. A common approach is to follow the steps of the systems development life cycle methodology: Initiation, Analysis, Design, Implementation, and Review. Some organizations have rigid descriptions of each of the steps involved in this process. Some organizations adopt a more flexible approach. Either way, this plan should outline the basic steps that need to be completed and an estimated schedule.

In the initial phase, it is also helpful to identify any potential risks to the project development. At various stages, ask what might go wrong. If you are aware of the potential problems, managers can monitor for them and can prepare solutions more quickly.

Activity: Create the Initial Project Plan

Action

Fill in the project milestone dates based on your school calendar.

Project plans and schedules are often shown with Gantt charts to illustrate how the various steps depend on each other. If you have access to software such as Microsoft Project, it is relatively easy to create the project plan. Figure 1.6 shows the basic steps that the labs will follow in building the application. Ultimately, you would estimate the times required for each step. However, until you have read the rest of the book and worked with the databases, it is difficult to estimate the times needed for each step. For now, evaluate the steps and try to identify any dependencies between the tasks. For example, is it possible to create the forms without having the database

Figure 1.6

1. Define the project and obtain approval.
2. Analyze the user needs and identify all forms and reports.
3. System Design
   a. Determine the tables and relationships needed.
   b. Create the tables and load basic data.
   c. Create queries needed for forms and reports.
   d. Build forms and reports.
   e. Create transaction elements.
   f. Define security and access controls.
4. Additional Features
   a. Create data warehouse to analyze data as needed.
   b. Handle distributed database elements as needed.
5. System Implementation
   a. Convert and load data.
   b. Train users.
   c. Load testing.
6. System review

tables and relationships? Assuming you have several people to help, reorganize the tasks so that as many tasks as possible can be done at the same time.

Feasibility

Feasibility studies are notoriously difficult. The concept is certainly simple: identify the potential costs and potential benefits of a system and compare them. The problem is that benefits might not be quantifiable, so it is difficult to attach meaningful numbers. Nonetheless, it is useful to at least write down the anticipated costs and expected benefits. Even if numbers are not available, managers at least can see a concise statement of the analysis.

Action
Create the feasibility plan for the project.

Figure 1.7

Assumptions			
Annual discount rate	0.03		
Project life/years	5		
Costs		Present Value	Subtotal
One time			
DBMS software			
Hardware			
Development			
Data entry			
Training			
Ongoing			
Personnel			
Upgrades/annual			
Supplies			
Support			
Maintenance			
Benefits			
Cost Savings			
Better inventory control			
Fewer clerks			
Strategic			
Increased sales			
Other?			
Net Present Value			

1. Windows update
   http://windowsupdate.microsoft.com

2. Office update
   http://office.microsoft.com/productupdates

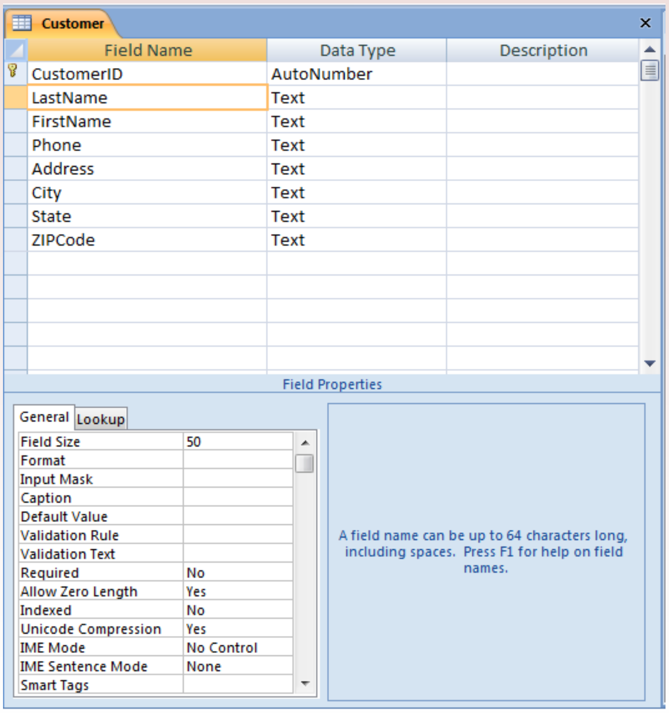
3. Data access component update
http://msdn2.microsoft.com/en-us/data/aa937730.aspx
Pick the highest numbered MDAC RTM.

Figure 1.8

Activity: Create the Feasibility Analysis

Figure 1.7 shows the basic elements of a feasibility study. You need to create a spreadsheet with these main categories. Use research to identify approximate costs of the various components. For example, assume that the shop will need to purchase a server to host the main database and two client computers for the sales staff. With Microsoft Access, several configurations are possible. Examine the software license to determine the number of copies you will need and the approximate cost. Other numbers, including benefits can be estimated. Remember that annual costs and benefits should be discounted to compensate for the time-value of money. Use the present value (PV) function in Excel. Although the benefits are relatively well defined, they can still be difficult to estimate. For example, how will the system reduce the need for sales clerks? How many or how many hours? How much do clerks earn? Likewise, in terms of inventory control, how

Figure 1.9


Customer

Field Name	Data Type	Description
CustomerID	AutoNumber	
LastName	Text	
FirstName	Text	
Phone	Text	
Address	Text	
City	Text	
State	Text	
ZIPCode	Text	

Field Properties

General | Lookup

Field Size	50
Format	
Input Mask	
Caption	
Default Value	
Validation Rule	
Validation Text	
Required	No
Allow Zero Length	Yes
Indexed	No
Unicode Compression	Yes
IME Mode	No Control
IME Sentence Mode	None
Smart Tags	

A field name can be up to 64 characters long, including spaces. Press F1 for help on field names.

much money will be saved by not having to slash prices at the end of the season to clear the unsold inventory? You need to know or estimate the number and value of items typically left at the end of the season. In practice, the managers might have answers to some of these questions, but you will still have to do additional research. In this example, be sure that you spell out your assumptions.

The Database Management System

Activity: Explore the DBMS

Action

Start Access and create the new table.

Create a new form with the Form Wizard.

Use the Customer table.

Select all of the columns.

Use the default column layout.

Test the form by entering data.

Save the form.

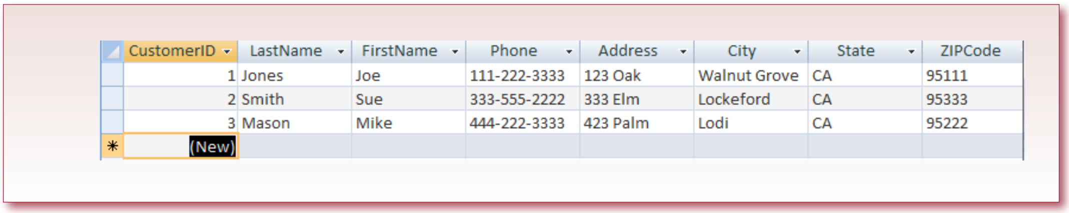
Two of the features that make Microsoft Access a popular database system are that it is relatively inexpensive and easy to install. If you are working in a classroom lab, your machines should already have Access installed. If you are working on your own computer, check your startup menu to be certain that Access is available. Note that Access is not shipped with the Small Business version of Office. More importantly, you need to make certain that you system is up-to-date. You should check the three sites listed in Figure 1.8 to update your computer in terms of the operating system, Microsoft Office, and the data access components (MDAC). Microsoft often releases updates on these three sites, so you should check them on a regular basis. Windows Vista should automatically check these sites.

To get a quick perspective of the various components of the DBMS, you need to build a simple database. Start Access and create a new, blank database. Because a database consists of tables, the first step is to create a table. Access 2007 opens a new database with a tool to help you create tables. However, this tool hides many of the details that you need, so it is best to close this screen and use the Create/Table Design option.

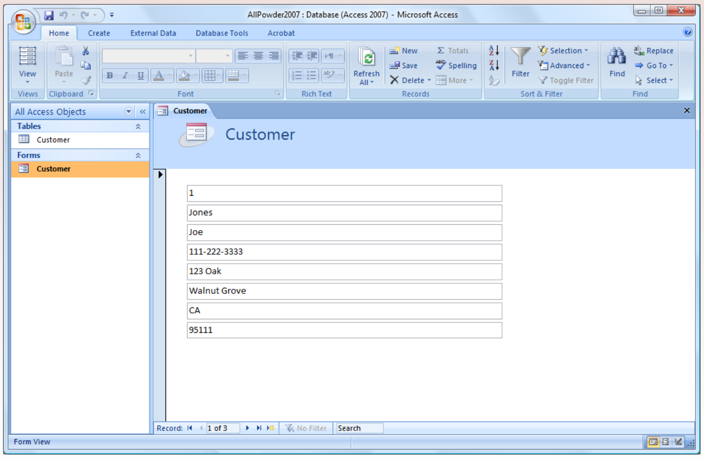
Figure 1.9 shows an initial definition of a customer table. Create a new table in Design view, and enter the names of the columns (fields) and select the data types as indicated. Then close and save the table design; name it "Customer" when asked.

Now open the table and enter some data for fake customers. You can copy the data from Figure 1.10 or just create your own. Access provides several tools in the Table window to examine the data. You can sort by columns or even filter the rows to see customers that meet some criteria. However, you will rarely give users direct access to tables. Instead, you will build forms and reports for managers to use. Close the Customer table.

Figure 1.10



CustomerID	LastName	FirstName	Phone	Address	City	State	ZIPCode
1	Jones	Joe	111-222-3333	123 Oak	Walnut Grove	CA	95111
2	Smith	Sue	333-555-2222	333 Elm	Lockeford	CA	95333
3	Mason	Mike	444-222-3333	423 Palm	Lodi	CA	95222
(New)							



1
Jones
Joe
111-222-3333
123 Oak
Walnut Grove
CA
95111

Figure 1.11

Access provides wizards to help build forms and reports. A common data entry form for the Customer table is relatively easy to create. Access 2007 has several options to create forms. For simple forms, one of the easiest is to select the Customer table in the list on the left side, then choose Create/Form from the menu. More commonly, you will use Create/Form/ More Forms/Form Wizard because it provides several useful options to create a form. The form in Figure 1.11 was created with the simple form builder. When you save the form it will appear in the list of objects on the left. However, you should click the display bar at the top of the list and choose All Access Objects.

Notice that as you type data into the form, the changes are written directly to the database table. The point of a form is to make it easier to enter and see data. From a usability standpoint, the layout should match the needs of the users.

Be default, Access 2007 puts form (and report) controls into a layout object that groups them together. You can switch to the Layout view or the Design view to see and modify the individual controls or the group. The purpose of the layout group is to make it easy to modify all of the controls at the same time. For example, using the Layout view you can change the size of one input box and all of them will adjust. An early hint: If you ever need to change the size or location of a single data control, you have to right-click it and choose Layout/Remove to take it out of the layout group. However, you can change basic display properties of a single box and still leave it in the Layout group. For practice, select the LastName textbox and click the Bold icon. You can also open the Property Sheet to see a list of the many properties you can assign. Save the form and run it to see the effect of your change. Similar wizards and properties are used to build reports, but there are not enough tables and data to justify writing a report yet.

Exercises

Many Charms

Madison and Samantha, friends of yours, have a small business selling charms for bracelets and necklaces. They buy some of the charms they sell; others they make. So far, they have run the business as a hobby, selling primarily to friends and relatives. But they have recently established a website to display pictures and prices of some of the charms. You have agreed to build a database for them to track their inventory, customers, and sales. Any orders they receive through the website will be e-mailed, so the website does not have to be connected live to the database. The database is a relatively traditional sales system, but it is slightly complicated by the nature of the charms. Charms come in a variety of shapes, sizes, and materials. For example, customers who want a quarter-moon charm have a choice of 4 mm or 8 mm; and of silver, gold, gold plate, bronze, or painted ceramic. Charms are also offered in categories such as animals, hearts, birthdays, and so on. Additionally, the duo offers a variety of chains and pins to hold the charms. Eventually, they want to track the sales by all of these categories, so they will know which items are selling the best and which make the most profit. Costs and prices tend to fluctuate. If they purchase items in large bulk, the per-piece cost is lower, but they need to know they can sell the entire shipment. If an item sits around too long, they find that they have to significantly cut the price just to clear out the stock. Of course, gold items are more expensive, making them more difficult to sell, and they are reluctant to tie up their money in high-priced merchandise.

1. Research similar sites on the Internet. Describe or sketch the major forms and reports that the company might use.
2. Create the initial proposal and feasibility study.

Standup Foods

Laura runs a catering company that focuses on Hollywood movie studios. Her chefs prepare hors d'oeuvres, sandwiches, and other food items that are served to the cast and crew of various movies and studios. To be fresh, the food is prepared each day in the main kitchens, and meals are then assembled and displayed on-site. For some clients, the company vans deliver fresh food every few hours. To hold costs down, many of Laura's employees are part time—only a few chefs and managers are full-time employees. Some of Laura's clients call at the last minute, so she maintains a large list of potential workers who can perform a variety of tasks, from driving to food preparation and display, as well as cleanup. The chefs and managers evaluate workers after each job in terms of timeliness, appearance, friendliness, and the ability to take orders and accomplish tasks. Workers often perform many tasks at a given event. For instance, a driver might also be a server. But some tasks require specific certifications. Not all workers are licensed to drive, and only a few have been trained to perform some tasks such as cutting meats. Most of the employee ratings are somewhat informal at the moment, but she would like to computerize them to help her select the best workers for future jobs. At some point, she would like to offer bonuses or higher pay to workers who routinely perform well. Another challenge Laura faces is that some clients are finicky about certain types of food. In particular, some movie clients have special preferences as well as some items that cause allergic reactions. The chefs currently keep these two lists in paper folders for some major performers and actors. But to be safe, Laura wants to computerize the lists and, ultimately, the recipe ingredients. Then when a chef plans the meals, the computer could check the list of main guests and their allergies against the recipe list to identify potential problems.

1. Research similar sites on the Internet. Describe or sketch the major forms and reports that the company might use.
2. Create the initial proposal and feasibility study.

EnviroSpeed

Brennan and Tyler are owner/managers of a consulting firm that specializes in environmental issues. In particular, the company's scientists are experts in clean-ups for chemical spills. For example, if a tanker crashes and spills chemicals on a highway, the company can quickly evaluate the potential problems and identify the best method to clean up the spill and prevent problems. The company itself does not clean up the spill, but it has contacts with several crews around the globe that it can call if local emergency workers need additional help. The primary focus of the company is to provide expert knowledge in the time of a crisis. This task requires specialized scientists, good communication systems, and in-depth training and practice. Brennan wants to improve the existing information system to maintain a database of case histories. Then, if a similar problem arises in the future, the scientists can quickly search the database and identify secondary problems to examine which solutions and ideas were successful and which ones caused more problems. Tyler has explained that at a minimum, the database has to hold the contact information for all of the scientists and emergency crews. It must also list the specialties, training, and skill levels of each person in a variety of areas. In terms of actual situations, the database should track the identities and roles of the various people and the key time frames (when reported, response time, and so on). Scientists also need the ability to list all of the chemicals involved and details about the terrain (hills, water, soil composition). More subjective data must also be captured, including comments by the onsite team and a description of the problem and secondary factors. All proposed solutions should be entered into the database, along with comments regarding their strengths and weaknesses as well as the final selections and an evaluation of the result. It is important to track potential solutions that were discarded. Even if they did not apply to the original problem, they might be useful for a future event with different circumstances.

1. Research similar sites on the Internet. Describe or sketch the major forms and reports that the company might use.
2. Create the initial proposal and feasibility study.

Final Project

The main textbook has an online appendix with several longer case studies. You should be able to work on one of these cases throughout the term. If you or your instructor picks one, do the following.

1. Research similar sites on the Internet. List the major forms and reports that the company might use.
2. Create the initial proposal and feasibility study.

Chapter 2

Database Design

Chapter Outline

Objectives

- Design the initial tables for the case.
- Create the design in the database design system.
- Determine the initial relationships for the case.
- Identify the data types needed for the attributes.

Database Design

You can design a database using paper and pencil. As you gain experience and become more skilled at the task, pencil and paper will be relatively easy to use. However, when you are learning, pencil and paper are tedious because you find that you often need to remove items from potential classes, or even alter the entire diagram. As an alternative, you might consider going directly to the DBMS and defining the tables or classes off the top of your head. This approach might work with Microsoft Access if you use the Relationships screen, but you will still find it time-consuming to continually revise the tables. With any other DBMS, trying to create tables from the top of your head is foolish because many systems do not allow you to make major changes to tables once they have been defined.

A few computer-assisted software engineering (CASE) tools remain that can help you define classes in a graphical environment. They are relatively powerful, and many have the ability to generate the final tables based on the class diagram. However, they are also expensive, hard to install, and cumbersome to learn. But if you work for a company that has invested in these tools, they are an excellent way to define the database classes.

To learn database design, there is a better tool. The database design system is an online expert system that enables students to create class diagrams graphically in a Java-enabled Web browser. The system makes it easy for you to create classes (entities) and build associations (relationships). More importantly, it provides immediate feedback on the design, which is the expert system part. The system runs on a custom Web server, and diagrams are stored in a central database. This approach means that you can access your diagrams from almost any computer. Changes you make in class or in your instructor's office are saved and are available when you return to a lab or to your own computer. From an instructional perspective, the best part is that the system contains some complex rules to provide feedback on your diagram. The system recognizes most design errors and points them out with suggestions to improve the design. Your instructor can obtain the database design system for your class. If it is available, you should use it to get the benefit of the immediate feedback. If it is not available, you can draw the class diagrams with paper and pencil, or with a graphics package such as Visio or even PowerPoint.

Access Data Types

As a database designer, your job is to define the database tables that efficiently store the organization's data and support the business rules. In this process, you will define the tables in terms of the data columns (attributes) and the table relationships (associations). You will also need to know what type of data will be stored in each column. Also, for some columns, you will want to specify constraints (for example, salary cannot be negative).

Selecting the proper data type can sometimes be a difficult step. Any DBMS supports only a limited number of domains and you have to understand the capabilities and limitations of each type. You must also understand the underlying business data—both the values collected today and the potential values that may be collected in the future. For example, workers may only use integer values to represent a quality rating. In the future, however, it is likely that the company will want to use fractional values as well. Although database types are becoming more standardized over time, each DBMS uses its own type names. Even more confusing, the actual values supported can be different even if the data type name is the same. The most common problem arises with the Integer data type. With some systems, an Integer is limited to 16 bits, while others default to 32 bits. With any DBMS, you should consult the help system to determine exactly which data types are provided and their limitations.

Figure 2.1 shows the main data types available in Microsoft Access. The types you will use most often are Text, Date/Time, Currency, and Long Integer. When you need to store date or time values, be sure to use the Date/Time type. It supports date arithmetic so users can subtract two dates to obtain the number of days between them. The OLE Object type can hold pictures, spreadsheets, or documents. However, the new Attachment data type stores images, spreadsheets, and similar files more efficiently.

The numeric subtypes are generally the most confusing. To truly understand the numeric types, you need to study the way that computers convert numbers into binary format, but that study is beyond the scope of this book. The easy answer is to split types into three groups: integers, floating point, and fixed point. Integers do not contain fractions or decimal points. The difference between the three types lies in the size of data that can be held. For example, a 16-bit integer cannot hold numbers larger than 32,767. So, if you have many items to sell or customers to track, you need to use a Long Integer, which can hold values to slightly over 2 billion. If you need fractional values, you need to choose the Single or Double data type. Again, the difference lies in the size of the number and the number of digits each type can hold. Single-precision numbers hold only seven significant digits, regardless of where the decimal point falls. Double-precision numbers can contain 14 significant digits. But, be careful. Even the 14 digits could cause you problems—if you try to use the Double type for money. Because of the way decimal numbers are converted to binary floating point, some numbers do not convert

Figure 2.1

	Name	Data	Bytes
Text (Characters)			
Fixed	NA		
Variable	Text	255	Variable
Memo	Memo	64K	Variable
Hyperlink	Hyperlink	1G	Variable
Numeric			
Byte (8 bits)	Number: Byte	0-255	1
Integer (16 bits)	Number: Integer	+/- 32767	2
Long (32 bits)	Number: Long	+/- 2 billion	4
(64 bits)	NA		
Fixed precision	Number: Decimal	+/- 1 E 28	12
Float	Number: Single	+/- 1 E 38	4
Double	Number: Double	+/- 1 E 308	8
Currency	Currency	+/- 900.0000 trillion	8
Yes/No	Yes/No	0/1	1 bit
Date/Time	Date/Time	1/1/100 – 12/31/9999 (1 sec)	8
Image	OLE Object	1 gigabyte	Variable
Attachment			
Generated Key	AutoNumber	Long (+/- 2 billion)	4
Replication ID	GUID	16 bytes	16

correctly and you can get round-off errors. If you need decimal point values and you need them to be exact, then you should use one of the fixed decimal types: Currency or Decimal. Currency is designed for monetary data and can accurately store 15 digits on the left side of the decimal point, and four on the right. The decimal type is both larger and more flexible than Currency. Numbers can have up to 28 total digits (specified by the precision), and you can specify how many digits fall to the right of the decimal point (given by the scale).

Although the many data types can be confusing, it is important that you choose the type carefully. In particular, for numeric types, make sure that you select a data type that will be able to hold the largest (or smallest) values that might exist in the data.

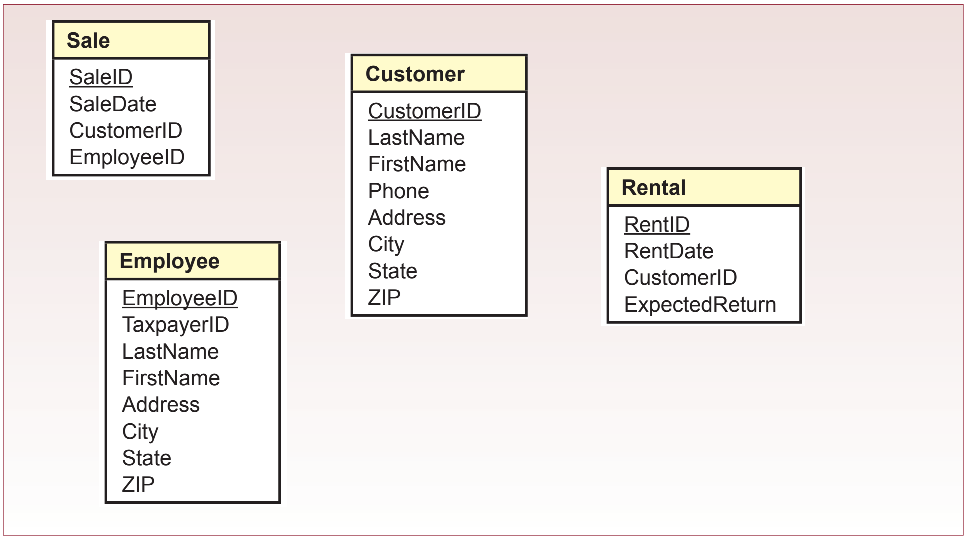
Case: All Powder Board and Ski Shop

With any database project, the first step is to understand the various elements of the organization and the components that will become part of the database application. This knowledge is critical, because the database design must reflect the business rules. In real life, you can ask workers about the processes and underlying assumptions. With a written case, it can be more challenging to determine all of the necessary rules. On the other hand, real life is messier and people often give inconsistent answers. It takes experience to learn to talk with users to identify exactly which components are the most important, and how the pieces relate to each other. Cases avoid this design complication but generally require you to make assumptions on your own. Since the goal is to make reasonable assumptions, you should search the Internet or read a few articles on snow boards and skis before you tackle the database design.

Business Objects: First Guess

One of the first steps in designing the database is to identify the business objects. In many ways, this case is a fairly typical business problem, so you would expect to see many of the traditional business objects, such as Customer, Employee, and Sale. Because the store also rents equipment, there will be a Rental object similar

Figure 2.2



to the Sale object. Figure 2.2 shows initial versions of these four classes. These objects are relatively standard, but some issues arise in this case. Notice that you must also begin to think about primary keys. In each of these four tables, the primary key is a new value that will be generated by the DBMS. In Microsoft Access, it will be an AutoNumber column. This decision was made because it ensures that these values are always going to be unique, and they can be created instantly by the DBMS. In most situations, the actual key values will be hidden from the users, and they will see only the relevant names.

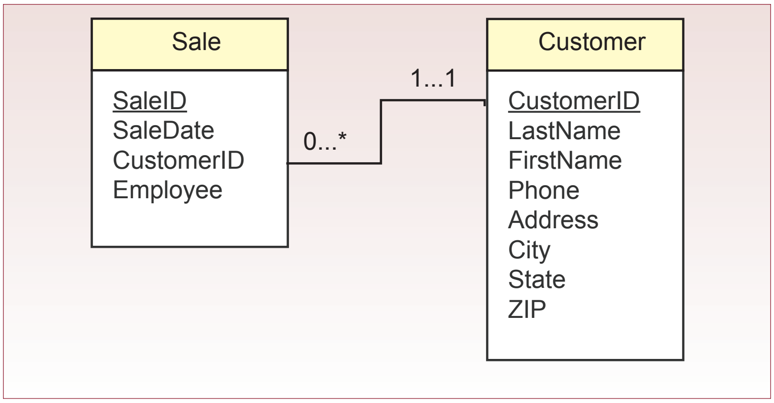
Notice that several attributes are missing from these initial classes. The main reason is that it is important to ensure that the columns you include at this stage are correct. If there is any doubt about a column in a potential class, leave it out and think about it. A few other classes should be relatively obvious for this case. In particular, several support tables are used to provide lookup data for other tables. Ultimately, you will have to define all of the objects, identify the columns for each table, and specify the data type for each column.

Relationships

Classes or entities are related to other classes. For example, notice that the Sale table contains a CustomerID property. Values in this column match entries in the Customer table, which is keyed by CustomerID. So, if you found a CustomerID value of 112 in the Sale table, you could look up the matching customer data by finding the row in the Customer table that has a primary key value of 112. This association also expresses several business rules. In particular, (1) each sale can be placed by only one customer, (2) a sale must be identified with a customer, (3) any given customer can participate in many sales, but (4) a customer might not have bought anything yet.

Relationships are displayed on the diagram by drawing connecting lines between the two tables involved. The business rules are shown as annotations at the end of each connection. Each side of the connection displays minimum and maximum values. Figure 2.3 shows the association between the Sale and Customer table. Notice that the annotations match the four business rules described in the previous paragraph. The 1…1 notation on the Customer side represents rules 1 and 2. At a minimum, each sale requires at least one customer, and, at a maximum, a sale can have no more than one customer. Likewise, the 0…* annotation represents rules 3 and 4. A customer can participate in zero to many sales. There is no maximum (*), so a customer can participate in any number of sales, and the

Figure 2.3



zero means that a customer might not have bought anything yet. As a database designer, your job is to identify the entities and relationships needed for this case.

Lab Exercise

Database Design System

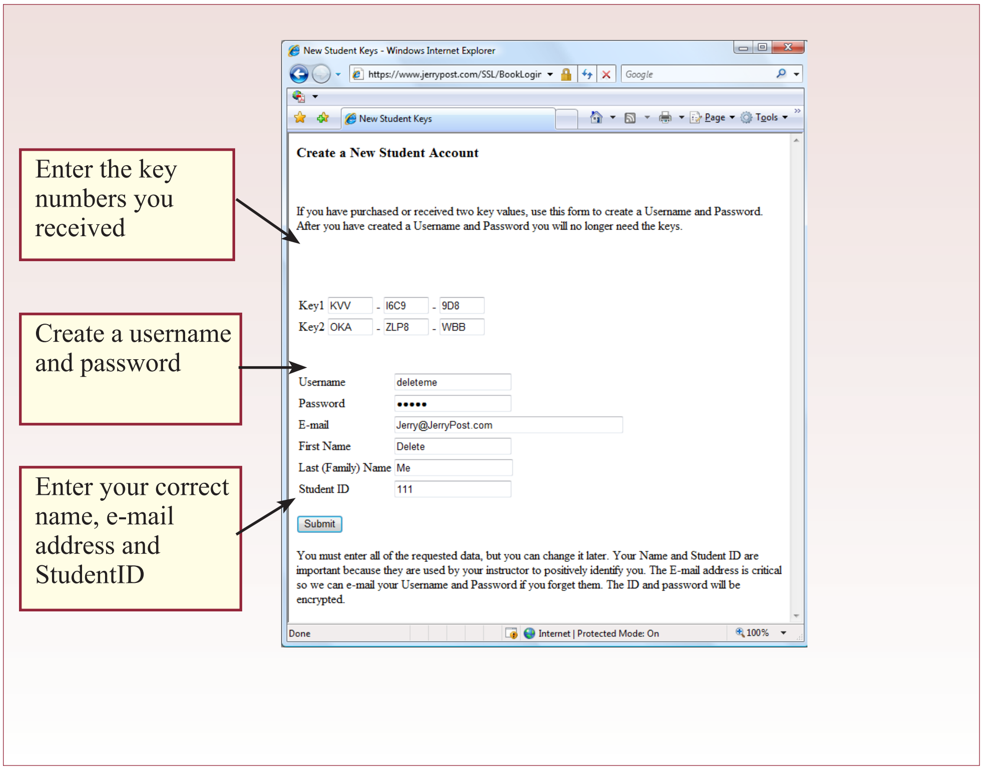
The database design system is designed as an instructional tool, so your instructor should have already registered to obtain an instructor account. The instructor also chooses and schedules assignments for the class. You will need a class code to register for a class, so be sure you get the correct admission code from your instructor. You will also need a set of numbers to create a new student account on the system. Check with your instructor to obtain these numbers. With the two sets of numbers, and the class admission code, you are ready to create your personal account.

Activity: Getting Started

> **Action**
> Browser: http://JerryPost.com/DBDesign
> New student who has two key numbers.

Use your browser to navigate to the database design website and select the link as a new student who has two key numbers. Figure 2.4 shows the form you need to fill out. First, enter the key numbers that you have. Next, create a username and password that you will remember. You must choose a username that is different from all others. Be sure that you enter your name, e-mail address, and Student ID number correctly. Your in-

Figure 2.4


Enter the key numbers you received
Create a username and password
Enter your correct name, e-mail address and StudentID
New Student Keys - Windows Internet Explorer
https://www.jerrypost.com/SSL/BookLogin
Google
New Student Keys
Page
Tools
Submit
Done
Internet | Protected Mode: On
100%

Create a New Student Account

If you have purchased or received two key values, use this form to create a Username and Password. After you have created a Username and Password you will no longer need the keys.

Key1 KVV - I6C9 - 9D8
Key2 OKA - ZLP8 - WBB

Username	deleteme
Password	•••••
E-mail	Jerry@JerryPost.com
First Name	Delete
Last (Family) Name	Me
Student ID	111

You must enter all of the requested data, but you can change it later. Your Name and Student ID are important because they are used by your instructor to positively identify you. The E-mail address is critical so we can e-mail your Username and Password if you forget them. The ID and password will be encrypted.

structor will use the name and ID number to correctly identify you so you receive credit for working on assignments. Note that your ID and password are encrypted on the Web site database to protect them. However, if your university still uses your Social Security number as an identifier, you might want to enter only a portion of the number—and then go ask your university to wake up and create a safer number. Your e-mail address is important so the system can send you the username and password in case you forget what you selected. When you have entered the data, click the Submit button. If you have an error in the key codes, or if your username has already been selected by someone else, you will receive a message and be asked to correct the items. Note that the key codes can only be used once and can be discarded after the account has been created.

Once you have successfully created the new account, you must register for the specific class. As shown in Figure 2.5, you simply choose your university and your correct class. Enter the admission code provided by the instructor and click the button to register for the class. If you do not have the proper code and are unable to register, you can get the code and return later. From the main page, enter your username and password to log in. If necessary, once you are logged in, you can click the link at the bottom of the main design page to register for a class. In fact, once you get to the design page, if you try to open a problem and the list is empty, it is most likely because you are not registered for a class.

All Powder Design

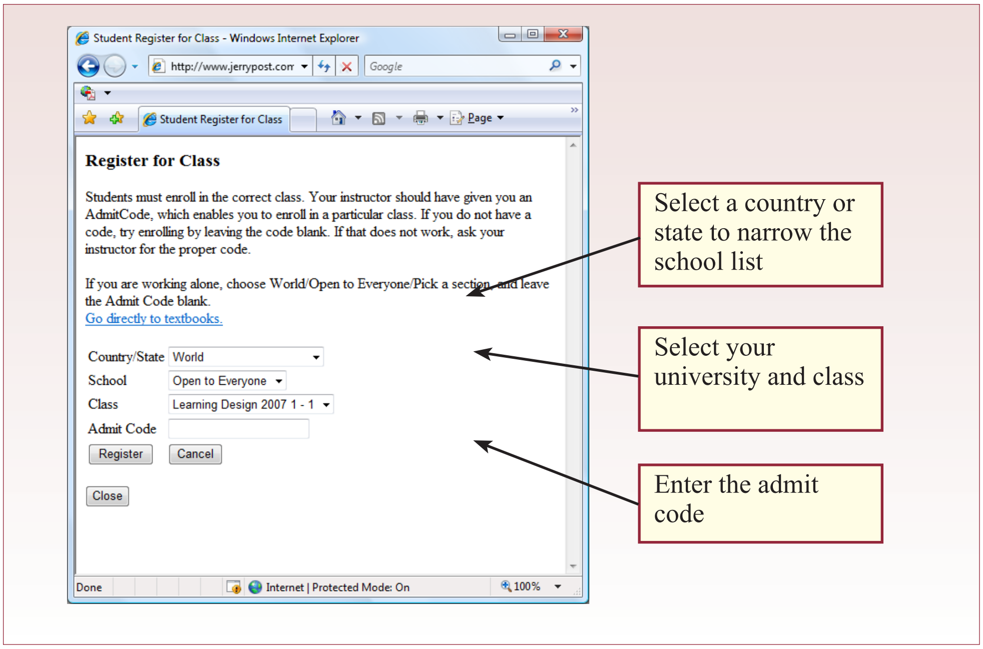
Activity: Create Tables and Columns

When you have created an account, registered for a class, and logged into the system, you are ready to begin designing the database. Figure 2.6 shows

Action

File/Open, choose All Powder case.
Right click/Add Table.
Type "Sale" as the new table name.
Drag columns from right onto table.
Right click name/set data type.

Figure 2.5


Student Register for Class - Windows Internet Explorer
http://www.jerrypost.com
Google
Student Register for Class
Page

Register for Class

Students must enroll in the correct class. Your instructor should have given you an AdmitCode, which enables you to enroll in a particular class. If you do not have a code, try enrolling by leaving the code blank. If that does not work, ask your instructor for the proper code.

If you are working alone, choose World/Open to Everyone/Pick a section, and leave the Admit Code blank.
Go directly to textbooks.

Country/State	World
School	Open to Everyone
Class	Learning Design 2007 1 - 1
Admit Code	

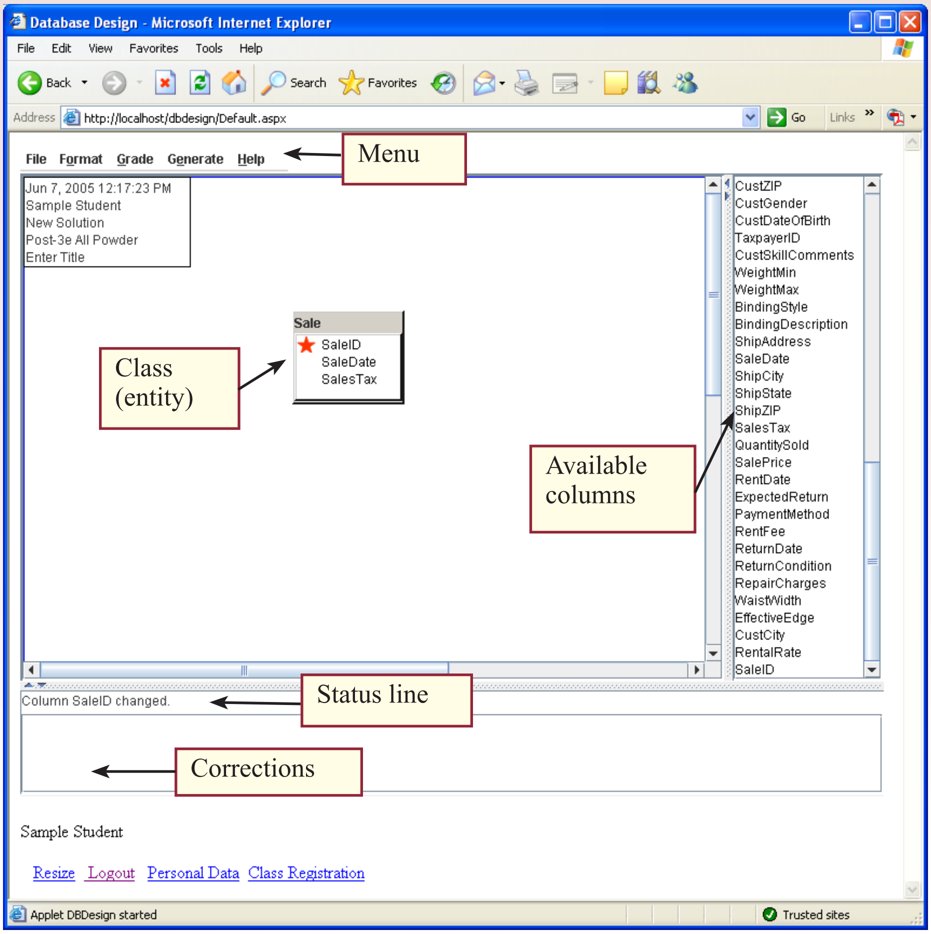
Register
Cancel
Close
Done
Internet | Protected Mode: On
100%
Select a country or state to narrow the school list
Select your university and class
Enter the admit code

the main elements of the system with the beginning of the solution. When you begin, the various windows will be empty. You must first open a problem using the File/Open menu choice and select the Workbook case. When the problem loads, the right-hand window will display a list of available columns. Initially, it will probably not include the key columns. You will add those in a minute.

You create a table (class/entity) by clicking the right mouse button on the main screen where you want the table located. Then select the Add Table option. Rename the table by typing "Sale" as the new name, and pressing the Enter key.

Now you get to add columns to the table. All columns are added to a table by dragging them from the right-hand window and dropping them onto the desired table. In the case of the Sale table, you will need to generate a new primary key column (SaleID). To create a generated key column, drag-and-drop the top label for Generate Key. Then, rename the newly created column. You rename columns by double-clicking the name either in the table or in the right-hand window. Be careful: Do not give two columns the same name, even if they are in different tables. You will not be able to tell them apart in the main list of the right-hand window. You might want to use an abbreviation and separator, such as Cust_LastName.

Figure 2.6


Database Design - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost/dbdesign/Default.aspx Go Links
File Format Grade Generate Help
Menu
Class (entity)
Available columns
Status line
Corrections
Resize Logout Personal Data Class Registration
Applet DBDesign started
Trusted sites

Jun 7, 2005 12:17:23 PM
Sample Student
New Solution
Post-3e All Powder
Enter Title

Sale
SaleID
SaleDate
SalesTax

CustZIP
CustGender
CustDateOfBirth
TaxpayerID
CustSkillComments
WeightMin
WeightMax
BindingStyle
BindingDescription
ShipAddress
SaleDate
ShipCity
ShipState
ShipZIP
SalesTax
QuantitySold
SalePrice
RentDate
ExpectedReturn
PaymentMethod
RentFee
ReturnDate
ReturnCondition
RepairCharges
WaistWidth
EffectiveEdge
CustCity
RentalRate
SaleID

Column SaleID changed.

Sample Student

Later, the system can remove the prefix when it generates Oracle tables. Now you can add some of the other columns needed in the Sale table. Look through the right-hand window to find the SaleDate and SalesTax entries. You can simplify your search if you sort the list by right-clicking on it and selecting Sort. Drag the desired column onto the Sale table. Once a column is in the table, you can change the order by dragging and dropping it higher or lower in the list.

At this point, you should set the data types of the columns in the table. The default type is Text, so in many cases you will not have to change it. However, you should choose Date/Time for the SaleDate, and Currency for the SalesTax column. Double-click on the column name within the table to open the column editor. You can select the data type and change the data size if desired. You can also add a constraint and default value, but you should probably do those later. The default value is straightforward, but the constraint has to be expressed in Oracle's format. Be sure to save your work every few minutes in case you lose the Internet connection or the server times out.

Activity: Create Relationships

Action

Add Customer and Sale tables.

Add GenerateKey to Customer table.

Rename it to CustomerID.

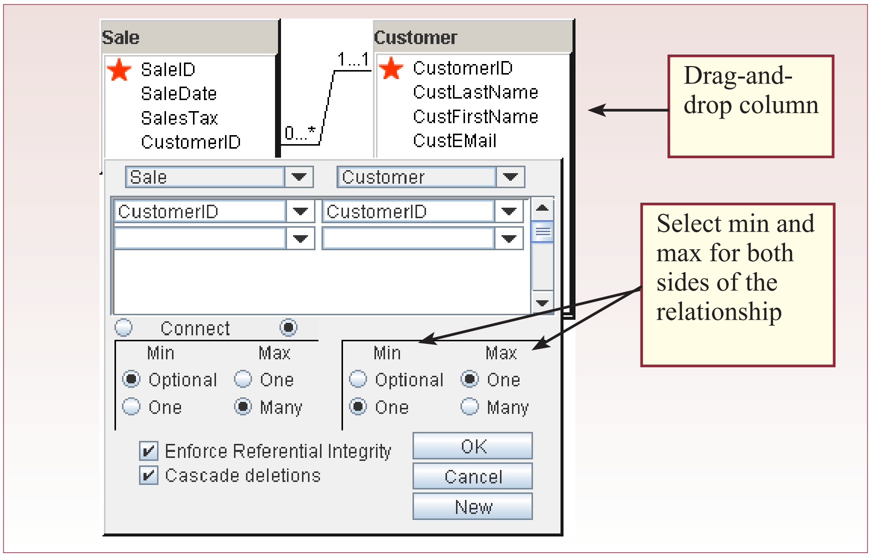
Drag new CustomerID from right side into Sale table.

Drag CustomerID from Customer and drop it on CustomerID in Sale table.

Fill out relationship box.

Associations or relationships are a key element of database design. In a relational database, columns in one table are connected to columns in other tables through common data. In the case, the Sale table needs to connect to a Customer table. Eventually, both tables will contain a CustomerID column. First, you have to create the Customer table, so right-click on the design screen, add a new table, and rename it. Again, to ensure that each customer is assigned a guaranteed unique identifier, add a Generate Key column to it. Rename this new column as the CustomerID. It is critical that you understand that this key value will be generated for each new customer added to the table. This value can only be generated in this table. You would never create another generated key column and call it CustomerID. Notice that the column is marked with a solid (red) star to indicate that it is a key with values generated in this table. How do you get CustomerID into the Sale table? Scroll the right-hand window to the bottom and notice that CustomerID has been added to the list of available columns. You could also sort the list and find it alphabetically. You can now drag this new column into the Sale table. Make sure its data type is Integer32 (Long). Before attempting to build the relationship, add the other customer properties to the Customer table by dragging them from the right-hand window. You can use the Shift or Ctrl key to select multiple columns at a time, but moving them takes a little practice. You can double-click the table heading to automatically resize the table design box to fit the columns it contains. Set the appropriate data types.

Now that you have both the Sale and Customer tables, and they both have a CustomerID column, you can build an association or relationship between them. Figure 2.7 shows how to create this relationship in the design system. Click on the CustomerID column in the Customer table and drag it to the Sale table. Release the mouse button to drop the cursor onto the CustomerID column in the Sale table. The relationship window then asks you to specify the minimum and maximum



Sale side: Min [✓ Optional] One; Max One [✓ Many]
Customer side: Min Optional [✓ One]; Max [✓ One] Many
[✓] Enforce Referential Integrity
[✓] Cascade deletions

Figure 2.7

values for each side of the relationship. These values specify the business rules, and are often the most difficult items to identify. In the sale case, the typical assumptions are that exactly one customer can place an order, and a customer can place from zero to many orders. So, on the Sale side of the window, select the Optional and Many buttons. On the Customer side, choose the One option for both Min and Max values. Note that if an option was selected by your instructor, the system will automatically attempt to create the correct relationship for you when you add the CustomerID column to the Sale table.

Remember that relationships generally involve at least one side in a primary key. The column names are often the same on each end, but they can be different. However, the data types do have to match, and the relationship has to be logical. For example, it would never make sense to connect an ItemID to a CustomerID, because that relationship would imply that a customer can also be an item and vice versa. Finally, notice that the integrity and cascade boxes are selected as the default. You should almost always leave these checked. In the database, cascade on delete means that if you delete a particular customer, all of the orders placed by that customer will also be deleted. If you do not specify the cascade, then you could end up with orders that contain a CustomerID, which has no matching customer data. After you close the relationship window with the OK button, you might have to refresh the display screen by right-clicking the design window and selecting Refresh.

Activity: Evaluate the Design

Action

Choose Grade/Grade and Mark.

Click messages in window.

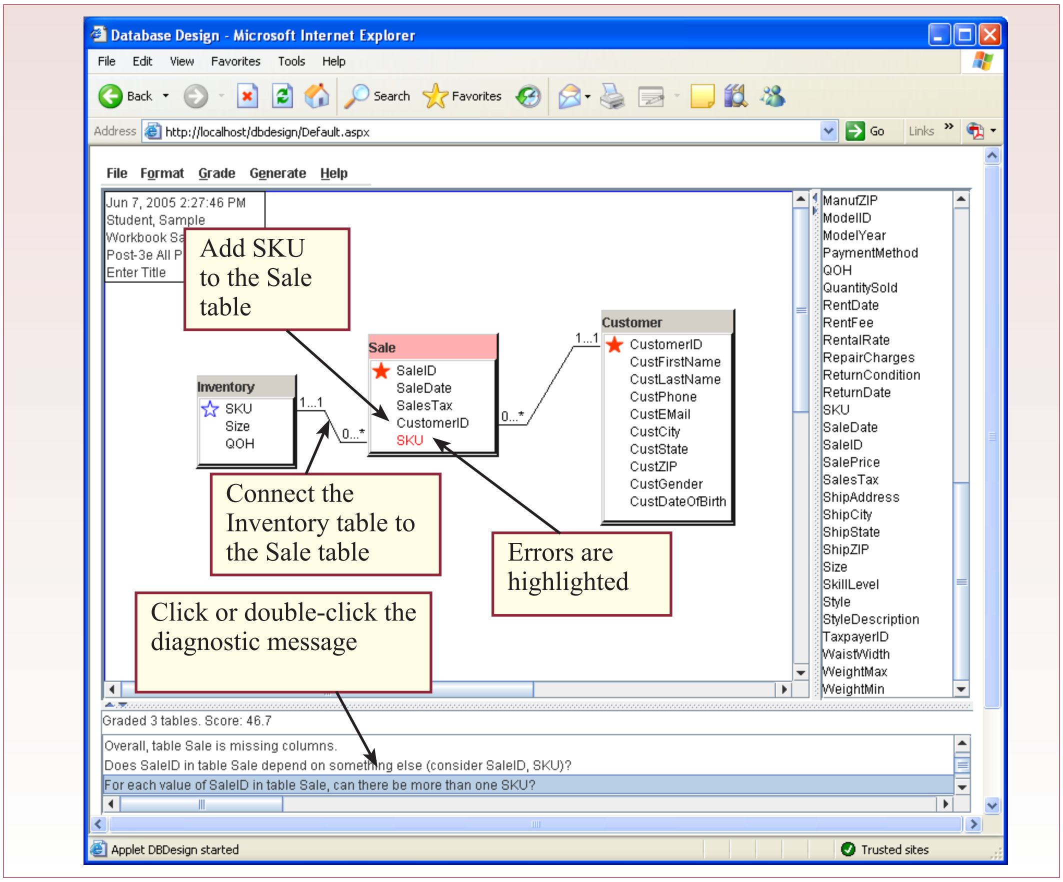
Fix errors by removing columns and adding new tables.

One of the most powerful aspects of the database design system is that it contains an expert system to help analyze your design for errors. You can quickly obtain comments by selecting the Grade/Grade and Mark option on the menu. At this point, you only have two tables partially created, so the most important comment you should receive is that overall, you are missing several tables. The system might also point out that you are missing columns from the Sales table, because you have not yet added the salesperson (employee) and the shipping information.

To illustrate the power of the system, you will add a new table (Item), and then build a new relationship that is incorrect. Add a new table for Inventory, and add the SKU column (a common retail abbreviation for stock-keeping unit) used to identify individual products. Right-click the SKU column in the Inventory table and set it as a key. Add the Size and QOH columns to the Inventory table. Set their data types to Single and Integer16 respectively. Now add the SKU column to the Sale table as an intentional error. Create a relationship from Inventory to Sale using the SKU columns.

Choose the Grade/Grade and Mark menu option to save the changes and obtain comments on the design. Again, the design is not finished, so focus on the other error messages. In particular, find the message "For each value of SaleID in table Sale, can there be more than one SKU?" and click it. Figure 2.8 shows the resulting diagnostic screen. The SKU column in the Sale table is highlighted as a potential problem. Indeed, it is an issue, because placing SKU into the Sale table as shown would mean that for each Sale, only one item (SKU) can be sold. You can usually double-click the comment to receive additional information about database design. In this case, notice that SKU is not part of the primary key. You might consider setting the SKU as a key column in the Sale table to solve the problem. But that would cause even more problems. For instance, the SaleDate depends only on the SaleID and not on the SKU. If you leave SaleDate in the table

Figure 2.8



Graded 3 tables. Score: 46.7

Overall, table Sale is missing columns.
Does SaleID in table Sale depend on something else (consider SaleID, SKU)?
For each value of SaleID in table Sale, can there be more than one SKU?

with both SaleID and SKU set as keys, you would be declaring that items within a single sale can be sold on different dates.

If you set SKU as a key and resubmit the problem for grading, it will return several messages. One of them will be the question "Does SaleDate in table Sales really depend on SKU?" Notice that sometimes a table has many errors, so you must carefully review the entire table to make sure you fix the primary problems first. The Grade menu also contains an option to generate a separate HTML file that lists all errors by table. This listing is easier to print.

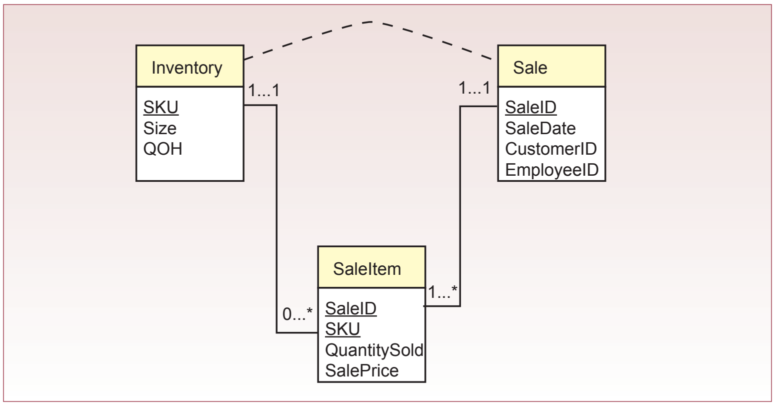
Primary keys are one of the most difficult things for students to understand when they first start designing databases. In particular, generated keys are tricky. In terms of the database design system, primary keys are critical because they are used to identify the tables. If you make major mistakes in the primary keys, the system will give confusing feedback because it cannot correctly identify your tables. For this reason, it is always best to begin with one or two tables, test them, and then slowly add more tables and relationships.

You still need to fix the problem with the Inventory and Sale table association. In a broad sense, it seems that there should be some type of connection between Inventory item and Sale to indicate which items were purchased by the customer. But placing the SKU attribute into the Sale entity appears to be a bad idea. The reason is straightforward. If there is an association between Inventory and Sale, it must be many-to-many. That is, a Sale can include many items (SKUs), and an Inventory item (SKU) can be sold many times. Relational databases do not handle many-to-many relationships directly. Instead, you must create an intermediary or junction table.

Figure 2.9 shows the creation of the intermediary table. It contains the key columns from both the Inventory (SKU) and Sale (SaleID) tables. Both columns are keyed in the new SaleItem table. Examining the keys within the SaleItem tables reveals that each sale can contain many items, and each item can appear on many sales. This is exactly the many-to-many relationship needed. The additional columns of QuantitySold and SalePrice indicate the number of items being purchased and any discounts applied—for an individual item on a specific sale. The dashed many-to-many line is never created, it is simply used here to show the goal of the two relationships.

The new SaleItem table corresponds to the repeating lines of items that you would see listed on a paper sale form. Examining the two new relationships reveals how the table works. Reading from the Sale to the SaleItem table, each sale

Figure 2.9



can contain from one to many items, and in reverse, each SaleItem line (SaleID and SKU) refers to exactly one sale. Essentially the same association exists from Inventory to SaleItem. However, since items might not have been sold, each item can appear on zero to many sales lines, and a given sales line refers to exactly one item. All many-to-many relationships must be split and joined with a junction table that contains the keys from both of the original tables.

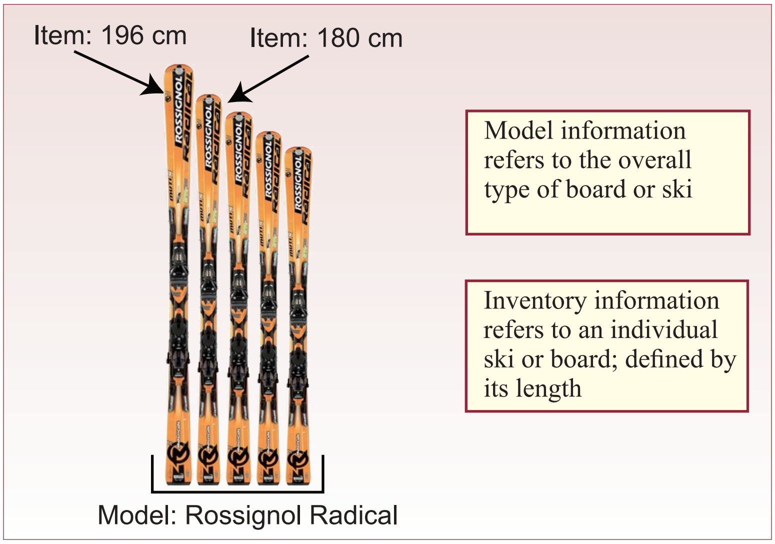
Activity: Fix Inventory Design

Action
Create the SaleItem table.
Create the ItemModel table.
Include the proper columns.
Set the keys.
Set the data types.
Grade/Grade and Mark.

Return to the database design system and delete the association between Inventory and Sale. Then remove the SKU column from the Sale table. Now you can create the SaleItem table. Simply drag the two keys (SaleID and SKU) into the table from the right-hand window—do not attempt to re-create them with a generate key. Double-click to the left of both names to add the simple key icon (unfilled blue star). Build the two new relationships in the Figure 2.9 example and add QuantitySold and SalePrice to the SaleItem table. Make sure the SalePrice data type is Currency and that the data size does not exceed 38, the maximum number of digits allowed in an Oracle number.

If you grade this version, you will see that the detail issues have been corrected. However, some design issues still exist in terms of handling inventory. The inventory for a ski shop is somewhat more complicated than for a typical retail store. In particular, snowboards and skis are sold in varying lengths to match the individual customer. Figure 2.10 shows the two concepts. A manufacturer produces a model line that exhibits certain characteristics such as width, flexibility, and side cut. For a model type, several different lengths are available. From the perspective of the All Powder store, the database has to keep information on each model, but the actual inventory must refer to a specific item or length within the model type. Each item will receive a different SKU. For example, SKU 1173 might refer to a

Figure 2.10


Item: 196 cm
Item: 180 cm
Model information refers to the overall type of board or ski
Inventory information refers to an individual ski or board; defined by its length
Model: Rossignol Radical

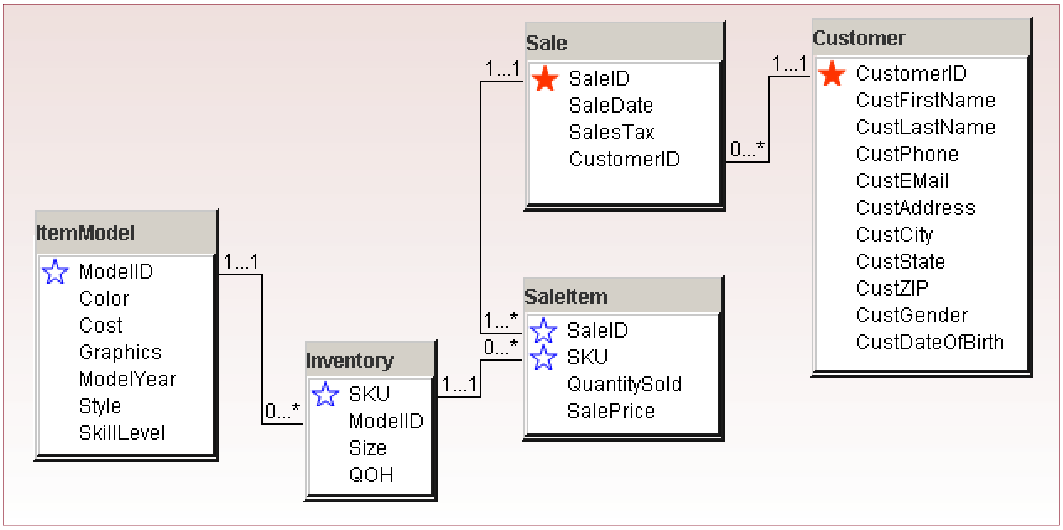


Figure 2.11

Rossignol Radical ski that is 196 cm in length, while SKU 1174 references a Rossignol Radical ski of 180 cm.

The catch is that it would waste considerable space to repeat all of the model data for every possible size of ski or board. Consequently, it is important to create two entities to handle the details: ItemModel and Inventory. Figure 2.11 shows the basic tables and the resulting relationships. Observe that each model results in many inventory items (multiple sizes of boards or skis), but each item can be only one model type. At this point, you should be able to add more attributes and more tables to the design, but the completion of the design will be left to the next chapter.

Exercises

Crystal Tigers

Crystal Tigers is a service club with about 150 members. The club primarily sponsors events such as community pancake breakfasts, local concerts, and sporting competitions. The club successfully uses the events to raise money for various charitable organizations. The club needs a database to help track the roles of the various members, both in terms of positions within the organization and their work at the events. The following form represents the basic data that needs to be collected.

	Year	Position/Title	Comment
Last Name, First Name Phone, Cell Phone Adderss City, State, ZIPCode			

	Member Activitie for Event			
	Date	**Hours**	**Activity**	**Comment**
Event title Start Date End Date Charity Charity contact Phone Amount raised				

1. Analyze the form and create the main classes and associations needed to maintain the data for this organization.

Capitol Artists

Capitol Artists is a partnership among several commercial artists that work on freelance and contract jobs for various clients. Some jobs are contracted at a fixed price, but complex jobs require billing clients for the number of hours involved in the project. To help the artists track the time spent on each project, the firm wants you to build an easy-to-use database. On a given day, the artist should be able to select the time slot, then choose a category and a job. All jobs are given internal numbers, and each job has only one client. But, it is helpful to list the client information on the form once the job has been selected. The artist then enters a short task description, the billing rate, and any out-of-pocket expenses. The billing rate is somewhat flexible and depends on the client, the job, the task, and the artist. For example, the company can charge higher rates for an artist's creative work time, but lower rates for copying papers. The following form contains the basic information desired.

Employee Last name, First name Date								
Time	**Category**	**Client**	**Job#**	**Task**	**Description**	**Hours**	**Rate**	**Expenses**
8:00 AM	Meeting	Name + Phone	1173					
8:30 AM								
9:00 AM								
9:30 AM								
...								

1. Analyze the form and create the main classes and associations needed to maintain the data for this organization.

Offshore Speed

The Offshore Speed company sells parts and components for high-performance boats. Some of the customers modify the boats for racing, others simply want faster boats for informal races. The engine parts tend to be highly specialized and new variations are released each year by manufacturers. Compatibility of parts is always a major issue, but most are tested by the manufacturers with data available from their websites. Customers tend to order parts through the store, but sometimes they will buy off-the-shelf components. The store also keeps many spare parts in stock because customers tend to break them often and the profit margins are good. The store also has arrangements with other firms that can help customers redesign and upgrade interiors and cabins, for example, provide new upholstery for seats and complete systems for beds and sinks for cabins. Lately, the store has been successful in selling and installing high-end GPS and communication systems. The form below is used to place custom orders for the clients. Discounts are given to customers based on several subjective factors that will not be entered into the database.

Customer Last name, First name Phone, E-mail Address City, State, ZIP			Employee Sale date Estiamted receive date			
Boat: Brand, year, # engines, length Engine 1: Brand, year, out drive, year Engine 2: Brand, year, out drive, year						
Manuf.	**Mfg Part No.**	**Category**	**Description**	**Quantity**	**List Price**	**Extended**
					Subtotal Tax Discount Total Due	

1. Analyze the form and create the main classes and associations needed to maintain the data for this organization.

Final Project

The main textbook has an online appendix with several longer case studies. You should be able to work on one of these cases throughout the term. If you or your instructor picks one, perform the following task.

1. Analyze the forms and create the main classes and associations needed to maintain the data for this organization.

Chapter 3

Data Normalization

Chapter Outline

Objectives

- Understand how to use generated AutoNumber keys.
- Create tables and specify data types.
- Create relationships and specify cascades.
- Establish column constraints and default values.
- Create lookup lists for columns.
- Estimate the data volume for the database.

Database Design

The main objective of database design is to define the tables, relationships, and constraints that describe the underlying business rules and efficiently store the data. The normalization rules are critical to properly identifying the columns that belong in each table. The first step is to make sure the keys are correct. A key uniquely identifies the rows in the table. If multiple columns are part of the key, it indicates a many-to-many relationship between the key columns. Note that if a base table contains a generated key column, it is the only column that may be keyed.

If you are uncertain about which columns should be keyed, write them down separately and evaluate the business rules between the two objects. Figure 3.1 shows a typical situation with orders and customers. First ask yourself: For a given order, can there ever be more than one customer? If the answer is "yes" based on the business rules, then you would mark the CustomerID column as key. However, most businesses have a rule that each order is placed by only one customer, so CustomerID should not be keyed. Second, reverse the question and ask yourself: For a given customer, can there be more than one order? Obviously, most businesses want customers to place repeat orders, so the answer is "yes." So you mark the OrderID as key. Since only OrderID is keyed, both columns belong in the CustomerOrder table.

Once the keys are correct, you need to check each nonkey column to ensure that it follows the three main normalization rules. First, each column must contain atomic or nonrepeating data. For example, a single phone number, but not multiple values of phone numbers. Second and third, each nonkey column must depend on the whole key and nothing but the key. You need to examine each potential table, determine that the keys are correct, then check each column to ensure that it depends on the whole key and nothing but the key. If there is a problem, you need to split the table. Remember that any time you make a change to the keys in a table, you have to reevaluate all of the columns in that table.

Generated Keys: AutoNumber

Key columns play a critical role in a relational database. The key values are used as a proxy for the rest of the data. For instance, once you know the CustomerID, the database can quickly retrieve the rest of the customer data. That is why you only need to place the CustomerID column in the CustomerOrder table. However, the database requires key values to be unique. Guaranteeing that key values are never repeated can be a challenging business problem. In some cases, businesses have separate methods to create key values. For instance, the marketing department might have a process to assign identifier numbers to customers and products. But the process must ensure that these values are never duplicated. In many situ-

Figure 3.1

OrderID CustomerID

CustomerOrder(OrderID, CustomerID, …)

ations, it is easier to have the database generate the key values automatically. In particular, orders often require keys that are generated quickly and accurately.

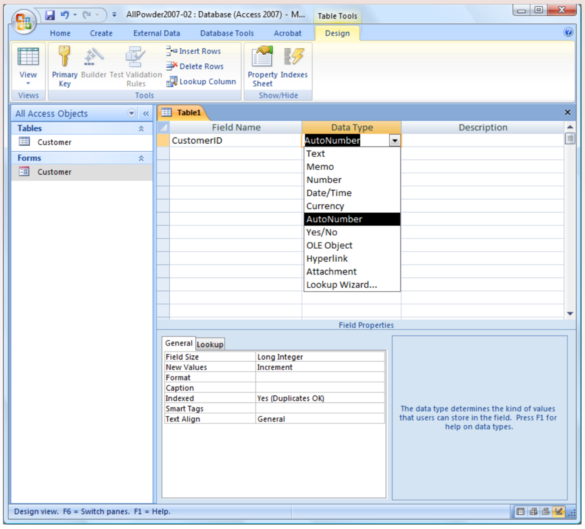
Microsoft Access provides the AutoNumber data type to generate new key values. You assign this type to the primary key in a table where you want the key value created. For instance, the CustomerID column in the Customer table, or the OrderID in the Order table, might be assigned the AutoNumber type. Whenever a row is added to a table with an AutoNumber column, a new key value is generated. The keys are long integers and the value is incremented by one each time a row is inserted. The 32-bit integer supports slightly more than 2 billion positive values and an equal number of negative values.

In general, the AutoNumber approach is relatively easy to use for most simple applications. Figure 3.2 shows how you select the AutoNumber data type when creating a table. No matter how the data is created, whenever a row is inserted a new key value is generated. The rest of the Access system utilizes the AutoNumber column, making it easy to build input forms. In most cases, you do not have to display the key value, so users never need to worry about it or even know that it exists.

The main drawback to the AutoNumber approach is that it is tricky to obtain the newly generated value. If you have an application that creates a new customer, you might have to obtain the newly generated key to use it in a second table. Access does not provide a function to obtain the new key value. There is a way to obtain the key in most cases, but you have to be careful, and it requires programming code.

There is one important step you have to take at this stage when you create tables with the AutoNumber data type. Since the column value can only be created in one table, you need to choose the Number/Long Integer data type if you use that column in a second table. For example, CustomerID can be AutoNumber in the Customer table, so new values are created when a customer is added. But to

Figure 3.2



Field Name	Data Type	Description
CustomerID	AutoNumber	

Field Size	Long Integer
New Values	Increment
Format	
Caption	
Indexed	Yes (Duplicates OK)
Smart Tags	
Text Align	General

The data type determines the kind of values that users can store in the field. Press F1 for help on data types.

Design view. F6 = Switch panes. F1 = Help.

use CustomerID in the Order table, it will have to be assigned the Number/Long Integer data type in that table.

Case: All Powder Board and Ski Shop

When you first approach a database design problem, you will often experience one of two perspectives: the project seems immensely complicated, or the project seems too easy. Usually, both perspectives are wrong. Even a difficult project can be handled if you divide it into small enough pieces, and few projects are as easy as they first appear. The main issue is to correctly identify the business rules. And there always seem to be complications with some of the rules. For the All Powder case, consider the issue of customer skill level. Whether a customer is renting or buying a board or skis, the salespeople need to match the person to the proper board or ski based on the customer's skill level. In terms of business decisions, managers need to identify the types of customers to plan for the models and inventory decisions for next season.

As shown in Figure 3.3, consider what happens if you try to place the Style (downhill, half pipe, and so on), and SkillLevel directly into the Customer table. The problem is that the business rules state that each customer can have one skill level in many styles, and each style can apply to more than one customer. For example, customer Jones could be an expert downhill skier but only a beginner in half-pipe snowboard. However, customer Sanchez is an expert at half pipe but has never tried any type of skiing. If you place Style and SkillLevel in the Customer table, you might try keying only CustomerID. But that action would state that each customer participates in only one style, with one skill level. On the other hand, if you key just the Style column, you would be indicating that each style can be performed by only one person. The only solution is to key both the CustomerID and the Style columns. Then each customer can participate in many styles (with one skill rating per customer per style), and each style can apply to many people (with possibly different skill ratings). But you cannot leave the Style and SkillLevel columns in the main Customer table along with columns such as LastName. It is clear that a customer's last name does not change for each different style. A customer's last name depends only on the CustomerID, so you need to split the tables.

Figure 3.3

Consider what happens if you (incorrectly) try to place Style and SkillLevel in the Customer table:

CustomerID, LastName, … Style, SkillLevel
CustomerID, LastName, … **Style**, SkillLevel

Business rule: Each customer can have one skill in many styles.
Business rule: Each style can apply to more than one customer.
Need a table with both attributes as keys.

CustomerID, LastName, … **Style**, SkillLevel

But you cannot include LastName, FirstName and so on, because then you would have to reenter that data for each customer skill.

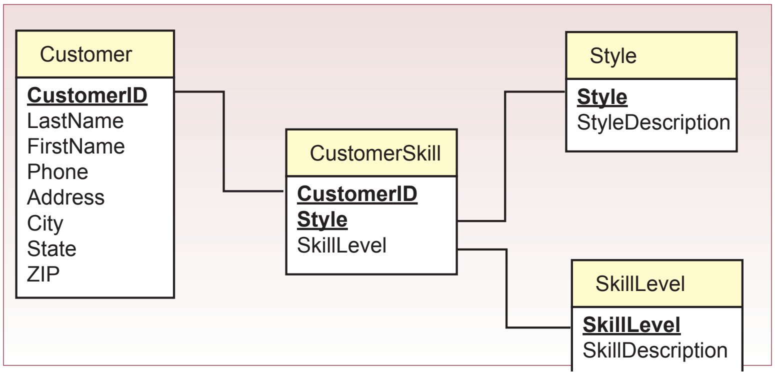


Figure 3.4

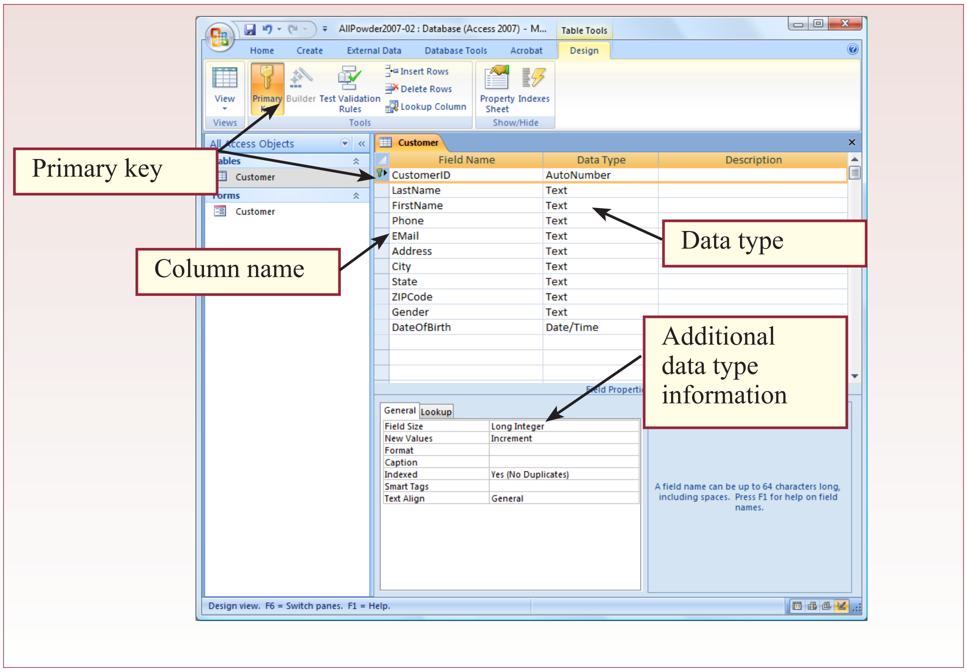
Figure 3.4 shows the resulting design. The Customer table is keyed only by CustomerID and contains attributes that describe each customer. The Style and SkillLevel tables are used as lookup tables to ensure that clerks select from the defined list of choices. Without them, the database would quickly become a mess because everyone would use different spellings and abbreviations for the entries. The CustomerSkill table contains the CustomerID and Style as key columns to support the business rules.

Lab Exercise

All Powder Board and Ski Database Creation

You should use the database design system to refine your table definitions. The system is designed to check the main design rules and ensure that your tables meet the requirements of good database design. However, if you make different

Figure 3.5



Field Name	Data Type	Description
CustomerID	AutoNumber	
LastName	Text	
FirstName	Text	
Phone	Text	
EMail	Text	
Address	Text	
City	Text	
State	Text	
ZIPCode	Text	
Gender	Text	
DateOfBirth	Date/Time	

Field Size	Long Integer
New Values	Increment
Format	
Caption	
Indexed	Yes (No Duplicates)
Smart Tags	
Text Align	General

A field name can be up to 64 characters long, including spaces. Press F1 for help on field names.

Design view. F6 = Switch panes. F1 = Help.

assumptions about the underlying business rules, you can create slightly different tables than those recommended by the design system.

Activity: Create Tables

Action

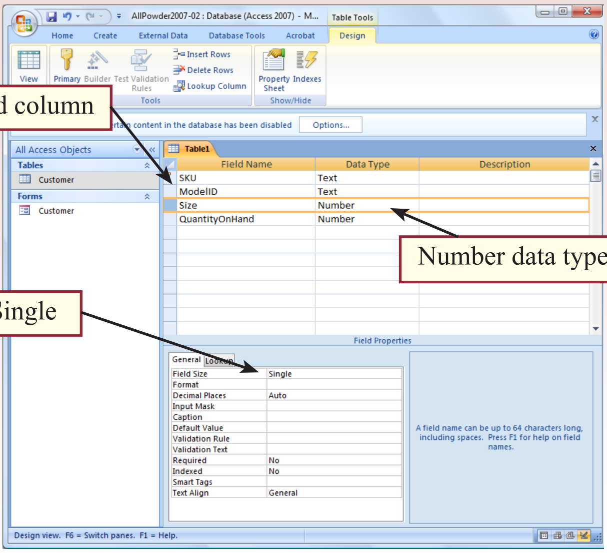
Create Customer table in Design view.
Enter column names.
Select data types.
Assign the primary key.
Save the table.

Once you have determined the overall Database Design, creating tables in Access is straightforward. The Design View visual editor makes it easy to enter column names and select the data types. One of the strengths of Access is that it allows you to change the table design later. So, if you find you made a mistake, you simply go back and edit the table. Of course, if you have already entered data into the table and need to make a major change, it can be more complicated because you will want to save the data so you do not have to reenter it.

Figure 3.5 shows the basic elements of the Table Design screen. As you enter the column (field) names, you select the data type from a drop-down list. Additional choices regarding the data type are made in the lower box. These options depend on the initial data type, so the elements in the box change. For example, the text data type is defined with a maximum of 50 characters by default. Access will efficiently store the data even if it takes less than 50 characters, but it will not allow anyone to enter a value with more than 50 characters. You will need to change this limit for some columns. For example, the Email column should probably allow up to 100 or perhaps 150 characters. Access supports a maximum of 255 characters in a text column. If you need more than that, you will have to switch to a Memo type of data. However, there are some limits on searching memo data, so do not use it unless you truly need the space.

You indicate the primary key by clicking the gray box to the left of the column name, and then clicking the Primary Key icon on the top menu bar. A small Key

Figure 3.6



…rtain content in the database has been disabled | Options...

Field Name	Data Type	Description
SKU	Text	
ModelID	Text	
Size	Number	
QuantityOnHand	Number	

Field Size	Single
Format	
Decimal Places	Auto
Input Mask	
Caption	
Default Value	
Validation Rule	
Validation Text	
Required	No
Indexed	No
Smart Tags	
Text Align	General

A field name can be up to 64 characters long, including spaces. Press F1 for help on field names.

Design view. F6 = Switch panes. F1 = Help.

icon is placed in the gray box to indicate the Key column. To set multiple column fields as keys, you first highlight each column field row that needs to be keyed. Then click the Key icon. You can select multiple rows by clicking and dragging the mouse across contiguous rows. Or you can use Shift + click or Ctrl + click to choose multiple rows. You give the table the appropriate name when you save it.

Although the Access Design view is relatively easy to use, you need to be careful when you select numeric data types. In Figure 3.6, the Size column is given the Number data type. Number, however, has several subtypes. You select the appropriate subtype in the lower window. In this case, the data type of Single indicates a relatively small floating point number. Most skis and boards are measured in centimeters, so the numbers are not overly large. However, some manufacturers might choose to use fractional lengths, so the single-precision floating point is appropriate. This step is sometimes difficult for beginners to catch. If you forget to choose the single or double precision subtype, you will not be able to enter fractional values (with decimal points). If you ever encounter that problem, simply return to the Design view and set the proper data type.

Activity: Create Constraints and Default Values

Action

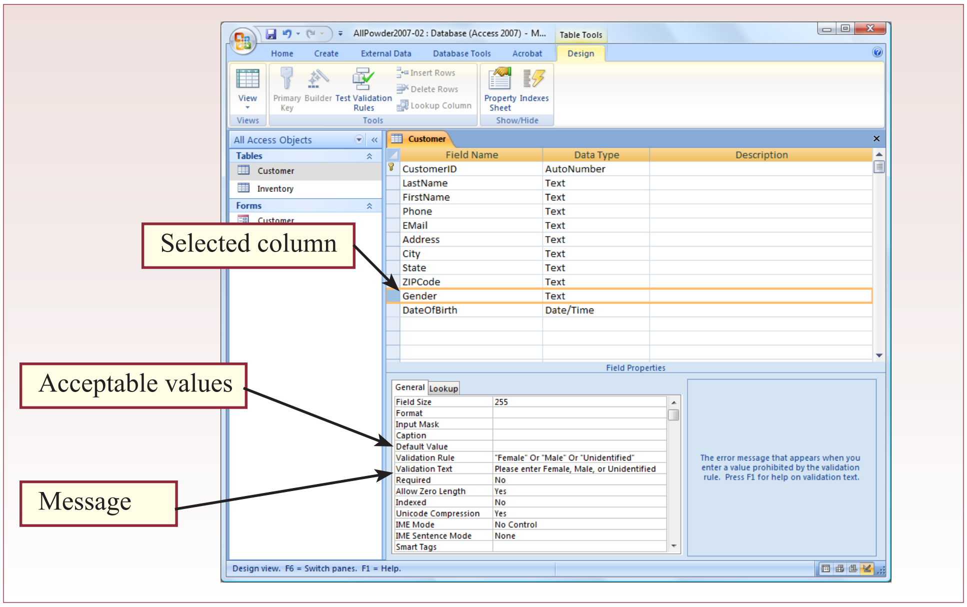
Select the Gender column.

Validation rule: “Female” Or “Male” Or “Unidentified”.

Validation text: Please enter Female, Male, or Unidentified.

In many cases, you will want the database to enforce the business rules. Placing the rules in the database means that they will be enforced in all situations, without relying on other programs. Figure 3.7 shows the primary elements for setting a condition to ensure gender data is entered consistently. First, select the column that needs to be controlled. Second, enter a validation rule that specifies the acceptable values or range of values. Many times, you will use rules that list acceptable values, such as the list of three items for gender. Other times, you might use a rule which specifies that values must fall within some range. For in-

Figure 3.7



Customer

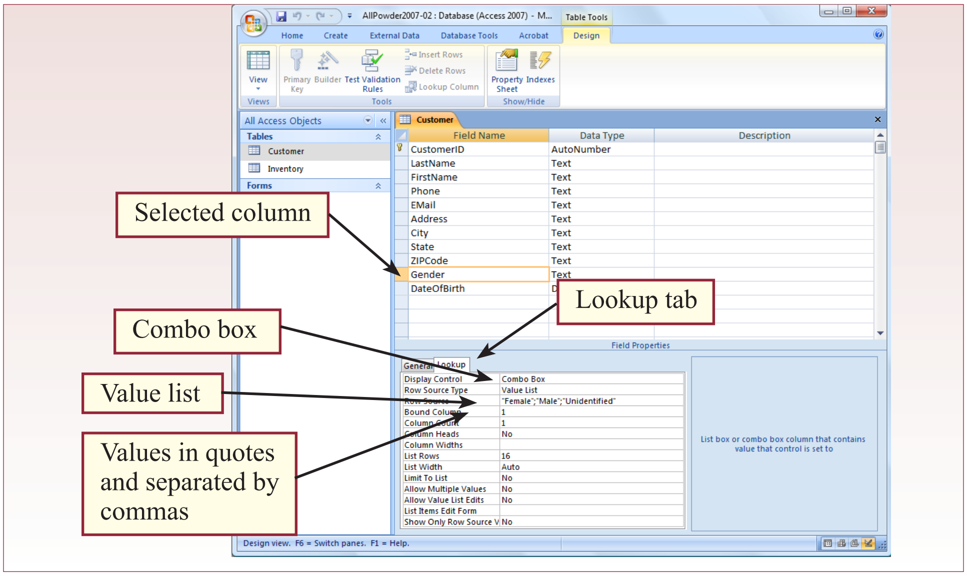
Field Name	Data Type	Description
CustomerID	AutoNumber	
LastName	Text	
FirstName	Text	
Phone	Text	
EMail	Text	
Address	Text	
City	Text	
State	Text	
ZIPCode	Text	
Gender	Text	
DateOfBirth	Date/Time	

Field Properties

General | Lookup

Field Size	255
Format	
Input Mask	
Caption	
Default Value	
Validation Rule	"Female" Or "Male" Or "Unidentified"
Validation Text	Please enter Female, Male, or Unidentified
Required	No
Allow Zero Length	Yes
Indexed	No
Unicode Compression	Yes
IME Mode	No Control
IME Sentence Mode	None
Smart Tags	

The error message that appears when you enter a value prohibited by the validation rule. Press F1 for help on validation text.


Selected column
Combo box
Value list
Values in quotes and separated by commas
Lookup tab
AllPowder2007-02 : Database (Access 2007) - M...
Table Tools
Home Create External Data Database Tools Acrobat Design
View Primary Key Builder Test Validation Rules Insert Rows Delete Rows Lookup Column Property Sheet Indexes
Views Tools Show/Hide
All Access Objects
Tables Customer Inventory Forms
Customer
Field Properties
General Lookup
Design view. F6 = Switch panes. F1 = Help.

Field Name	Data Type	Description
CustomerID	AutoNumber	
LastName	Text	
FirstName	Text	
Phone	Text	
EMail	Text	
Address	Text	
City	Text	
State	Text	
ZIPCode	Text	
Gender	Text	
DateOfBirth	D	

Display Control	Combo Box
Row Source Type	Value List
Row Source	"Female";"Male";"Unidentified"
Bound Column	1
Column Count	1
Column Heads	No
Column Widths	
List Rows	16
List Width	Auto
Limit To List	No
Allow Multiple Values	No
Allow Value List Edits	No
List Items Edit Form	
Show Only Row Source V	No

List box or combo box column that contains value that control is set to

Figure 3.8

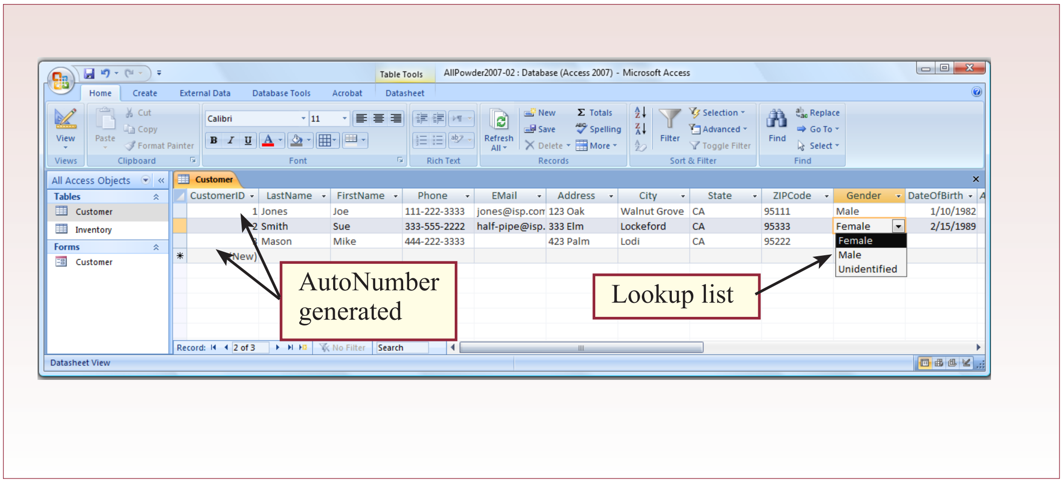
stance, to indicate that a salary column must be greater than zero, select the Salary column and enter >0 as the validation rule. Finally, enter a message that will be displayed to anyone who tries to enter an invalid value. This message can be any text message, but keep it precise and friendly.

Notice that it is also easy to specify default values. These are values you want displayed whenever a new row is created. The user can override the default value and enter something else, but it is often convenient to display a commonly used value to save time for users who are entering data. For example, a SaleDate can be set to the Now( ) function so that the current date is automatically entered.

Activity: Create Lookup Lists

Ultimately, you want to make the data entry process even easier for users. Instead of waiting for users to guess what values they might enter for gender, you can create a simple lookup list that contains the three acceptable values (Female, Male,

Figure 3.9


AutoNumber generated
Lookup list
Table Tools
AllPowder2007-02 : Database (Access 2007) - Microsoft Access
Home Create External Data Database Tools Acrobat Datasheet
View Paste Cut Copy Format Painter Calibri 11
Views Clipboard Font Rich Text
Refresh All New Save Delete Totals Spelling More
Records
Filter Selection Advanced Toggle Filter
Sort & Filter
Find Replace Go To Select
Find
All Access Objects
Tables Customer Inventory Forms Customer
Customer
Record: 2 of 3 No Filter Search
Datasheet View

CustomerID	LastName	FirstName	Phone	EMail	Address	City	State	ZIPCode	Gender	DateOfBirth
1	Jones	Joe	111-222-3333	jones@isp.com	123 Oak	Walnut Grove	CA	95111	Male	1/10/1982
2	Smith	Sue	333-555-2222	half-pipe@isp.	333 Elm	Lockeford	CA	95333	Female	2/15/1989
3	Mason	Mike	444-222-3333		423 Palm	Lodi	CA	95222		
(New)										

Female
Male
Unidentified

or Unidentified). Figure 3.8 shows the choices once you select the column and click the Lookup tab. You can choose between a combo box and a list box. The combo box is usually preferred because it uses less screen space: the choices are displayed when the user clicks the familiar drop-down arrow. For the simple list of pre-defined terms, choose the Value list and enter the values in the next row. Each item is entered in quotes and separated by semicolons.

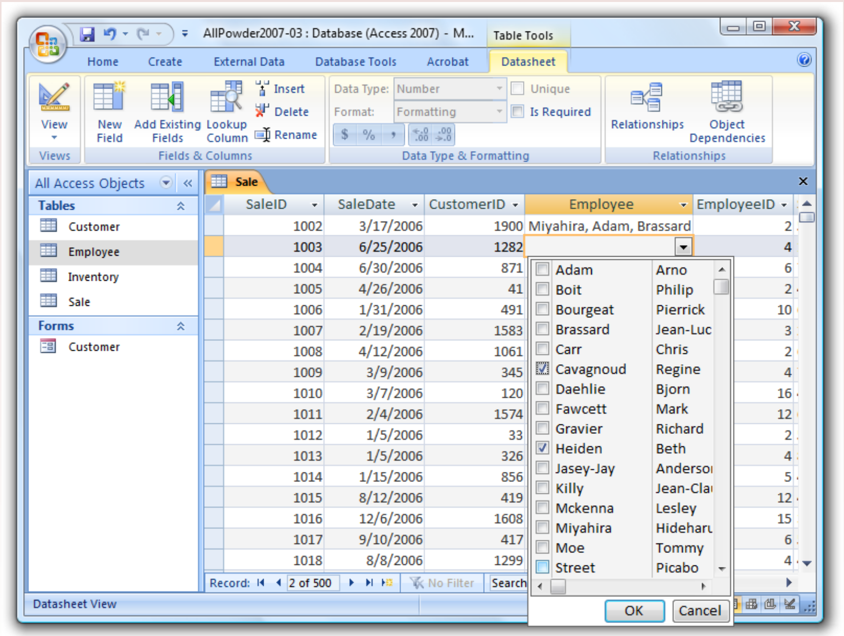
Action

Select the Gender column.
Select the Lookup tab.
Display Control: Combo Box.
Row Source Type: Value List.
Row Source:
“Female”;”Male”;”Unidentified”.
Switch to datasheet view and test it.

There is a major drawback to entering a list of values into the table definition: This list is difficult to change later. Consequently, you should use the predefined list only for items that rarely change. If you suspect the list will need to be updated later, you should create a separate lookup table and store each possible entry as a row within that table. Then you can use this same process to point to that table instead of a fixed list, by selecting Table/Query for the Row Source Type.

Switching to data entry mode, Figure 3.9 shows the effect of this design on the Customer table. In terms of gender, users see a defined list of choices and are not able to enter other values. Even if a programmer attempted to enter a value not in the list, the validation rule would prevent the change. Also, notice the effect of the AutoNumber column for the primary key. Initially, no value is displayed. As data is entered into the row, Access automatically creates a new key value that is one unit higher than the prior value. However, note that if you delete a row, the numbering does not start over. Eventually, some numbers will be missing from the list.

Figure 3.10



SaleID	SaleDate	CustomerID	Employee	EmployeeID
1002	3/17/2006	1900	Miyahira, Adam, Brassard	2
1003	6/25/2006	1282		4
1004	6/30/2006	871		6
1005	4/26/2006	41		2
1006	1/31/2006	491		10
1007	2/19/2006	1583		3
1008	4/12/2006	1061		2
1009	3/9/2006	345		4
1010	3/7/2006	120		16
1011	2/4/2006	1574		12
1012	1/5/2006	33		2
1013	1/5/2006	326		4
1014	1/15/2006	856		5
1015	8/12/2006	419		12
1016	12/6/2006	1608		15
1017	9/10/2006	417		6
1018	8/8/2006	1299		4

☐	Adam	Arno
☐	Boit	Philip
☐	Bourgeat	Pierrick
☐	Brassard	Jean-Luc
☐	Carr	Chris
☑	Cavagnoud	Regine
☐	Daehlie	Bjorn
☐	Fawcett	Mark
☐	Gravier	Richard
☑	Heiden	Beth
☐	Jasey-Jay	Andersoı
☐	Killy	Jean-Clau
☐	Mckenna	Lesley
☐	Miyahira	Hideharu
☐	Moe	Tommy
☐	Street	Picabo

Action

Tools/Relationships (or button).

Add all tables.

Move and resize them to fit.

Drag and drop key columns on foreign keys.

Check integrity and cascade boxes.

Define all relationships.

Since the actual value does not matter, only the uniqueness, do not worry about any missing values.

Access 2007 added a new feature that is of somewhat dubious value, but it does have a specific use when you want to integrate data with Microsoft SharePoint. Access 2007 now supports a multivalued field. In simple terms, it violates first normal form by enabling you to store multiple entries in a single cell. For example, perhaps the owner wants to store data in the Sale table so that multiple employees can participate in a sale, creating a many-to-many relationship between Sale and Employee. Technically, you should store this data in a third, intersection table that contains both SaleID and EmployeeID as columns in the primary key. Actually, this is exactly how Access handles the problem by creating a hidden table. Behind the scenes, Access draws data from this table to make it appear that multiple items can be selected for the SalesPerson.

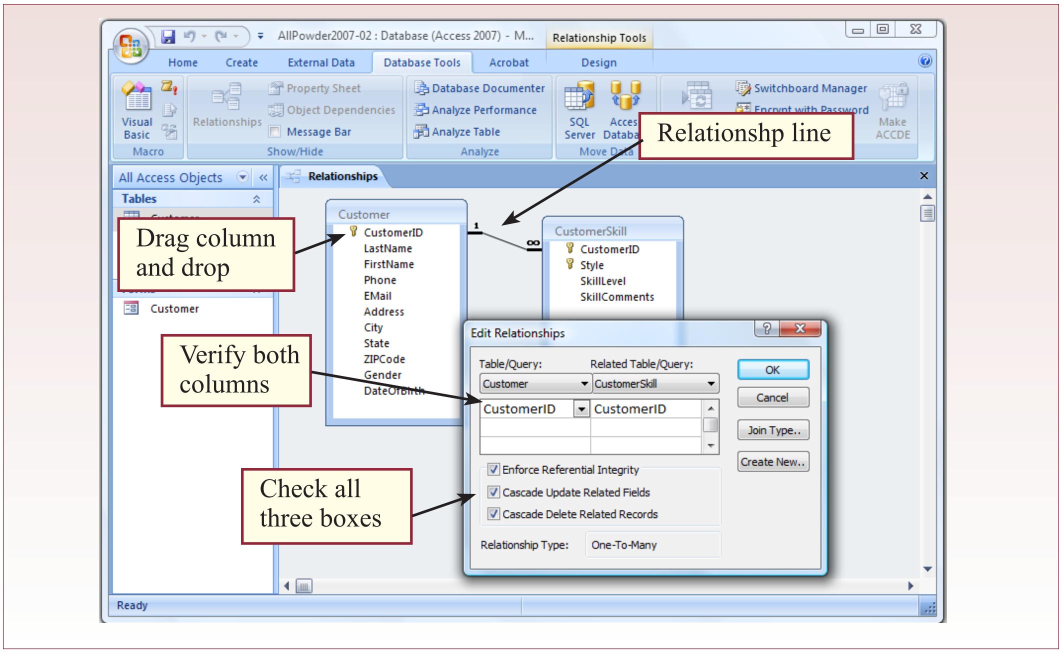
Figure 3.10 shows how the multivalued field is presented to the user to make it easier to enter data. In particular, a list of checkboxes is displayed and the user selects the employees who participated in the sale. The point of the list is to make it easier for user to enter data and save the developer a few steps by automatically creating the list of checkboxes. In most situations, you are probably better off explicitly defining the intersection table and creating your own forms to handle the data entry. Retrieving data with queries involving the new field adds more complications. Access created a new Employee.Value option to display and search the list of individual employees.

Relationships

Activity: Define Relationships

When all of the tables have been created, you need to define the relationships. For now, create at least the CustomerSkill table with two columns in the primary key. You can click the Relationships button under the Database Tools menu. Select the tables to be displayed on the screen. Eventually you will need all of the tables, but sometimes it is easier to start with a smaller group and add tables one at a time with the Add Tables button. As shown in Figure 3.11, to build a new relationship, click-and-drag a column from one table and drop it on the corresponding column of the second table. To edit an existing relationship, double-click the relationship line. In the Edit Relationships window, verify that the column names are correct. If a relationship needs to join across multiple columns, you must select the additional columns in the list. When they are correct, check all three of the integrity boxes. These boxes enforce the integrity rules so that users can only enter data that exists in the original base table. For example, it does not make sense to enter skill data for a customer number that does not yet exist, so the database will enforce that rule and display an error message if someone tries to violate it. Click the OK button to establish the new relationship.

Continue adding tables and building the required relationships. Eventually, all of the tables should be tied together. An important consequence of building the relationships is that it forces you to enter data in a specific sequence. In the ex-



Edit Relationships

Table/Query: Customer — Related Table/Query: CustomerSkill

CustomerID — CustomerID

[✓] Enforce Referential Integrity
[✓] Cascade Update Related Fields
[✓] Cascade Delete Related Records

Relationship Type: One-To-Many

Figure 3.11

ample, you must first enter data into the SkiBoardStyle, SkillLevel, and Customer tables before you can create an entry in the CustomerSkill table. In general, you should first enter data into the base lookup tables, like the SkillLevel table. This data is generally relatively static and well-defined. Notice that it could be difficult to explain this process to users. To look ahead a little, users should never enter data directly into tables. Instead, you will create forms that make it easy for users to enter data properly.

Figure 3.12

```
CREATE TABLE Customer (
        CustomerID          Long,
        LastName            Text(50),
        FirstName           Text(50),
        Phone     Text(50),
        Email     Text(150),
        Address   Text(50),
        State     Text(50),
        ZIP                 Text(15),
        Gender    Text(15),
        DateOfBirth         Date,
        CONSTRAINT pk_Customer PRIMARY KEY (CustomerID)
)
CREATE TABLE Rental (
        RentID    Long,
        RentDate Date,
        CustomerID          Long,
        ExpectedReturn      Date,
        PaymentMethod       Text(50)
        CONSTRAINT pk_Rental PRIMARY KEY (RentID)
        CONSTRAINT fk_RentalCustomer FOREIGN KEY (CustomerID)
                  REFERENCES Customer(CustomerID)
)
```

As of the XP/2002 version of Access, you can create tables and relationships using direct SQL statements. However, the syntax does not support all of the SQL standard options. For instance, only a single command can be issued at a time. Figure 3.12 shows how you could use SQL to create two of the tables for the case. Note that you will have to create each table separately, but they are displayed together to highlight the relationship. You can open a new query and switch to SQL view, then enter the CREATE TABLE command and execute the query to create the table.

Why would you want to use this method to create tables—particularly when the Table Design view screen is so easy to use? The answer is that the Design View screen requires someone to create each table and column by hand. That is fine when you are initially creating tables and if you never have to re-create them. However, the text-based approach enables you to save the table definitions so that you can quickly re-create the tables in a new database. It is a useful way to save backup copies of the table definitions. These backup copies are useful when you need to modify the tables at some point, or create a new database, or have someone else re-create the database on a different machine. The method is also useful when a design system generates the CREATE TABLE statements for you.

If you create a file listing all of the tables, be careful about the order in which the tables will be created. Some tables must be created before others. In particular, a table can only establish a foreign key reference to an existing table. In the example, the Customer table has to be created before the Rental table can reference it. Also, keep in mind that the CREATE TABLE command is a recent feature in Microsoft Access, so it does not fully support the SQL standard. In particular, it does not appear to support the ON DELETE CASCADE and ON UPDATE CASCADE options for the foreign key relationship. You will still have to go to the Relationships screen, edit the relationship, and check those boxes separately. Also, Access 2007 does not support the CHECK and DEFAULT options to specify individual column constraints and default values.

Activity: Estimate the Database Size

Action

Create a spreadsheet.
Enter table names as rows.
Add columns for: Bytes, Rows, Totals.
Calculate the bytes per table row.
Estimate the number of rows.
Compute the table and overall totals.

At some point, you need to estimate the size of the database project. Of course, any estimate at this early stage will be very rough. Your goal is not to be perfect but to be able to categorize the overall project size. The information will help you identify the basic category of database server and perhaps narrow your choice of tools. In particular, it will help you determine whether the project is too large to be handled in Access, or if you should move up to SQL Server or Oracle. You still might want to use Access as a front-end tool, but it would be helpful to know if you need to move the database to a larger server.

To estimate the database size, you begin by estimating the size of each data table. You must already know which columns belong to each table. Figure 3.13 shows the process for the Customer table. Some of the column size estimates are straightforward. Look back to Chapter 2 to see that a long integer uses 4 bytes of storage in Access. The text columns are a little trickier. For instance, although the database will allow up to 50 characters of text for the last name, almost no names will actually be that long. Instead, you need to estimate the average length

CustomerID	Long	4
LastName	Text(50)	30
FirstName	Text(50)	20
Phone	Text(50)	24
Email	Text(150)	50
Address	Text(50)	50
State	Text(50)	2
ZIP	Text(15)	14
Gender	Text(15)	10
DateOfBirth	Date	8

Average bytes per customer	212
Customers per week (winter)	*200
Weeks (winter)	*25
Bytes added per year	1,060,000

Figure 3.13

of customer last names. You could use existing data or perhaps take a sample from a phone book. Perhaps an average last name is 15 characters long. But the DBMS stores text in Unicode format, which requires 2 physical bytes of storage for each character, so the average storage space needed for a last name is 30 bytes. Use a similar process to estimate the number of bytes needed to store an average row of customer data.

Next, you need to estimate how many new customers will arrive each year. In a real case, you could look at past records or talk with the expert users. Here, assume it is about 200 per week, but there are only 25 weeks of the ski season; so estimate 5,000 new customers a year. Multiplying the estimated number of customers by the size of an average row yields the initial data size of the customer table of about 1 million bytes.

Follow a similar process for all of the tables in the case. Figure 3.14 lists some of the basic assumptions you can use. You should build a spreadsheet that lists each table, the average number of bytes per row, the estimated number of rows, and the total estimated size for the table. There is still some flexibility in the final number, but your estimate should be around 5 to 6 megabytes. Remember that this is data for only one year. Also, additional space will be required for indexes, overhead, queries, forms, and reports. But even if the final number is closer to 20 megabytes, Microsoft Access should be able to handle the database on a typically configured computer.

Figure 3.14

200 customers per week for 25 weeks
2 skills per customer
2 rentals per customer per year
3 items per rental
20 percent of customers buy items
4 items per sale
100 manufacturers
20 models per manufacturer
5 items (sizes) per model

Exercises

Many Charms

Samantha and Madison want you to build the database for their charms sales. They emphasized that the system has to be easy to use. They also pointed out that a key element of their business is tracking all of the products and the various suppliers, then monitoring the costs so they can set their prices accurately. They are also concerned about monitoring how quickly their charms sell. They figure they will need to start with at least 200 basic charms, but most charms come in two sizes, along with the different metals and finishes. When asked, the women indicate they are uncertain how many customers they will have but would like to get at least 50 sales a week. Although some of the sales might be small, they hope to build a solid list of clients who return for new purchases on a monthly basis. To encourage return customers, they are thinking about offering some type of frequent-buyer program, where customers receive discounts or maybe a free charm, after purchasing a specified number of charms.

1. Define the final tables needed for this case.
2. Create the database.
3. Estimate the size of the database for one year of operation.

Standup Foods

Laura's business has been established for several years. Many of her clients are old customers, and she has a couple of thousand in her files—although some have gone out of business. Her business has grown considerably based on referrals from existing clients. She gets so many good comments and referrals, she is thinking that she needs to track which customers pass her name on to others so she can call them or send thank-you gifts. But, her more immediate concern is tracking employees. Over the course of a year, she has a relatively high turnover in some positions. Other employees have been with her for years. In total, she probably deals with 400 to 500 employees a year. Employees are rated after each job, and typically employees work 15 to 20 jobs a year for her. On average, employees tend to have three tasks per event. For instance, a driver will also be a server, and possibly also a busboy or dishwasher. They are evaluated on 10 items for each task they perform, as well as given an overall rating. Client food preferences are somewhat more complex, so Laura wants the capability to add free-form comments to cover extreme cases. For common elements, such as allergies to nuts, she wants to keep itemized lists—both for desired items and forbidden items. Some clients are easy going, but this is Hollywood, so many have long lists of items—often ranging to 50 or even up to 100 items.

1. Define the final tables needed for this case.
2. Create the database.
3. Estimate the size of the database for one year of operation.

EnviroSpeed

For good or bad, Tyler and Brennan have been busy. Their firm has been averaging four to five cleanups a week. Although there are not many permanent employees (fewer than 100), they have close associations with about 200 experts in various areas. All of these people need access to the environmental documents and other information. Additionally, about 400 crews around the world are called in to work on various problems. The crews consist of 10 to 20 people. Initially, experts contribute the most information. Sometimes an expert will contribute hundreds of pages of documents and comments. Once an incident is opened, most of the new data and the searches come from the emergency crews. Time schedules, environmental factors, and comments can arrive quickly from all of the crew members. Some of the notes are on paper and saved until the emergency is over, when clerks enter the basic data to the database. A typical incident can generate dozens of pages of notes and schedules from each crew member. Although there are hundreds of possible chemicals, the firm has found that only about 50 major chemicals are typically involved in critical incidents. One important aspect of this case is the need for experts and crew members to search through documentation based on key words. For example, crews will need to search for certain chemicals, possibly in combination with other chemicals, and often include the type of problem, such as water or road spill. Brennan estimates a typical document needs to include at least 20 keywords to identify the exact purpose of the document.

1. Define the final tables needed for this case.
2. Create the database.
3. Estimate the size of the database for one year of operation.

Final Project

The main textbook has an online appendix with several longer case studies. You should be able to work on one of these cases throughout the term. If you or your instructor picks one, perform the following instructions.

1. Finalize your database design.
2. Create the tables in the DBMS.
3. Estimate the amount of data that might be generated for one year.

Chapter 4

Database Queries and SQL

Chapter Outline

Objectives

- Create or import sample data into a database.
- Create basic queries to answer common business questions.
- Use joins to create multitable queries.
- Use queries to perform simple calculations.
- Answer business questions involving totals and subtotals.

Database Queries

Relational databases are designed to store data efficiently. Efficiency results in splitting the data into many tables, interconnected by the data. Consequently, you need a good query system to retrieve data. SQL is a powerful standard designed to perform several tasks in retrieving and manipulating data in relational database systems. Most modern systems implement some version of SQL. The catch is that the standard continues to evolve, and it takes time for the DBMS vendors to catch up. Also, vendors tend to include proprietary extensions to provide additional features. Because the SQL syntax can be difficult for managers, most vendors also include some type of query-by-example (QBE) system to make it easier to construct common queries. The visually oriented QBE also saves some typing effort and provides lists of tables and columns so you do not have to memorize them. But, ultimately, you need to learn SQL because it is a standard and because some queries are easier to understand in SQL. With Access, the query designer makes it easy to switch from QBE to SQL, so you should get in the habit of checking the SQL to see that the query is being built correctly.

This chapter focuses on the data retrieval aspects of queries. SQL can also be used for data definition (e.g., CREATE TABLE), and for data manipulation (e.g., UPDATE and DELETE). These features and more complex queries are covered in Chapter 5. Once you learn the foundations of queries presented in this chapter, the other topics are easier to understand.

In any database, it helps to have a copy of the class (relationship) diagram handy when you are writing queries. One of the more difficult aspects to creating a query is to find which tables hold the data you need. This problem is one of the reasons it is so important to label your tables and columns carefully when you create the database. Managers need to be able to identify the tables and columns that match the business questions. With dozens or even hundreds of tables with confusing or abbreviated names, it can be difficult to find the correct data.

Case: All Powder Board and Ski Shop

Before you can build queries, you need data in the tables. Even with a small number of tables, it is time-consuming to create reasonable data. You have to match the foreign keys across the relationships. For instance, it is straightforward to create basic customer data, although it would take a while to type in data for a thousand customers. But then, when you want sales data, you have to select CustomerID values from the existing list. You also have to create ski and board models and then generate data for items with appropriate attributes, then choose the proper ID values for the sales and rentals. In a typical business project, you can test the database with a few dozen examples, and then wait for the business to generate real data to analyze. In a class setting, it is better to use sample data. For that reason, sample data is available for the tables in the All Powder case. The one catch is that your tables might not contain exactly the same columns. So you might have to edit the data slightly in Excel before you import it into your database. This data was randomly generated with specially built generators. The business interpretations might not be useful, but the dataset is consistent.

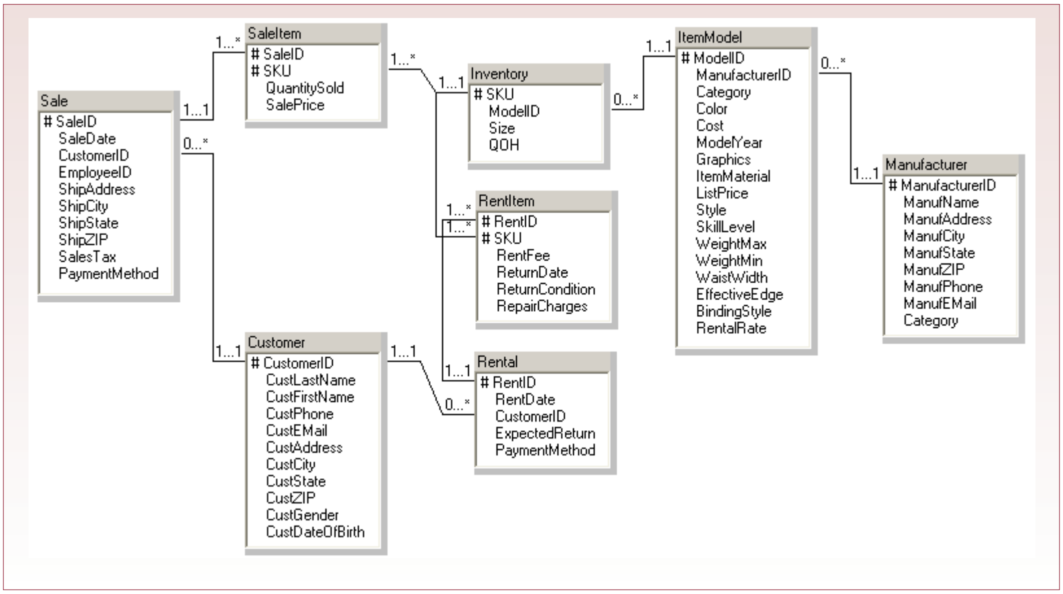


Figure 4.1

Lab Exercise

All Powder Board and Ski Data

At this point, the main tables of your database should be similar to those in Figure 4.1, although several supporting tables have been removed from the figure. The Manufacturer, Customer, Sale, and SaleItem tables are common to most business databases. The Rental and RentItem tables simply mirror the sale aspects. The Inventory and ItemModel tables arose because of the characteristics of the board and ski products.

To save time and effort, a sample database is included with initial data that you can use to test queries. In real-world cases, you will need to import data. Access is quite flexible about important data, but it still takes time to setup the database. sample data files are provided for each of these tables, plus the common supporting tables. Access can easily import the data files into your tables. To import data, you will first have to extract the files from the compressed archive. Also, if your table definitions are different from these, you will have to edit the data files. For example, your ItemModel table might not have all of the columns. In this case, you can open the data file in a spreadsheet program, delete the unneeded columns, and save the file in comma-delimited format.

Activity: Create Basic Queries

Creating a query requires that you translate a business question into a form that the query system can process. Sometimes this step is straightforward, other times it is difficult. It helps if you format your query in terms of the four main questions: (1) What do you want to see? (2) What do you know or what are the constraints? (3) What tables hold the data? (4) How are the tables connected? You can use the query grid in Access to reduce the typing, or you can switch to SQL. Either way, you have to answer the four questions. Figure 4.2 shows the basic

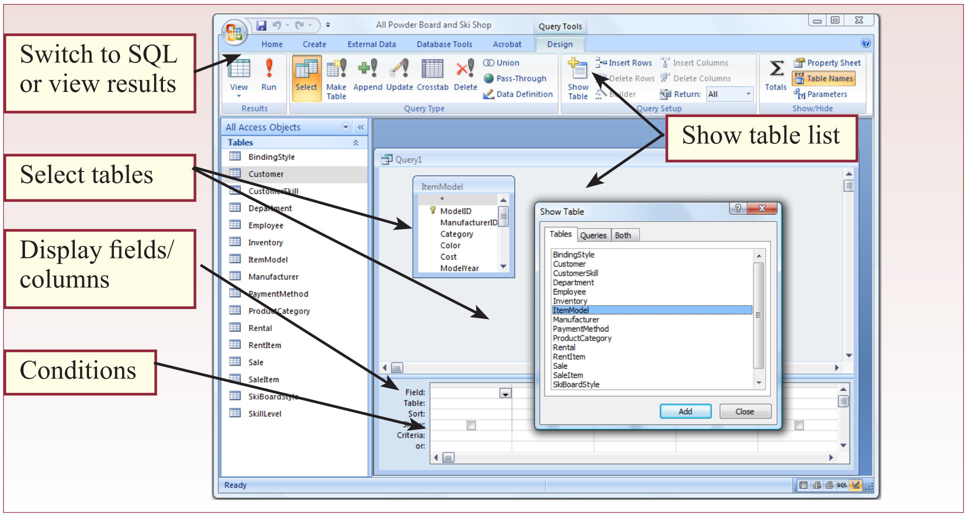


Figure 4.2

elements of the query grid. Tables that have been selected are displayed in the top half of the grid screen. Connections among them will be displayed as lines. Columns or computed fields that you want to see are displayed in the top row of the grid. Conditions that apply to individual columns are displayed in the Criteria row. You can also sort the results by selecting the appropriate sort order in any column.

Action

Choose Create/Query/Query Design.

Add the ItemModel table.

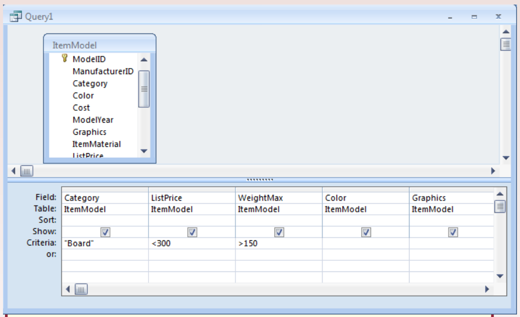
Select columns for Category, ListPrice, WeightMax, Color, and Graphics.

Enter conditions for Board, ListPrice, and Weight.

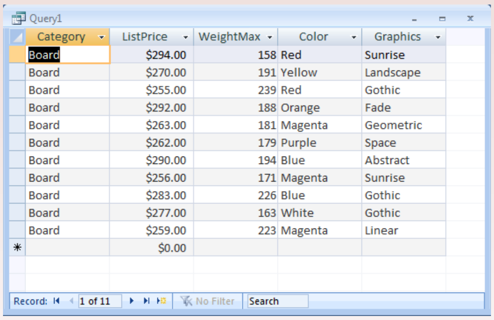
Run the query.

Begin with a straightforward query: Display the snowboards with a list price under $300 for riders over 150 pounds. The potential buyer wants to know what color and graphics are available for boards that meet those conditions. The most difficult step in this query is to identify the table and columns that match the conditions. For example, snowboards are identified by the Category column in the ItemModel table. If you examine the data, you will see a Board entry for each item that is a snowboard. The list price, maximum weight, color, and graphics columns are also in the ItemModel table.

Figure 4.3 shows the basic query, both using the query grid and switching to the SQL. In the grid, once you have selected the ItemModel table, you can double-click or drag the desired columns down to the display grid. In SQL, you would type the names of the columns you want to see immediately after the SELECT command. Notice that Access automatically includes the name of the table (ItemModel.Category). With only one table, this step is not required, but Access plans ahead in case you add more tables that might have similar column names. The conditions you are given (snowboard, list price, and weight) are entered on the Criteria row of the grid, or typed immediately after the WHERE clause of the SQL statement. The grid conditions can be tricky at first. Notice that you enter < 300 in the ListPrice column. Since it is in the ListPrice column, this line is read as ListPrice < 300. Because the three conditions are on the same Criteria row, they are connected with ANDs, meaning all three conditions must be true. The SQL



Field:	Category	ListPrice	WeightMax	Color	Graphics
Table:	ItemModel	ItemModel	ItemModel	ItemModel	ItemModel
Sort:					
Show:	✓	✓	✓	✓	✓
Criteria:	"Board"	<300	>150		
or:					



Category	ListPrice	WeightMax	Color	Graphics
Board	$294.00	158	Red	Sunrise
Board	$270.00	191	Yellow	Landscape
Board	$255.00	239	Red	Gothic
Board	$292.00	188	Orange	Fade
Board	$263.00	181	Magenta	Geometric
Board	$262.00	179	Purple	Space
Board	$290.00	194	Blue	Abstract
Board	$256.00	171	Magenta	Sunrise
Board	$283.00	226	Blue	Gothic
Board	$277.00	163	White	Gothic
Board	$259.00	223	Magenta	Linear
*	$0.00			

Display snowboards with a list price under $300 and max weight over 150 pounds.

SELECT ItemModel.Category, ItemModel.ListPrice, ItemModel.WeightMax, ItemModel.Color, ItemModel.Graphics

FROM ItemModel

WHERE (ItemModel.Category="Board") AND (ItemModel.ListPrice<300) AND (ItemModel.WeightMax>150)

Figure 4.3

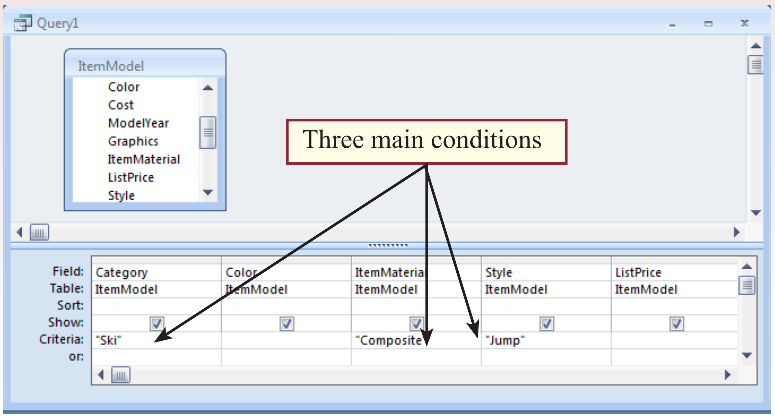
WHERE clause is a little easier to read, but it has more parentheses than necessary. At any time you can switch between the grid and the SQL by selecting the desired view. The results show the 11 boards that meet the desired conditions.

Activity: Create and Test Multiple Boolean Conditions

Interpreting business questions can sometimes be difficult because of the ambiguity of natural languages. It is one of the reasons SQL remains so important. SQL requires you to specify exactly what you want to see and to write the conditions mathematically. Of course, these conditions can become relatively long when the business question is complex. Consider a customer who wants skis for jumping. She wants them made from composite materials, and the main color can be red

Figure 4.4

Ski for jumping.
Composite material.
Red or Yellow as main color.
Yellow skis must be under $300.
Red skis must be under $400.



Field:	Category	Color	ItemMaterial	Style	ListPrice
Table:	ItemModel	ItemModel	ItemModel	ItemModel	ItemModel
Sort:					
Show:	✓	✓	✓	✓	✓
Criteria:	"Ski"		"Composite"	"Jump"	
or:					

Action

Create a new query in Design View.

Add the ItemModel table.

Add columns: Category, Color, ItemMaterial, Style, and ListPrice.

Set requested conditions.

Check the SQL.

Run the query.

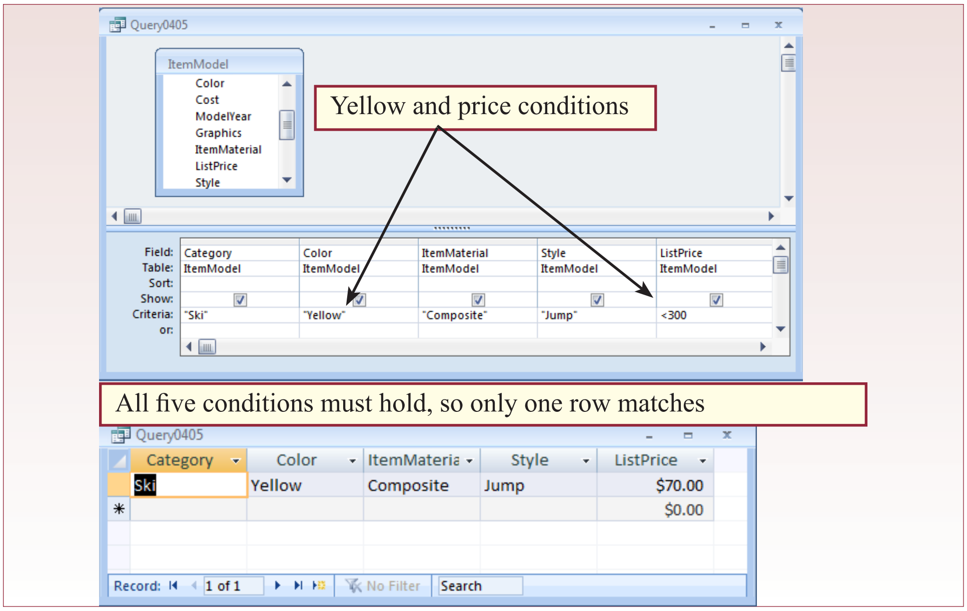
or yellow. She does not want to spend more than $300, but if they are red, she is willing to pay up to $400.

Begin with a new query, and again recognize that all of the attributes are in the ItemModel table. Looking through the data, the first three conditions are straightforward: the Category is Ski, the ItemMaterial is Composite, and the Style is Jump. The colors appear to be straightforward, except that the choice is connected with Or. Whenever a query contains both And and Or conditions, you must be careful, so start with basic conditions and check the results as you go. Figure 4.4 shows the initial query with the three main conditions that must always hold (ski, jump, and composite).

Now you can think about how to add the other two aspects of the question. If yellow skis must cost less than $300, what happens if you add both conditions to the query? Figure 4.5 shows the query and the results. Since all of the conditions are on the same Criteria row, all five must be true at the same time. The query therefore returns only yellow skis.

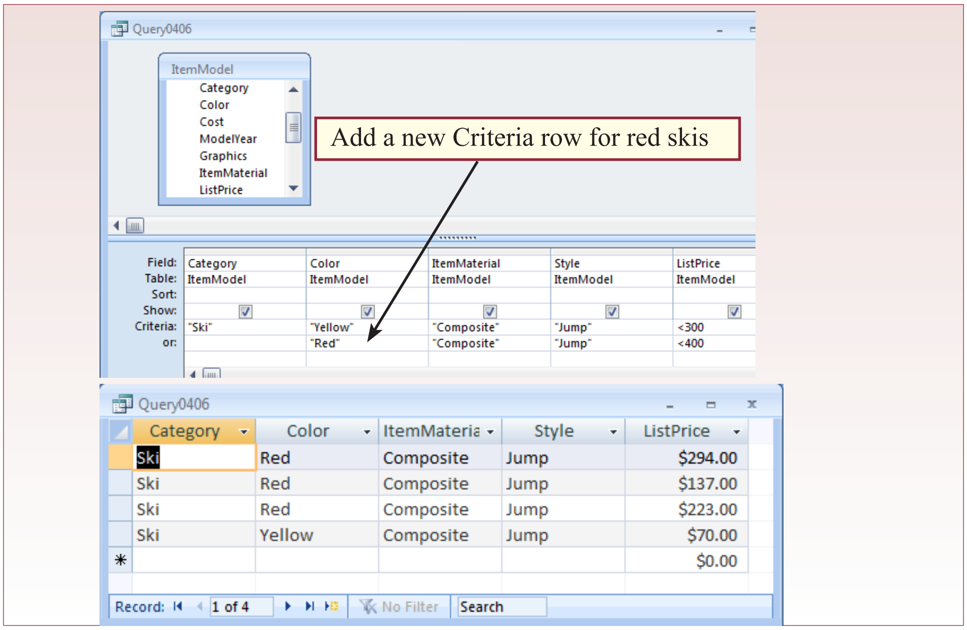
To see the red skis, you somehow have to add the option of Red as a color, but you also have to establish the higher acceptable price for red skis. The solution is to add a new criteria row. Again, the Category, ItemMaterial, and Style are set to Ski, Composite, and Jump. As shown in Figure 4.6, the Color this time is set to Red and the ListPrice must be under $400. Conditions on multiple rows are connected with Or clauses, so the final query displays skis that are yellow or red. The yellow skis must be composite jumping skis priced below $300. The red skis must be composite jumping skis priced below $400. Note that the grid requires you to duplicate the main conditions that hold true across the yellow and red skis.

Figure 4.5



Field:	Category	Color	ItemMaterial	Style	ListPrice
Table:	ItemModel	ItemModel	ItemModel	ItemModel	ItemModel
Sort:					
Show:	✓	✓	✓	✓	✓
Criteria:	"Ski"	"Yellow"	"Composite"	"Jump"	<300
or:					

Category	Color	ItemMateria	Style	ListPrice
Ski	Yellow	Composite	Jump	$70.00
				$0.00



Field:	Category	Color	ItemMaterial	Style	ListPrice
Table:	ItemModel	ItemModel	ItemModel	ItemModel	ItemModel
Sort:					
Show:	✓	✓	✓	✓	✓
Criteria:	"Ski"	"Yellow"	"Composite"	"Jump"	<300
or:		"Red"	"Composite"	"Jump"	<400

Query0406

Category	Color	ItemMaterial	Style	ListPrice
Ski	Red	Composite	Jump	$294.00
Ski	Red	Composite	Jump	$137.00
Ski	Red	Composite	Jump	$223.00
Ski	Yellow	Composite	Jump	$70.00
				$0.00

Record: 1 of 4 No Filter Search

Figure 4.6

If you switch from the grid view to the SQL view, the SQL query looks complex. Part of the problem is the excessive use of parentheses, but the query is also longer than it needs to be because Access duplicates the three main conditions. Figure 4.7 shows the SQL version generated from the query grid. However, you would never type the query in that way. Instead, the query can be simplified by removing the excess parentheses and the unneeded table name prefixes. The lower half of Figure 4.7 shows the simpler SQL query. You should enter this query into the SQL editor in Access and verify that it produces the same answers as the grid query. The SELECT statement lists the columns that you want to see. The FROM clause lists the single table. The WHERE clause begins with the three items that must be true. It also includes the two choices of colors with their associated price limits.

Figure 4.7

```
SELECT ItemModel.Category, ItemModel.Color, ItemModel.ItemMaterial,
ItemModel.Style, ItemModel.ListPrice
FROM ItemModel
WHERE (((ItemModel.Category)=”Ski”) AND ((ItemModel.Color)=”Yellow”)
AND ((ItemModel.ItemMaterial)=”Composite”) AND ((ItemModel.Style)=”Jump”)
AND ((ItemModel.ListPrice)<300)) OR (((ItemModel.Category)=”Ski”) AND
((ItemModel.Color)=”Red”) AND ((ItemModel.ItemMaterial)=”Composite”) AND
((ItemModel.Style)=”Jump”) AND ((ItemModel.ListPrice)<400));

Simplified:

SELECT Category, Color, ItemMaterial, Style, ListPrice
FROM ItemModel
WHERE (Category=”Ski” AND ItemMaterial=”Composite” AND Style=”Jump”)
AND ( ( Color=”Yellow” AND ListPrice<300)
OR (Color=”Red” AND ListPrice<400) );
```

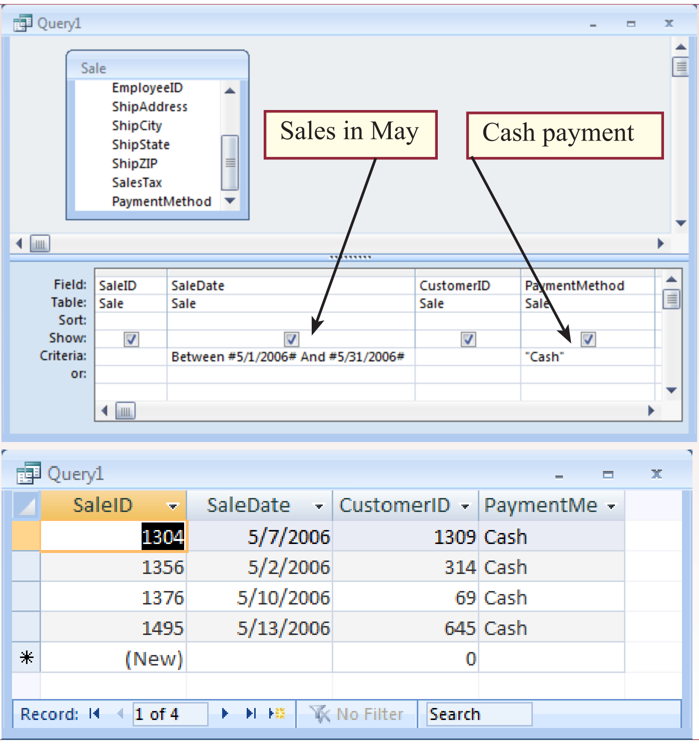
Activity: Use Multiple Tables in a Query

Action
Create a new query in Design View.
Choose the Sale table.
Select columns: SaleID, SaleDate, CustomerID, and PaymentMethod.
Set conditions for Cash sales in May.
Choose Query/Show Table (or button).
Add the Customer table.
Add columns: LastName, FirstName.
View the SQL.
Run the query.

Relational databases require the tables to be carefully designed so that the DBMS can efficiently store large amounts of data. This process entails placing data into multiple tables. Consequently, a key feature of SQL is its ability to join the tables to make it easy to retrieve data from many tables with one query. The Access query grid makes it easy to join tables. Once the tables are joined, you can generally treat the columns as if they came from a single table.

To understand the joining process, create a new query and add just the Sale table. The objective is to find all of the sales in May that were made with a cash payment. Figure 4.8 shows the initial query. Note the use of the Between clause to specify the month of May. Also observe that Access automatically places pound signs (#) around the dates. These marks enable you to enter dates in any common date format. Notice that the query returns the CustomerID, but no one is going to memorize CustomerID numbers. Instead, you need to look up matching customer names. If you look at the relationship diagram (part of it is shown in Figure 4.1), you find that the CustomerID and matching names are stored in the Customer table. You could take each of the ID values returned by the Sale query and create a new query on the Customer table

Figure 4.8



Field:	SaleID	SaleDate	CustomerID	PaymentMethod
Table:	Sale	Sale	Sale	Sale
Sort:				
Show:	✓	✓	✓	✓
Criteria:		Between #5/1/2006# And #5/31/2006#		"Cash"
or:				

SaleID	SaleDate	CustomerID	PaymentMe
1304	5/7/2006	1309	Cash
1356	5/2/2006	314	Cash
1376	5/10/2006	69	Cash
1495	5/13/2006	645	Cash
(New)		0	

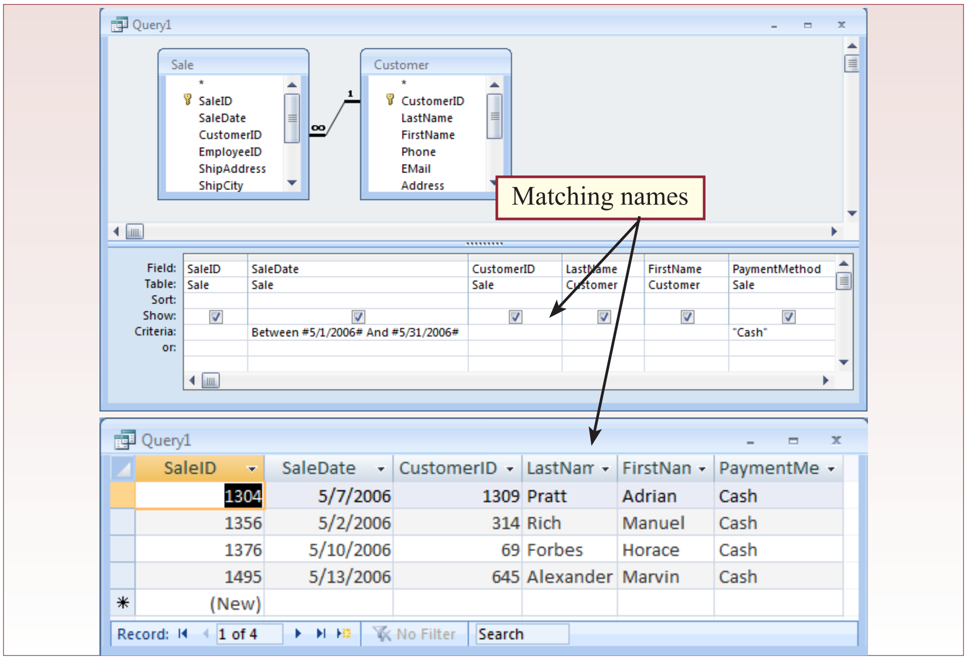
to find the names; however, the table JOIN command is much easier and more powerful.

With the Sale query in Design view, click the Show Table button on the menu. Then add the Customer table to the query. The Customer table is added to the top half of the query screen, and the join to the Sale table is shown with a connecting line between the CustomerID columns. This connection exists because the relationship was created when the tables were designed. Most of the time, these automatically created joins will be correct. But sometimes you will have to delete or add connections. You can remove a join by right-clicking on the line and selecting the delete option. You can create a new join by dragging a column name from one table and dropping it onto the matching column in the second table.

Figure 4.9 shows the basic query design. Once the tables are joined correctly, you can move any column down to the query grid. In this case, place the Customer LastName and FirstName columns in the grid. Run the query to see that the DBMS automatically looks up the names that match the ID values. If you want to double-check the lookup, you can add the CustomerID column from the Customer table and see that it matches the CustomerID values from the Sale table.

Figure 4.10 shows the SQL for the same query. The FROM and INNER JOIN statements specify the tables and how they are joined. If you read the FROM statement carefully, you can see that it provides the same information as the query grid join. It lists the two tables and specifies that they are connected by the CustomerID column. Because the column has the same name in both tables, you are required to specify the full name of the column in the ON statement. For two tables, the FROM and INNER JOIN syntax is relatively easy to follow. With more tables, the join begins to look a little messier. Consequently, it is often easiest to join tables using the visual connections in the query grid, and then switch to SQL view to ensure that the query is correct.

Figure 4.9



Field:	SaleID	SaleDate	CustomerID	LastName	FirstName	PaymentMethod
Table:	Sale	Sale	Sale	Customer	Customer	Sale
Sort:						
Show:	✓	✓	✓	✓	✓	✓
Criteria:		Between #5/1/2006# And #5/31/2006#				"Cash"
or:						

SaleID	SaleDate	CustomerID	LastNam	FirstNan	PaymentMe
1304	5/7/2006	1309	Pratt	Adrian	Cash
1356	5/2/2006	314	Rich	Manuel	Cash
1376	5/10/2006	69	Forbes	Horace	Cash
1495	5/13/2006	645	Alexander	Marvin	Cash
(New)					

```
SELECT Sale.SaleID, Sale.SaleDate, Sale.CustomerID, Customer.LastName,
Customer.FirstName, Sale.PaymentMethod

FROM Customer INNER JOIN Sale ON Customer.CustomerID = Sale.
CustomerID

WHERE (Sale.SaleDate Between #5/1/2006# And #5/31/2006# AND Sale.
PaymentMethod="Cash");
```

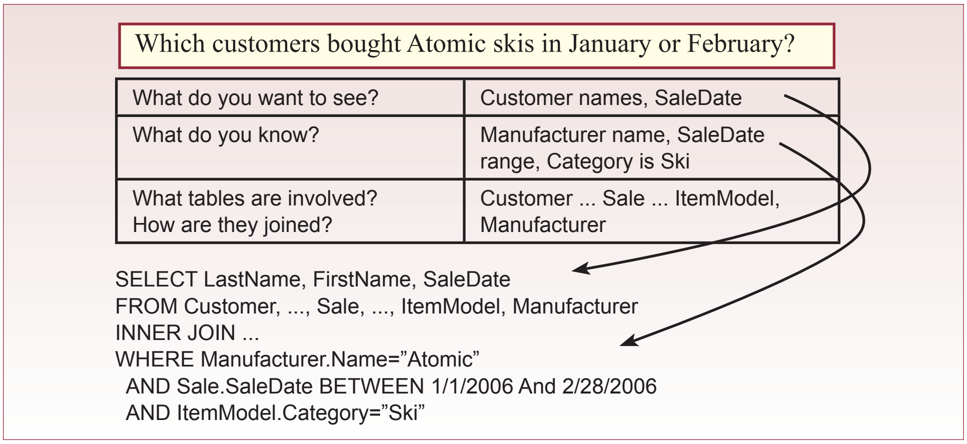
Figure 4.10

To see the power of the SQL joins, consider a slightly more challenging business question: Which customers bought Atomic skis in January or February? Note that Atomic is the name of a ski manufacturer. Before leaping into the Access query screen, it is best to think about the query and look at the relationship screen for a minute. As shown in Figure 4.11, you can begin with what you want to see: the names of the customers. These are in the Customer table. Now, what facts do you know? In this case, you are given the name of the manufacturer, the ItemModel.Category, and the range for the SaleDate. You should also begin writing down the tables you need to provide these facts: Customer, Sale, ItemModel, and Manufacturer so far. When you examine the relationships for the database, you will see that these four tables are not enough—they do not connect together. You will also need the SaleItem and Inventory tables.

Figure 4.12 shows the final query in Design view. Notice the large number of tables involved. But you need to verify that each connection is correct for the specific problem. Once the tables have been selected and joined, you can quickly place the columns you need on the query grid, and then enter the desired conditions. Running the query reveals the two people who meet the desired conditions.

Figure 4.13 shows the matching SQL for the same query. The INNER JOIN and ON statements seem complex, but you do not need to worry about the order of the statements, so it is more flexible than it first appears. The main thing is to make sure you list all of the tables and include the correct connections within the ON statements. Nonetheless, it is easier to join tables with the visual design, and let it enter the matching SQL statements. You can always switch to the SQL view to examine the statements or make minor changes.

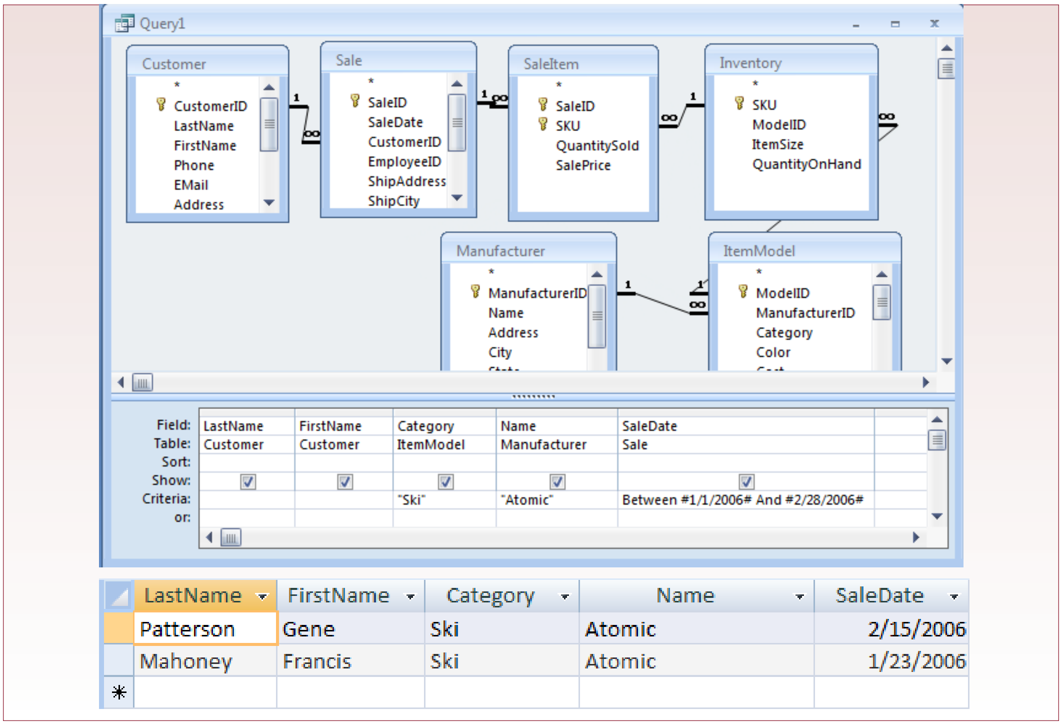
Figure 4.11



Which customers bought Atomic skis in January or February?

What do you want to see?	Customer names, SaleDate
What do you know?	Manufacturer name, SaleDate range, Category is Ski
What tables are involved? How are they joined?	Customer ... Sale ... ItemModel, Manufacturer

```
SELECT LastName, FirstName, SaleDate
FROM Customer, ..., Sale, ..., ItemModel, Manufacturer
INNER JOIN ...
WHERE Manufacturer.Name="Atomic"
  AND Sale.SaleDate BETWEEN 1/1/2006 And 2/28/2006
  AND ItemModel.Category="Ski"
```



Field:	LastName	FirstName	Category	Name	SaleDate
Table:	Customer	Customer	ItemModel	Manufacturer	Sale
Sort:					
Show:	✓	✓	✓	✓	✓
Criteria:			"Ski"	"Atomic"	Between #1/1/2006# And #2/28/2006#
or:					

LastName	FirstName	Category	Name	SaleDate
Patterson	Gene	Ski	Atomic	2/15/2006
Mahoney	Francis	Ski	Atomic	1/23/2006

Figure 4.12

```
SELECT Customer.LastName, Customer.FirstName, ItemModel.Category,
Manufacturer.Name, Sale.SaleDate

FROM Manufacturer INNER JOIN (ItemModel INNER JOIN (Inventory INNER
JOIN (SaleItem INNER JOIN (Sale INNER JOIN Customer ON Customer.
CustomerID = Sale.CustomerID) ON Sale.SaleID = SaleItem.SaleID) ON
Inventory.SKU = SaleItem.SKU) ON ItemModel.ModelID = Inventory.ModelID)
ON Manufacturer.ManufacturerID = ItemModel.ManufacturerID

WHERE (((ItemModel.Category)=”Ski”) AND ((Manufacturer.Name)=”Atomic”)
AND ((Sale.SaleDate) Between #1/1/2004# And #2/29/2004#));
```

Figure 4.13

Computations and Subtotals

Activity: Compute Values with Queries

In general, it does not make sense to store some columns in the database. In particular, the DBMS query system has the ability to perform common calculations. Figure 4.14 shows how the query system can easily calculate the profit margin for each item. In this case, the table holds the item’s list price and the acquistion cost. The profit is simply the

Action

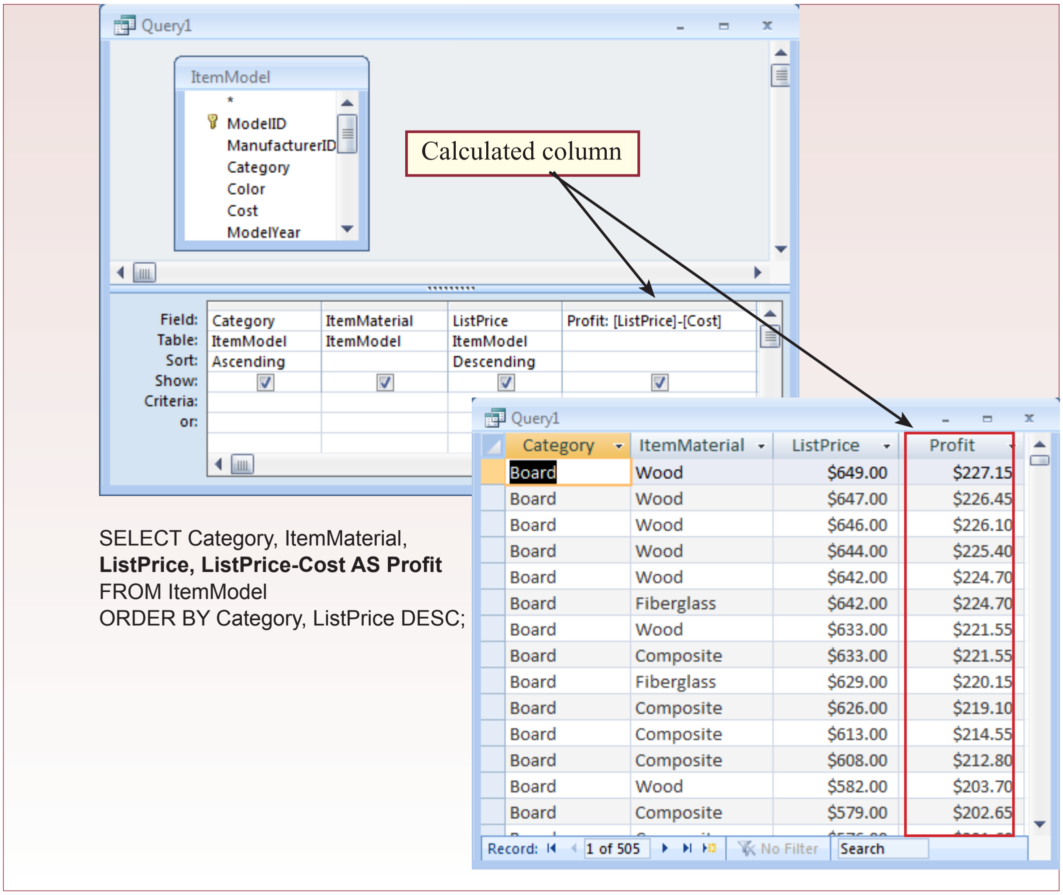
Create a new query in Design view.

Add the ItemModel table.

Select columns: Category, ItemMaterial, and ListPrice.

Create new column as Profit: [ListPrice]-[Cost]

Run the query.



Field:	Category	ItemMaterial	ListPrice	Profit: [ListPrice]-[Cost]
Table:	ItemModel	ItemModel	ItemModel	
Sort:	Ascending		Descending	
Show:	✓	✓	✓	✓
Criteria:				
or:				

SELECT Category, ItemMaterial,
ListPrice, ListPrice-Cost AS Profit
FROM ItemModel
ORDER BY Category, ListPrice DESC;

Category	ItemMaterial	ListPrice	Profit
Board	Wood	$649.00	$227.15
Board	Wood	$647.00	$226.45
Board	Wood	$646.00	$226.10
Board	Wood	$644.00	$225.40
Board	Wood	$642.00	$224.70
Board	Fiberglass	$642.00	$224.70
Board	Wood	$633.00	$221.55
Board	Composite	$633.00	$221.55
Board	Fiberglass	$629.00	$220.15
Board	Composite	$626.00	$219.10
Board	Composite	$613.00	$214.55
Board	Composite	$608.00	$212.80
Board	Wood	$582.00	$203.70
Board	Composite	$579.00	$202.65

Figure 4.14

difference between the list price and the cost. In the grid, you enter the name of the new column (Profit), followed by a colon (:) and the calculation (ListPrice - Cost). Observe that Access will place square brackets around the column names. It does this to be cautious. Anytime a column name is a reserved word or contains special characters (such as a space or the # sign), you must place the brackets around the column name. Notice that the query is sorted by Category and List-Price. You make these selections on the Sort row. The SQL is also straightforward. To add the computed column to the display, enter the calculation in the SELECT line, followed by AS and the name of the new column.

Calculations written in this form are always performed on data on the same row. It does not calculate across rows. You can use the standard mathematical operators (add, subtract, divide, and multiply). You can also use several standard functions built into Access. Figure 4.15 shows some of the commonly used functions. Most are straightforward, but the date functions require a little explanation and

Action

Create a new query in Design view.

Add the Sale table.

Select columns: SaleID and SaleDate.

Create new column as LateDate: [SaleDate] + 30.

Create a nother new column as LateMonth: DateAdd(“m”,1,[SaleDate]).

Run the query.

Lcase	To lower case
Len	Length/number of characters
Mid	Get substring
Trim	Remove leading and trailing spaces
Ucase	To upper case
Date	Current date
DateAdd	Add days, months, years to a date
DateDiff	Subtract two dates
Format	Highly detailed formatting
Now	Current date and time
Abs	Absolute value
Cos	Cosine, all common trig functions
Int	Integer, drop decimal values
Sgn	Signum
Round	Round-off

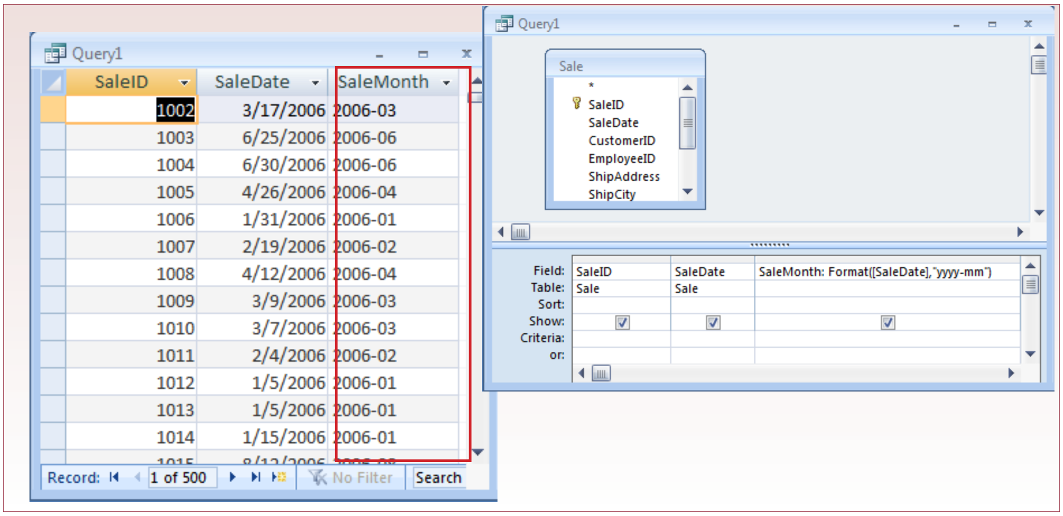
Figure 4.15

practice. The Format function enables you to specify detailed formats for date and numeric columns.

To illustrate the power of some of the date functions, create a new query using the Sale table and display the SaleID and SaleDate columns. Now, as shown in Figure 4.16, add a new column as SaleMonth: Format(SaleDate,"yyyy-mm"). Be sure to enter the quoted format correctly—it controls the way the date will be displayed. In this case, it will display the four-digit year, followed by a two-digit number for the month. You may want to format months this way to ensure that they sort correctly. The Format function has many options, and you will have to consult the Access Help documentation for details.

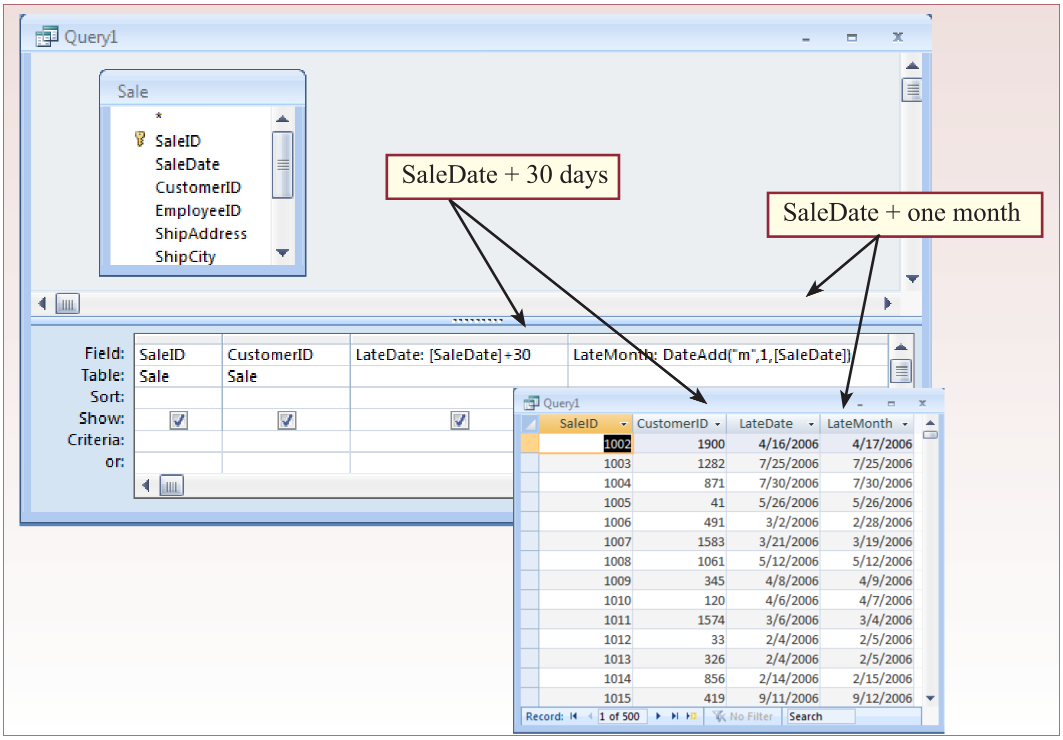
The DateAdd and DateDiff functions are even more versatile. Create a new query, again using the Sale table. Display the SaleID and the SaleDate. Then, as shown in Figure 4.17, add the column LateDate: SaleDate + 30. This calculation generates the new date that is 30 days in the future. Now, add the column Late-Month: DateAdd("m", 1, SaleDate). Run the query to compare the two calculations. The first one adds 30 days to the SaleDate. The second adds one month to

Figure 4.16



SaleID	SaleDate	SaleMonth
1002	3/17/2006	2006-03
1003	6/25/2006	2006-06
1004	6/30/2006	2006-06
1005	4/26/2006	2006-04
1006	1/31/2006	2006-01
1007	2/19/2006	2006-02
1008	4/12/2006	2006-04
1009	3/9/2006	2006-03
1010	3/7/2006	2006-03
1011	2/4/2006	2006-02
1012	1/5/2006	2006-01
1013	1/5/2006	2006-01
1014	1/15/2006	2006-01

Field:	SaleID	SaleDate	SaleMonth: Format([SaleDate],"yyyy-mm")
Table:	Sale	Sale	
Sort:			
Show:	✓	✓	✓
Criteria:			
or:			



Field:	SaleID	CustomerID	LateDate: [SaleDate]+30	LateMonth: DateAdd("m",1,[SaleDate])
Table:	Sale	Sale		
Sort:				
Show:	✓	✓	✓	
Criteria:				
or:				

SaleID	CustomerID	LateDate	LateMonth
1002	1900	4/16/2006	4/17/2006
1003	1282	7/25/2006	7/25/2006
1004	871	7/30/2006	7/30/2006
1005	41	5/26/2006	5/26/2006
1006	491	3/2/2006	2/28/2006
1007	1583	3/21/2006	3/19/2006
1008	1061	5/12/2006	5/12/2006
1009	345	4/8/2006	4/9/2006
1010	120	4/6/2006	4/7/2006
1011	1574	3/6/2006	3/4/2006
1012	33	2/4/2006	2/5/2006
1013	326	2/4/2006	2/5/2006
1014	856	2/14/2006	2/15/2006
1015	419	9/11/2006	9/12/2006

Figure 4.17

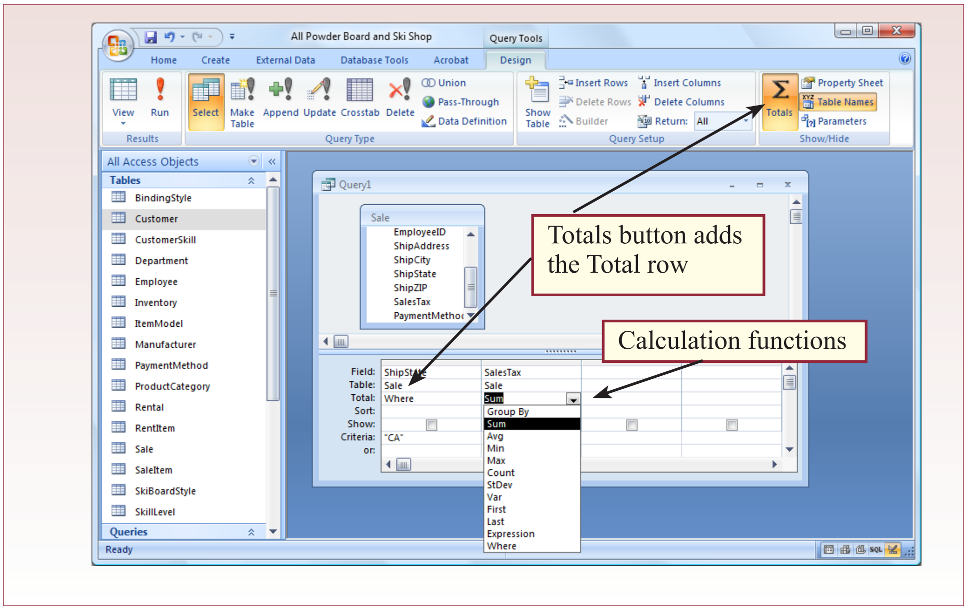
the SaleDate. Of course, you could specify any number of days or months and the system would correctly compute the date across month or year boundaries. For many business questions, you will only need the standard SQL date arithmetic. But occasionally, you will find it useful to apply the power of DateAdd or DateDiff. DateDiff is similar to DateAdd, but it subtracts two dates. In both the addition and subtraction, you can specify how you want the date arithmetic performed—by dates ("d"), months ("m"), years ("yyyy"), or other options. Check the Access documentation to see how to count the number of workdays or even the number of Fridays between two dates. Note: You might have to open the Visual Basic Editor before the Help system can find the documentation for some of these specialized functions. To open the VB Editor, open a form in Design view and then select View/Code.

Activity: Calculate Totals and Subtotals

Business managers often need to compute totals across rows of data. SQL provides several aggregation functions to perform these tasks. The most commonly used functions are Sum, Average, and Count. Of the three, the Count function can be the most confusing to use. Just remember that it simply counts the number of rows, while Sum adds up the num-

Action

Create a new query in Design view.
Add the Sale table.
Select columns: ShipState and SalesTax.
View/Totals (or Totals button).
Select "Where" for Total row in State.
Enter "CA" as a criteria.
Select "Sum" for SalesTax Total row.
Run the query.



Field:	ShipState	SalesTax
Table:	Sale	Sale
Total:	Where	Sum
Sort:		
Show:		
Criteria:	"CA"	
or:		

Figure 4.18

bers within a row. The challenge is to identify when you need to use Count instead of Sum.

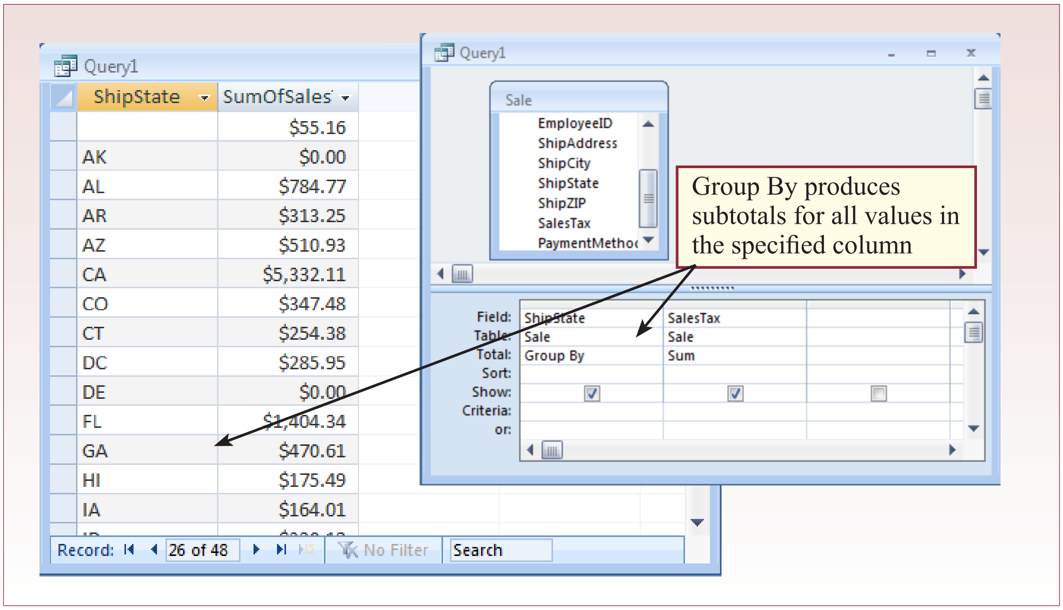
The Sum function is straightforward. For example, how much sales tax does the company owe to the State of California? Begin by creating a new query based on the Sale table, because it has the ShipState and SalesTax columns. As a criterion for ShipState, enter the CA abbreviation for California. Run the query, and you should see two columns: each row will have CA in the state and a value for the SalesTax. To compute the total, return to Design view. You now need to indicate that you want to compute totals by clicking the Totals button on the menu bar. As shown in Figure 4.18, this button adds the Total row to the grid. Select the Sum option under the SalesTax column and the Where option for the ShipState column. The Sum option makes sense since you want the total of the sales tax. When you run the query, you will see a single value—the total amount of sales tax collected for the state of California.

Figure 4.19 shows the SQL syntax for the query. Note that setting the Where option for the SaleState was not strictly necessary, but it makes the SQL clearer. Note the use of the column alias SumOfSalesTax to provide a name for the output column. You can set this name to almost anything, but it should indicate the business interpretation of the column. Of course, you can use multiple tables with the FROM and INNER JOIN syntax.

To understand some of the power of SQL, what if you want to see the total tax owed to each state? Of course, it would be possible to edit the CA condition and

Figure 4.19

```
SELECT Sum(Sale.SalesTax) AS SumOfSalesTax
FROM Sale
WHERE Sale.ShipState="CA"
```



ShipState	SumOfSalesTax
	$55.16
AK	$0.00
AL	$784.77
AR	$313.25
AZ	$510.93
CA	$5,332.11
CO	$347.48
CT	$254.38
DC	$285.95
DE	$0.00
FL	$1,404.34
GA	$470.61
HI	$175.49
IA	$164.01

Field:	ShipState	SalesTax	
Table:	Sale	Sale	
Total:	Group By	Sum	
Sort:			
Show:	✓	✓	☐
Criteria:			
or:			

Figure 4.20

replace it with each state, but there is an easier way. Close this query and begin a new query the same way. Add the Sales table and use the ShipState and SalesTax columns, but do not specify any limiting conditions. Click the Totals button to see the totals row. Select the Sum option under the SalesTax column, but do not change the setting for the ShipState. The default option for the Totals row is Group By. This choice instructs the DBMS to identify each unique entry in that column and compute the selected subtotal for each element. Figure 4.20 shows the query and the result. The result lists each state, followed by the total sales tax collected for that state.

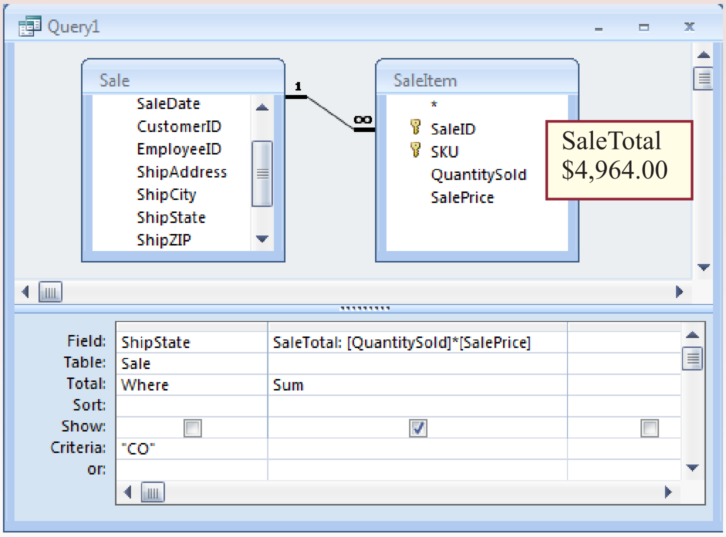
Action
Create a new query in Design view.
Add the Sale table.
Select columns: ShipState and SalesTax.
View/Totals (or Totals button).
Select "Sum" for SalesTax Total row.
Run the query.

Figure 4.21 shows the corresponding SQL statement. The main addition is the actual GROUP BY clause. Note that any column listed in the GROUP BY clause must be included in the SELECT clause. It would not make sense to compute subtotals and then not display the values you are grouping. Of course, you could compute the average or count the number of items in a group just as easily. In fact, you can compute multiple functions at the same time, just by including multiple copies of the desired column and selecting a different aggregation function.

For practice, you should compute the total value of sales to customers in Colorado (the state code is CO). Create a new query and add the Sale and SaleItem

Figure 4.21

```
SELECT Sale.ShipState, Sum(Sale.SalesTax) AS SumOfSalesTax
FROM Sale
GROUP BY Sale.ShipState;
```



Field:	ShipState	SaleTotal: [QuantitySold]*[SalePrice]	
Table:	Sale		
Total:	Where	Sum	
Sort:			
Show:	☐	☑	☐
Criteria:	"CO"		
or:			

```
SELECT Sum([QuantitySold]*[SalePrice]) AS SaleTotal
FROM Sale INNERJOIN SaleItem ON Sale.SaleID = SaleItem.SaleID
WHERE Sale.ShipState="CO";
```

Figure 4.22

tables. Use the ShipState column from the Sale table. To compute the total value of the actual sale is slightly trickier. You need to multiply the QuantitySold by the SalePrice from the SaleItem table, then compute its sum. To be safe, first do the multiplication and check your progress. In a new field, enter the name SaleTotal, followed by a colon, and the formula: QuantitySold * SalePrice. Enter CO as the criteria to select the state, then run the query and check the results to see if they make sense. You might want to list the QuantitySold and SalePrice separately, and then use a calculator or spreadsheet to verify some of the calculations. Returning to Design view, you need to compute the total. Click the Totals button and select Sum for the SaleTotal column, and Where for the ShipState column. Technically, the query will work if you leave the ShipState as Group By, but the SQL is clearer and potentially faster if you use Where. Figure 4.22 shows the query.

Exercises

Crystal Tigers

Enter sample data for the Crystal Tigers service club database. You can make up data, but remember that it has to be consistent. You might want to share data with other students so that everyone has a larger database to work with. Then create queries to provide the following business information.

1. List all of the members who have been president of the organization.
2. List the charities for which the club has raised more than $1,000.
3. Pick an event and list all of the members who worked at that event.
4. Count the number of events and the amount of money raised for each charity.
5. List the total number of service hours provided in the latest year.

6. List the number of service hours provided by each member.
7. List the members who have held the most number of officer positions.

Capitol Artists

Enter sample data for the Capitol Artists business. You can create random data, but remember that it has to be consistent. You might want to share data with other students so that everyone has a larger database to work with. Then create queries to provide the following business information.

1. Pick a date and an employee and list all of the tasks by that person on that date.
2. List all of the tasks performed for a specific job (e.g., Job #1173).
3. List all of the client jobs that had active tasks on a specific date.
4. Count the number of meetings held regarding one client (pick any client).
5. List the employees who have attended the most number of meetings.
6. Pick a job and compute the amount of money billed (hours * rate).
7. List the clients in order of the ones that have provided the greatest revenue (billing + expenses).

Offshore Speed

Enter sample data for the Offshore Speed company. You can create random data, but remember that it has to be consistent. You might want to share data with other students so that everyone has a larger database to work with. Then create queries to provide the following business information. If you have not created data that matches these questions, either add more data, or change the query to match your data. For instance, if you do not have any sales of propellers, pick a category of item that you have sold several times.

1. Pick a month and list all of the customers who purchased propellers (Category).
2. List all of the parts sold on a particular day.
3. What is the most expensive steering wheel we have sold?
4. List the manufacturers sorted by the number of parts we sell from each one.
5. List the employees to identify the best salespeople in terms of value.
6. List the brands of boat for which we sell the most oil pumps (Description).
7. For a given order, compute the total value of the order and the sales tax, assuming a 6 percent tax rate.

Final Project

The main textbook has an online appendix with several longer case studies. You should be able to work on one of these cases throughout the term. If you or your instructor picks one, perform the following tasks.

1. Create a few rows of sample data for all of the tables.
2. Identify at least five business questions that a manager would commonly ask and provide the queries to answer those questions. At least two of the questions should involve subtotals or averages.
3. Exchange three business questions with other students in your class and write the queries for the questions you receive.

Chapter 5

Advanced Queries

Chapter Outline

Objectives

- Create more complex SELECT queries using subqueries.
- Understand the role of INNER and LEFT joins.
- Create theta joins using inequalities to match categories.
- Use a UNION statement to merge rows of data.
- Use DDL to CREATE and DROP tables.
- Use DML to INSERT, UPDATE, and DELETE data.

Advanced Database Queries

SQL is a powerful language. For many queries, you will not need the full power of SQL, but some seemingly innocent business questions can be tricky to answer. In these cases, you need some additional capabilities. Some of these capabilities can be challenging to understand, but if you follow the examples carefully, you should be able to use the ideas to create similar queries in the future.

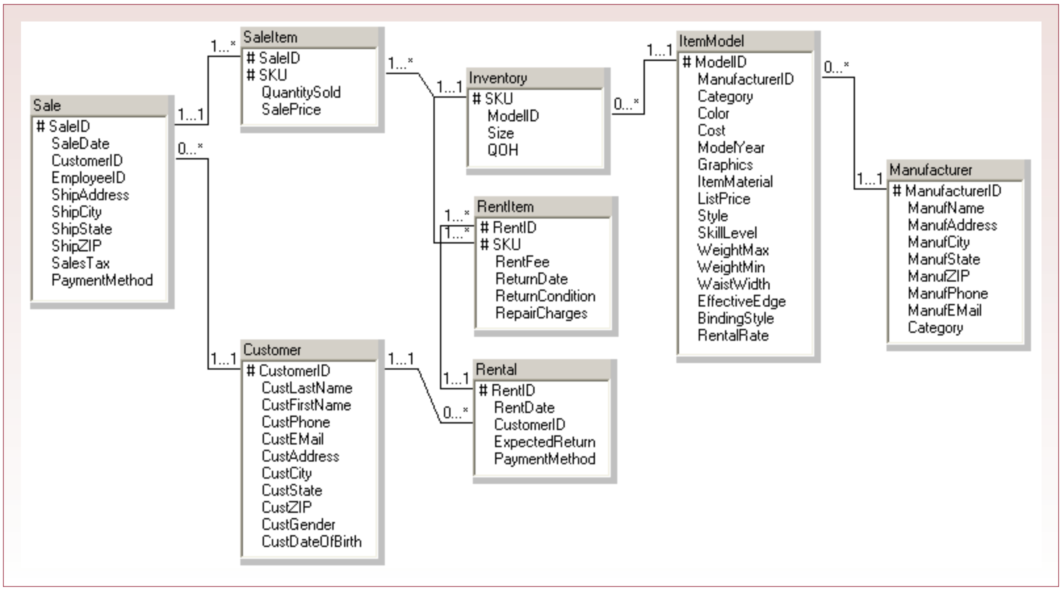
Subqueries are one of the more interesting features of SQL. A subquery is a query that calls a second query to obtain additional data. Instead of looking up a second set of numbers yourself, you can add a second query to do the work automatically.

Joins offer other powerful options. Joins are commonly used as a lookup link between tables, making it easy for you to build a query that uses data from multiple tables. However, joins have several options to help you answer even more complex questions. It is especially important that you understand the difference between inner and outer joins.

One of the strengths of SQL is that it operates on sets of data. Instead of thinking in terms of individual rows, you can concentrate on collections of rows that meet specified conditions. SQL offers some interesting set-operation commands that provide detailed control over rows of data. For example, the UNION statement combines rows of data from multiple SELECT statements.

Chapters 2 and 3 hinted at the ability of SQL to define tables. In fact, you can use a DBMS without the fancy visual screens. Generally, everything you need to do can be handled with SQL commands. For example, the data definition command CREATE TABLE is a powerful method to create new tables. Storing these SQL statements in a text file makes it easy to quickly rebuild or transfer a database. Similarly, the data manipulation commands (INSERT, DELETE, and UPDATE) are powerful tools for copying and deleting rows, and quickly changing thousands or millions of pieces of data.

Figure 5.1



Case: All Powder Board and Ski Shop

As the queries become more complex, it is better to work from a common set of data. Figure 5.1 shows the primary tables for the All Powder Board and Ski Shop. Your tables and sample data should be very close to these tables. Note that several supporting tables are not displayed in this diagram, but you will also need those in your database. As explained in Chapter 4, you can import the sample data to these tables. If you add more data, your query results may be slightly different from the ones shown in this chapter. Although the query is more important than the actual results, the results are useful to help you decide if you have constructed the query properly.

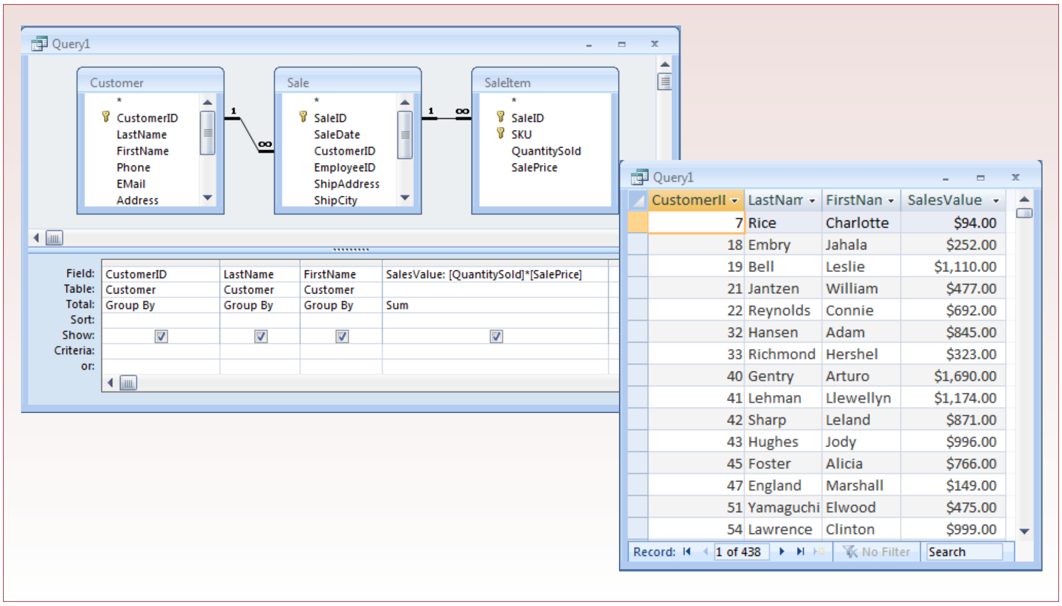
One of the greatest challenges with any database query is that most queries return values, but they might not be answers to the question you thought you were asking. You must learn to build the queries and test each intermediate step carefully so that you can be sure the final result is an accurate answer to the question being asked.

Lab Exercise

All Powder Board and Ski Data

Subqueries are used to create a second (or more) query to look up additional data that can be used in the primary query. The value is often used within a WHERE clause to make comparisons in more depth. For example, Katy the manager wants to identify the best customers of the shop. In particular, she would like to know which customers have placed the most sales. You could just give her the complete list of customers and the sales made by each person. Eventually, however, this list would be too long. Instead, she wants a list that displays the customers whose total purchases are larger than the average purchase per customer. Although the business question is reasonable, this question is slightly tricky because you have to build the query in pieces.

Figure 5.2



Field:	CustomerID	LastName	FirstName	SalesValue: [QuantitySold]*[SalePrice]
Table:	Customer	Customer	Customer	
Total:	Group By	Group By	Group By	Sum
Sort:				
Show:	✓	✓	✓	✓
Criteria:				
or:				

CustomerII	LastNam	FirstNan	SalesValue
7	Rice	Charlotte	$94.00
18	Embry	Jahala	$252.00
19	Bell	Leslie	$1,110.00
21	Jantzen	William	$477.00
22	Reynolds	Connie	$692.00
32	Hansen	Adam	$845.00
33	Richmond	Hershel	$323.00
40	Gentry	Arturo	$1,690.00
41	Lehman	Llewellyn	$1,174.00
42	Sharp	Leland	$871.00
43	Hughes	Jody	$996.00
45	Foster	Alicia	$766.00
47	England	Marshall	$149.00
51	Yamaguchi	Elwood	$475.00
54	Lawrence	Clinton	$999.00

Activity: Create a Subquery

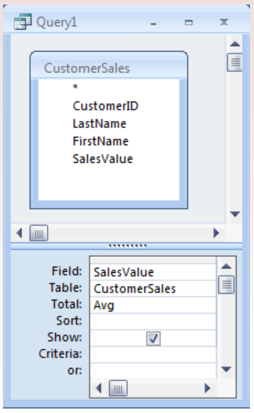
> **Action**
>
> Create a new query in Design view.
>
> Tables: Customer, Sale, SaleItem
>
> Columns: CustomerID, LastName, FirstName, SalesValue: [QuantitySold] * [SalePrice].
>
> Sum the SalesValue, Group By the rest.
>
> Save query as CustomerSales.
>
> Create new query in Design view.
>
> Table: CustomerSales query.
>
> Columns: SalesValue.
>
> Set Totals and select Avg.
>
> Run the query.

The first step in the query is to recognize that you need to compute total sales by customer. The phrase "by customer" is an indication that you need to compute subtotals using the GROUP BY clause. Figure 5.2 shows the initial query that computes these subtotals. Of course, it lists the sales for every customer, and Katy wants only the ones with greater-than-average sales. But this query is an important step and needs to be saved as "CustomerSales".

The next step is to use this first query to compute the average amount of sales for customers. This computation is straightforward. Simply build a new query using CustomerSales as the only table, and calculate the average of the sales column. Figure 5.3 shows the basic query and the result based on the current data. Notice that the SQL is straightforward. In this case, the SQL is critical for the next step. It is not necessary to save this query, but you might want to leave the SQL window open for the final step.

The last step is to create a new query that answers the overall question to determine which customers spend more than average. The new query will also be based on the CustomerSales query created in the first step, so just add that query. This time, select the LastName, FirstName, and SalesValue columns. If you ran the query at this point, you would get the same results as in the first query. Instead, you want to add a criterion to only display the customers with a SalesValue greater than the average. The simple approach is to enter the value 942.11 as a condition in the query. Although this approach works this time, it does not work very well over time. It would require the owner to run the average query first, then copy the value into the Design view of the main query. It makes more sense to automate the entire process. So instead of entering the actual number as the condition, you need to enter the subquery calculation. You can write the complete SQL state-

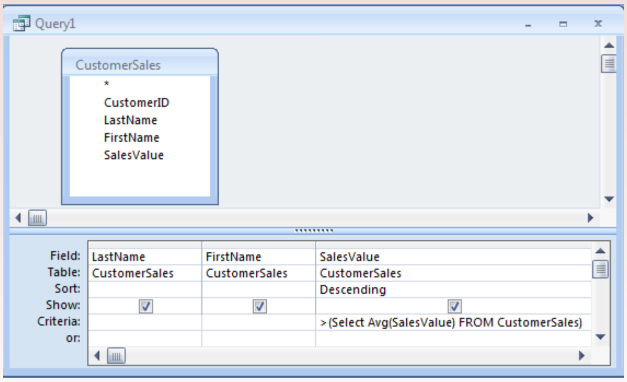
Figure 5.3



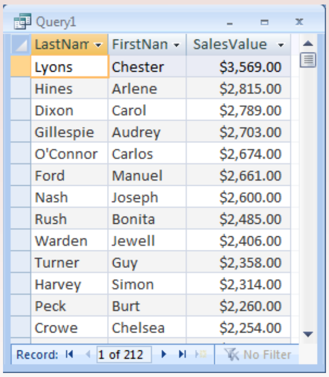
Field:	SalesValue
Table:	CustomerSales
Total:	Avg
Sort:	
Show:	✓
Criteria:	
or:	

```
SELECT Avg(CustomerSales.SalesValue)
AS AvgOfSalesValue
FROM CustomerSales;
```

$942.11



Field:	LastName	FirstName	SalesValue
Table:	CustomerSales	CustomerSales	CustomerSales
Sort:			Descending
Show:	✓	✓	✓
Criteria:			>(Select Avg(SalesValue) FROM CustomerSales)
or:			



LastName	FirstName	SalesValue
Lyons	Chester	$3,569.00
Hines	Arlene	$2,815.00
Dixon	Carol	$2,789.00
Gillespie	Audrey	$2,703.00
O'Connor	Carlos	$2,674.00
Ford	Manuel	$2,661.00
Nash	Joseph	$2,600.00
Rush	Bonita	$2,485.00
Warden	Jewell	$2,406.00
Turner	Guy	$2,358.00
Harvey	Simon	$2,314.00
Peck	Burt	$2,260.00
Crowe	Chelsea	$2,254.00

```
SELECT LastName, FirstName, SalesValue
FROM CustomerSales
WHERE SalesValue > (SELECT Avg(SalesValue) FROM
CusmterSales)
ORDER BY SalesValue DESC;
```

Figure 5.4

ment, but it must be contained within parentheses. Figure 5.4 shows the final query that you can give to Katy. Notice that it is sorted in descending order by SalesValue so that the customers with the largest total purchases are listed at the top. Also, always remember to put the subquery inside parentheses; otherwise the query will not run at all. If you want to save some typing and reduce errors, you should create the subquery first in a separate query to test it. When it is correct, you can copy the SQL statement and paste it into the WHERE clause for the final query. Again, remember to add the parentheses around the subquery.

Action

Create a new query in Design view.

Table: CustomerSales query.

Columns: LastName, FirstName, SalesValue.

Criteria for SalesValue (enter in SQL)

>(SELECT Avg(SalesValue) FROM CustomerSales).

Activity: Build Outer Joins

Joining tables is one of the more complex issues in SQL. Up to this point, the joins have been simple equality joins designed to show how a column in one table links to data stored in a related table. It is important that you understand the effect of this join. Jim the sales manager and David the rental manager want to know if customers who rent equipment also purchase items for sale. As with many questions, there are several different ways to build this query. Figure 5.5 shows the effect of an inner join. Build a new query and add the Rental and Sale tables. Join these tables through CustomerID by dragging and dropping the CustomerID from one table onto the column in the other table. When you display both CustomerID values in the query and run it, you can see that they are the

Action

Create a new query in Design view.

Tables: Rental and Sale.

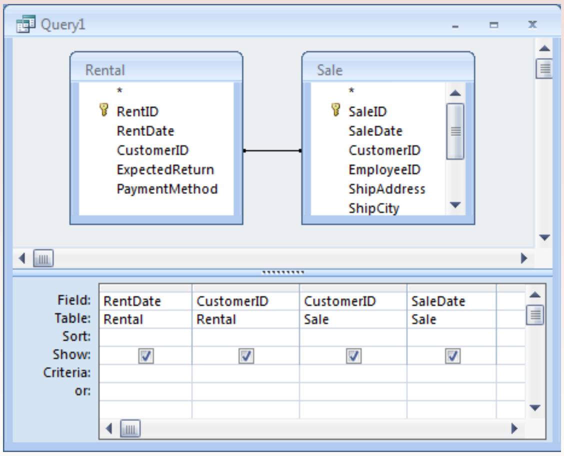
Columns: RentDate, SaleDate, and CustomerID from both tables.

Join the tables on CustomerID.

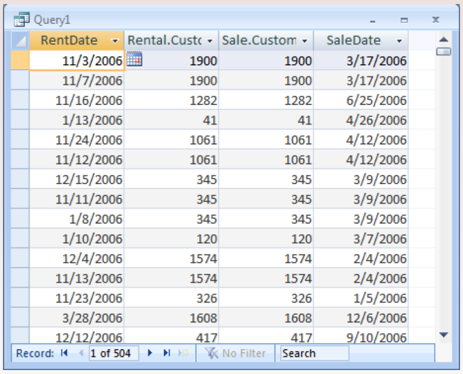
Run the query.

Join the tables on RentDate=SaleDate.

Run the query.



Field:	RentDate	CustomerID	CustomerID	SaleDate
Table:	Rental	Rental	Sale	Sale
Sort:				
Show:	✓	✓	✓	✓
Criteria:				
or:				



RentDate	Rental.Custc	Sale.Custom	SaleDate
11/3/2006	1900	1900	3/17/2006
11/7/2006	1900	1900	3/17/2006
11/16/2006	1282	1282	6/25/2006
1/13/2006	41	41	4/26/2006
11/24/2006	1061	1061	4/12/2006
11/12/2006	1061	1061	4/12/2006
12/15/2006	345	345	3/9/2006
11/11/2006	345	345	3/9/2006
1/8/2006	345	345	3/9/2006
1/10/2006	120	120	3/7/2006
12/4/2006	1574	1574	2/4/2006
11/13/2006	1574	1574	2/4/2006
11/23/2006	326	326	1/5/2006
3/28/2006	1608	1608	12/6/2006
12/12/2006	417	417	9/10/2006

SELECT Rental.RentDate, Rental.CustomerID, Sale.CustomreID, Sale.SaleDate
FROM Rental INNER JOIN Sale ON Rental.CustomerID=Sale.CustomerID

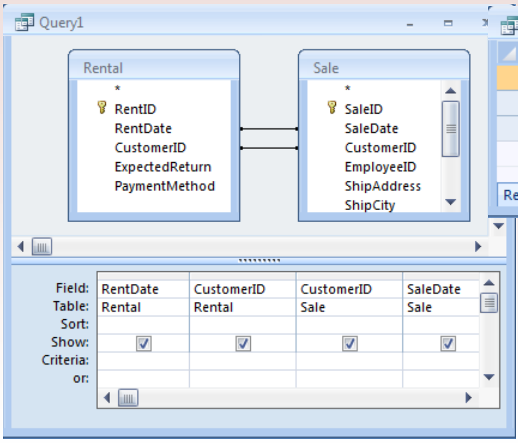
Figure 5.5

same. The effect of this join is that the results show the customers (ID only) who participated in a sale and a rental—at any time.

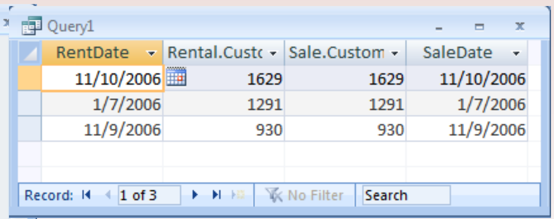
If you want to know which customers made a purchase and rental on the same day, you could add a condition that RentDate equals SaleDate. Or you could add a second join that connects RentDate and SaleDate. Figure 5.6 shows the query with the second join condition. Notice the use of the AND in the join statement. This query demonstrates the effect of the inner join. In many respects, it is equivalent to a WHERE clause. The inner join restricts the rows that you will see by forcing values to be equal.

On the other hand, perhaps Jim would like to see a list of all of the customers who participated in sales, and then check to see which of those have rented items. You need to build a new query. This time include the Customer table so their names can be displayed. Then add the Sale and Rental tables. Delete the join from Customer to Rental. That join would force all of the CustomerIDs to be equal which is not what Jim wants. Then connect Rental to Sale by CustomerID,

Figure 5.6



Field:	RentDate	CustomerID	CustomerID	SaleDate
Table:	Rental	Rental	Sale	Sale
Sort:				
Show:	✓	✓	✓	✓
Criteria:				
or:				



RentDate	Rental.Custc	Sale.Custom	SaleDate
11/10/2006	1629	1629	11/10/2006
1/7/2006	1291	1291	1/7/2006
11/9/2006	930	930	11/9/2006

```
SELECT RentDate, Rental.CustomerID,
Sale.CustomerID, SaleDate
FROM Rental
INNER JOIN Sale
  ON Rental.RentDate = Sale.SaleDate
  AND Rental.CustomerID=Sale.CustomerID
```

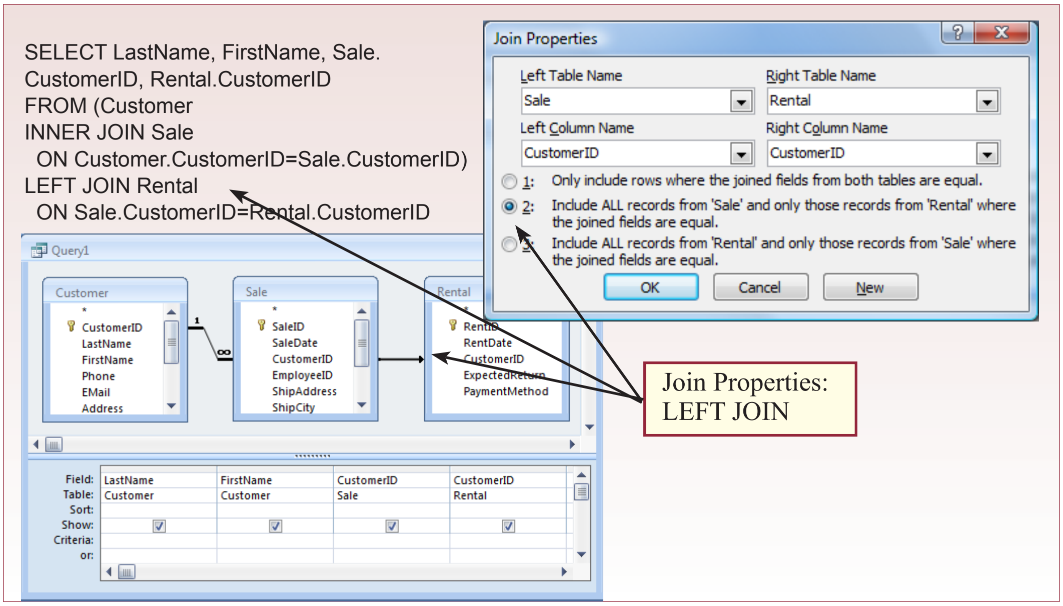
but this time, double-click the resulting line to modify the join properties. Figure 5.7 shows the basic query. Select the option to display all of the values from the Sale table and only the matching values from the Rental table. As shown in the SQL, this option sets up a LEFT JOIN, which displays all values in the Sale table (the left table in the SQL query list), even if the customer never rented items. If you have problems running the query, you might have to remove the Customer table from the query. Sometimes Access cannot figure out how to establish left joins when more than two tables are in the query. In these cases, you build the left join with only two tables, save the query, then create a second query based on the saved query and any other tables needed.

Action

Create a new query in Design view.
Tables: Customer, Sale, Rental.
Columns: LastName, FirstName, and CustomerID from Sale and Rental.
Delete join from Rental to Customer.
Add join from Sale to Rental.
Double-click this new join.
Select option to include all from Sale.
Run the query.

Figure 5.8 shows some of the results from running the query. Notice that several of the rows show missing values for the Rental.CustomerID. These are the customers who purchased items, but have never rented an item. If you want to see only this list, you can add the condition that Rental.CustomerID Is Null. Observe that the full list from the main query might not include all of the customers. To review your knowledge of joins, you should be able to identify the customers that might not be in this list. Looking at the design, notice that there is still an inner join between the Customer and Sale tables. Consequently, customers who have not participated in sales at all will not be displayed in this list. If you truly wanted a list of all customers, you would have to use a left join from the Customer to the Sale table. However, you will probably have to do one of the joins at a time, save the query, and then do the second join.

Figure 5.7



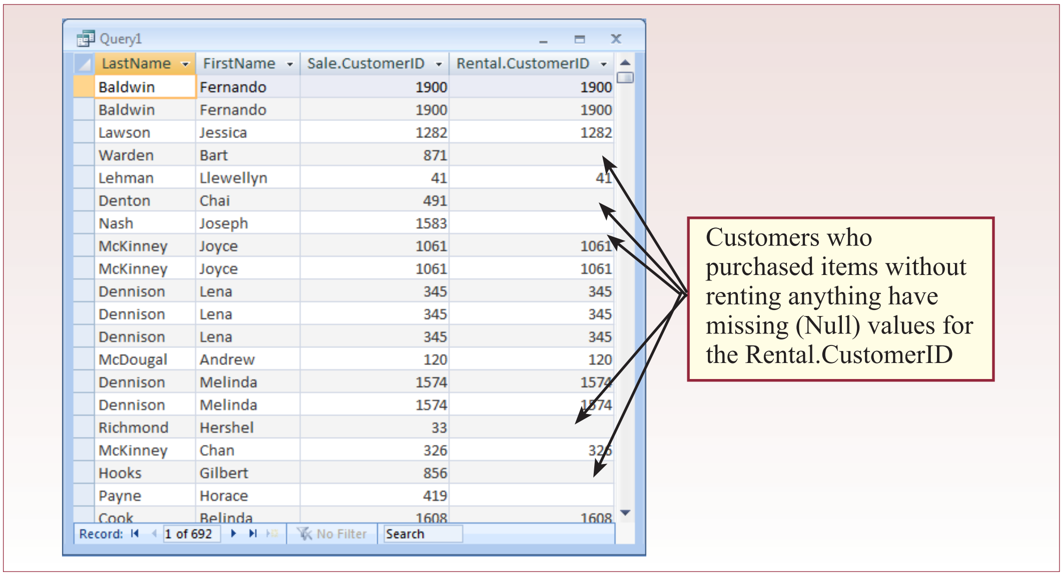
```
SELECT LastName, FirstName, Sale.
CustomerID, Rental.CustomerID
FROM (Customer
INNER JOIN Sale
  ON Customer.CustomerID=Sale.CustomerID)
LEFT JOIN Rental
  ON Sale.CustomerID=Rental.CustomerID
```

Join Properties

Left Table Name: Sale
Right Table Name: Rental
Left Column Name: CustomerID
Right Column Name: CustomerID

1: Only include rows where the joined fields from both tables are equal.
[✓ 2: Include ALL records from 'Sale' and only those records from 'Rental' where the joined fields are equal.]
3: Include ALL records from 'Rental' and only those records from 'Sale' where the joined fields are equal.

Field:	LastName	FirstName	CustomerID	CustomerID
Table:	Customer	Customer	Sale	Rental
Sort:				
Show:	✓	✓	✓	✓
Criteria:				
or:				



LastName	FirstName	Sale.CustomerID	Rental.CustomerID
Baldwin	Fernando	1900	1900
Baldwin	Fernando	1900	1900
Lawson	Jessica	1282	1282
Warden	Bart	871	
Lehman	Llewellyn	41	41
Denton	Chai	491	
Nash	Joseph	1583	
McKinney	Joyce	1061	1061
McKinney	Joyce	1061	1061
Dennison	Lena	345	345
Dennison	Lena	345	345
Dennison	Lena	345	345
McDougal	Andrew	120	120
Dennison	Melinda	1574	1574
Dennison	Melinda	1574	1574
Richmond	Hershel	33	
McKinney	Chan	326	326
Hooks	Gilbert	856	
Payne	Horace	419	
Cook	Belinda	1608	1608

Customers who purchased items without renting anything have missing (Null) values for the Rental.CustomerID

Figure 5.8

Recall the question of listing the customers who have purchased items but have not rented anything. With the left join, it is straightforward to get this list by adding the Is Null condition. But you must be very careful when creating this query. If you forget to specify the left join and stick with the standard inner join, the query will indicate that no customers match that condition. The

Action

Create a new query in Design view.

Tables: Customer and Sale.

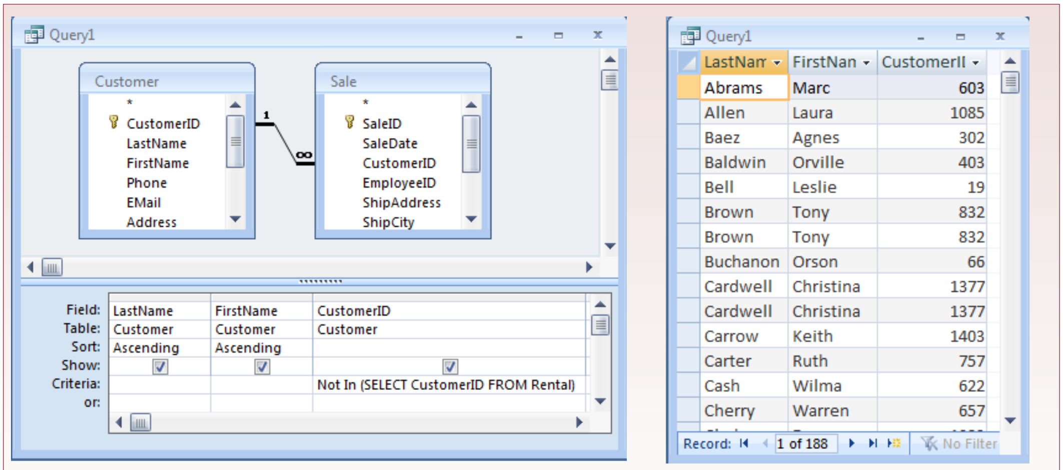
Columns: LastName, FirstName, and CustomerID.

Criteria for CustomerID (use SQL): Not In (SELECT CustomerID FROM Rental).

Check the SQL.

Run the query.

Figure 5.9



Field:	LastName	FirstName	CustomerID
Table:	Customer	Customer	Customer
Sort:	Ascending	Ascending	
Show:	✓	✓	✓
Criteria:			Not In (SELECT CustomerID FROM Rental)
or:			

LastNam	FirstNan	CustomerI
Abrams	Marc	603
Allen	Laura	1085
Baez	Agnes	302
Baldwin	Orville	403
Bell	Leslie	19
Brown	Tony	832
Brown	Tony	832
Buchanon	Orson	66
Cardwell	Christina	1377
Cardwell	Christina	1377
Carrow	Keith	1403
Carter	Ruth	757
Cash	Wilma	622
Cherry	Warren	657

```
SELECT LastName, FirstName, Customer.CustomerID
FROM Customer INNER JOIN Sale ON Customer.CustomerID=Sale.CustomerID
WHERE Customer.CustomerID Not In (SELECT CustomerID FROM Rental)
```

reason for this result is that an inner join automatically leaves out the customers you are searching for. This question can also be answered with a subquery. Figure 5.9 shows the subquery approach. Start a new query and add the Customer and Sale tables. Sort the columns by LastName and FirstName. Then add the condition CustomerID Not In (SELECT CustomerID FROM Rental). As always, remember to put the subquery in parentheses. This query will retrieve all Customers who have participated in sales but have not rented any items. You should compare the results from this version to the left join version to ensure that both queries return the same results. Most systems support either method to answer the question, but there can sometimes be performance differences between the two approaches. Access seems to be faster using left join, but you would have to time large sets of data to measure the difference.

Activity: Create Complex Joins

Jim the sales manager is concerned about excess inventory. He wants to be able to monitor the status of quantity on hand (QOH) for all inventory items. He is particularly concerned about identifying which models are selling quickly versus models that have large numbers of items sitting around. Remember that models are product lines from the manufacturers, while individual items are specific sizes within a model group. He wants the totals for the model. To see if there is a problem, construct a new query that totals the quantity on hand and sorts it in descending order by ModelID. Figure 5.10 shows the total QOH for the various models. Save the query as "ModelsOnHand."

Action

Create a new query in Design view.

Table: Inventory.

Columns: ModelID and QuantityOnHand.

Sum the QuantityOnHand and sort it in descending order.

Run the query.

Save it as ModelsOnHand.

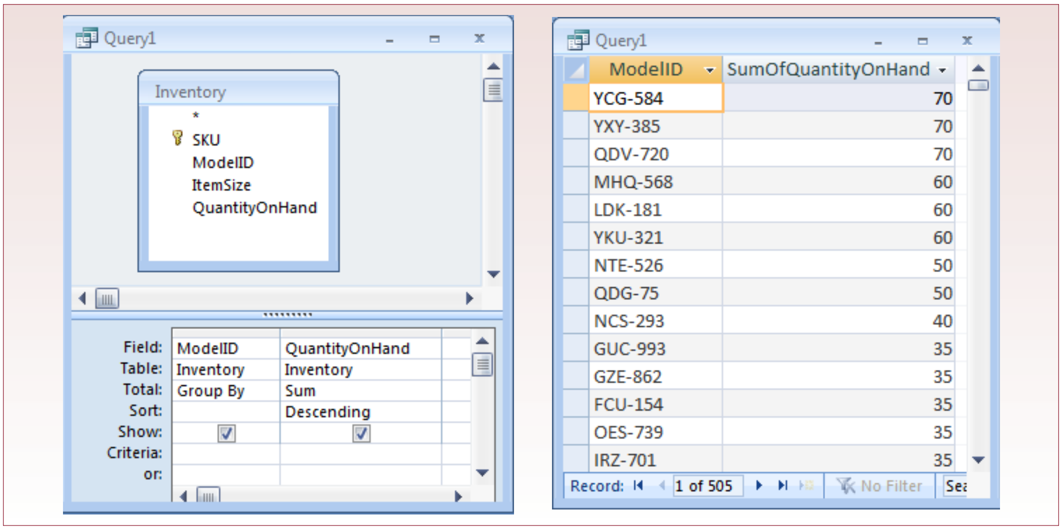
Create a new table in Design view.

Columns: CategoryID, CategoryName, LowLimit, HighLimit.

Save it as SalesCategory.

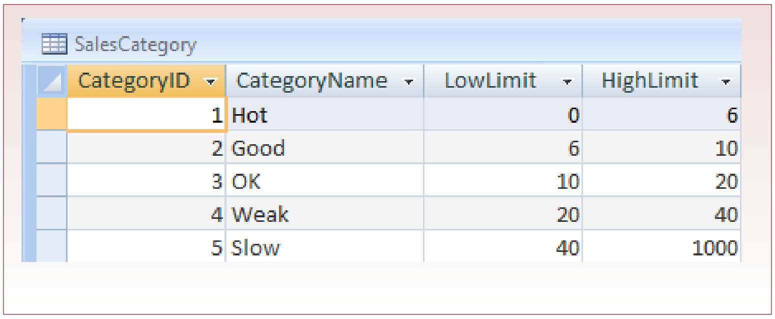
Enter data.

Figure 5.10



Field:	ModelID	QuantityOnHand
Table:	Inventory	Inventory
Total:	Group By	Sum
Sort:		Descending
Show:	✓	✓
Criteria:		
or:		

ModelID	SumOfQuantityOnHand
YCG-584	70
YXY-385	70
QDV-720	70
MHQ-568	60
LDK-181	60
YKU-321	60
NTE-526	50
QDG-75	50
NCS-293	40
GUC-993	35
GZE-862	35
FCU-154	35
OES-739	35
IRZ-701	35



SalesCategory

CategoryID	CategoryName	LowLimit	HighLimit
1	Hot	0	6
2	Good	6	10
3	OK	10	20
4	Weak	20	40
5	Slow	40	1000

Figure 5.11

But Jim does not want to wade through the entire query every day. Instead, he is proposing a categorical system, where items with more than a certain QOH will be called slow sellers, and items with minimal QOH will be hot sellers. He also wants a few categories in-between. While you have the tools to build this query, there is one catch: he wants the ability to fine-tune the numbers on the ranges for each category. The solution is to create a new table that defines the category and the upper and lower limits for each category:

Action

Create a new query in Design view.

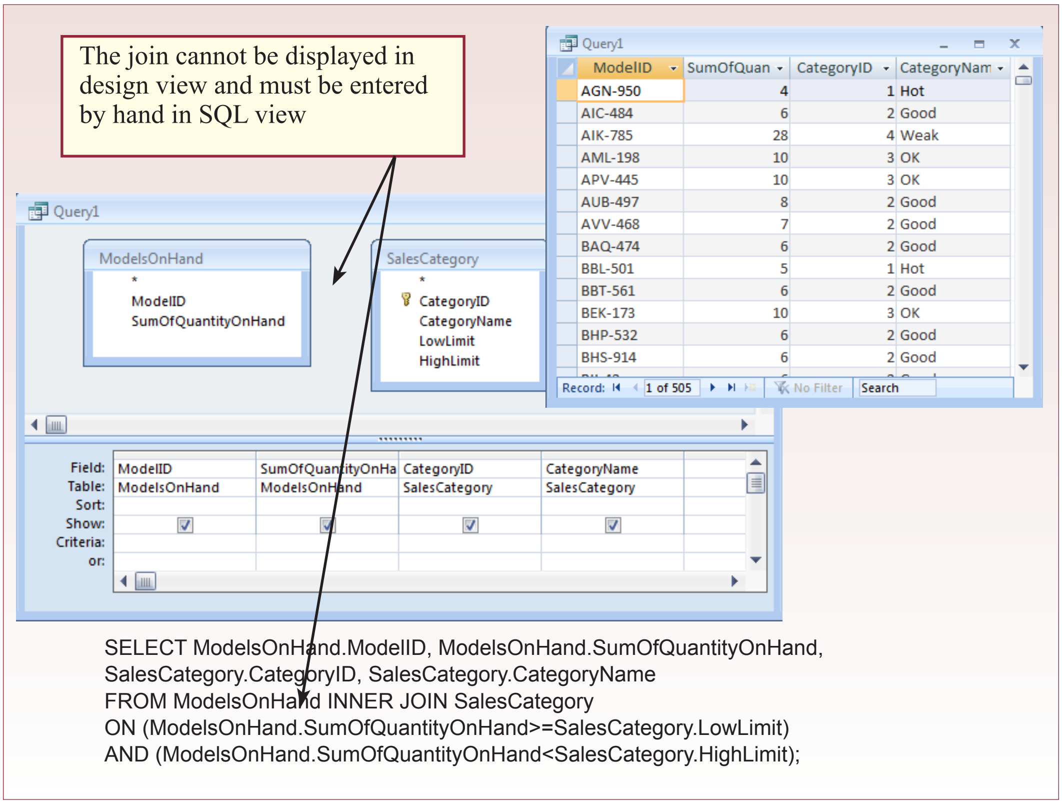
Tables: ModelsOnHand and SalesCategory.

Columns: ModelID, SumOfQuantityOnHand, CategoryID, and CategoryName.

In SQL view add the inequality join.

Run the query.

Figure 5.12



The join cannot be displayed in design view and must be entered by hand in SQL view

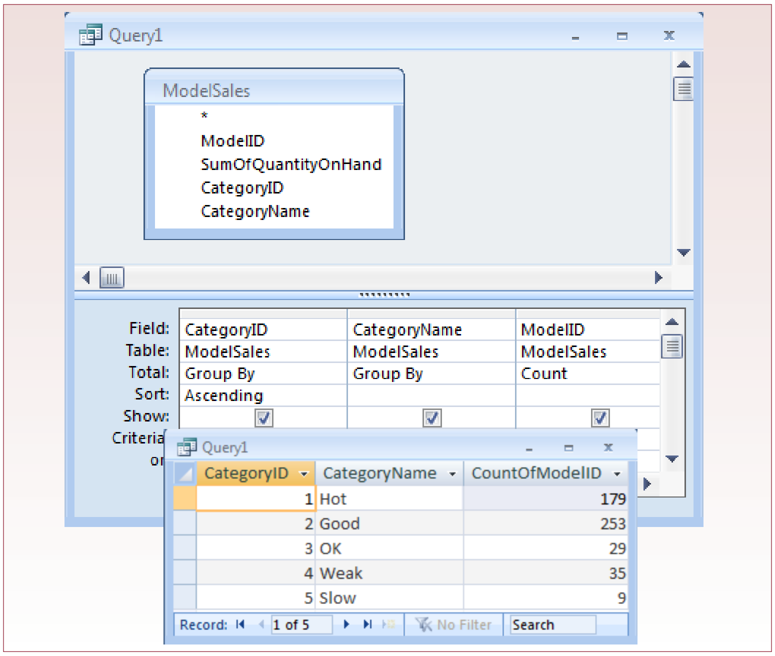
Query1

ModelID	SumOfQuan	CategoryID	CategoryNam
AGN-950	4	1	Hot
AIC-484	6	2	Good
AIK-785	28	4	Weak
AML-198	10	3	OK
APV-445	10	3	OK
AUB-497	8	2	Good
AVV-468	7	2	Good
BAQ-474	6	2	Good
BBL-501	5	1	Hot
BBT-561	6	2	Good
BEK-173	10	3	OK
BHP-532	6	2	Good
BHS-914	6	2	Good

Record: 1 of 505

Field:	ModelID	SumOfQuantityOnHa	CategoryID	CategoryName
Table:	ModelsOnHand	ModelsOnHand	SalesCategory	SalesCategory
Sort:				
Show:	✓	✓	✓	✓
Criteria:				
or:				

```
SELECT ModelsOnHand.ModelID, ModelsOnHand.SumOfQuantityOnHand,
SalesCategory.CategoryID, SalesCategory.CategoryName
FROM ModelsOnHand INNER JOIN SalesCategory
ON (ModelsOnHand.SumOfQuantityOnHand>=SalesCategory.LowLimit)
AND (ModelsOnHand.SumOfQuantityOnHand<SalesCategory.HighLimit);
```



Field:	CategoryID	CategoryName	ModelID
Table:	ModelSales	ModelSales	ModelSales
Total:	Group By	Group By	Count
Sort:	Ascending		
Show:	✓	✓	✓

CategoryID	CategoryName	CountOfModelID
1	Hot	179
2	Good	253
3	OK	29
4	Weak	35
5	Slow	9

Record: 1 of 5

Figure 5.13

SalesCategory(CategoryID, CategoryName, LowLimit, HighLimit). If the QOH for a model is greater than or equal to the LowLimit and less than the HighLimit, it falls into the specified category. The CategoryID ensures a unique key and could be used to sort the rows if necessary. Figure 5.11 shows the initial categories.

Using the categories in a query requires slightly tricky join conditions. You need to use inequality (theta) joins. Begin with a new query based on the ModelsOnHand query and the SalesCategory table. Display the ModelID and SumOfQuantityOnHand along with the CategoryName. But do not attempt to join the tables in Design view. Instead, switch to SQL view and modify the FROM clause to match the Figure 5.12 inequality join statement. Access can handle inequality joins but cannot display or edit them in Design view. Figure 5.12 also shows the sample result from the query. Save the query as ModelSales so Jim can perform some additional analysis on the data.

Jim might create a new simpler query that counts the number of models that fall within each of the categories. Figure 5.13 shows the basic query. It is built simply using the results of the previous query. This query hides the complicated details as Jim needs to see only the simple data results. The final aggregation query uses the CategoryID to sort the results logically; otherwise, they would be sorted alphabetically by the category name. Fortunately, most of the models appear to be in the categories indicating that they sell relatively quickly. Although, the category defini-

Action

Create a new query in Design view.

Tables: Customer and Sale.

Columns: CustomerID, LastName, FirstName, and SaleDate.

Set January sale date in criteria row.

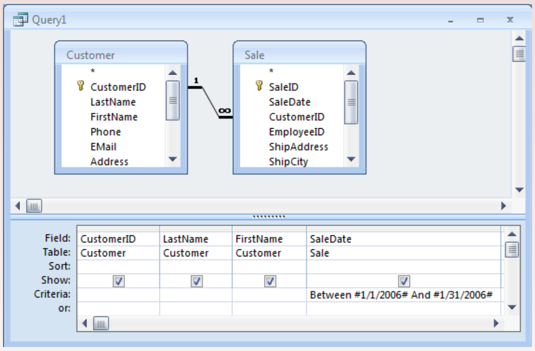
Switch to SQL.

Copy the entire statement.

Add the word Union.

Paste the SELECT statement and change the date condition to March.

Run the query.



Field:	CustomerID	LastName	FirstName	SaleDate
Table:	Customer	Customer	Customer	Sale
Sort:				
Show:	✓	✓	✓	✓
Criteria:				Between #1/1/2006# And #1/31/2006#
or:				

List customers who bought items in January or in March.

It could be done with simple conditions, but it is good practice for UNION.

```
SELECT Customer.CustomerID, LastName, FirstName, “Jan” As SaleMonth
FROM Customer INNER JOIN Sale ON Customer.CustomerID = Sale.CustomerID
WHERE (((Sale.SaleDate) Between #1/1/2004# And #1/31/2004#))
UNION
SELECT Customer.CustomerID, LastName, FirstName, “Mar” As SaleMonth
FROM Customer INNER JOIN Sale ON Customer.CustomerID = Sale.CustomerID
WHERE (((Sale.SaleDate) Between #3/1/2004# And #3/31/2004#));
```

Figure 5.14

tions might not be accurate, Jim can quickly alter the range numbers and rerun the query to see the results.

Activity: Combine Data Rows with UNION

You need to understand the role of the UNION command. It is designed to combine rows from multiple queries. Read that line carefully. It says combine rows not columns. If you have two queries that retrieve similar columns of data, the UNION statement will combine the results into one set of data. To illustrate the process, consider a request that Katy made to see a single list of customers who purchased items in January or in March. You could build this query using simple WHERE conditions, but if you want to list people twice if they bought items both in January and in March, the UNION query is easier.

As shown in Figure 5.14, create a new query using the Customer and Sale tables. Display the CustomerID, LastName, and FirstName columns. Add the SaleDate column, but uncheck the box to display the date. Add the condition to select sales only in January. If you run the query at this point, you will see a list of customers who bought items in January. To get the March customers, switch to SQL view. Mark the entire SQL statement and copy it. Delete the semicolon at the end, add the word UNION after the existing query, then below that, paste a copy of the query. Now modify the dates in this copy to indicate March instead of January. Finally, in the first SELECT statement (January), add a computed column to display “Jan” As SaleMonth. Do the same thing for the second SELECT statement, but display “Mar” for March. This column will identify each row to indicate the month for the sale. Run the query, and you will see a combination of rows from both queries. If you want to sort the data by Customer or by date, first you will have to save

Action

Create a new query in Design view.

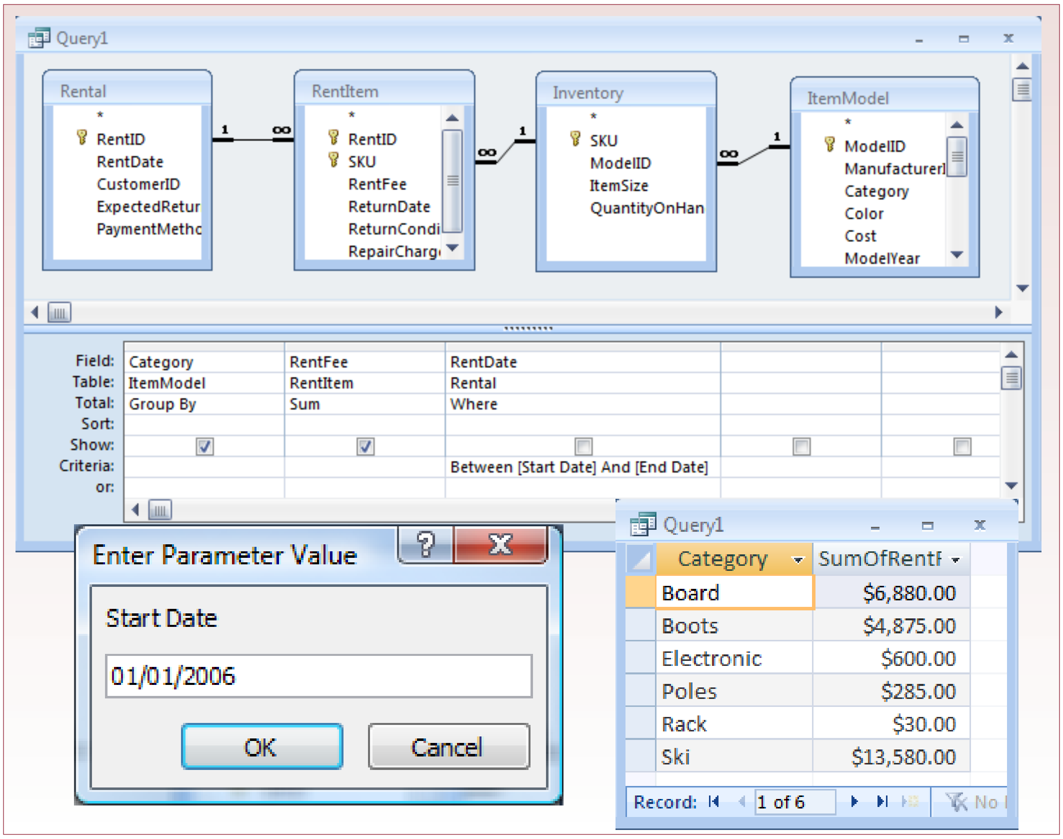
Tables: Rental, RentItem, Inventory, and ItemModel.

Columns: RentDate, Category, RentFee.

Set totals to sum RentFee.

Set “Where” and criteria for RentDate to Between [Start Date] And [End Date].

Run the query.



Rental: RentID, RentDate, CustomerID, ExpectedRetur, PaymentMethc

RentItem: RentID, SKU, RentFee, ReturnDate, ReturnCondi, RepairChargı

Inventory: SKU, ModelID, ItemSize, QuantityOnHan

ItemModel: ModelID, Manufacturerl, Category, Color, Cost, ModelYear

Field:	Category	RentFee	RentDate		
Table:	ItemModel	RentItem	Rental		
Total:	Group By	Sum	Where		
Sort:					
Show:	✓	✓	☐	☐	☐
Criteria:			Between [Start Date] And [End Date]		
or:					

Start Date

01/01/2006

Category	SumOfRentF
Board	$6,880.00
Boots	$4,875.00
Electronic	$600.00
Poles	$285.00
Rack	$30.00
Ski	$13,580.00

Figure 5.15

the query, then you can build a second query based on the first and sort the columns as needed.

Activity: Create Parameter Queries

Parameter queries are useful when you need to create a query that a manager runs on a regular basis but needs to change some of the constraints. For instance, you often use parameters to set starting and ending dates so the manager can easily select a range of data without having to know anything about building queries. The example in Figure 5.15 shows a query that displays the total rental income by Category for a specified range of dates. Save the query as RentalsByTime, so David the rental manager can run the query whenever he needs. When the query is run, two boxes pop up to request the starting date and ending date.

You can build complex queries and insert parameters to request specific data from the person running the query. In your query, simply enter a brief description in brackets, and treat that parameter as any other number or date. When the query runs, the user will enter a value and it will be applied to the query. It is a useful method to quickly build queries that users can control without having to alter the query.

Action
Create a new query in Design view.
Do not select any tables.
Switch to SQL view.
Enter the CREATE TABLE command.
Run the query.

```
CREATE TABLE Contacts
(
   ContactID          Long,
   ManufacturerID     Long,
   LastName           Text(25),
   FirstName          Text(25),
   Phone              Text(15),
   Email              Text(120),
      CONSTRAINT pk_Contacts PRIMARY KEY (ContactID),
      CONSTRAINT fk_ContactsManufacturer FOREIGN KEY (ManufacturerID)
        REFERENCES Manufacturer(ManufacturerID)
);
```

Figure 5.16

SQL Data Definition and Data Manipulation

Activity: Create Tables

Although Access makes it easy to create and delete tables using the visual designer, sometimes you will need to know how to create a table using the data definition language (DDL) CREATE TABLE command. After working with the database for a while, you realize that it would be nice to have a separate table that lists salespeople and other contacts at the manufacturers. Each person has a direct phone number and an e-mail address. To practice building tables, Figure 5.16 shows the CREATE TABLE command for the new Contacts table. Essentially, you list each desired column along with its data type. Note that you have to choose the basic data type, and not the higher-level type you might select in the designer. For instance, you have to enter "Long" instead of just "Number."

To enter the code, you must open the SQL Query window. Create a new query and close the Table Selection window without choosing any tables. Then click the SQL button or select it from the View option. Delete any statements in the window and type the CREATE TABLE query as shown.

You should also create the primary key constraint to indicate the ContactID is the sole primary key column. If you need multiple columns, simply create a comma-separated list. The foreign key constraint is similar, but you must also specify the table and column that is referenced by the foreign key. The Access 2002 version does not support the ON DELETE CASCADE or ON UPDATE CASCADE options for the foreign key, so you should add them by hand in the relationships screen. If you have a more recent version of Access, you should try adding these two lines immediately after the foreign key constraint to see if support has been added. To run the query, you must use the exclamation point Run button. You can only run this query one time, otherwise it will try to create duplicate copies of the same table with the same name—which is not allowed.

Figure 5.17

```
CREATE TABLE MyTemp
(
    ID          Long,
    LName       Text(25),
    FName       Text(25)
);
```

Generally with Access, it is easier to create tables with the design screen. However, sometimes you will want to create temporary tables—eventually within a section of program code. For the next section, you will need a temporary table to transfer data. Figure 5.17 shows the table that you need to create.

The SQL ALTER TABLE command can also be used to add new columns to an existing table. However, you rarely need this command, since it is easier to use the table design view in Access to add a new column to a table.

Activity: Insert, Update, and Delete Data

Action

Create a new query in Design view.

Do not select any tables.

Switch to SQL view.

Type the INSERT command: INSERT INTO Customer (LastName, FirstName, City, Gender) VALUES ('Jones', 'Jack', 'Nowhere', 'Male');.

Run the query.

SQL also provides data manipulation language (DML) commands to insert, update, and delete rows of data. Consider the INSERT command first. The simple version of the command shown in Figure 5.18 inserts a single row into one table. Notice that you specify the table columns in the first list and the corresponding values in the second list. By listing the column names, you choose to enter the data in any order and to skip columns. Of course, you will rarely enter data this way, but occasionally it comes in handy. More importantly, the SQL statement can be generated using programming code with complex routines to extract data from one source, clean it up, and transfer it to the desired table.

On the other hand, a second version of the INSERT command is more useful because of its power. You use it to transfer large blocks of data from one table into a second table. Note that the second table must already exist. The example in Figure 5.19 copies some data from the Customer table and transfers it to the temporary MyTemp table you created in the previous section. Again, you list the columns for the new table that will hold the data, then write a SELECT statement that retrieves matching data for those columns.

Figure 5.18

```
INSERT INTO Customer (LastName, FirstName, City, Gender)
VALUES ('Jones', 'Jack', 'Nowhere', 'Male');
```

You should keep in mind that the SELECT statement can be as complex as you wish. It can include calculations, multiple tables, complex WHERE conditions, and subqueries. For complex queries you should first build the SELECT statement on its own and test it to ensure that it retrieves exactly the data you want. Then switch to the SQL view and add the INSERT INTO line at the top. The ability to perform calculations has another benefit. You can add a constant to the SELECT

Figure 5.19

```
INSERT INTO MyTemp (ID, LName, FName)
        SELECT CustomerID, LastName, FirstName
        FROM Customer
        WHERE City='Sacramento'
;
```

statement that will be inserted as data into the second table. For example, you might write SELECT ID, Name, "West" to insert a region name into a new table. The INSERT INTO command is useful when you need to expand a database or add new tables. You can quickly copy selected rows and columns of data into a new table.

Action

Create a new query in Design view.

Table: ItemModel.

Columns: Category, ModelYear, and Cost.

Criteria: Category="Board" And ModelYear=2004.

Run the query.

Choose Query/Update Query.

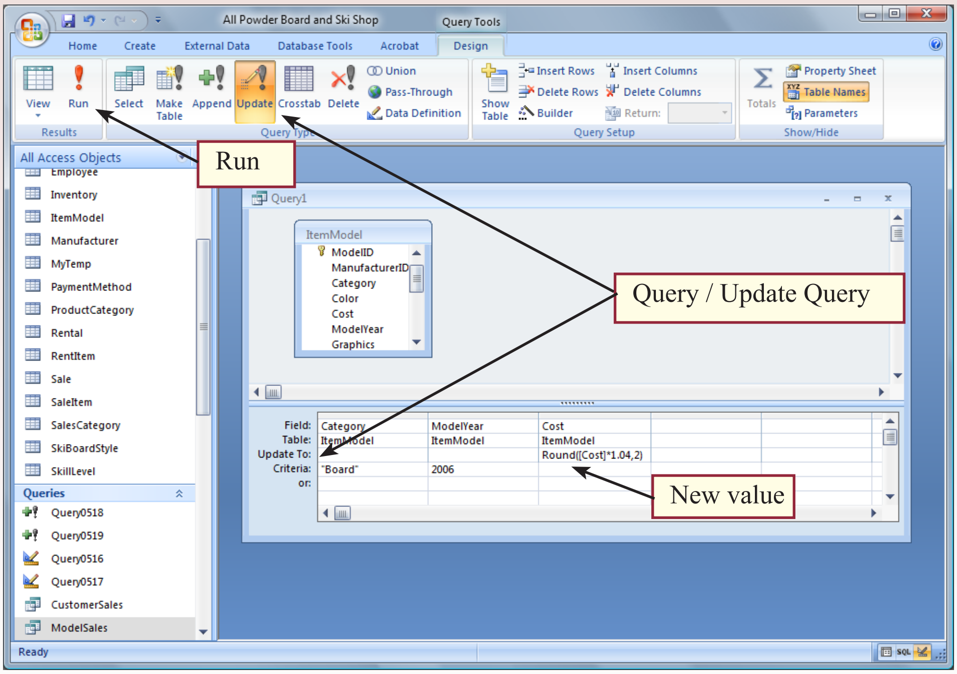
Under Cost set Update To: Round([Cost]*1.04,2).

Run the query.

The UPDATE command is used to change individual values for specified rows. It is a powerful command that affects many rows. You must always be cautious when using this command because it can quickly change thousands of rows of data. To illustrate the power of the command, consider that the manufacturers have announced that costs will increase by 4 percent for the 2006 boards. The ItemModel table contains an estimate of the Cost for each model, so you need to increase this number by 4 percent, but only for the boards.

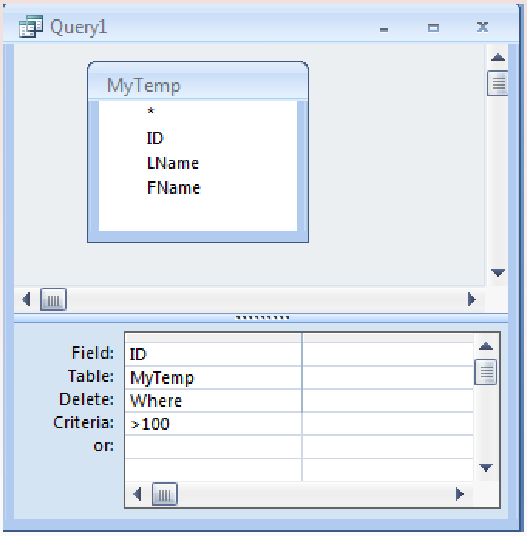
To be safe, begin by creating a query that displays the Cost for the 2006 boards. You should run the query to ensure that it returns exactly the data that you want to update. Next, as shown in Figure 5.20, select the Query/Update Query option on the main menu to display the Update To row on the grid. In the Cost field, enter

Figure 5.20



Field:	Category	ModelYear	Cost
Table:	ItemModel	ItemModel	ItemModel
Update To:			Round([Cost]*1.04,2)
Criteria:	"Board"	2006	
or:			

```
UPDATE ItemModel SET Cost = Round([Cost]*1.04,2)
WHERE (Category="Board") AND (ModelYear=2004);
```



Field:	ID
Table:	MyTemp
Delete:	Where
Criteria:	>100
or:	

```
DELETE
FROM MyTemp
WHERE ID > 100;
```

Figure 5.21

the new calculation as Round([Cost] * 1.04, 2) to indicate the 4 percent cost increase. Note that you will have to type the brackets around the Cost column name. If you do not, Access might place the entire formula in quotes, which would result in an attempt to write the formula text into the column instead of the actual values. The Round function is used to ensure that the final Cost value is rounded off to cents instead of extended fractions. You can run the query by clicking the Run button on the main toolbar.

Notice that the SQL statement is straightforward. It is also easy to change multiple columns at one time. On the grid, just add the new column and the new formula. In SQL, add a comma after the first calculation and enter a new one in the SET clause. Of course, you can use multiple tables. In SQL, they are added with the standard join statement; but it is generally easier to add the tables in the Design view grid and let Access build the join statement. If necessary, you can then switch to SQL to modify the lines.

The DELETE command is similar to the INSERT and UPDATE commands, but it is more dangerous. It is designed to delete many rows of data at a time. Keep in mind that because of the relationships, when you delete a row from one table, it can trigger cascade deletes on additional tables. For the most part, these deletes are permanent. If you are not careful, you could wipe out a large

Action

Create a new query in Design view.
Table: MyTemp.
Columns: ID, LastName, FirstName.
Criteria: ID > 100.
Test the query.
Choose Query/Delete Query.
Run the query.

Figure 5.22

```
DROP TABLE MyTemp;
```

chunk of your data with one delete command. To minimize the impact of these problems, you should always make backup copies of your database—particularly before you attempt major delete operations. You should probably take a break and do that now. Use Windows to copy the main database file.

To be extra safe, this example is just going to delete data from the temporary table that was created in the previous section. Create a new query in Design view using the MyTemp table. To see the rows you are going to delete, add all three columns to the display and set a condition to show only rows with an ID > 100. You can then use the Query/Delete Query option to setup the delete query. If you switch to the SQL view, you will see that the individual columns are listed. These columns are not necessary because the query will always delete the entire row. Figure 5.21 shows both the Design view and the basic SQL needed to delete the specified rows. In practice, it is best to stick with simple WHERE clauses when possible. However, it can be complex and include subqueries. Particularly in the complex cases, you should first build a SELECT statement using the same WHERE clause to ensure that you are deleting exactly the rows you want to delete. Then convert the query into a Delete Query, or delete the SELECT statement and replace it with the DELETE command.

The DROP TABLE command is even more dangerous. It removes the entire table and all of its data. Generally, you should only use it for temporary tables. As shown in Figure 5.22, the syntax is straightforward; just make sure you enter the correct table name. Again, it would be wise to make a backup copy of your database before removing tables.

The main aspect to remember about these commands is that they operate on sets of rows that you control with the WHERE clause. The WHERE clause can be complex and can include subqueries with detailed SELECT commands. All of the power of the SELECT command is available to you to control inserting, updating, and deleting rows of data.

Exercises

Many Charms

You will need to create some additional sample data for each table. Madison and Samantha know that they will want certain information on a weekly basis, but they will not be able to build complex queries to retrieve the data. You will have to build a few queries for them that they can run when they want to see the results or need to change prices. Some of the queries should be parameter queries so they can easily select the values they need to control the results. Note: You will have to modify the queries slightly to match the data that you have entered.

1. Which customers who ordered bracelets have not ordered necklaces?
2. Which customers bought more gold charms than silver ones?
3. Which categories generated the most profit over a parameterized time period?
4. Are expensive charms more profitable than mid-priced or low-priced charms? Hint: Create categories based on the prices.
5. Create a parameterized query to enable Samantha to increase the prices of a certain category of charms by a given percentage.

6. Create a new table with SQL and copy into it all of the customers who have not purchased items within the last three months.
7. Delete customers from the new table in the prior exercise who have spent more than $100 in the past year.

Standup Foods

You will need to create some additional sample data for each table. Laura knows she will want certain information on a weekly basis, but she will not be able to build complex queries to retrieve the data. You will have to build a few queries for her that can be run to display results or change prices. Some of the queries should be parameter queries so Laura can easily select the values she needs to control the results. Note: You will have to modify the queries slightly to match the data that you have entered.

1. Identify the employees who have below-average overall job evaluations.
2. Identify the main menu items that have not been served to a particular director or other celebrity. (Pick one from your list who wants something different.)
3. Which customers have not yet referred her business to other clients?
4. Create a category table to segment the employee ratings (excellent, good, average, poor). Use the table to identify the employees with excellent evaluations as both server and dishwasher.
5. Create a temporary table and copy into it information about employees who have worked as drivers but have not driven within the last month.
6. Delete from the temporary table in the previous question the drivers whose average evaluations are less than 6 (on the 10-point scale).
7. Write a parameterized query that enables Laura to increase the base wage rate of employees by specifying a category, a minimum overall average evaluation, and the percentage increase.

EnviroSpeed

You will need to create some additional sample data for each table. Brennan and Tyler know that they will want certain information on a weekly basis, but they will not be able to build complex queries to retrieve the data. You will have to build a few queries for them that they can run when they want to see the results or need to change prices. Some of the queries should be parameter queries so they can easily select the values they need to control the results. Note: You will have to modify the queries slightly to match the data that you have entered.

1. List the experts who have worked with two or more crews in the same month.
2. Which experts have not contributed any documents within the last three months?
3. List the crews that are more than 25 percent larger than the average crew.

4. Create a table to categorize the expense of cleanups. For example, spills costing more than $1 million to clean up are expensive; cleanups costing $500,000 to $1 million are merely costly; and so on. Create a query to apply these categories to the actual spills.
5. Write a query that retrieves documents based on a list of keywords entered by a user. The keywords might appear anywhere in the document, and the final query should sort the list based on the number of matches.
6. Write a parameterized query to update a severity value for an incident by allowing the user to enter a chemical name and a point increase in severity.
7. Write a query to copy the data on experts to a new table who have participated in a total of at least three incidents in the last year.

Final Project

The main textbook has an online appendix with several longer case studies. You should be able to work on one of these cases throughout the term. If you or your instructor picks one, answer the questions below. You will have to create sample data for each of the tables.

1. Identify and create at least two parameter queries that would be useful to managers. Share the business question (not the query) with other students and solve their queries.
2. Identify a business question to list items greater (or less) than average. Write the query to return the results.
3. Create a temporary table and write a query to copy some rows of data from one table into the new table.
4. Write a delete query to remove a few rows of data from the temporary table.
5. Write an update query using parameters to change the value of one of the numeric columns in a table based on a percentage and conditions entered by the user.

Chapter 6

Forms and Reports

Chapter Outline

Objectives

- Create forms (main, grid, and subforms) that make it easy for users to enter data.
- Create reports to display and summarize data.
- Build applications that connect forms and reports.
- Add toolbars and menus to forms.
- Add help files to the database application.

Applications

The main purpose of the DBMS is to store data efficiently and provide queries to retrieve data to answer business questions. But from the perspective of businesses, the true value of the DBMS lies in the applications that can be built on top of the database. One of the strengths of Access is the tools provided to build forms and reports. Once the database is designed, you can use the wizards to quickly build basic forms and reports. You will still need to edit the designs to clean them up to make them easier to use..

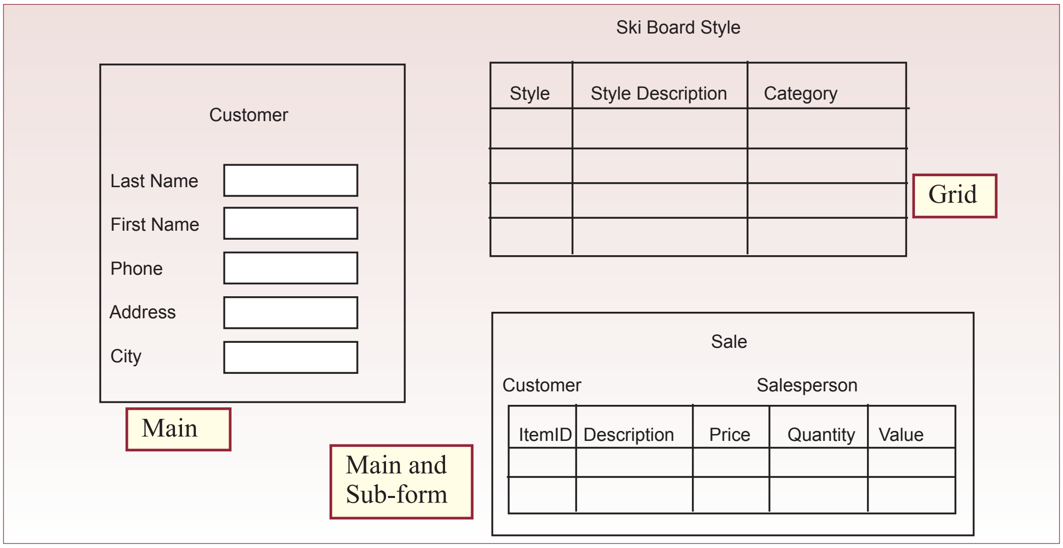
Forms are used to make it easier for users to enter data. You would never want users to enter data directly into the tables. For example, look again at the Sales table. It contains mostly ID numbers, and you cannot expect workers to memorize thousands of ID numbers. Instead, you build forms to match the processes and styles of the business. Likewise, you rarely ask managers to build queries themselves. Instead, you create reports that display details and subtotals within a layout that is easy to read. You can even include charts to make it easy to compare values or examine trends over time.

Access provides some support for Web pages. However, these tools will work only on internal servers. For the most part, you cannot use Access forms and reports to provide data on the Internet to typical users. For those types of tasks, you should switch to Visual Studio and ASP.NET.

Case: All Powder Board and Ski Shop

The primary application at All Powder Board and Ski Shop is the need to track sales and rentals. Of course, these applications also require you to build forms and reports for inventory items and customers as well. Eventually you will have forms that store data into each of the tables in the relationship diagram. However, before you leap to the Forms wizard, make sure you understand the three major form types shown in Figure 6.1: main form, grid form, and main with subform. A main form shows one row of data at a time, such as a form to edit basic information about one customer. A form appears similar to the Table view in that it shows sev-

Figure 6.1



eral rows at one time. Main and subforms combine the two: the main form shows one row of data from one table and the grid subform shows matching rows from a related table. The classic business example is the Sale form and SaleItem grid, where the main form shows data from one sale, and the grid shows the repeating items purchased and stored in the SaleItem table. At this point, your responsibility is to examine the business operations and determine the best type of form to handle each operation.

Lab Exercise

All Powder Board and Ski Shop Forms

Many of the forms in an application are straightforward main forms. Users want to see data for one row—such as one customer or one employee. You generally create main forms when you need more control over the layout.

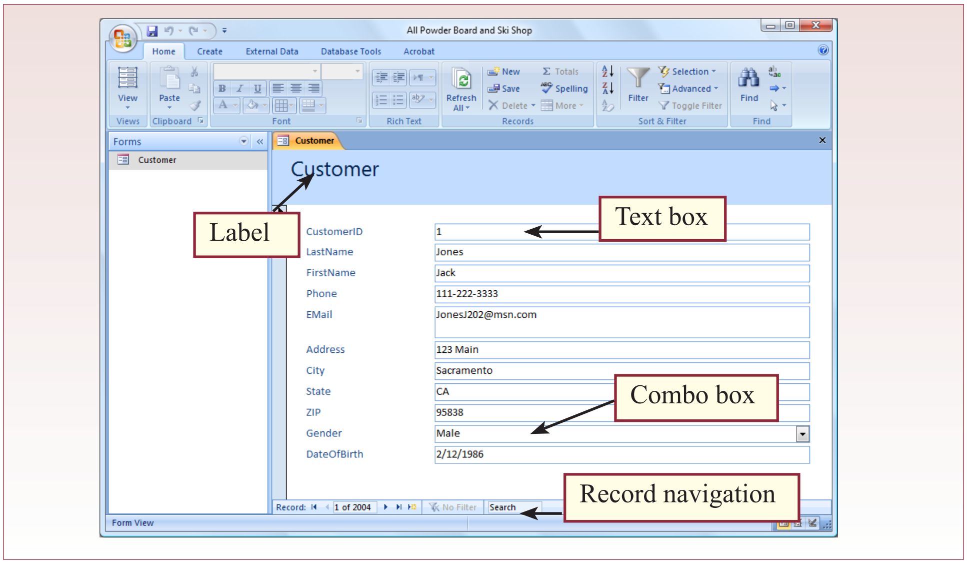
Activity: Create Basic Main Forms

Action

Start the form wizard.
Select the Customer table.
Transfer all columns to the right window.
Use the Columnar layout.
Select a style.
Enter a name for the form.

Figure 6.2 shows a simple version of the form to edit customer data. In its simplest layout, the main form contains labels and text boxes for each column in the table. You can enter any text into the label to help tell the user what data is to be entered into each text box. The data on the form is bound to the database table. Changes made to the data in the text boxes are automatically written to the database table. However, these changes are written only at certain times—such as when the user moves to a new row. The importance of the main form is that you have considerable control over the layout and presentation of the items. You can change the image of the form by setting the properties for the form or the controls to control things such as size, position, and color. You can add new con-

Figure 6.2



Customer

CustomerID	1
LastName	Jones
FirstName	Jack
Phone	111-222-3333
EMail	JonesJ202@msn.com
Address	123 Main
City	Sacramento
State	CA
ZIP	95838
Gender	Male
DateOfBirth	2/12/1986

Action

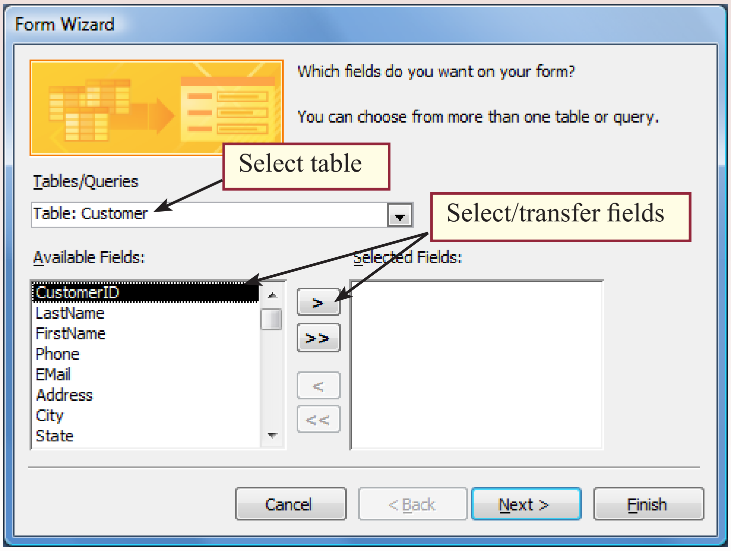
Switch Customer form to design view.
Add a command button.
Select Record Operation/Delete Record.
Add a command button to insert records.
Run the form and test the buttons.
View/Tab Order and verify sequence.
Save the form.

trols to display images or include buttons to delete or find records.

Access 2007 introduced two main options for forms: (1) a tabbed display of multiple forms, and (2) the layout control on a form that treats controls as a group. Both of these options are used in this form. The tabs option is set for the entire database, not for individual forms. Its use generally comes down to individual preference but it tends to force all forms to be the same size. It is set or removed using the Office Button in the top-left corner of Access and clicking the Access Options button. Under the Current Database list, you can set either Overlapping Windows or Tabbed Documents.

You created a version of this form in Chapter 1 using the simple form builder. To explore some of the options, this chapter focuses on using the Form Wizard. Creating a main form is straightforward using the wizard. To build the Customer form, create a new form using Create/Forms/More Forms/Form Wizard. Figure 6.3 shows the most important step of the wizard: choosing the tables and fields. For this form, select the Customer table and choose all of its columns by transferring them to the box on the right. In the next step, select the Columnar form, which is how Access describes the main form type. Then you can select the overall style. If you are ambitious, you can create your own template styles for other developers in your company to use. The style simply applies some basic design elements to the form, such as font size and background colors. For the most part, you want to pick a style that is easy to read. Also, be sure that you remain consistent. All of the forms in an application should have the same style. Figure 6.2 shows a Customer form that should be similar to yours. If you finish the wizard and realize that you

Figure 6.3


Form Wizard

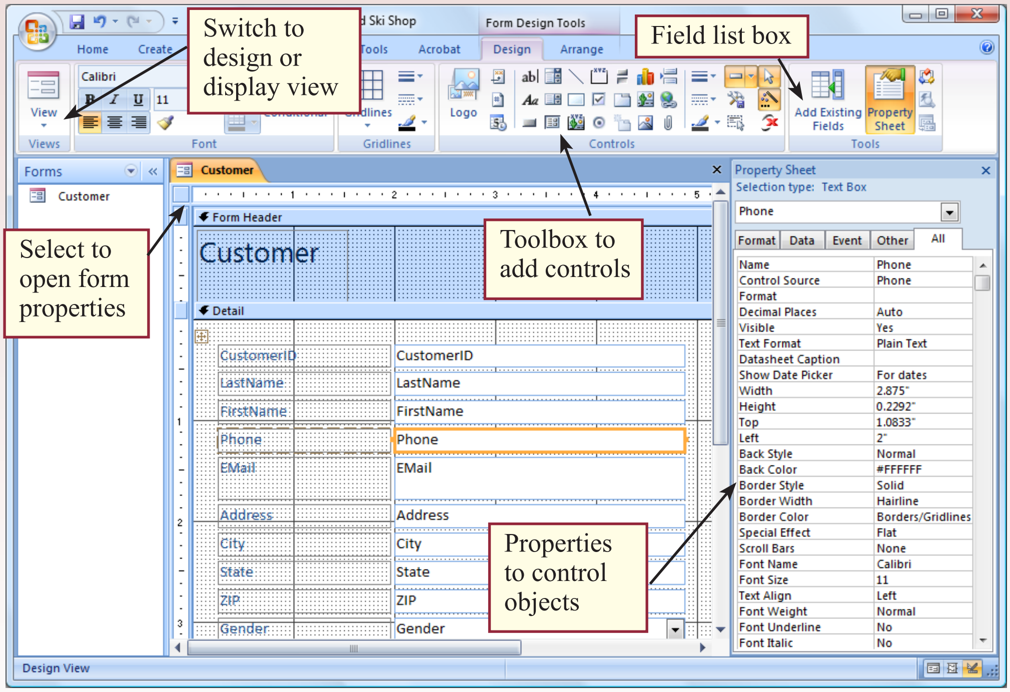
Which fields do you want on your form?

You can choose from more than one table or query.

Tables/Queries: Table: Customer

Available Fields: CustomerID, LastName, FirstName, Phone, EMail, Address, City, State

Selected Fields:



CustomerID	CustomerID
LastName	LastName
FirstName	FirstName
Phone	Phone
EMail	EMail
Address	Address
City	City
State	State
ZIP	ZIP
Gender	Gender

Property Sheet

Selection type: Text Box

Phone

Format | Data | Event | Other | All

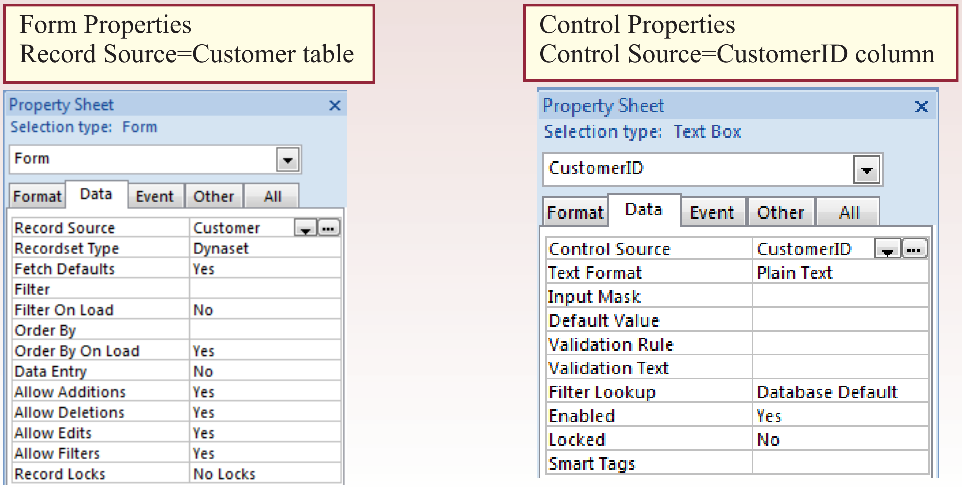
Name	Phone
Control Source	Phone
Format	
Decimal Places	Auto
Visible	Yes
Text Format	Plain Text
Datasheet Caption	
Show Date Picker	For dates
Width	2.875"
Height	0.2292"
Top	1.0833"
Left	2"
Back Style	Normal
Back Color	#FFFFFF
Border Style	Solid
Border Width	Hairline
Border Color	Borders/Gridlines
Special Effect	Flat
Scroll Bars	None
Font Name	Calibri
Font Size	11
Text Align	Left
Font Weight	Normal
Font Underline	No
Font Italic	No

Figure 6.4

made a mistake or simply want to change the design, the easiest way to make the change is to delete the form and start over.

The wizard does a decent job of displaying the data for the form, but invariably you will want to modify the design of the form. Sometimes you simply need to change the layout, formatting, and colors. At other times you want to add but-

Figure 6.5


Form Properties
Record Source=Customer table
Control Properties
Control Source=CustomerID column

Property Sheet

Selection type: Form

Form

Format | Data | Event | Other | All

Record Source	Customer
Recordset Type	Dynaset
Fetch Defaults	Yes
Filter	
Filter On Load	No
Order By	
Order By On Load	Yes
Data Entry	No
Allow Additions	Yes
Allow Deletions	Yes
Allow Edits	Yes
Allow Filters	Yes
Record Locks	No Locks

Property Sheet

Selection type: Text Box

CustomerID

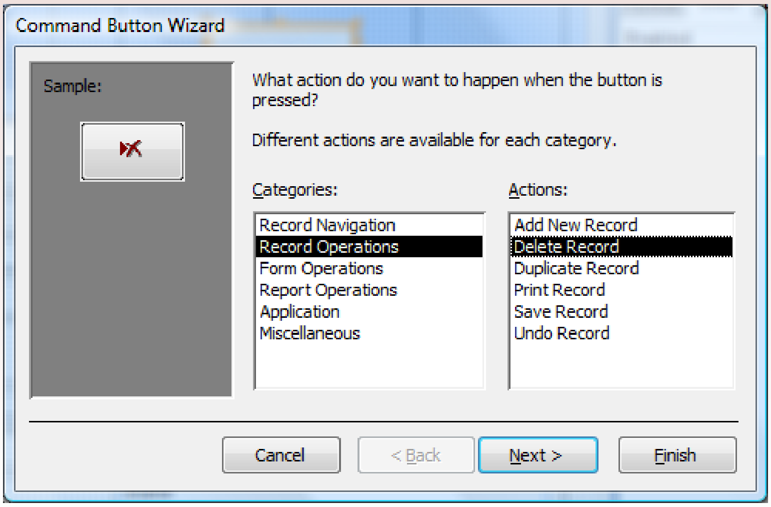
Format | Data | Event | Other | All

Control Source	CustomerID
Text Format	Plain Text
Input Mask	
Default Value	
Validation Rule	
Validation Text	
Filter Lookup	Database Default
Enabled	Yes
Locked	No
Smart Tags	

tons to open additional forms or reports, or to add or delete data rows. As shown in Figure 6.4, you can switch to Design view to edit the details of the form and its controls. The layout view is useful for changing alignments or sizes of group objects. Design view provides more detailed control over each individual object. Right-click a control, the form background, or the small square in the top-left corner of the form to set the various properties. The property box shows you which properties can be set for each item and helps you select the appropriate values. The toolbox contains additional controls that you can place on the form. The field list box is a quick way to add a standard data field to the form. For example, if you delete one of the existing fields and its label, you can open the field list box and drag the field back onto the form.

To understand how the form works, you should look at two main properties: the Record Source for the form and the Control Source for a text box. As shown in Figure 6.5, in this example, the Record Source is the Customer table and the Control Source is the CustomerID. The Record Source is a table or query that retrieves all of the columns that can be displayed and edited on the form. The field list box uses this query to show you the fields available to be placed on the form. In this example, remember that only the Customer table was selected when the form was created. You could have added columns from additional tables, in which case the Record Source would be a query. You can still make changes—simply click the Ellipses button (…) and you will be shown a query edit screen in which you can add more tables and columns. However, you must be careful to make sure the query is updatable. The main trick is to use the primary key from only one table. You can pull in other columns from additional tables, but use the primary key from only one table. This condition is critical to understanding the role of forms: Each form is designed to edit only one table at a time. You can stretch that statement a little and actually change some data values in several tables, but the main form should only be used to add data rows to a single table. Once you have the table or query chosen, you can set the Control Source for each individual control. Of course, it is much easier to let the wizard or the field list box do the work, but

Figure 6.6


Command Button Wizard

Sample:

What action do you want to happen when the button is pressed?

Different actions are available for each category.

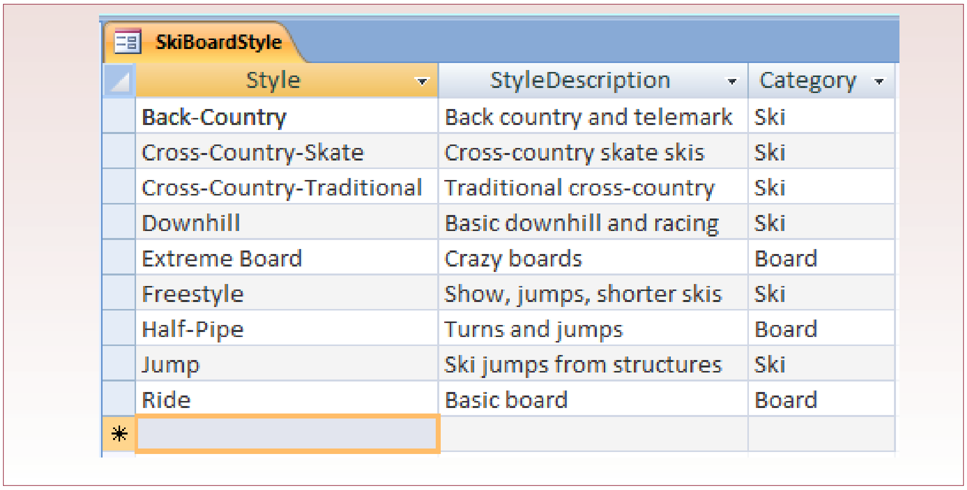
Categories:	Actions:
Record Navigation	Add New Record
Record Operations	Delete Record
Form Operations	Duplicate Record
Report Operations	Print Record
Application	Save Record
Miscellaneous	Undo Record

once in a while you might have to edit a Control Source—particularly if you go back and change the name of a column in a table.

Adding labels and command buttons to the form is a common step. For instance, you might want to add buttons to make it easier to add and delete a data row. If you run the form, you will see that the menu provides options to delete and add rows, but users sometimes forget to check the menu. They also might not know that they should click the selection bar at the left side of the form. It is often useful to place buttons on the form to delete and add new rows of data. To begin the process in Design view, select the command button in the toolbox and click a desired location on the form. Figure 6.6 shows the main command button wizard screen. It provides many common options that you might want to use on a form. In this case, select the Record Operations and then Delete Record options. You can use one of the default icons or enter a word that will be placed on the button. The wizard writes the code necessary to carry out the desired command. This code is stored in a subroutine on the form, and you should give it a descriptive name so you can find it later if needed. In this case, cmdDelete is a useful name, where the cmd prefix indicates that the routine is tied to a command button. The wizard then creates the button and the necessary code. If you selected an icon for the button, set the button's control tip property to add a short description of the button's role. In this case, "Delete this customer" is a good start. It will be displayed when the user rolls the cursor over the button. Follow the same procedure to create a button to add a new customer. To see the code that was created, right-click the button and choose Build Event. For simple events, the Access 2007 wizard relies on macro commands. Macros are more difficult to edit and have fewer options than VBA code. Later, when you want to add more complex processing to your form, you will probably want to write in VBA code instead of using macros. For buttons without macros, the Build Event link provides an option to display code, or you can click the View Code icon in the Tools section of the ribbon.

One last element of the form is important. The controls have a tab order that specifies how the focus shifts when the user presses the Enter or Tab keys. Generally, the tab order should be set so that the user's focus moves from the top to the bottom of the form. When you add controls to a form, this order is often altered. Right-click the form and use the Tab Order option to rearrange the sequence.

Figure 6.7



Style	StyleDescription	Category
Back-Country	Back country and telemark	Ski
Cross-Country-Skate	Cross-country skate skis	Ski
Cross-Country-Traditional	Traditional cross-country	Ski
Downhill	Basic downhill and racing	Ski
Extreme Board	Crazy boards	Board
Freestyle	Show, jumps, shorter skis	Ski
Half-Pipe	Turns and jumps	Board
Jump	Ski jumps from structures	Ski
Ride	Basic board	Board

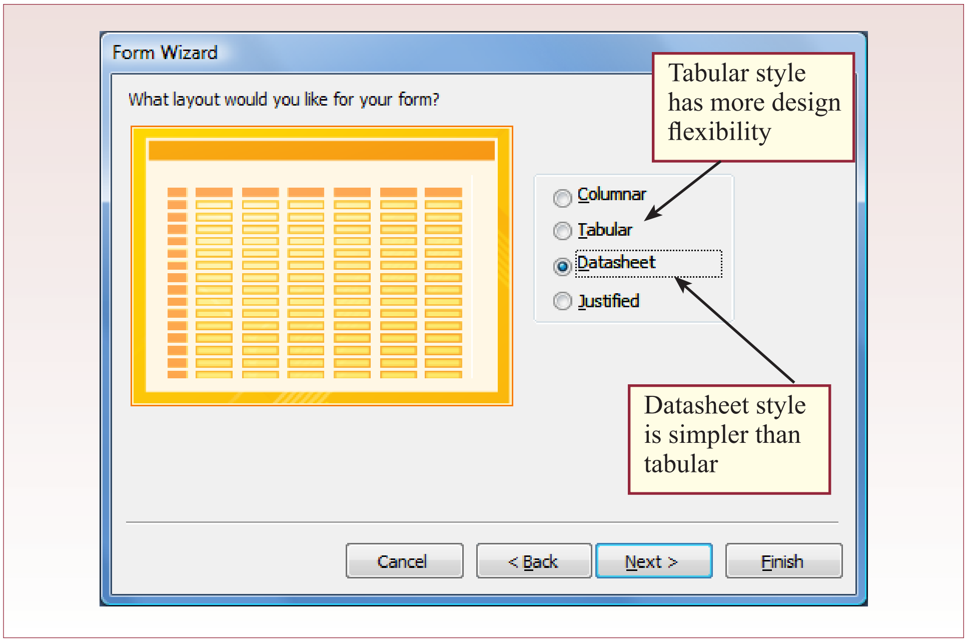

Form Wizard
What layout would you like for your form?
Columnar
Tabular
[✓ Datasheet]
Justified
Cancel
< Back
Next >
Finish
Tabular style has more design flexibility
Datasheet style is simpler than tabular

Figure 6.8

Activity: Create Grid Forms

Action

Create a new form with the wizard.

Table: SkiBoardStyle.

Columns: All.

Select the Datasheet layout.

Test the form.

Grid forms are another simple type of form. They are used when a table has a limited number of columns and rows. The columns should all fit on one screen because users find it difficult to edit data if they have to scroll horizontally. The number of rows should be limited because the grid form has few methods for searching, and users should not be forced to scroll through thousands of rows to change one piece of data. Figure 6.7 shows an initial grid form for the SkiBoardStyle table. Notice that the data in this table is generally used only to provide consistent values to other tables. This form will generally be used only by an administrator once in a while to modify or add a style. The data all fit on one screen, making it easy for you to find the items to be altered and to compare the various entries across the rows. In practice, you will use grid forms for similar tasks aimed at administration. Think hard before you use one of these forms for general users. Although you have some control over the form design, your options are limited, so users need to know what they are doing.

You create a grid form in much the same way as a main form. Start the wizard, select the table, and choose the columns you want on the form. As shown in Figure 6.8, the different step is to select the Datasheet option. You could also choose the Tabular style which provides a few more design options but tends to have thicker lines between rows and columns. The datasheet style is similar to editing data directly in a table, but you can impose more limits and controls on the user. So, if you find that you want to give users the ability to edit data directly, create a datasheet form instead.

Action

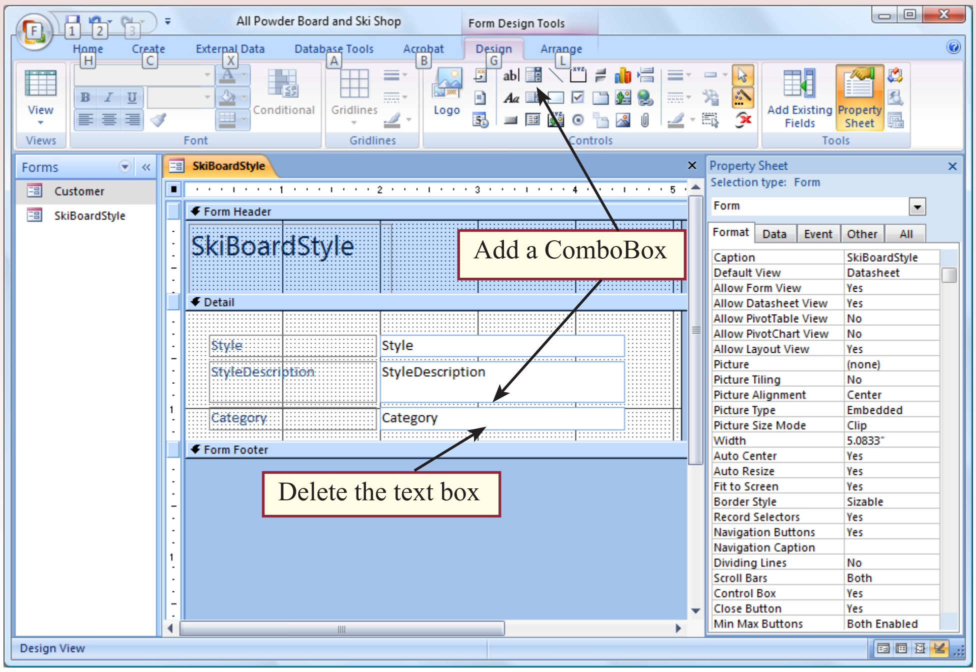
Switch to form Design view.
Delete the box for Category.
Add a combo box.
Select the first lookup option.
Choose the ProductCategory table.
Choose the Category and CategoryDescription columns.
Set column widths.
Select the Category column.
Store value into the Category column.
Rename the combo box.
Set the tab order.
Run the form and test the combo box.

Look closely at the data in Figure 6.7 and you will see that the Category data actually comes from a second table: ProductCategory. Of course, you should also build a grid form to enable administrators to update this table as well. However, you also need to make it easier for people to enter data into the Category column of the main SkiBoardStyle table. If users have to retype the data for every row, they might abbreviate or misspell the entries—leading to inconsistent data that is difficult to search. Instead, you want to create a combo box on the form that makes it easy for users to select the desired category.

As shown in Figure 6.9, switch the form to Design view and you will see that it looks much like a main form. However, the layout of the form is ignored when it is displayed as a datasheet. In particular, the order of the columns is set by the tab order and not by the position on the form. The goal is to replace the Category text box with a combo box, so the first step is to delete the Category text box and its associated label. Then, you can use the toolbox to place a combo box on the form.

The combo box wizard supports three uses: to look up data from a second table, to select from a fixed list of values, or to search the existing form for matching

Figure 6.9

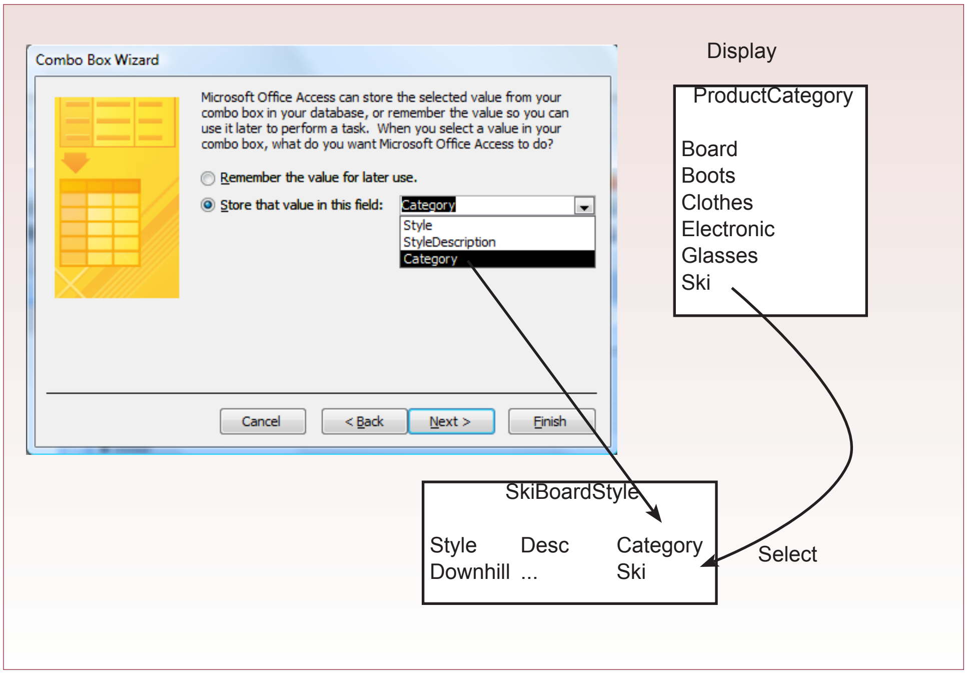


Property Sheet
Selection type: Form

Form

Format | Data | Event | Other | All

Caption	SkiBoardStyle
Default View	Datasheet
Allow Form View	Yes
Allow Datasheet View	Yes
Allow PivotTable View	No
Allow PivotChart View	No
Allow Layout View	Yes
Picture	(none)
Picture Tiling	No
Picture Alignment	Center
Picture Type	Embedded
Picture Size Mode	Clip
Width	5.0833"
Auto Center	Yes
Auto Resize	Yes
Fit to Screen	Yes
Border Style	Sizable
Record Selectors	Yes
Navigation Buttons	Yes
Navigation Caption	
Dividing Lines	No
Scroll Bars	Both
Control Box	Yes
Close Button	Yes
Min Max Buttons	Both Enabled

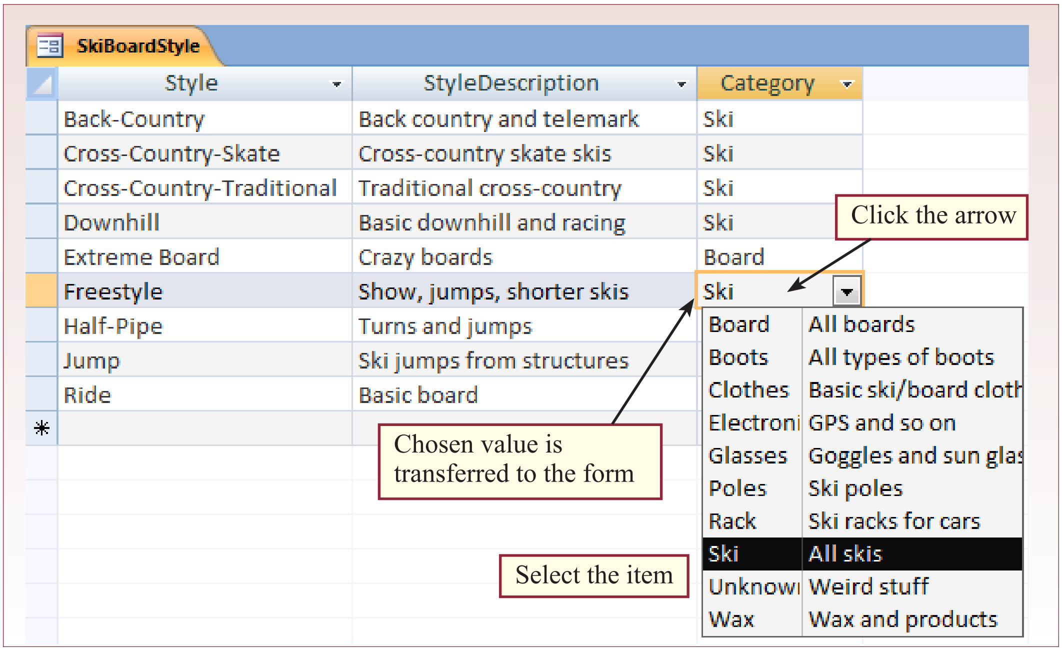


Microsoft Office Access can store the selected value from your combo box in your database, or remember the value so you can use it later to perform a task. When you select a value in your combo box, what do you want Microsoft Office Access to do?

Figure 6.10

data. In most cases, you want the first option. You should almost always avoid the second option because it is difficult to change the values later. You generally want to place the values within a separate table and build a form so they can be altered later without having to modify this form. The third option is sometimes useful on main forms but can be confusing because it is used to search for data instead of

Figure 6.11



Style	StyleDescription	Category
Back-Country	Back country and telemark	Ski
Cross-Country-Skate	Cross-country skate skis	Ski
Cross-Country-Traditional	Traditional cross-country	Ski
Downhill	Basic downhill and racing	Ski
Extreme Board	Crazy boards	Board
Freestyle	Show, jumps, shorter skis	Ski
Half-Pipe	Turns and jumps	
Jump	Ski jumps from structures	
Ride	Basic board	

Board	All boards
Boots	All types of boots
Clothes	Basic ski/board cloth
Electroni	GPS and so on
Glasses	Goggles and sun glas
Poles	Ski poles
Rack	Ski racks for cars
Ski	All skis
Unknow	Weird stuff
Wax	Wax and products

entering data. The first option is the one needed in this case, so select the table that holds the data that will be displayed in the combo box list (ProductCategory). Choose both the Category and the CategoryDescription columns to be displayed in the combo box list. The wizard gives you the option to hide the key column, which is commonly done for numeric AutoNumber keys. In this case, the key column is text, so make sure it is displayed.

Figure 6.10 shows the role of the combo box and one of the trickiest steps in the combo box wizard. The purpose of a combo box is to display a list of predefined items from a table or query (ProductCategory in this case). To enter data into the SkiBoardStyle table, users select one of the items from the complete list, and the chosen value is entered into the Category column of the SkiBoardStyle table. In the screen of this wizard you specify the Category column of the SkiBoardStyle table as the place to hold the chosen value. The screen immediately prior to this one is where you select the key column in the display list—the Category column in the ProductCategory. The screen in Figure 6.10 is the one that generally causes the most problems. You must be sure to store the value in the appropriate column or the combo box will not work. Once the wizard has completed, you should set the properties of the combo box to give it a better name (cboCategory in this case), and double-check the tab order of all of the controls.

As shown in Figure 6.11, when you run the form, the datasheet rows appear similar to the original version. However, when you click on an entry in the Category column, you will see the arrow box for the combo box. Clicking on the arrow generates the list shown in the figure. Selecting an item transfers it to the form and the table. It is important that you understand the role of the combo box in relational databases. Remember that normalization splits the database into tables that are connected through primary and foreign keys. Generally, these keys are numeric, and often they are generated by the DBMS, so they have no overt business interpretation. It would be difficult for users to remember these numbers, but they have to be entered correctly into many tables. The combo box solves the usability problem by displaying a list of data. When an item is chosen, the matching key value is entered into the foreign key column automatically. With numeric keys, you generally want to hide the key; users do not even need to know that it exists.

Activity: Create Main Forms and Subforms

Action

Create a new form with the wizard.

Sale table, all columns.

Customer table use name, phone, and e-mail columns.

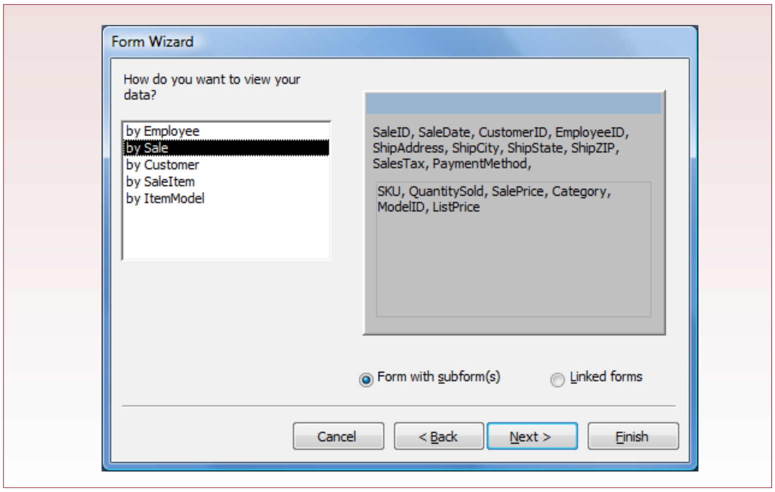
Employee table, use the name.

SaleItem table, all columns except SaleID.

ItemModel table, use Category, ModelID, and ListPrice.

Finish the wizard and test the form.

Now that you understand the main and grid forms, it is time to combine them into a main form and subform. Remember where this process began—with business forms, particularly the sale form. A typical business sale form has data for the sale (SaleID, SaleDate) and customer (name, address, and so on). It also has a section of repeating data to hold the specific items being purchased by the customer. This repeating section was split into the SaleItem table, with some elements placed in the Inventory and ItemModel tables. The purpose of the main form and subform is to recombine these tables. Keep in mind that each form can be associated directly with only one table. In this case, the sale form will



SaleID, SaleDate, CustomerID, EmployeeID, ShipAddress, ShipCity, ShipState, ShipZIP, SalesTax, PaymentMethod,

SKU, QuantitySold, SalePrice, Category, ModelID, ListPrice

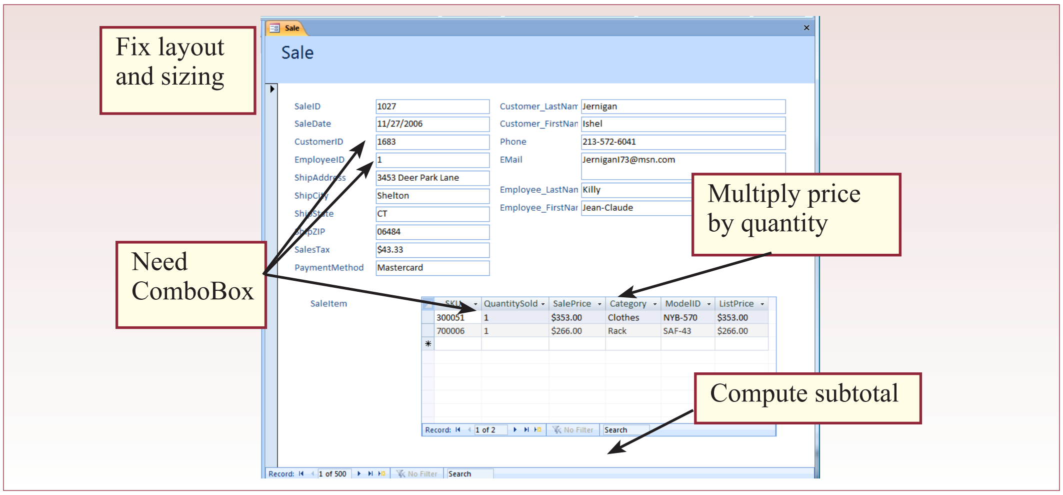
Figure 6.12

be based primarily on the Sale table, and the subform will be based primarily on the SaleItem table. Additional data from the other tables can be displayed on the forms, but only the primary keys from those two tables will be used.

> **Action**
>
> Switch to Design view/
>
> Replace CustomerID, EmployeeID, and PaymentMethod with combo boxes.
>
> Use the Row Source property of the combo boxes to set a sort order.
>
> Set tab order and run the form.

The wizard is again used to start the forms. Begin by selecting all of the columns from the Sale table. Then select the LastName, FirstName, Phone, and Email columns from the Customer table. Remember, do not use the Customer.CustomerID column. Also, you should add the Employee LastName and FirstName so they can be displayed on the form for the salesperson. Next, use all of the columns from the SaleItem table except the SaleID column, because you do not want to have it repeated on every row of the subform. From the ItemModel table, bring the Category, ModelID, and ListPrice columns so they can be displayed as a description on each row of the subform.

Figure 6.13


Fix layout and sizing
Need ComboBox
Multiply price by quantity
Compute subtotal
Sale
Record: 1 of 500 No Filter Search
Record: 1 of 2 No Filter Search

Sale

SaleID	1027
SaleDate	11/27/2006
CustomerID	1683
EmployeeID	1
ShipAddress	3453 Deer Park Lane
ShipCity	Shelton
ShipState	CT
ShipZIP	06484
SalesTax	$43.33
PaymentMethod	Mastercard
Customer_LastNam	Jernigan
Customer_FirstNar	Ishel
Phone	213-572-6041
EMail	JerniganI73@msn.com
Employee_LastNan	Killy
Employee_FirstNar	Jean-Claude

SaleItem

SKU	QuantitySold	SalePrice	Category	ModelID	ListPrice
300051	1	$353.00	Clothes	NYB-570	$353.00
700006	1	$266.00	Rack	SAF-43	$266.00

Action

Open the sub form in Design view.

Insert a combo box for the SKU.

Add a text box to compute Value=QuantitySold*SalePrice.

Add a text box to the footer to compute the subtotal: =Sum(QuantitySold*SalePrice).

Set tab order.

Test the sub form calculations.

Open the main form in Design view.

Add subtotal text box and use the expression builder to copy the subtotal value from the subfrom subtotal.

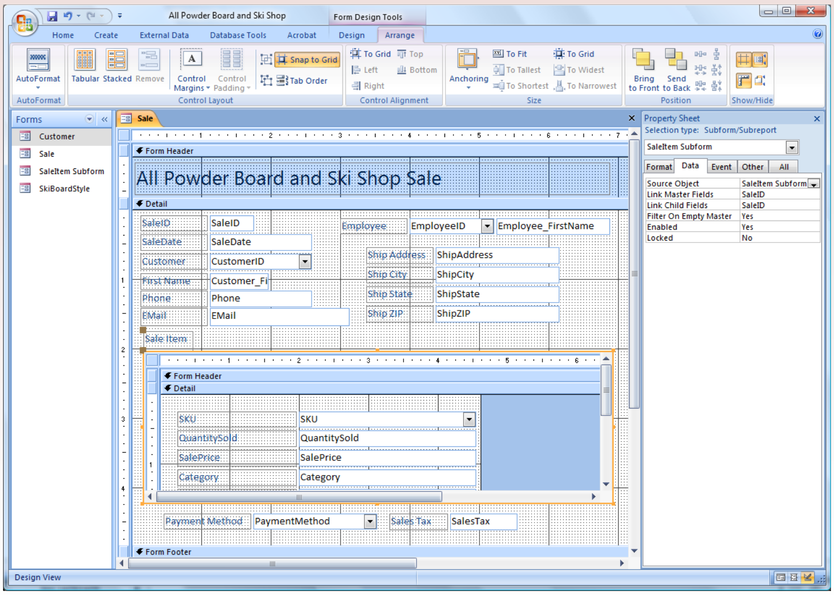
Test the form.

The next step in the wizard is to make sure that it recognizes the main/subform relationship by Sale. Figure 6.12 shows the step where you can tell the wizard how to organize the form layout. If your database relationships are correct, it generally picks the correct layout, but sometimes you have to override this choice. Here, the Sale is the main form, and the repeating sections are moved to the subform. You can use the default options for the other choices in the wizard.

The initial form generated by the wizard will work, but it usually needs some work. You will have to fix the layout, the widths of the text boxes, and add combo boxes. For subforms that contain numeric data (price and quantity), you will also want to compute the subtotal of the value and display it on the main form. Figure 6.13 shows the initial form. Before trying to improve the form, take a minute to run the form and see how the subform works. Notice that as you switch to a different sale, the subform picks up the items for just that sale. The subform is linked to the main form through the SaleID, and the form processor knows how to retrieve and display just the matching data.

To improve the form, begin by replacing the foreign key ID text boxes with combo boxes. Make sure that you assign a new name using the combo box properties. Note that all of the controls on the form are contained within a layout group. When you delete a textbox, the others immediately move to cover the space. Do not worry about it yet. Simply add the combo box and enter the details. When fin-

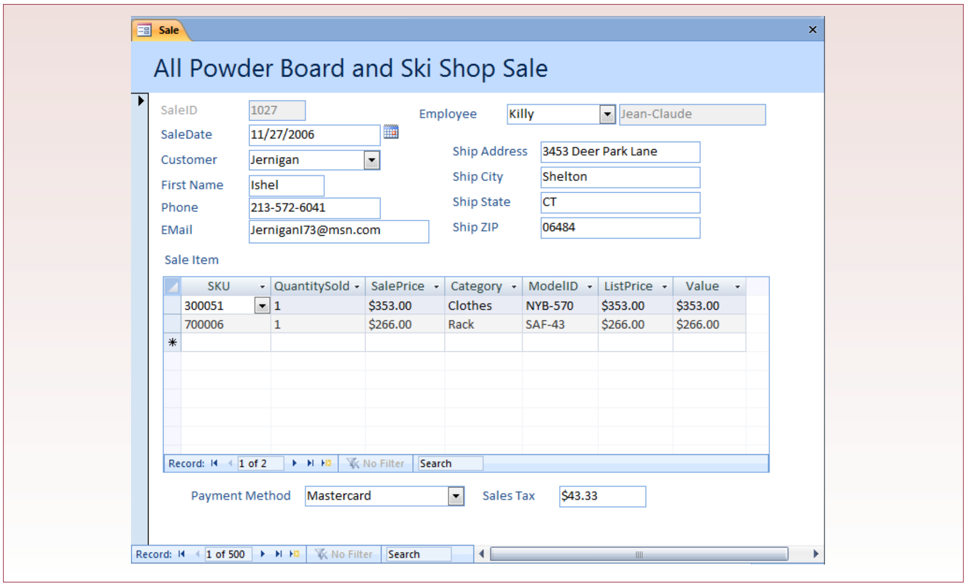
Figure 6.14



Property Sheet
Selection type: Subform/Subreport
SaleItem Subform

Format | Data | Event | Other | All

Source Object	SaleItem Subform
Link Master Fields	SaleID
Link Child Fields	SaleID
Filter On Empty Master	Yes
Enabled	Yes
Locked	No



Sale

All Powder Board and Ski Shop Sale

SaleID	1027	Employee	Killy / Jean-Claude
SaleDate	11/27/2006	Ship Address	3453 Deer Park Lane
Customer	Jernigan	Ship City	Shelton
First Name	Ishel	Ship State	CT
Phone	213-572-6041	Ship ZIP	06484
EMail	JerniganI73@msn.com		

Sale Item

SKU	QuantitySold	SalePrice	Category	ModelID	ListPrice	Value
300051	1	$353.00	Clothes	NYB-570	$353.00	$353.00
700006	1	$266.00	Rack	SAF-43	$266.00	$266.00

Record: 1 of 2 No Filter Search

Payment Method Mastercard Sales Tax $43.33

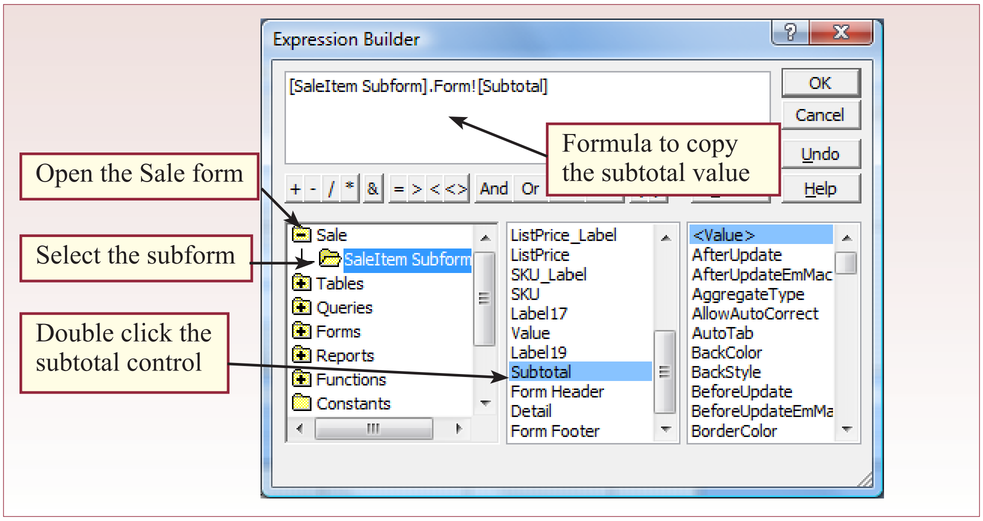
Record: 1 of 500 No Filter Search

Figure 6.15

ished, you can drag the combo box to the desired location until it gets joined into the layout group. However, the layout group makes it difficult to arrange the items aesthetically. For more flexibility, you should remove the controls from the layout group. Simply select them and use right-click to choose Layout/Remove. You can select all of them and then use right-click to save time. You should now be able to drag the controls on the main form to better locations.

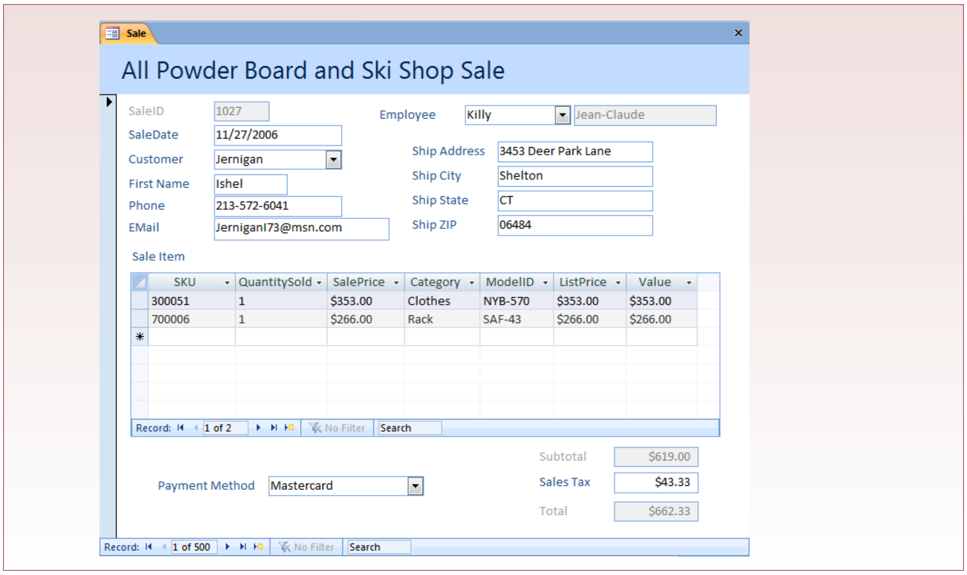
Figure 6.14 gives you an idea of how to rearrange and resize the main form controls. Be sure that you set the tab order. You also might consider adding command buttons to delete and add sales. Also, set the default value for the SaleDate to be =Date( ) so that the current day will always be entered for new sales. Run the form and check that the combo boxes work correctly and that you sorted the data correctly when you created them. If you need to adjust them, you can simply de-

Figure 6.16



Expression Builder

[SaleItem Subform].Form![Subtotal]


Sale

All Powder Board and Ski Shop Sale

SaleID	1027	Employee	Killy / Jean-Claude
SaleDate	11/27/2006	Ship Address	3453 Deer Park Lane
Customer	Jernigan	Ship City	Shelton
First Name	Ishel	Ship State	CT
Phone	213-572-6041	Ship ZIP	06484
EMail	JerniganI73@msn.com		

Sale Item

SKU	QuantitySold	SalePrice	Category	ModelID	ListPrice	Value
300051	1	$353.00	Clothes	NYB-570	$353.00	$353.00
700006	1	$266.00	Rack	SAF-43	$266.00	$266.00

Record: 1 of 2 No Filter Search

Payment Method	Mastercard
Subtotal	$619.00
Sales Tax	$43.33
Total	$662.33

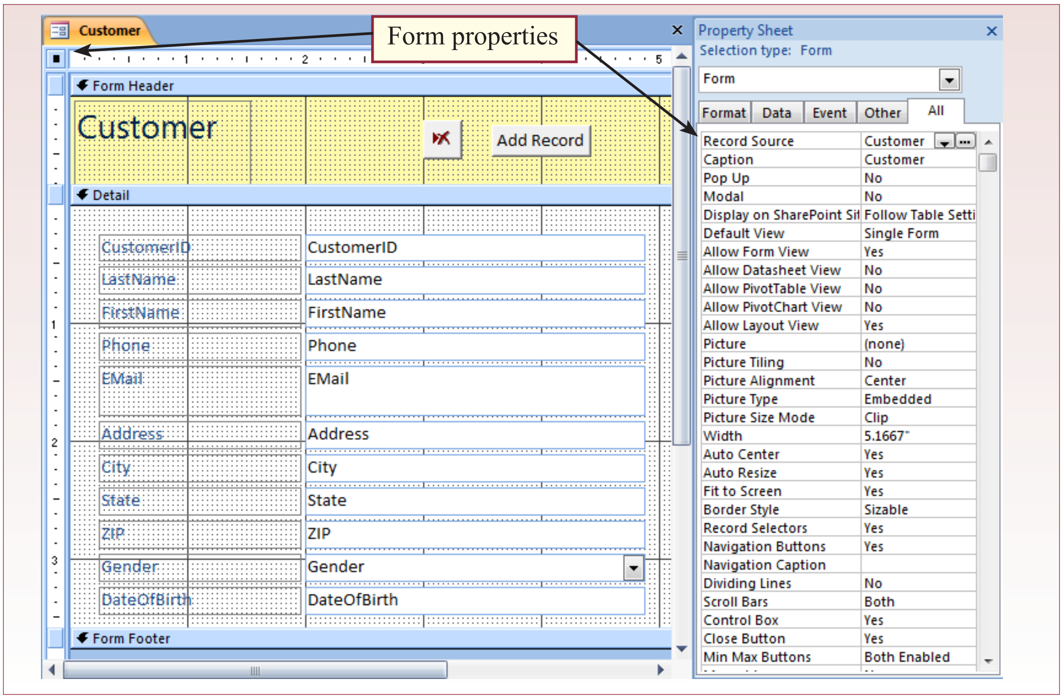
Record: 1 of 500 No Filter Search

Figure 6.17

lete one and start over. Or, adjust the properties for the combo box. For example, the Row Source property contains a SELECT query and you can edit it by adding an ORDER BY LastName, FirstName to the end of the query. You can use the Ellipses button (…) to open the query in Design view to add the sort order.

The next step is to improve the subform. It is a little challenging because it is difficult to get all of the columns to fit horizontally on the form. You might have to widen the form and the subform box. You set the column widths while the form is running by dragging or double-clicking the column dividing lines. You can also use the layout view to help align items while the form is displaying data. The next changes are often easier if you close the main form and open the subform alone in Design view. First, you should replace the SKU box with a combo box to minimize data entry errors. Next, add a text box to the bottom of the form, and enter "= SalePrice * QuantitySold" as the Control Source. Give it the name "Value" in its properties. Next, expand the form footer and add a new text box to it. It's formula should be "= Sum(SalePrice * QuantitySold)", and give it the name "Subtotal." Format them both as Currency with two decimal places. Figure 6.15 shows how the forms will appear when you run the main Sale form.

Notice that the subtotal is not displayed on the main form. To get it there, you need to copy its value from the subform. Return to the main form in Design view and add a text box beneath the subform. The copy is made dynamically by using a formula, but the formula is slightly complex. Instead of trying to memorize the formula syntax, you can use the Expression Builder to create the formula for you. On the properties for the Sale form Subtotal text box, click the Ellipses button to open the Expression Builder. Figure 6.16 shows the rest of the steps. Open the Sale form and select the subform to see its controls. Find the Subtotal control and double-click it. The proper formula will be entered into the edit box. Click the OK button to return to the properties. Set the format to Currency and run the form to test it. You should also add one more text box to compute the total. Enter the formula as "= Subtotal + SalesTax" and set its format to currency.



Record Source	Customer
Caption	Customer
Pop Up	No
Modal	No
Display on SharePoint Sit	Follow Table Setti
Default View	Single Form
Allow Form View	Yes
Allow Datasheet View	No
Allow PivotTable View	No
Allow PivotChart View	No
Allow Layout View	Yes
Picture	(none)
Picture Tiling	No
Picture Alignment	Center
Picture Type	Embedded
Picture Size Mode	Clip
Width	5.1667"
Auto Center	Yes
Auto Resize	Yes
Fit to Screen	Yes
Border Style	Sizable
Record Selectors	Yes
Navigation Buttons	Yes
Navigation Caption	
Dividing Lines	No
Scroll Bars	Both
Control Box	Yes
Close Button	Yes
Min Max Buttons	Both Enabled

Figure 6.18

Figure 6.17 shows the final form. Your version should be similar to this one, but there is always room for flexibility within a design. Notice that some of the text boxes are disabled to prevent clerks from changing the data. In particular, the SaleID and totals are generated by the system and should not be modified by the users. The controls have an Enabled property that you can set to False.

Activity: Customize Form Properties

Action

Make a copy of the Customer form.

Set the background color of the header to yellow.

The form wizards are easy to use and do a decent job. However, you eventually will want to customize the forms to improve the layout or add more features. In some cases, you might need to make major changes to the data connections, or you might even want to build a form from scratch. Properties are the key to all of these issues. Once you know where to look, you will see that the wizards simply take the answers to your questions and assign values to properties on the form. Once you understand these various properties, you can edit them yourself to obtain more control over the forms.

Figure 6.18 shows the basic Customer form in design view. Right-click the small square in the upper left-hand corner of the form and choose the Properties option to open the Properties window that is also displayed. The Record Source property is one of the most important properties affecting the overall form. This property defines the data that will be displayed and edited on the form. Currently, it is set to the simple Customer table. If you select that property, you will see a small ellipses button (…). Click that button and the designer will open a query edit window. You can add items to the query or you can sort the data to control the order in which it will be displayed. Be careful with adding tables to the query—

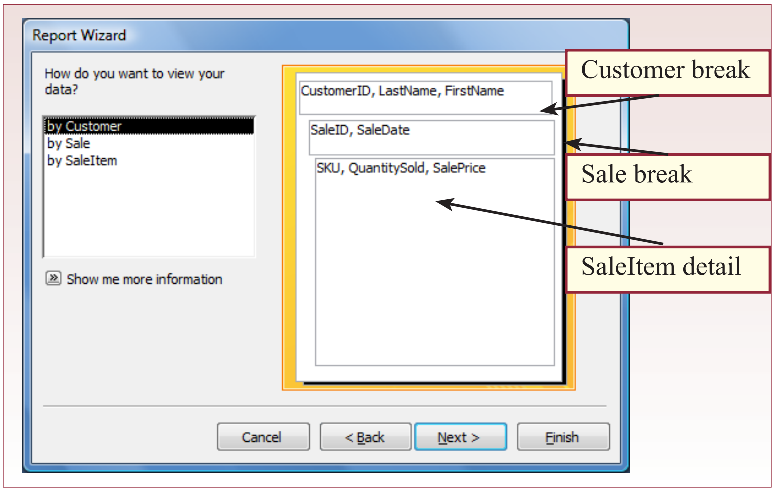
some queries are not updateable, and the form will no longer be able to collect data. However, this trick is useful when you want to display additional data on a form. The Sale form is a common example. The main Sale form would be based on the Sale table, but you could also include the customer name and phone number columns from the Customer table.

Once you modify the data source, the form will have access to the additional columns. However, they will not be displayed on the form. There are two ways to add text boxes to a form: (1) You can click the Text Box icon in the Toolbox and click a location on the form, or (2) You can open the Field List box on the forms toolbar and drag the desired column onto the form. The advantage of the second approach is that it automatically applies the appropriate styles if you are using a template. If you use the first method, you will have to set the text box display properties yourself. However, the Format Painter simplifies the task. Simply click a text box that has the properties you want, click the paint brush in the toolbar, and finish by clicking the new text box to paint it with those properties. To get more information about any property, simple select it in the Property window and press the F1 key for help. Using the form's Record Source, and the text box (or List Box) you can build a form from scratch and customize it anyway you want. However, it takes considerable time. Remember to use the Format/Align commands to ensure everything lines up properly.

The form has three sections (header, detail, and footer), and you can set foreground and background colors separately for each section. Each control item on the form also has its own collection of properties. The properties vary depending on the type of control, but most have a Control Source property that links them to a specific column, along with visual properties such as fonts, location, and colors.

For practice with properties, in the Customer form set the background color of the header section to yellow. Right-click the header background and select Properties. Select the Background property, click the ellipses button and choose a color. Run the form to test it. Adjust the layout or colors to improve the appearance.

Figure 6.19



All Powder Basic Reports

Activity: Create Reports with Subtotals

Action

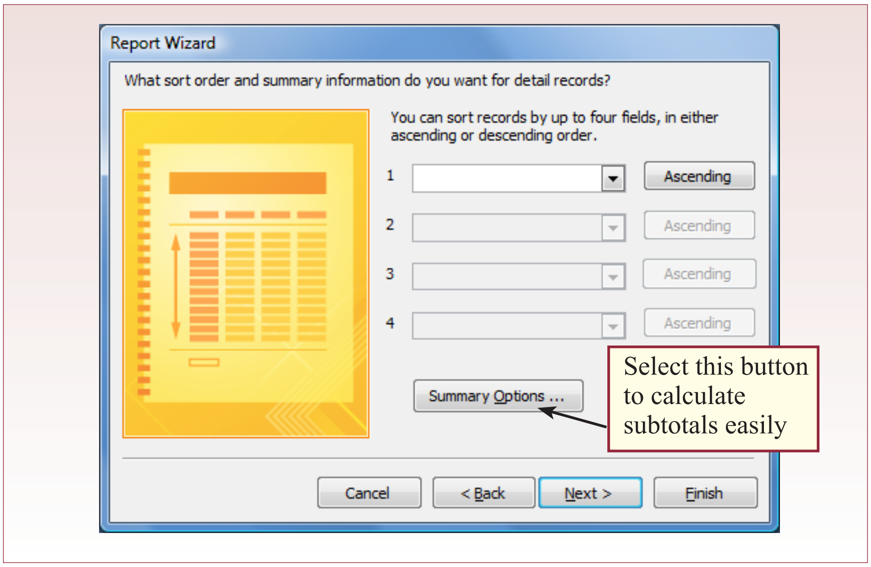
Create a new report with the wizard.
Customer table, use all columns.
Sale table, use all columns.
SaleItem table, use all columns.
Verify the group breaks.
Click the Summary Options button.
Compute totals for the Quantity.
Finish the wizard and switch to design view.

Most managers want reports so they can evaluate the progress of their business. Today, much of the business data could be displayed within forms—if the managers have sufficient access to the online system and if they are comfortable with reading the data on the screen instead of on paper. However, reports are also useful when managers need to see lists of items with subtotals. Remember that queries can print detailed data rows or summary totals, but not both at the same time. And query results are difficult to format. Instead, you want to use the Report Writer to format results, draw lines, and compute subtotals.

The first issue in building a report is to identify the level of detail that will be needed. The Report Writer can always compute subtotals across groups, but you need to ensure that your query retrieves the level of detail desired by the managers. As an example, consider a basic sales report by customer. Managers want to list each customer, followed by the sales placed by that customer. If they also want to include the individual items purchased on each sale, that level of detail is different than if they simply want to see the total value of the sale. For now, assume that they want to see the detailed item list.

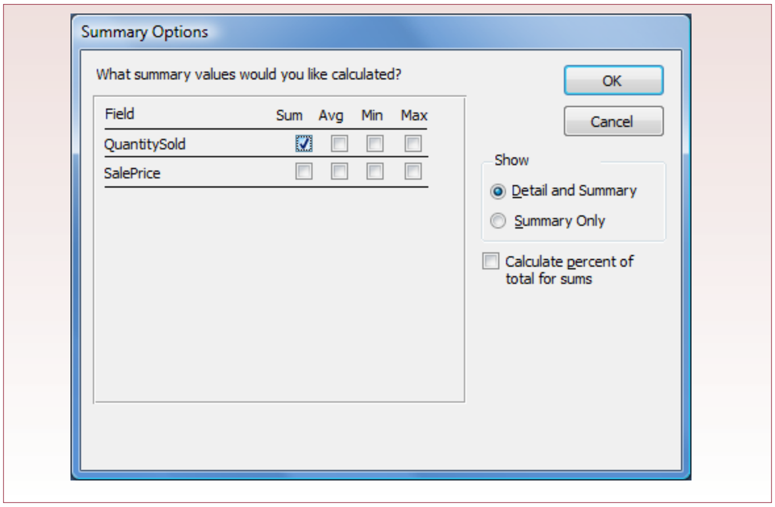
You can use the report wizard to build this report. Similar to building forms, your first step is to choose the tables and columns of data that you want on the report. In some ways, queries for reports are simpler because you do not have to worry about keeping the query updatable. However, the form wizard in Access

Figure 6.20


Report Wizard

What sort order and summary information do you want for detail records?

You can sort records by up to four fields, in either ascending or descending order.



Summary Options

What summary values would you like calculated?

Field	Sum	Avg	Min	Max
QuantitySold	☑	☐	☐	☐
SalePrice	☐	☐	☐	☐

Show

(●) Detail and Summary

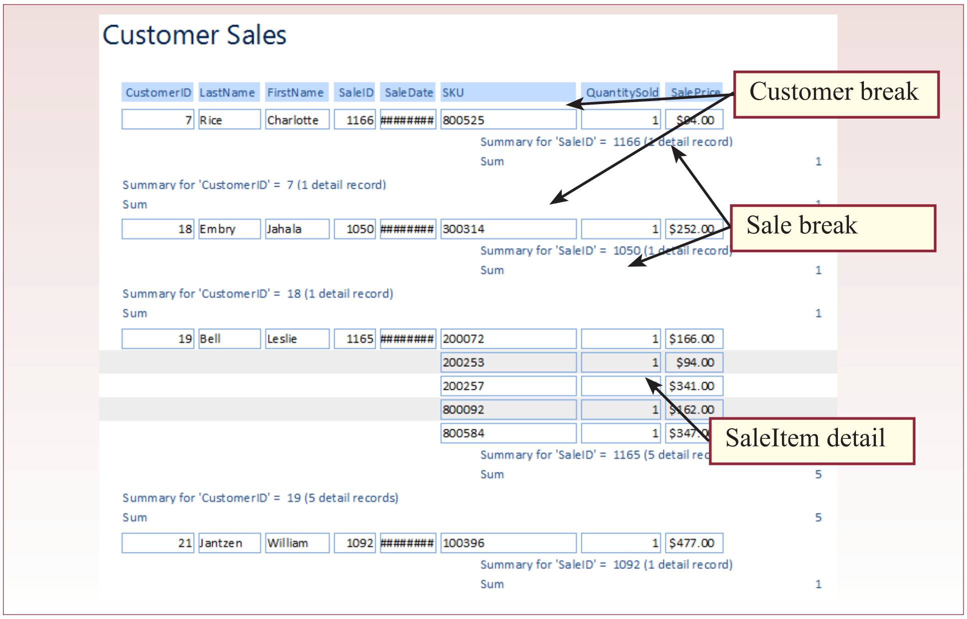
( ) Summary Only

☐ Calculate percent of total for sums

Figure 6.21

2007 also relies on layout groups. The results are unwieldy if you include too many columns. It is best to choose a small set of three or four columns (with primary keys) from each desired table. When the wizard has created the base form, you can add the other columns. In this case, take the CustomerID and name columns from the Customer table, the SaleID and SaleDate from the Sale table, and all but the SaleID table from the SaleItem table. If you want to display additional descriptions of the items, you would also have to include the Inventory and possibly ItemModel tables, but for now, leave those out. Figure 6.19 shows one of the more important screens of the report wizard. You need to make sure that the wizard picks up the breaks correctly, and that your detail section is correct.

Figure 6.22



Customer Sales

```
CustomerID LastName FirstName SaleID SaleDate SKU     QuantitySold SalePrice
         7 Rice     Charlotte   1166 ######## 800525             1   $84.00
                                 Summary for 'SaleID' =  1166 (1 detail record)
                                 Sum                                              1
Summary for 'CustomerID' =  7 (1 detail record)
Sum                                                                               1
        18 Embry    Jahala      1050 ######## 300314             1  $252.00
                                 Summary for 'SaleID' =  1050 (1 detail record)
                                 Sum                                              1
Summary for 'CustomerID' =  18 (1 detail record)
Sum                                                                               1
        19 Bell     Leslie      1165 ######## 200072             1  $166.00
                                          200253             1   $94.00
                                          200257                $341.00
                                          800092             1  $162.00
                                          800584             1  $347.00
                                 Summary for 'SaleID' =  1165 (5 detail records)
                                 Sum                                              5
Summary for 'CustomerID' =  19 (5 detail records)
Sum                                                                               5
        21 Jantzen  William     1092 ######## 100396             1  $477.00
                                 Summary for 'SaleID' =  1092 (1 detail record)
                                 Sum                                              1
```

> **Action**
>
> Move items around to improve the look.
>
> Add a text box in the Detail section called Value to compute [QuantitySold]*[SalePrice]
>
> Set currency format.
>
> Edit the Sum calculations to be Sum([QuantitySold]*[SalePrice].
>
> Edit the Record Source to add columns
>
> Run the form and clean up the layout.

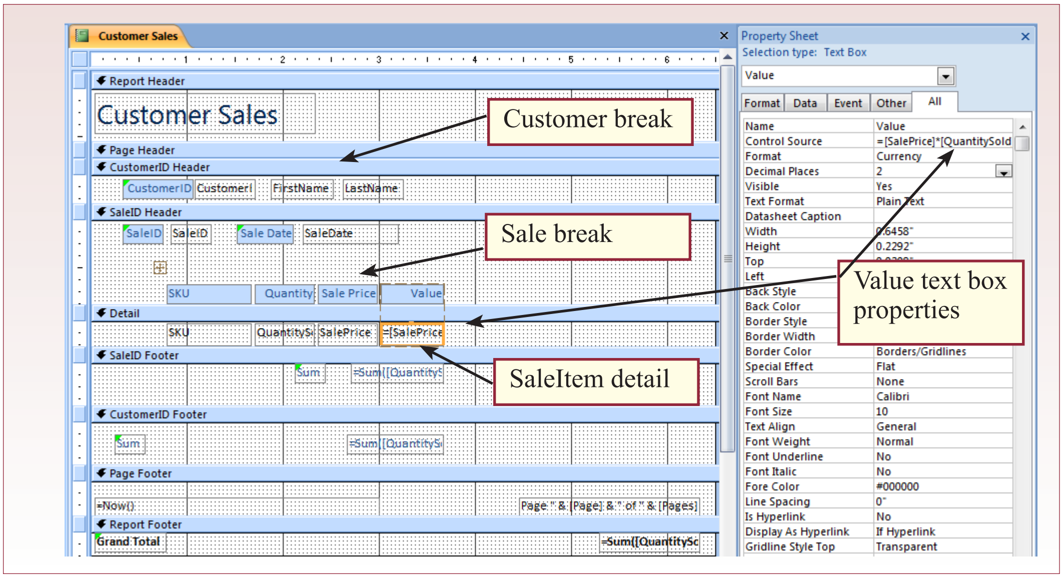
As shown in Figure 6.20, the next step is a little tricky. It is easy to miss the Summary Options button. Clicking this button gives you options shown in Figure 6.21 to automatically calculate subtotals. The button only appears when you have numeric columns that can be summed or averaged. You then select the columns to compute subtotals or averages.

In this particular example, it does not make sense to sum either of the two numeric columns (quantity and price). You really need to sum the value of price times quantity, which can be done later on the form. If you want to do it at this step, you need to base the report on a query that already computes a column to multiply the two values, then you can sum the resulting column. However, in this case, it is useful to temporarily sum the QuantitySold column and then modify the formula later.

Figure 6.22 shows the initial report created by the wizard. Notice that it has the three sections defined by the group breaks. However, even with the limited number of columns, the report is a little messy—largely because it relies on the layout groups. It also needs a column in the detail section to multiply price by quantity, and then compute the total in the footer sections. It would also be useful to split the output so that sales for each customer appear on separate pages.

The report can be cleaned up by switching to Design view. First, you have to move all controls inside the 6.5 inch margin and drag the right edge inside that limit so everything fits on one page. Second, you have to expand the header sections; then place columns into those sections and arrange them, instead of having them squeezed onto one line. For example, place the customer name in the Customer header. You will have to remove the label and the text box from the lay-

Figure 6.23



Name	Value
Control Source	=[SalePrice]*[QuantitySold
Format	Currency
Decimal Places	2
Visible	Yes
Text Format	Plain Text
Datasheet Caption	
Width	0.6458"
Height	0.2292"
Top	[illegible]
Left	
Back Style	
Back Color	
Border Style	
Border Width	
Border Color	Borders/Gridlines
Special Effect	Flat
Scroll Bars	None
Font Name	Calibri
Font Size	10
Text Align	General
Font Weight	Normal
Font Underline	No
Font Italic	No
Fore Color	#000000
Line Spacing	0"
Is Hyperlink	No
Display As Hyperlink	If Hyperlink
Gridline Style Top	Transparent

out group to move them. You might consider creating new layout groups within the header sections, but it is not necessary. You can move the labels from the Page Header to the Detail Header by using the Layout/Move Down a Section command.

Action

Create a new query to total sales by customer.

Save the query.

Create a new report based on the query.

For the Customer group, set the Keep Together property to Whole Group.

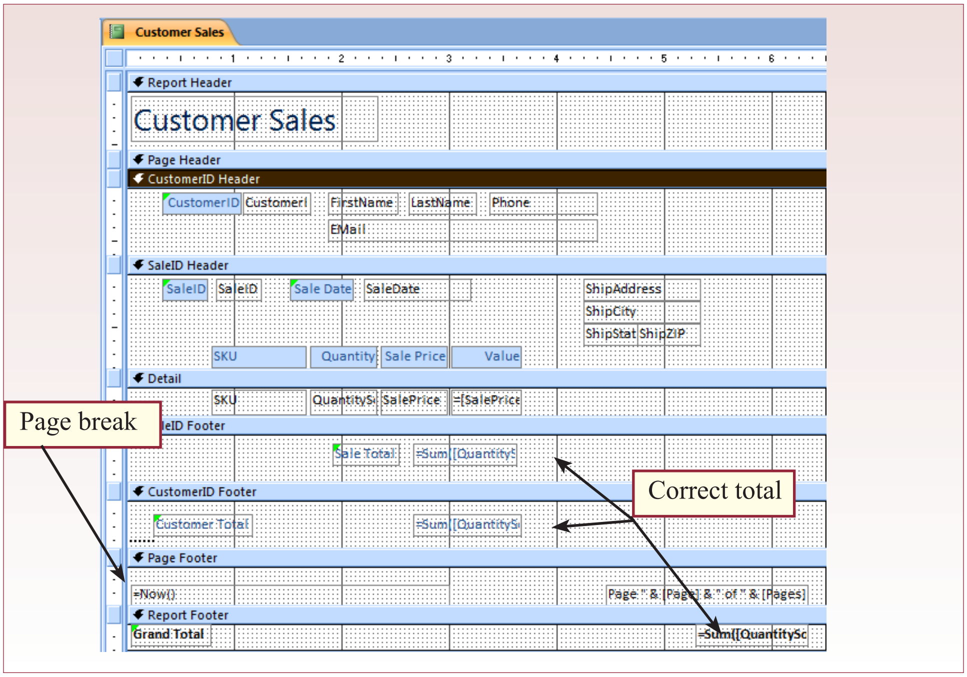
Run the report.

Figure 6.23 shows some of the ways to rearrange the form. Notice the sections that correspond to the group breaks. The Customer and Sale breaks have headers and footers. The titles are displayed in the headers, and the sum in the footers. The figure also shows the addition of the Value text box in the detail section. It computes QuantitySold * SalePrice to get the value of the items being purchased.

You can add more columns to the report by editing the record source for the report. Click the square in the top-left of the report. In the Property Sheet, click the ellipses button for the Record source to open the query editor. Add columns from the three tables that you want to display on the report, such as phone and e-mail from Customer, and shipping address from Sales. When you close the editor, save the changes to the property sheet. Switch the window to Add Existing Fields, and drag the new fields onto the report.

Figure 6.24 shows the final design for the Customer Sale form. Notice the cleaner layout of the controls. Also, notice the page break at the bottom of the Customer footer. It ensures that each new set of customer data will begin on a new page. The calculations for the sum have been corrected so that they total QuantitySold * SalePrice instead of just QuantitySold. The interesting aspect of the sums is that the formula for all three controls is exactly the same. The only difference

Figure 6.24



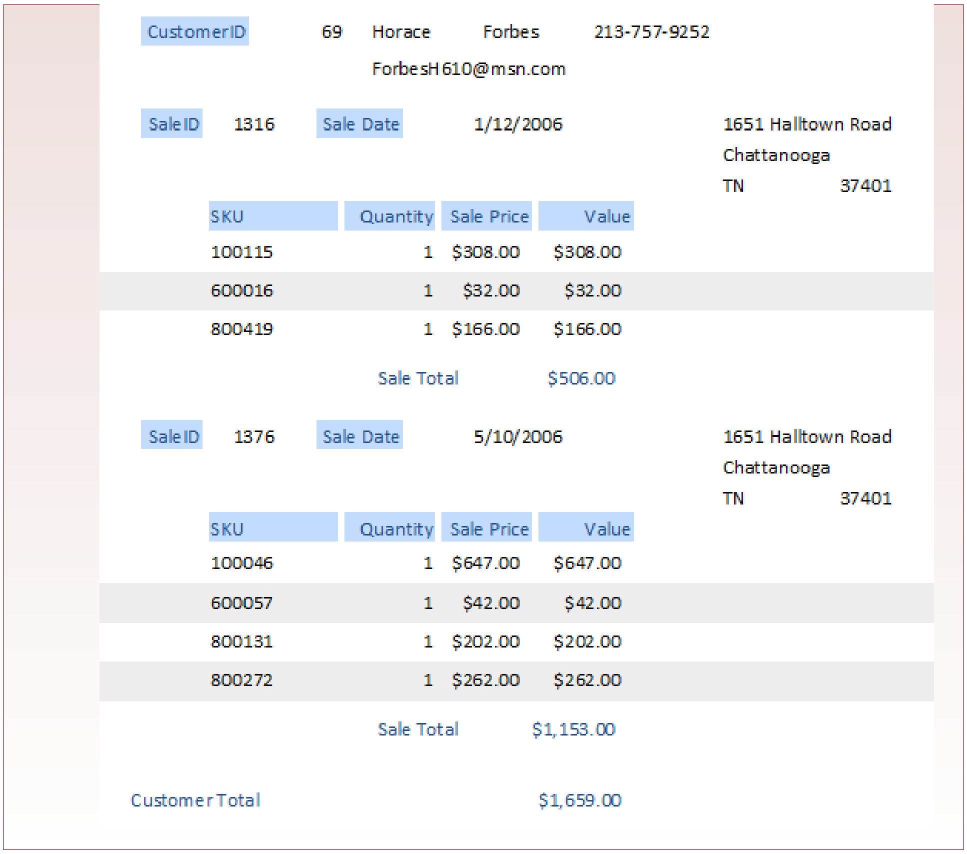
is the section on the report. The total on the Sale footer computes the sum just for that sale, while the total control on the Customer footer computes the total for the customer, across all sales for that customer. The total control on the Report footer computes and displays the total for all sales printed on the report.

Figure 6.25 shows a portion of the final version of the report. Notice that the page shows sales for only one customer. The Value calculation is correct, and the sums for the Sale and Customer are correct. Although the report will use a considerable number of pages, the details and summary values can be read easily. If the managers truly want this level of detail, the report should work well.

It is possible that managers do not need the detailed list of items sold. Perhaps they need only the total value of the sale for each customer. The report would certainly be shorter. Because this level of report requires a different level of detail, you will first have to create a query that extracts the main data and computes the sales value and the total value for each sale. The Report Writer will then compute the totals per customer.

Figure 6.26 shows the query used to create the total value of the items for each sale. The design is straightforward—add the desired columns from the Customer, Sale, and SaleItem table. Then add the computed column Value to compute SalePrice by QuantitySold. Save the query and build the report based on that query. Be sure that the wizard creates a group break based on the CustomerID. You can clean up the report in Design view. Move the columns for LastName, FirstName, Phone, and Email from the detail section to the Customer header section.

Figure 6.25



```
CustomerID        69   Horace    Forbes    213-757-9252
                       ForbesH610@msn.com

SaleID  1316   Sale Date     1/12/2006            1651 Halltown Road
                                                  Chattanooga
                                                  TN          37401
        SKU             Quantity  Sale Price     Value
        100115                 1   $308.00     $308.00
        600016                 1    $32.00      $32.00
        800419                 1   $166.00     $166.00

                       Sale Total              $506.00

SaleID  1376   Sale Date     5/10/2006            1651 Halltown Road
                                                  Chattanooga
                                                  TN          37401
        SKU             Quantity  Sale Price     Value
        100046                 1   $647.00     $647.00
        600057                 1    $42.00      $42.00
        800131                 1   $202.00     $202.00
        800272                 1   $262.00     $262.00

                       Sale Total            $1,153.00

CustomerTotal                                $1,659.00
```

```
SELECT Customer.CustomerID, Customer.LastName, Customer.FirstName,
Customer.Phone, Customer.EMail, Sale.SaleID, Sale.SaleDate, Sale.ShipCity,
Sale.ShipState, Sum([SalePrice]*[QuantitySold]) AS [Value]
FROM (Customer INNER JOIN Sale ON Customer.CustomerID = Sale.
CustomerID) INNER JOIN SaleItem ON Sale.SaleID = SaleItem.SaleID
GROUP BY Customer.CustomerID, Customer.LastName, Customer.FirstName,
Customer.Phone, Customer.EMail, Sale.SaleID, Sale.SaleDate, Sale.ShipCity,
Sale.ShipState
ORDER BY Customer.LastName, Customer.FirstName;
```

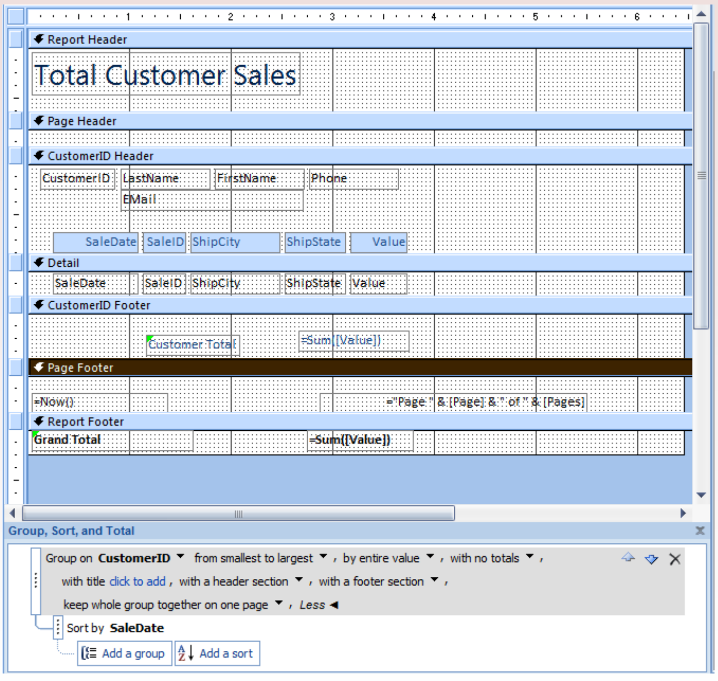
Figure 6.26

Figure 6.27 shows the Design view for the new report after it has been cleaned up. Clicking the Sorting and Grouping button on the tool bar brings up the options to control how the grouping is handled. In this case, for the CustomerID, set the Keep Together option to Whole Group. This value tells the Report Writer to avoid splitting customer data across pages. If data for a customer would be split by a page break, the Report Writer will move the entire group to a new page. These options are useful and can provide detailed control over the report layout. However, you have to be careful. If a customer has so many sales that the data will not fit on one page, the system will still be forced to spread the data across multiple pages. The options are particularly useful for relatively small groups of data.

Activity: Create Subreports

Access supports subreports, which are similar to subforms in that a repeating group can be displayed as a linked report within the main report. It is primarily useful when you need two or more repeating groups of data. For example, for each customer you might want to display a list of sales and a list of rentals.

Figure 6.27


Report Header
Total Customer Sales
Page Header
CustomerID Header
CustomerID LastName FirstName Phone
EMail
SaleDate SaleID ShipCity ShipState Value
Detail
SaleDate SaleID ShipCity ShipState Value
CustomerID Footer
Customer Total =Sum([Value])
Page Footer
=Now() ="Page " & [Page] & " of " & [Pages]
Report Footer
Grand Total =Sum([Value])

Group, Sort, and Total
Group on **CustomerID** from smallest to largest, by entire value, with no totals,
with title click to add, with a header section, with a footer section,
keep whole group together on one page, *Less*
Sort by **SaleDate**

Action

Create a simple report based on the Customer table.

Create a Sale sub-report from the Sale and SaleItem tables, grouping it by SaleItem..

Create a Rental sub-report from the Rental and RentalItem tables.

Open the Customer report in design view.

Drag-and-drop the two subreports onto the Detail section of the Customer report.

Use properties of the subreports to set the Link Child and Link Master Fields to CustomerID.

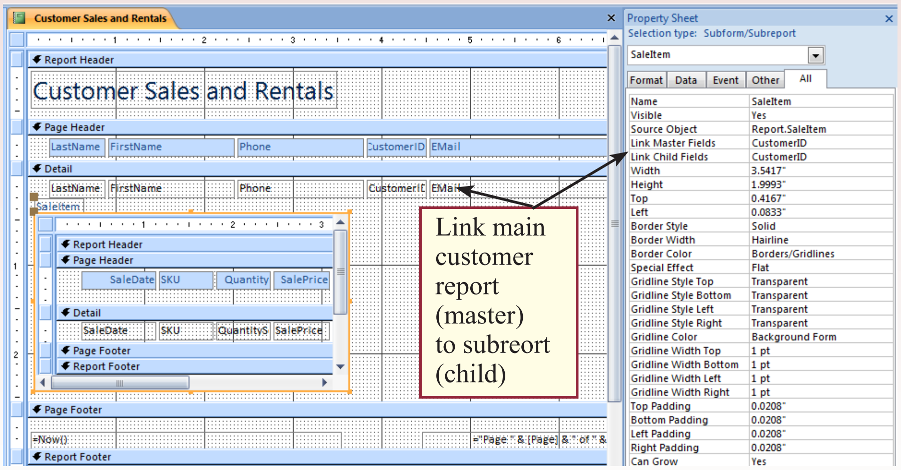
Run the report.

With subreports, you can display the lists side by side. The two subreports are linked by CustomerID to the main customer group, but not to each other since sales and rentals are not necessarily related.

The wizard does much of the work, but not all of it. This report needs to be built in three sections: (1) the Customer report, (2) the Sales subreport, and (3) the Rentals subreport. You can use the wizard to build each report, and then drag the subreports onto the main Customer report. Begin by creating the Customer report with some of the basic data from the Customer table. Then build the Sale subreport with the SaleID, CustomerID, and SaleDate from the Sale table and the columns from the SaleItem table. Group the report by SaleItem instead of the Sale table. Third, build the Rental subreport in a simlar manner using the RentalID, CustomerID, and RentalDate from the Rental table, along with the columns from the RentalItem table.

To build the final report, open the new Customer report in Design view. Expand the detail section so it has room to hold the subreports. Drag the SaleItem subreport onto the detail section and resize it horizontally and vertically so it fits on the page. On the subreport, delete and shrink any report and page header and footer sections. Add a text box to compute and display the value = SalePrice * QuantitySold, and resize the width of the lines and the page so you will be able to fit the two reports on the page. To link the subreport to the main report, you need to set the link properties on the subreport. Carefully, right-click on the border of

Figure 6.28



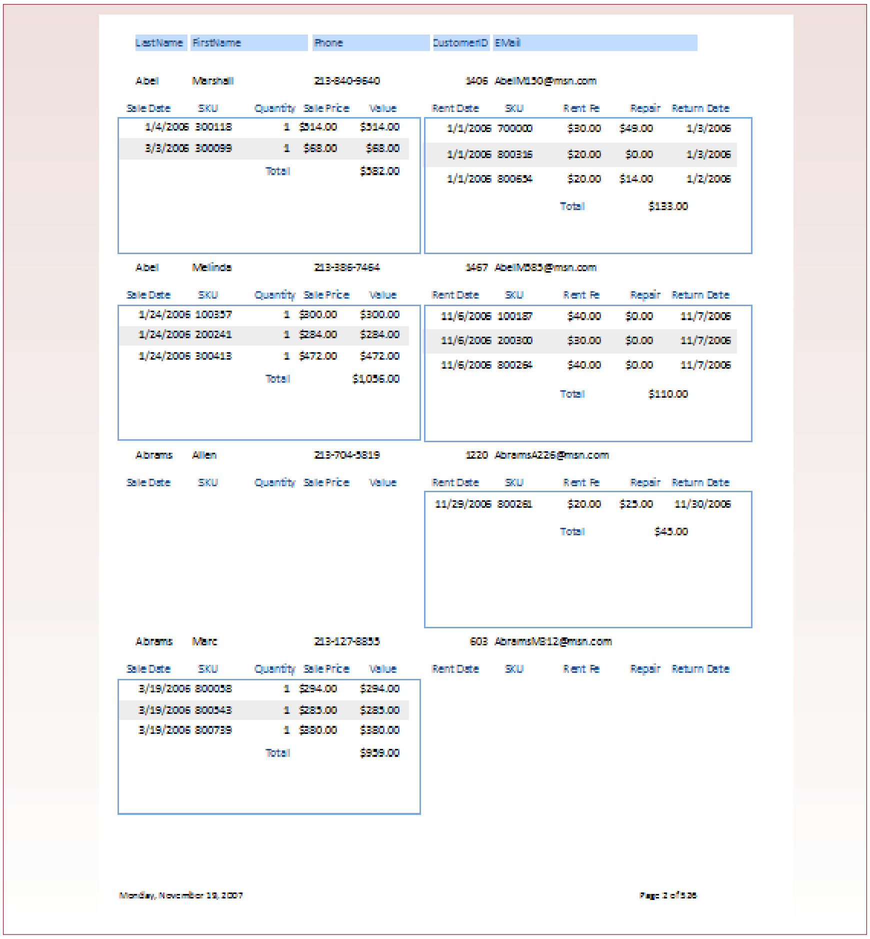
Property Sheet

Selection type: Subform/Subreport

SaleItem

Format | Data | Event | Other | All

Name	SaleItem
Visible	Yes
Source Object	Report.SaleItem
Link Master Fields	CustomerID
Link Child Fields	CustomerID
Width	3.5417"
Height	1.9993"
Top	0.4167"
Left	0.0833"
Border Style	Solid
Border Width	Hairline
Border Color	Borders/Gridlines
Special Effect	Flat
Gridline Style Top	Transparent
Gridline Style Bottom	Transparent
Gridline Style Left	Transparent
Gridline Style Right	Transparent
Gridline Color	Background Form
Gridline Width Top	1 pt
Gridline Width Bottom	1 pt
Gridline Width Left	1 pt
Gridline Width Right	1 pt
Top Padding	0.0208"
Bottom Padding	0.0208"
Left Padding	0.0208"
Right Padding	0.0208"
Can Grow	Yes



```
LastName  FirstName          Phone               CustomerID  EMail

Abel      Marshall           213-840-9640              1406  AbelM150@msn.com

Sale Date    SKU     Quantity  Sale Price    Value   Rent Date    SKU      Rent Fe   Repair   Return Date
 1/4/2006  300118           1     $314.00  $314.00    1/1/2006  700000      $30.00   $49.00      1/3/2006
 3/3/2006  300099           1      $68.00   $68.00    1/1/2006  800316      $20.00    $0.00      1/3/2006
                        Total              $382.00    1/1/2006  800654      $20.00   $14.00      1/2/2006
                                                                       Total        $133.00

Abel      Melinda            213-386-7464              1467  AbelM583@msn.com

Sale Date    SKU     Quantity  Sale Price    Value   Rent Date    SKU      Rent Fe   Repair   Return Date
1/24/2006  100357           1     $300.00  $300.00   11/6/2006  100187      $40.00    $0.00     11/7/2006
1/24/2006  200241           1     $284.00  $284.00   11/6/2006  200300      $30.00    $0.00     11/7/2006
1/24/2006  300413           1     $472.00  $472.00   11/6/2006  800264      $40.00    $0.00     11/7/2006
                        Total            $1,056.00                     Total        $110.00

Abrams    Allen              213-704-3819              1220  AbramsA226@msn.com

Sale Date    SKU     Quantity  Sale Price    Value   Rent Date    SKU      Rent Fe   Repair   Return Date
                                                    11/29/2006  800261      $20.00   $25.00    11/30/2006
                                                                       Total         $45.00

Abrams    Marc               213-127-8835               603  AbramsM312@msn.com

Sale Date    SKU     Quantity  Sale Price    Value   Rent Date    SKU      Rent Fe   Repair   Return Date
3/19/2006  800058           1     $294.00  $294.00
3/19/2006  800343           1     $285.00  $285.00
3/19/2006  800739           1     $380.00  $380.00
                        Total              $959.00

Monday, November 19, 2007                                                                  Page 2 of 536
```

Figure 6.29

the subreport and open the property list. Then set the master and child properties to CustomerID, which tells the subreport (child) to display only the data for the matching customer. Figure 6.28 shows the layout of the form and the property values needed to link the forms. The next step is to add the subreport for rentals following the same process. Be sure to set the link master and child properties. You will also want to rearrange the text boxes so that the two subreports fit side by side on one page. You should also consider adding a border around each of the subreports to highlight them on the main report. You will probably want to add totals on each of the subreport report footers.

Figure 6.29 shows a sample page from the new report. The subreport also supports properties to control automatically expanding the report to handle additional rows, or autosizing to restrict it if there are only a few rows in the report. You also might want to include a page break at the bottom of the detail section so that each customer report is printed on a separate page, but that decision will depend on the sizes of the subreports, which depend on the number of items purchased or rented. One of the most time-consuming elements in creating reports is setting the layout, sizing, and design elements to make the report compact, yet readable. The main aspect of this report is the dual column layout to show the separate sale and rental items.

Exercises

Crystal Tigers

The Crystal Tigers club is mostly interested in tracking members and events. The officers who will use the system do not know much about computers, but they can enter data into forms. They are also interested in a few key reports. For instance, they want to be able to get totals for the amount of hours their members devoted to charity events. They also want monthly summaries of the amount of money raised. The vice president also wants to be able to print a simple listing of the officers with their phone numbers and e-mail addresses. Sometimes she also wants a similar list of members who have participated in the initial steps of an event. She wants to be able to carry the list with her when the event starts so she knows who to contact if problems arise.

1. Create the basic forms needed to enter data into the database.
2. Build a form similar to the one defined in Chapter 2.
3. Create the main reports needed by the organization.

Capitol Artists

Job tracking is the most important aspect of the application needed by Capitol Artists. In particular, the employees need to be able to quickly select a job and enter the time and expenses for the task performed. This data is then used to create a monthly billing report for the client. Consequently, you need to focus on creating the forms to capture this data. You need to make sure they are fast and easy to use. The managers also want weekly reports showing the hours and money generated by each employee so they can use the data for personnel evaluations.

1. Create the basic forms needed to enter data into the database.
2. Build a form similar to the one defined in Chapter 2.
3. Create the main reports needed by the managers.

Offshore Speed

Special orders have always been a complex problem for the Offshore Speed managers. Customers come to the shop because it is one of the few that can obtain the custom parts they want. But the company has always had problems training employees to collect all of the order data and to keep track of getting the orders placed and delivered in a timely manner. Some of these orders include contracts with other local firms to perform customization and finish work on the boats. Although these firms do excellent work, most are terrible at keeping records. Consequently, the managers want to use the system to generate reports on individual boats for each contract shop that can be used to remind the other owners of the details. The company also needs reports on the inventory status of the specialized parts. They have trouble keeping some items in stock, and other items seem to sit on the shelves forever. They have no good way of keeping track at the moment.

1. Create the basic forms needed to enter data into the database.
2. Build a form similar to the one defined in Chapter 2.
3. Create the main reports needed by the managers.

Final Project

The main textbook has an online appendix with several longer case studies. You should be able to work on one of these cases throughout the term. If you or your instructor picks one, perform the following tasks.

1. Create the main forms needed for the database, including forms that will be used by administrators.
2. Build the forms similar to the ones used to define the project. That is, build database forms that match the existing user forms.
3. Create the main reports needed. Think about the analysis that managers will want to do and provide reports that help them. Consider adding charts to compare data.

Chapter 7

Database Integrity and Transactions

Chapter Outline

Objectives

- Define customized functions.
- Improve forms by responding to form events.
- Execute customized SQL statements from code.
- Define transactions.
- Create new rows and use the generated key value.
- Write cursor-based programs that compare data across rows.
- Set up and handle optimistic and pessimistic locking conditions.

Program Code in Microsoft Access

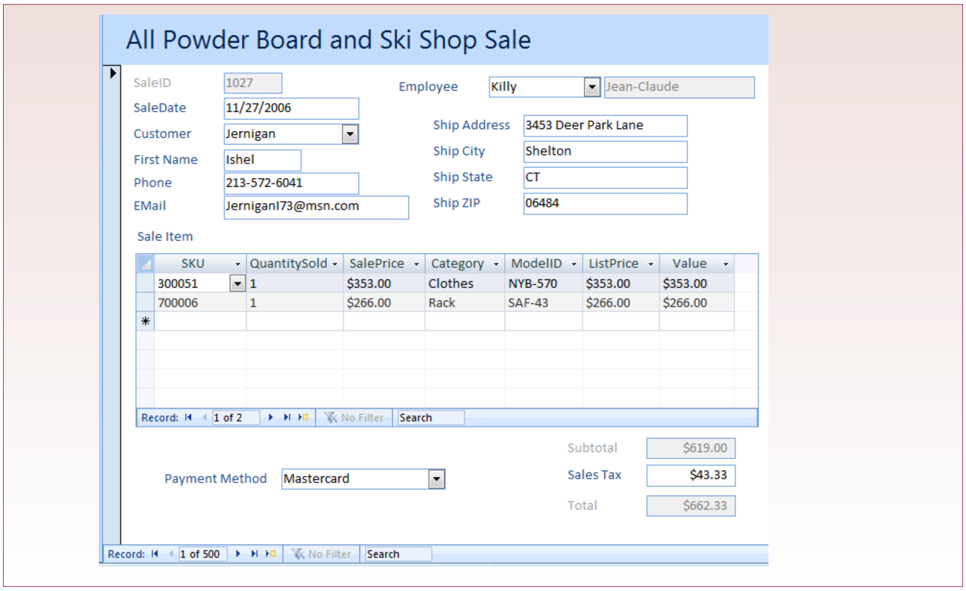
Access supports code modules that contain functions that you can use throughout your application. Access also supports detailed code behind each form that is activated in response to events on the form. However, Access does not support data triggers that attach to database events. Because of this emphasis on forms, data integrity code, locking, transactions, and key generation are handled within the necessary forms. For instance, you will generally have to write code within a sales form to handle transactions and data locking. In some ways, this approach is easier to understand because you write the code within each form. On the other hand, similar code might have to be written several times, and it is up to each application developer to ensure that problems are handled correctly in every form.

To understand how the code and event models work, this chapter begins with some easy examples. Pay close attention to the code and where it is located. For example, code written in a module can be accessed throughout the application, but code written within a form is generally only called in response to events on that form.

Case: All Powder Board and Ski Shop

Figure 7.1 shows the Sales form developed in the last chapter. Notice that it has a box to enter the sales tax. If you look at the underlying Sale table, you will see that it contains a column to hold the sales tax amount for each sale. You could argue that the sales tax does not have to be stored, since it can always be computed from the other sales data. But what happens if the tax rate changes? Or, what if the round-off computation is modified? Then the company's sales tax records will no longer exactly match the data filed with the state and local governments. It is safer to store the actual tax amount collected to ensure consistency. However, now you need a method to compute the sales tax on each sale—you certainly cannot expect clerks to compute the amount or even to look it up correctly in a table. Instead, you need to write a function that will compute the sales tax correctly and transfer

Figure 7.1



All Powder Board and Ski Shop Sale

SaleID	1027	Employee	Killy / Jean-Claude
SaleDate	11/27/2006	Ship Address	3453 Deer Park Lane
Customer	Jernigan	Ship City	Shelton
First Name	Ishel	Ship State	CT
Phone	213-572-6041	Ship ZIP	06484
EMail	Jernigan173@msn.com		

Sale Item

SKU	QuantitySold	SalePrice	Category	ModelID	ListPrice	Value
300051	1	$353.00	Clothes	NYB-570	$353.00	$353.00
700006	1	$266.00	Rack	SAF-43	$266.00	$266.00

Record: 1 of 2 No Filter Search

Payment Method	Mastercard
Subtotal	$619.00
Sales Tax	$43.33
Total	$662.33

Record: 1 of 500 No Filter Search

it to the form and the database. Sales taxes can be highly complex. Some items are taxable, while others are not. Since each state and local district is different (and there are several thousand tax districts in the U.S. alone), this presentation is simplified and assumes a single tax rate to be applied to all sale and rental items.

The first question you must answer when creating custom code is to determine where it belongs. In this example, you might consider putting it on the Sales form, but since the code will also be useful for rentals, it makes more sense to generalize it and place it in a module so that it is available to any form, query, or report within the application. Placing the code in a special module also makes it easier to find later.

Lab Exercise

All Powder Board and Ski Data

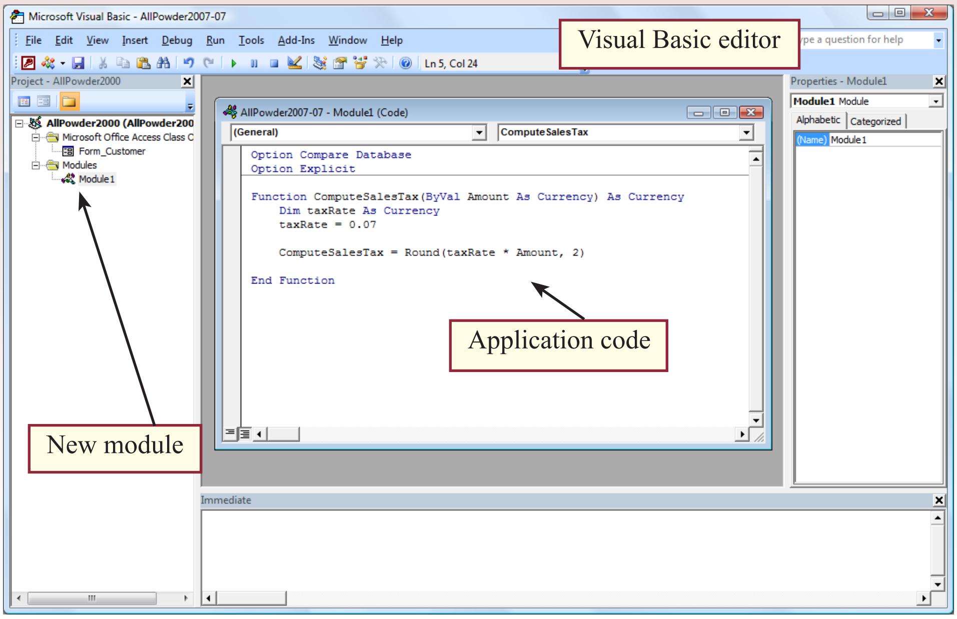
Most forms require some amount of programming. Code is used to improve data integrity and to make the forms easier to use. Functions and modules are used to centralize code so it is easy to find and change elements later.

Activity: Create Sales Tax Function

Begin by creating a new module, which brings up the Visual Basic editor. Before defining the function to create the sales tax, add the command Option Explicit at the top of the module. This option forces you to declare (Dim) all variables that you use, and helps spot typographical errors. Figure 7.2 shows that the function is a simple calcula-

Action
Create/Other/Macro/Module
At the top add: Option Explicit.
Add the function ComputeSalesTax.
Use Debug/Compile to find errors.
Close the module.

Figure 7.2



```
Option Compare Database
Option Explicit

Function ComputeSalesTax(ByVal Amount As Currency) As Currency
    Dim taxRate As Currency
    taxRate = 0.07

    ComputeSalesTax = Round(taxRate * Amount, 2)

End Function
```

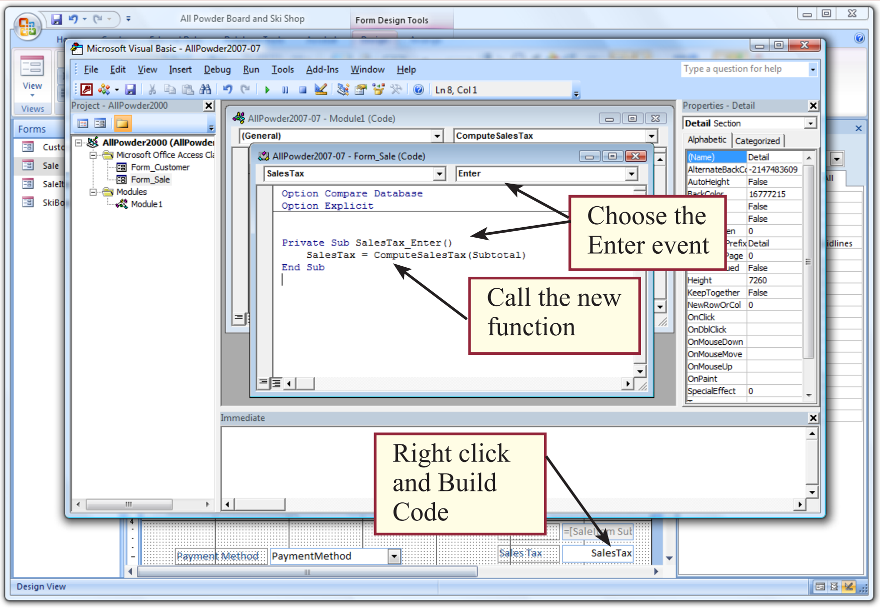
> **Action**
>
> Edit the Sale form in Design view.
>
> Right click the SalesTax box and select the Build Event/Code Builder option.
>
> Select the Enter event
>
> Add the line: SalesTax = ComputeSalesTax(Subtotal).
>
> Run the form.
>
> Click on the SalesTax box to test the calculation.

tion. Be sure to use a variable for the tax rate, since it makes the code easier to understand and reduces errors when someone tries to modify it later. Also, be sure you use the Round function to truncate the tax due at two decimal places. Save and close the module. You can now use this function in queries and forms just as you would use any other function.

The next step is to use the formula to automatically compute the sales tax on the Sales form. The challenge at this step is to identify when you want to compute the tax. Why does that matter? The problem is that there are times you do not want to compute the tax. For example, if a sale has been completed and a manager is simply reviewing the form, you should not recompute the tax because the rate might have changed. So, you only want to compute it for a new sale. Realistically, it only needs to be computed when all of the sale items have been selected, but the form has no way to know when the sale is completed. It is possible to recomputed the sales tax every time an entry is changed in the SaleItem subform, but that process is a little complicated. Probably the easiest solution is to compute the sales tax due when the user clicks on the SalesTax box. For new orders, a simple click generates the correct value and the order total.

You need to attach a line of code to the Enter event of the SalesTax box (see Figure 7.3). Right-click on the text box and choose the Build Code option. Use the Code Builder option to open the Visual Basic editor. Select the Enter event from the drop-down list of events. Enter the computation as "SalesTax = ComputeSalesTax(SubTotal)," where SubTotal is the name of the box on the form

Figure 7.3



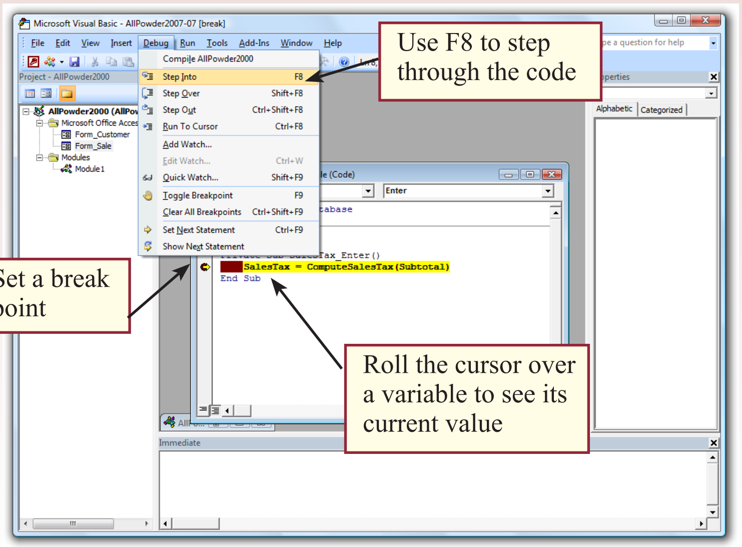
```
Option Compare Database
Option Explicit


Private Sub SalesTax_Enter()
    SalesTax = ComputeSalesTax(Subtotal)
End Sub
```

Properties - Detail

Detail Section

Alphabetic | Categorized

(Name)	Detail
AlternateBackC	-2147483609
AutoHeight	False
BackColor	16777215
	False
	False
en	0
Prefix	Detail
Page	0
ued	False
Height	7260
KeepTogether	False
NewRowOrCol	0
OnClick	
OnDblClick	
OnMouseDown	
OnMouseMove	
OnMouseUp	
OnPaint	
SpecialEffect	0



```
Private Sub SalesTax_Enter()
    SalesTax = ComputeSalesTax(Subtotal)
End Sub
```

Figure 7.4

that copies the subtotal amount from the subform. The Enter event is fired whenever the user clicks on or tabs to the specified text box.

To test the code, run the form and create a new order. Select an employee as the salesperson and choose the customer from the list. Select a product to sell (100003), enter the QuantitySold (1) and the SalePrice (300). Select a payment method. Initially, the Sales Tax entry will be empty. Click on the tax box and the 7 percent tax will be computed and entered. The total sale value will also be updated.

In real life, program code rarely runs correctly the first time. To find mistakes, you need to use the debugger. Open the Sales form in Design view and return to the code (Use the View Code icon on the Design/Tools menu.). Click on the gray column just to the left of the calculation line in the subroutine. The line should be highlighted in red as a break point. Return to the form and run it so that you can click on the Sales Tax box. Figure 7.4 shows that this time, the Visual Basic editor pops up, with the interpreted stopped on the break line. You can use the F8 debug key to step through line-by-line to see how the program behaves. Rolling the cursor over a variable displays its current value.

Activity: Update Inventory with Form Events

Maintaining quantity-on-hand statistics for inventory is one of the trickiest elements in programming business forms. Reexamine the Inventory table and notice that it contains the column QuantityOnHand. This value represents the current number in stock for a specific item. The value of the column is that clerks can quickly check the column to see if certain sizes are available. Also, managers can get a quick look at the

Action

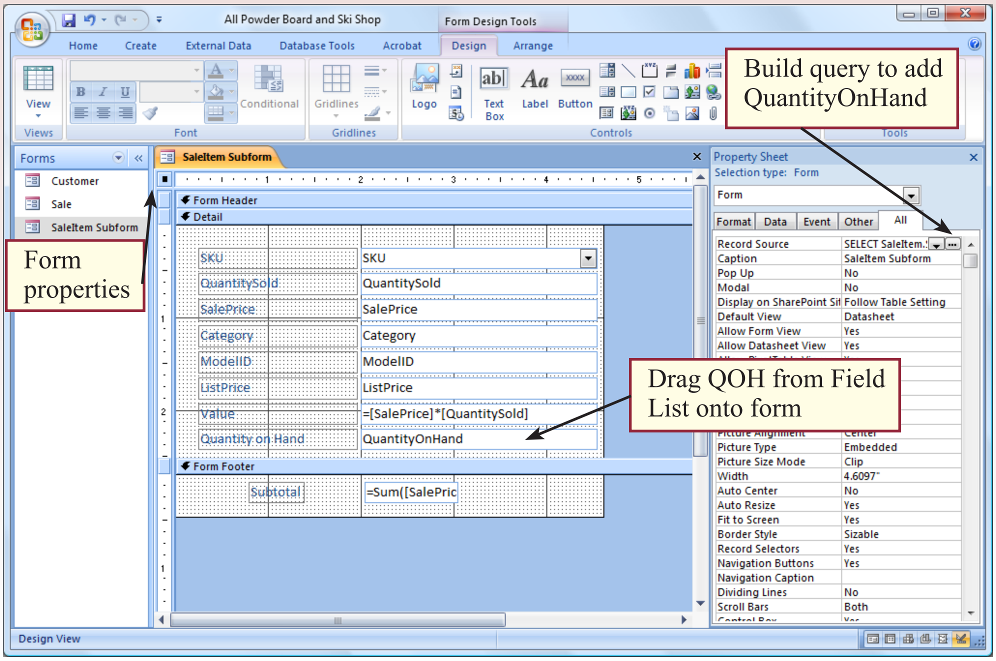
Open the SaleItem subform in Design view.

Open the form properties and modify the Record Source query to add the QuantityOnHand column.

View the Field List box.

Drag QuantityOnHand onto the form.

Run the form and improve the layout.



SKU	SKU
QuantitySold	QuantitySold
SalePrice	SalePrice
Category	Category
ModelID	ModelID
ListPrice	ListPrice
Value	=[SalePrice]*[QuantitySold]
Quantity on Hand	QuantityOnHand
Subtotal	=Sum([SalePric

Record Source	SELECT SaleItem.
Caption	SaleItem Subform
Pop Up	No
Modal	No
Display on SharePoint Sit	Follow Table Setting
Default View	Datasheet
Allow Form View	Yes
Allow Datasheet View	Yes
Picture Type	Embedded
Picture Size Mode	Clip
Width	4.6097"
Auto Center	No
Auto Resize	Yes
Fit to Screen	Yes
Border Style	Sizable
Record Selectors	Yes
Navigation Buttons	Yes
Navigation Caption	
Dividing Lines	No
Scroll Bars	Both

Figure 7.5

list of items that might be under- or overstocked. Technically, this value would not have to be stored in the database—if you have a complete list of all purchases, sales, and adjustments, you could use a query to compute the total number currently in stock. However, with thousands of items and sales, this query might take too long to run. Consequently, you need a mechanism to update this value on the fly. Whenever an item is sold, the corresponding quantity should be subtracted from the quantity on hand. In Access, this subtraction has to occur on the Sale form.

The first step in modifying the Sale form is to add the QuantityOnHand column to the SaleItem subform so clerks can see the status of each item. Since QuantityOnHand is already in the Inventory table, you simply have to modify the subform to display it. Open the SaleItem Subform and click on the Form Properties button. Click the Ellipses button on the Record Source property to build the query. Add the QuantityOnHand column from the Inventory table. Close and save the query, which returns it to the form. Figure 7.5 reviews the steps needed to add the field to the subform. You will need to resize the Sale form to display the new column.

Now that the QuantityOnHand is displayed on the sale subform it is relatively easy to update as items are selected. Remember that you can attach code to any event on the form or events on specific controls. Before attempting to write the code to modify the QOH, it is best to look at some of the form events. The Visual Basic editor gives you a complete list of form and control events through the drop-down lists. However, the editor does not explain the sequence of events. Figure 7.6 shows the primary events that are triggered as a form is opened and closed. The Current event of the form fills the form with data from the current row. Then controls trigger Enter and GotFocus events. Controls also experience

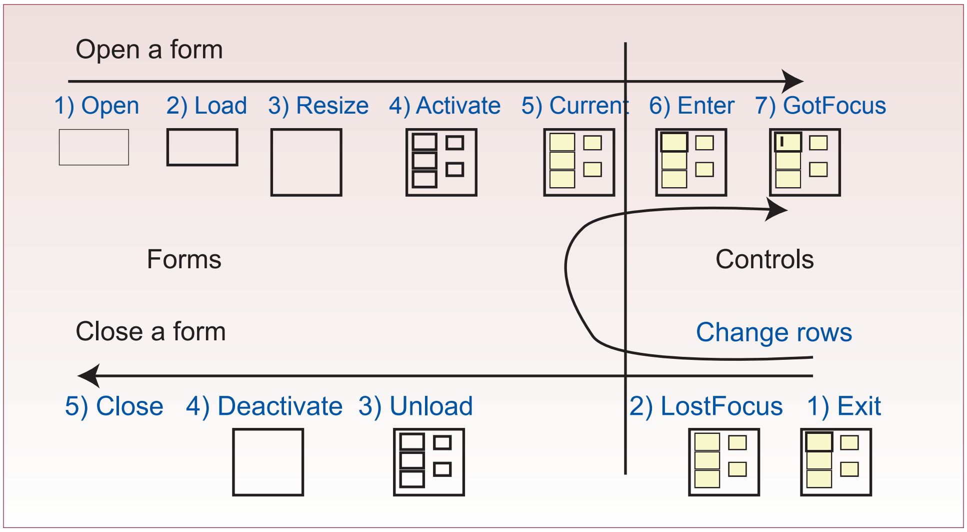
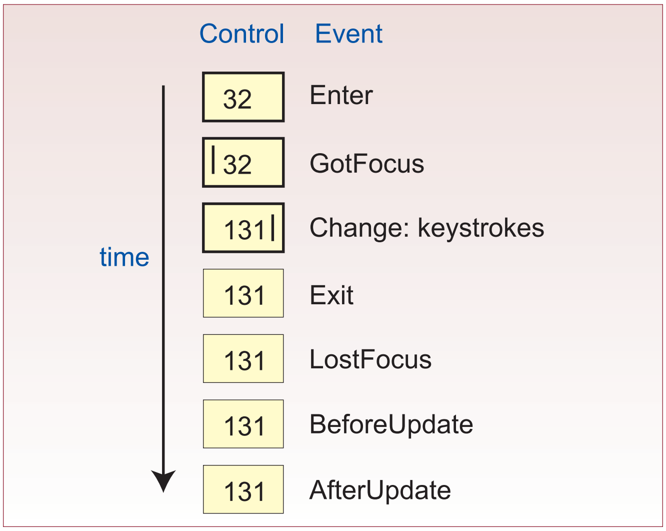


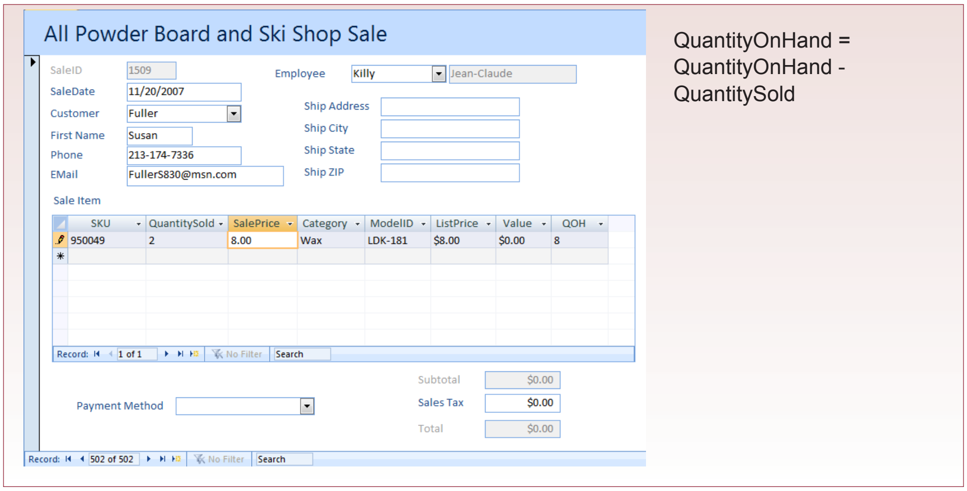
Figure 7.6

BeforeUpdate and AfterUpdate events, which are triggered just before and just after the data is transferred to the database when it is changed.

It is important to understand the event sequence for a control being edited. Figure 7.7 shows the main events as sample data is changed in a text box. You can write code that is activated in response to each event. The trick is to determine the proper event. In many cases, the BeforeUpdate and AfterUpdate events are useful. The BeforeUpdate event is often used to perform complex validation checks on the value before writing it to the database. You can cancel the update if something is incorrect. The AfterUpdate event is used when you want to make sure that the change was correctly recorded and need to take some action in response to the change. The AfterUpdate event is useful for handling the change to QuantityOnHand. The inventory level will be changed only when the initial item selection has been successful.

Figure 7.7





All Powder Board and Ski Shop Sale

SaleID	1509	Employee	Killy Jean-Claude
SaleDate	11/20/2007	Ship Address	
Customer	Fuller	Ship City	
First Name	Susan	Ship State	
Phone	213-174-7336	Ship ZIP	
EMail	FullerS830@msn.com		

Sale Item

SKU	QuantitySold	SalePrice	Category	ModelID	ListPrice	Value	QOH
950049	2	8.00	Wax	LDK-181	$8.00	$0.00	8

Record: 1 of 1 No Filter Search

Subtotal	$0.00
Sales Tax	$0.00
Total	$0.00

Payment Method

Record: 502 of 502 No Filter Search

```
QuantityOnHand =
QuantityOnHand -
QuantitySold
```

Figure 7.8

You begin by adding the subtraction code to the AfterUpdate event of the QuantitySold field. Figure 7.8 shows the two lines of code you need to enter at this stage. The Refresh command forces newly added rows to be saved before attempting to change the QOH. You should open the Sales form and create a new order: select a salesperson and a customer. Choose product 950049 because it has a QOH of 10. Next, enter a value of 2 for the QuantitySold and see what happens to the QuantityOnHand column. When you exit the QuantitySold column, the value is written to the database, your new code is executed, and two units are subtracted from the QuantityOnHand.

Before celebrating and moving on, you need to think about some potential problems. These problems may highlight the event-driven nature of the form, but they also exist with a database trigger approach. In particular, you should determine what happens if the clerk makes a mistake: instead of two items sold, he meant to enter one, so he returns to the QuantitySold box and changes the value. Try this and see what happens. Your code dutifully subtracts another unit from the QuantityOnHand, but, the new value of seven items in is wrong. Since the company really sold only one item, it should be nine units. The problem is that you have to add the original two units back to the total. In more general terms, if the QuantitySold value is not zero, your code must add that original value back to the total, and then subtract the new QuantitySold. That is, your calculation should be QuantityOnHand = QuantityOnHand – QuantitySold + OldQuantity. The only problem is to find the OldQuantity.

Look again at the Figure 7.7 events. If you wait until the AfterUpdate event, the only value you can get is the new number that was entered. In fact, any event after the Change event is too late—the original value will be gone. Actually, GotFocus is even too late, because someone might change the data, leave the text box and return, triggering a new GotFocus event and losing the original value. That leaves the Enter event. When the user first enters the box, you can obtain the original value.

Action

Edit the SaleItem subform in Design view.

Right click the QOH box and build code.

Select the AfterUpdate event.

Add code to refresh the row and subtract QuantitySold from QuantityOnHand.

Test the form.

```
Option Compare Database
Option Explicit

Dim OldQuantity As Integer

Private Sub QuantitySold_AfterUpdate()
   Me.Refresh                       ‘ Save data for new rows
   QuantityOnHand = QuantityOnHand - QuantitySold + OldQuantity
End Sub

Private Sub QuantitySold_Enter()
   If IsNull(QuantitySold) Then
      OldQuantity = 0
   Else
      OldQuantity = QuantitySold
   End If

End Sub
```

Figure 7.9

But it does not do you any good within the Enter event. As soon as the event is finished, the subroutine ends and the number will be discarded. The AfterUpdate event cannot refer to data in other events. You need a place to temporarily store the OldQuantity value so that it will be available to the AfterUpdate calculation. The solution is to create a new variable on the form code page that is outside of all of the events, making it accessible to all of them. Before modifying the code, you should open the Inventory table and correct the total to nine units.

Figure 7.9 shows the new code to update the inventory. The value for Old-Quantity is assigned in the Enter event. By declaring the variable outside of the individual routines, it is global to the form and exists as long as the form is open, and can be used by any routine on the form page. Notice the use of the condition to handle the case where the QuantitySold is missing (null). Without this test, the code might crash if the data is missing. Null values in operations lead to more null values, or sometimes a total crash if Null cannot be converted to an integer. When you have entered the code, save and close the subform. Run the Sale form and return to the new sale. Change the QuantitySold value from one to two and watch the QuantityOnHand change from nine to eight, just as it should.

This trick with events is impressive, but not perfect. What else could go wrong? Think about other possible changes. What happens if the clerk selects a different SKU after the QuantitySold has been updated and the QuantityOnHand has been changed? To handle this problem, you would have to add the quantity back to the QuantityOnHand for the original SKU, and then subtract it for the new SKU. Or what if the user deletes the SKU item or even the entire order? You need to add the original QuantitySold back to the QuantityOnHand before the row is deleted. Although these issues seem to be similar to the first problem, they are harder because they require changing a QuantityOnHand value for an SKU that is no longer displayed on the form. You will have to write SQL commands that can execute changes directly to the database.

Action

Open the SaleItem subform in Design view.

Choose View/Code.

Add and modify the code as specified to handle changes in quantity.

Test the form.

Activity: Building and Executing SQL Commands in Code

Action

Open the SaleItem subform in Design view.

Add the oldSKU variable.

Add the SKU event code as described.

Place a breakpoint in the code before the Execute line.

Test the form by changing an SKU number.

Examine the SQL line before it runs.

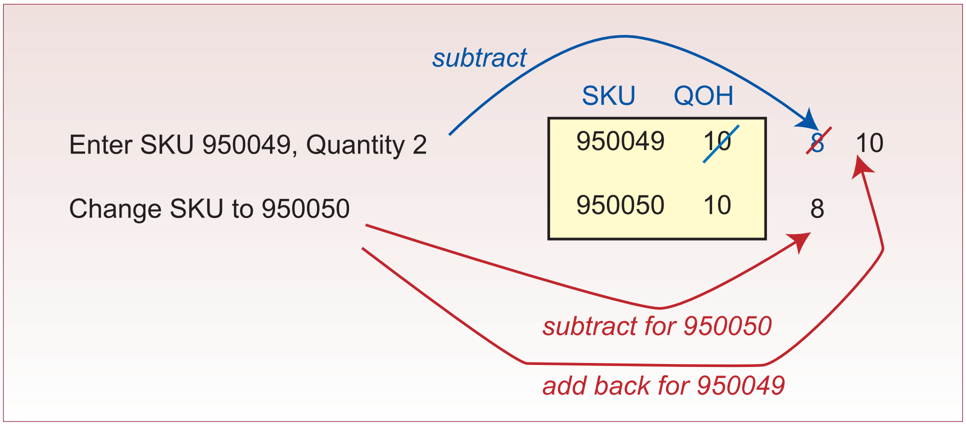
Add the delete code as shown.

Test multiple changes to validate the code.

Remember the power of the DML SQL commands (INSERT, UPDATE, and DELETE). In most cases, they are too powerful to trust to typical users. (Oops, half the database has just been deleted!) On the other hand, they are useful within your code when you need to make changes automatically. You might have a form that collects a few pieces of information and then builds an SQL command that can be executed immediately. Or, as in the inventory case, your code might have to update a row of data that is not displayed on the form. This latter situation arises when a clerk changes the value of the SKU item being sold. Figure 7.10 shows the changes needed in the Inventory table. When the first SKU (950049) is entered, your code subtracts the two units from the quantity on hand. When the clerk changes the SKU to 950050, you need code to add the two units back to the 950049 row and subtract them from the 950050 row. The code for subtracting the 2 units from the new row is straightforward because it is the current row and the corresponding quantity on hand is directly accessible. The catch is that the old row (950049) is no longer active, and you will have to write an SQL statement to change the value.

Again, you need to keep track of the original value of the SKU. As shown in the first part of Figure 7.11, your code for the combo box Enter event stores the current SKU value in a temporary variable. Be sure you create the new variable at the top of the form. Then, if the SKU is changed in the combo box, the code first updates the QOH for the current SKU (950050). It then creates an SQL statement to update the QOH for the prior SKU. Before attempting to write the code, think about how you would write the SQL statement by hand. If you want to add two units back to QuantityOnHand for the 950049 SKU, it would be: UPDATE Inventory SET QuantityOnHand = QuantityOnHand + 2 WHERE SKU = '950049'.

Figure 7.10



The UPDATE statement is straightforward, but you have to be careful to include the quotes around the SKU value. The table definition declares the SKU as a text column—just in case the store wants to use nonnumeric codes in the future. Consequently, all SKU values have to be enclosed in quotes within an SQL statement. Figure 7.11 shows how the SQL string is built in pieces. First, the literal portion “UPDATE …” is created. Next, the value of the QuantitySold is appended with an ampersand (&), followed by more literal text “WHERE …” that also includes the first single quote mark. The old SKU value is then appended, followed by the trailing single quote character. To test this code, you should place a breakpoint on this line, then examine the string that gets assigned to make sure it is correct. In particular, double-check that there are spaces between the words. If you have a complex SQL statement that does not seem to work but you cannot identify the problem, you can display the value in the Immediate window. Then copy the statement and return to Access to build a new query. Paste the SQL statement into the SQL query window and run it. Access will give you more detailed error messages. The final part of the code simply creates a new Active Data Object Database (ADODB) command. It connects to the current database and runs the SQL statement.

Maintaining quantity on hand is a good way to learn the details of form events. It forces you to think about how users interact with the form and the effects of user changes. As a developer, you must always think of different ways that people will use your application. Always assume that someone will follow a different path and consider how that will affect the data. Now, what happens if a user deletes an item from the list of sale items? If your code has already subtracted units from inventory, you must restore those values. This process is similar to the two cases already examined. But it is more difficult because Access allows users to delete multiple rows at one time. You need code that captures each SKU and the corresponding quantity. Then, when the rows are deleted, your code goes through the list of deleted items and restores the quantity on hand for each entry. Handling

Figure 7.11

```
Private Sub cboSKU_Enter()
   If IsNull(cboSKU.Value) Then
      OldSKU = “-1”
   Else
      OldSKU = cboSKU.Value
   End If
End Sub
Private Sub cboSKU_AfterUpdate()
   If OldSKU <> “-1” And Not IsNull(QuantitySold) Then
      QuantityOnHand = QuantityOnHand - QuantitySold
      Dim sql As String
      sql = “UPDATE Inventory SET QuantityOnHand = QuantityOnHand + “ _
         & QuantitySold & “ WHERE SKU=’” & OldSKU & “’”

      Dim cmd As ADODB.Command
      Set cmd = CreateObject(“ADODB.Command”)
      cmd.ActiveConnection = CurrentProject.Connection
      cmd.CommandText = sql
      cmd.Execute
   End If
End Sub
```

```
Dim DelQuantity() As String
Dim DelSKU() As String
Dim nDel As Integer
Private Sub Form_Delete(Cancel As Integer)
   If IsNull(cboSKU.Value) Or IsNull(QuantitySold) Then Exit Sub
   If (nDel = 0) Then
      ReDim DelQuantity(Me.CurrentView)
      ReDim DelSKU(Me.CurrentView)
   End If
   DelSKU(nDel) = cboSKU.Value
   DelQuantity(nDel) = QuantitySold
   nDel = nDel + 1
End Sub
Private Sub Form_AfterDelConfirm(Status As Integer)
   Dim i As Integer
   For i = 0 To nDel - 1
      DeleteOneRow DelSKU(i), DelQuantity(i)
   Next i
   nDel = 0
End Sub
Private Sub DeleteOneRow(ByVal SKU As String, ByVal Qty As String)
   Dim sql As String
   sql = "UPDATE Inventory SET QuantityOnHand = QuantityOnHand + " _
      & Qty & " WHERE SKU='" & SKU & "'"
   Dim cmd As ADODB.Command
   Set cmd = CreateObject("ADODB.Command")
   cmd.ActiveConnection = CurrentProject.Connection
   cmd.CommandText = sql
   cmd.Execute
End Sub
```

Figure 7.12

this list efficiently requires a dynamically created array. A complete description of the code would be too long for this lab. However, Figure 7.12 shows the code so your application can be complete. Notice that two events are used: Delete and AfterDeleteConfirm. The Delete event is triggered once for each row being deleted, so this is where the SKU and QuantitySold values are collected and stored in temporary arrays. The AfterDeleteConfirm event is fired once—when users agree to delete the rows. At this time, a loop updates the QOH for each of the saved values, using an SQL statement that is executed in a separate routine.

Activity: Define Transactions

Transactions consist of multiple changes that must succeed or fail together. Access provides some support for transactions using the ADODB transaction code. However, you need to know that Access does not provide a backup journal log. If the system crashes in the middle of a transaction, it wipes out the transaction point. That is, all of the updates will fail: there is no easy way to find the point at which they failed or to know which transactions to restart.

Action

Create the Rental Discount form in Design view.

Add the text boxes and button.

Save the form.

Open the Rental form in Design view.

Add a button to open the Discount form.

Modify the open code as indicated.

Run the form and test the button.

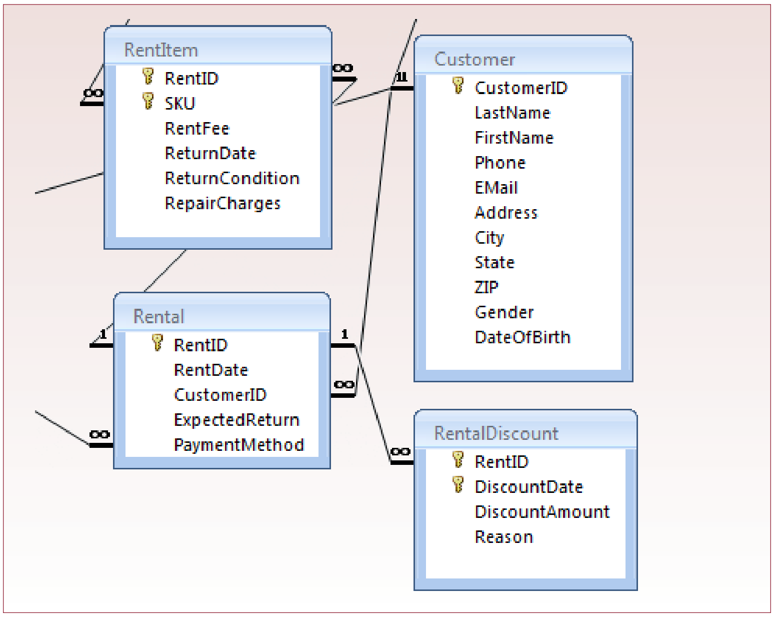
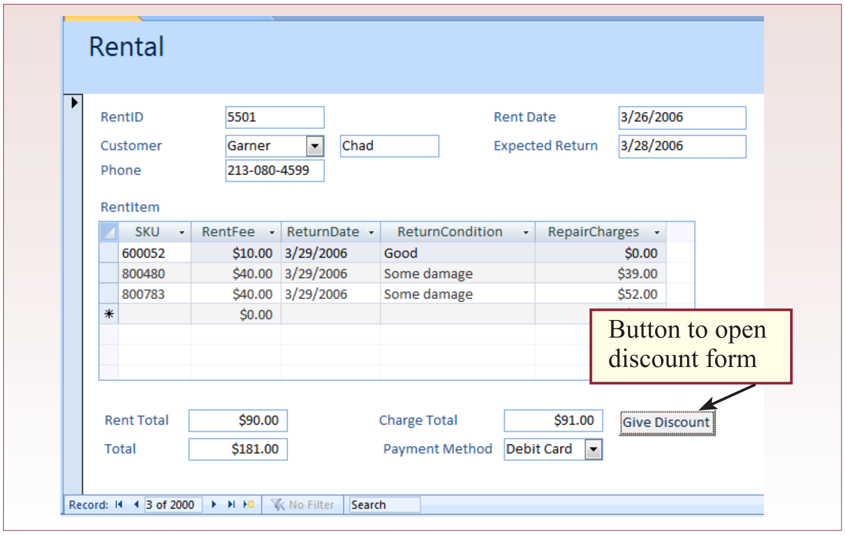


Figure 7.13

But, at least you will not end up with half a transaction completed and the rest missing.

Katy, the manager at All Powder, has noticed that many customers do not like being charged for damages caused to the rental equipment. Some of them believe that the equipment has simply worn out and failed. She has also noticed that there can be several complaints about a specific rental—particularly when it involves multiple items. David, the rental manager, agrees, but still wants to be able to track the cumulative charges. He has suggested that any reduction in the damage charge be recorded as a discount to that customer. That way, he can track the total damages, as well as which customers might receive the most discounts. Katy also likes the discount idea, because she wants to implement a discount program

Figure 7.14

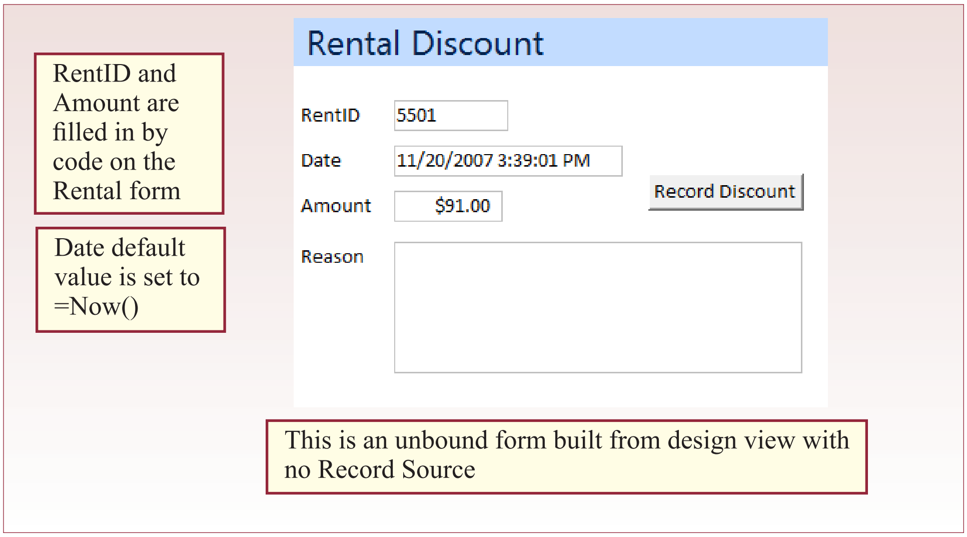

Rental

RentID	5501		Rent Date	3/26/2006
Customer	Garner	Chad	Expected Return	3/28/2006
Phone	213-080-4599			

RentItem

SKU	RentFee	ReturnDate	ReturnCondition	RepairCharges
600052	$10.00	3/29/2006	Good	$0.00
800480	$40.00	3/29/2006	Some damage	$39.00
800783	$40.00	3/29/2006	Some damage	$52.00
	$0.00			

Rent Total	$90.00	Charge Total	$91.00
Total	$181.00	Payment Method	Debit Card



RentID and Amount are filled in by code on the Rental form

Date default value is set to =Now()

This is an unbound form built from design view with no Record Source

Figure 7.15

for employees who rent equipment. Since multiple discounts can be applied to a single rental, a new table is needed. Figure 7.13 shows the table keyed by both RentID and DiscountDate.

You can build a form to handle data entry for the employee discounts, but do not do that now. It is a little more complicated to correctly handle the customer discounts for disagreements over the damage charges. You need a transaction that decreases the repair charges and adds a row to the RentalDiscount table for the same amount. To begin, you need to create a Rental form similar to the Sales form. Figure 7.14 shows a standard Rental form. Notice that it needs subtotals for the rental amount and for the charges. Any repair charges would be entered when the items are returned. Eventually, you also need to add a standard command button to open the form to give the discounts, but it is easier to create the form first and then return to add the button on the Rental form.

Figure 7.16

```
Private Sub cmdDiscount_Click()
On Error GoTo Err_cmdDiscount_Click
    Dim stDocName As String
    Dim stLinkCriteria As String

    stDocName = “GiveRentDiscount”
    DoCmd.OpenForm stDocName, , , stLinkCriteria
    Forms!GiveRentDiscount!txtRentID = RentID
    Forms!GiveRentDiscount!txtDiscountAmount = SubTotalCharges

Exit_cmdDiscount_Click:
    Exit Sub

Err_cmdDiscount_Click:
    MsgBox Err.Description
    Resume Exit_cmdDiscount_Click
End Sub
```

Action

Open the Rental Discount form in Design view.

Add the specified code to the button click event.

Test the forms.

Figure 7.15 shows the Discount form (GiveRentDiscount). It is built from the blank form design view and not tied to the database. Add the text boxes by hand and be sure to give them names, such as txtRentID. Set the default value on the date field to Now(), so the current date and time are entered by default. You can manually add the button, but if the wizard creates any code for the Click event, keep the subroutine, and delete all of the code from inside of it. Save the form.

The next step is to place a button on the Rental form that will open this Discount form and transfer two values automatically: RentID and Amount. Figure 7.16 shows the code used on the Rental form button. The wizard to open a form generated most of the code automatically. You only have to add the two lines that transfer the RentID and Charges to the Discount form. You do that by assigning the values on the current Rental form to the corresponding text boxes on the discount form by using the name of the Discount form and the desired text box. Save the forms and test the data transfer.

The code for handling the actual discount is a little more complex. One complicating factor is the question of how to apply the discount. Should the charges be subtracted from one of the rental items or several of them? If the managers want to subtract only some of the charges, you will need to implement a mechanism on the Discount form to identify which charges should be reduced and by what amount. You might add a drop-down list to do one item at a time, or a subform with text boxes on each line. Either way, the process becomes too complicated for

Figure 7.17

```
Private Sub cmdDiscount_Click()
   Dim cmd As ADODB.Command
   Dim SQL1 As String, SQL2 As String
   Set cmd = CreateObject("ADODB.Command")
   cmd.ActiveConnection = CurrentProject.Connection
   SQL1 = "UPDATE RentItem SET RepairCharges=0 WHERE RentID=" & txtRentID
   SQL2 = "INSERT INTO RentalDiscount (RentID, DiscountDate, DiscountAmount, Reason)" & _
      " VALUES (" & txtRentID & _
      ", #" & txtDiscountDate & "#" & _
      ", " & txtDiscountAmount & _
      ", '" & txtReason & "')"
   On Error GoTo Err_DiscountTrans
   cmd.ActiveConnection.BeginTrans
   cmd.CommandText = SQL1
   cmd.Execute
   cmd.CommandText = SQL2
   cmd.Execute
   cmd.ActiveConnection.CommitTrans
   lblMessage.Caption = "Changes recorded."
Exit1:
   Exit Sub
Err_DiscountTrans:
   cmd.ActiveConnection.RollbackTrans
   lblMessage.Caption = Err.Description
   Resume Exit1
End Sub
```

this lab. The code in Figure 7.17 assumes that all repair charges are being reduced to zero. The first SQL statement updates the RentItem table and sets the repair charges to zero for the specified RentID. The second SQL statement records the change by inserting a new row into the Discount table. The string is built by appending the values from the form text boxes. Notice that the date is surrounded by pound signs (#), and the reason by single quotes ('). Since the code requires changes to two tables, the BeginTrans function is called on the database connection to establish the start of the transaction. If no problems arise, the code executes linearly and commits the two changes to the database. A simple message is then displayed on the form and the code exits.

Notice the use of the ON ERROR GOTO statement to handle potential errors. This line states that if something goes wrong in the processing, the computer will immediately jump to the specified location in the code (Err_DiscountTrans). Here, the error message is written to the form, and the transaction is rolled back. The code then exits the subroutine. This error control structure is commonly used in this version of Visual Basic. In fact, the command button wizards write this code for all of the routines. However, you will have to add customized code to the error routines. For example, you should at least display the error message so you and the users know something went wrong. Also try to handle the error and clean up the database where possible.

Database Cursors, Keys, and Locks

Activity: Read Rows of Data

Action

Create a new query in Design view.

Tables: Sale and SaleItem.

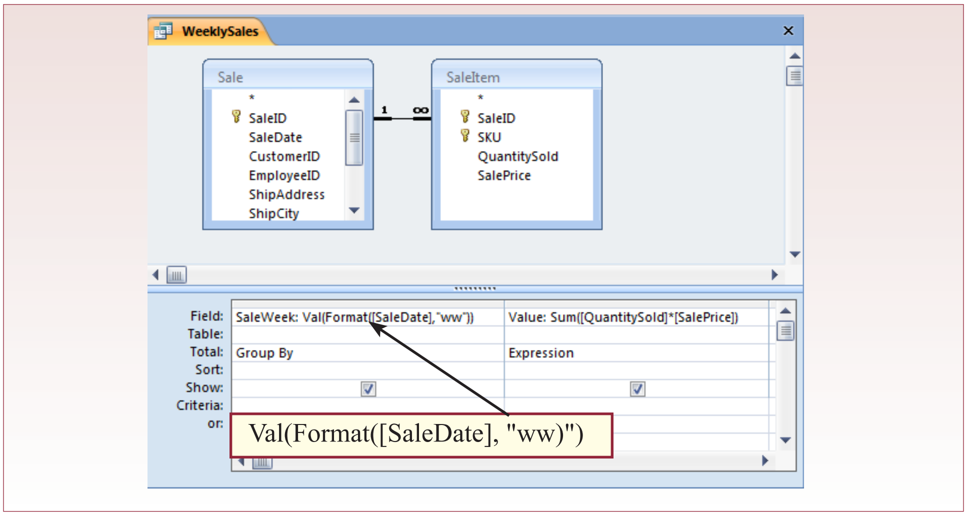
Create column SaleWeek:
Val(Format([SaleDate],"ww")).

Create column Value:
[QuantitySold]*[SalePrice]).

Sum the Value column.

Direct SQL commands are useful for DML issues where you need to change or delete rows of data. When you need program code that has to examine several rows of data, you should use database cursors. Consider the business question of sales by week. Katy wants to know if weekly sales increase more in the first part of the year or in the last part. In particular, she wants to know the average percent increase in weekly sales for the first weeks (1 to 15) compared to the last 15 weeks (38 to 52). Remember that SQL can perform calculations on data within the same row. SQL can also compute subtotals for groups of data. However, it is difficult to get SQL to compare data by subtracting values across two rows. Instead, it is easier to write a query that does the main computations, and then use cursor code to do the comparisons.

Begin by creating a query that computes total sales by week. Figure 7.18 shows that the query is slightly tricky. You can use the Format(SaleDate, "ww") statement to convert the date to week of the year. However, because the Format function returns a string for the week number, it does not sort correctly. So you have to use the Val function to convert the weeks into values that are sorted numerically. Run the query and you will see that it returns the total value of sales for each week of the year. Note that there are 53 weeks instead of the expected 52, but the last week does not have seven days. The issue of 52 and 53 weeks per years is important when setting up accounting systems for retailers but can be ignored for this lab. Save the query as qryWeeklySales and close it.



Field:	SaleWeek: Val(Format([SaleDate],"ww"))	Value: Sum([QuantitySold]*[SalePrice])
Table:		
Total:	Group By	Expression
Sort:		
Show:	✓	✓
Criteria:		
or:		

Val(Format([SaleDate], "ww"))

Figure 7.18

Action

Create a new blank form.

Add a button.

Add a text box named txtAverage.

Add the specified code to the button

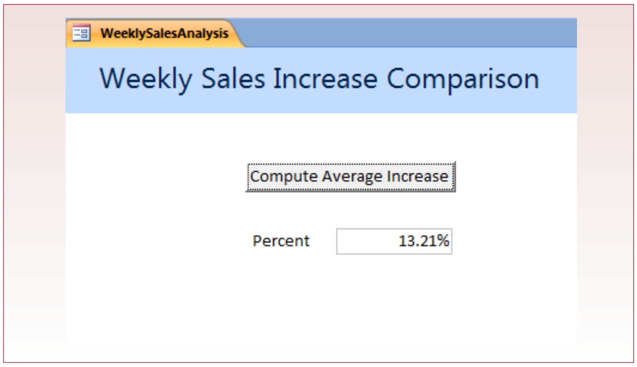
Run the form and test the code.

Place a breakpoint and step through the code as it runs.

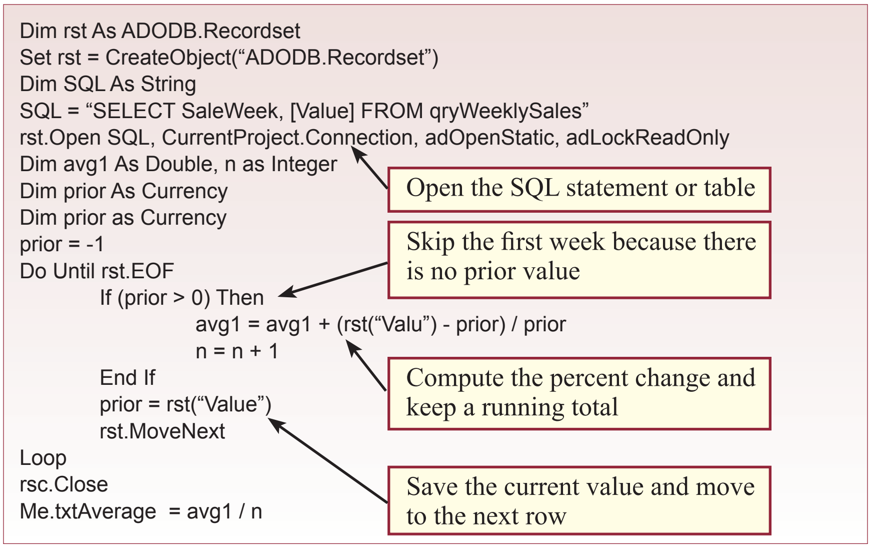
The next step is to compute the percentage change between the rows. Because you will need code for this step, begin by creating a new form in Design view. Add a command button and a text box to display the result. Give meaningful names to both controls, such as txtAverage. Figure 7.19 shows the basic form.

The next step is to write the code that computes the average percent increase. For each pair of rows, the code needs to subtract the two values and divide by the value in the prior row to yield a percentage change. This percentage needs to be summed and eventually divided by the number of calculations to obtain the average percent increase. Figure 7.20 shows the main code. The SQL statement is opened as a recordset, which retrieves one row of data at a time. The avg1 variable keeps the running total of the percentage increase, while n counts the number of operations. The role of the “prior” variable is the most important. At the end of the

Figure 7.19



Percent 13.21%



```
Dim rst As ADODB.Recordset
Set rst = CreateObject("ADODB.Recordset")
Dim SQL As String
SQL = "SELECT SaleWeek, [Value] FROM qryWeeklySales"
rst.Open SQL, CurrentProject.Connection, adOpenStatic, adLockReadOnly
Dim avg1 As Double, n as Integer
Dim prior As Currency
Dim prior as Currency
prior = -1
Do Until rst.EOF
        If (prior > 0) Then
                avg1 = avg1 + (rst("Valu") - prior) / prior
                n = n + 1
        End If
        prior = rst("Value")
        rst.MoveNext
Loop
rsc.Close
Me.txtAverage  = avg1 / n
```

Open the SQL statement or table
Skip the first week because there is no prior value
Compute the percent change and keep a running total
Save the current value and move to the next row

Figure 7.20

loop, it is assigned the value obtained from the current row. When the next row is retrieved, the program can now compare the current (new) value to the old (prior) value. This trick is useful for many cursor-based programs, so you should study the code until you understand it. You should also place a breakpoint at the start of the code and then step through it line by line so you can see how it operates.

Activity: Generate and Use Keys

Action

Create a new blank form.

Add boxes for CustomerID, EmployeeID, SKU, and txtSaleID as the generated key.

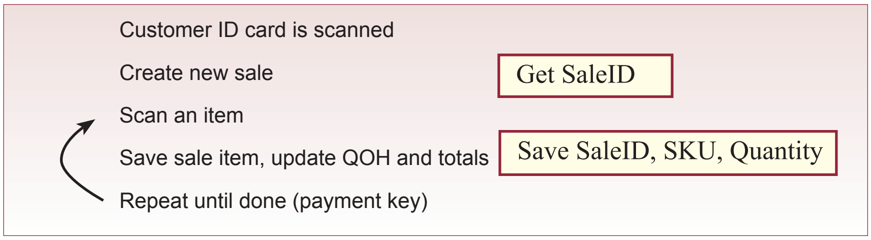
Create a command button and add the indicated code.

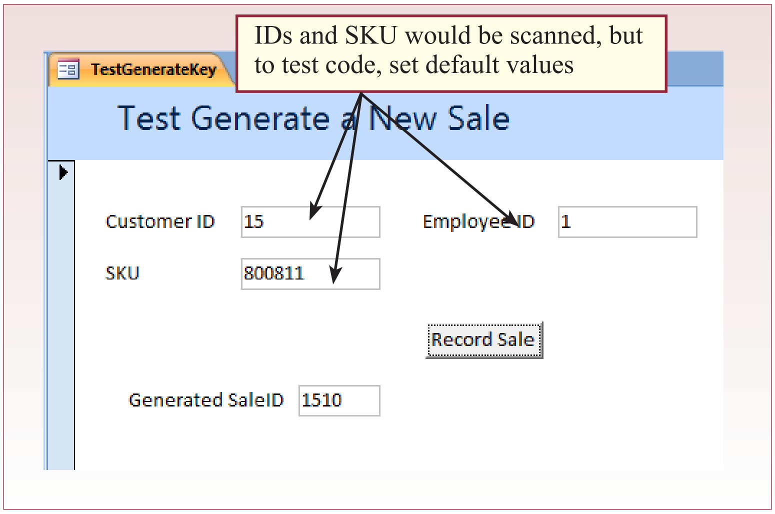
Test the form.

Place a breakpoint at the top of the code and step through the code.

Access uses the AutoNumber field type to indicate when a key should be generated. For standard forms, the process is relatively automatic. For example, the Customer table uses AutoNumber for the CustomerID. Whenever a new customer is added in the Customer form, a new value is generated and stored automatically. For many common operations, this process works well and is mostly invisible to the user and developer. However, some cases cause more problems—particularly when you need to perform operations outside the common forms. Consider a case where you need custom code to generate each sale and

Figure 7.21


Customer ID card is scanned
Create new sale
Get SaleID
Scan an item
Save sale item, update QOH and totals
Save SaleID, SKU, Quantity
Repeat until done (payment key)


IDs and SKU would be scanned, but to test code, set default values
TestGenerateKey
Test Generate a New Sale
Record Sale

Customer ID	15	EmployeeID	1
SKU	800811		
Generated SaleID	1510		

Figure 7.22

enter the sale items. For example, perhaps you have a barcode scanner and want to automate as much of the checkout process as possible.

Figure 7.21 outlines the basic events that will occur. Notice that when the new Sale is created, Access will generate a new key value. You need to get this value so that you can save it in the SaleItem table for each scanned item. The immediate question is how do you get this value? This question is somewhat challenging to answer and highlights one of the biggest problems with using AutoNumber. The other issue with AutoNumber is that the techniques are not portable to other systems.

The most important issue to recognize is that you cannot get a reliable value for the generated key if you use an SQL DML command. For example, if you issue an INSERT command on the Sale table, Access will automatically generate and insert a new key value for SaleID. However, there is no way to know what value was generated. You may consider issuing a select statement using the other columns in a complex where clause. But if you do, there is no guarantee that those values are unique, so you cannot be completely certain that you will obtain the correct SaleID value. However, there is a way to use cursor code to obtain the newly generated value.

Even if you have a scanner handy, you probably do not want to write the interface code for it. To simulate the data from the scanner, begin by creating a form in Design view that has text boxes for the three main keys: CustomerID, EmployeeID, and SKU. Figure 7.22 shows a sample form with default values that will work. Add a command button and delete the main wizard code. Also, add a text box to display the SaleID that will be generated within the code.

Figure 7.23 shows the code that runs the process. First, a query is used to look up the ListPrice of the scanned SKU. Second, a new sale row is created and the foreign key values of CustomerID and EmployeeID are entered into the row along with the current date. When these values are added, Access generates the new SaleID key value, so you need a line of code to capture this value and save it in a variable. Otherwise, the value will disappear when the sale recordset is closed. Third, a new SaleItem row is added. The SaleID stored in the temporary variable is placed into the data row, along with the values for the SKU, ListPrice as the

```
Dim sqlSale As String, sqlItem As String, sqlSaleItem As String
Dim rstSale As ADODB.Recordset, rstModel As ADODB.Recordset
Dim rstSaleItem As ADODB.Recordset
Set rstSale = CreateObject("ADODB.Recordset")
Set rstModel = CreateObject("ADODB.Recordset")
Set rstSaleItem = CreateObject("ADODB.Recordset")

sqlSale = "SELECT SaleID, CustomerID, EmployeeID, SaleDate FROM Sale"
sqlItem = "SELECT ModelID, ListPrice FROM Inventory INNER JOIN ItemModel ON " & _
        "Inventory.ModelID = ItemModel.ModelID WHERE SKU='" & SKU & "'"
sqlSaleItem = "SELECT SaleID, SKU, SalePrice, QuantitySold FROM SaleItem"

Dim cnn As ADODB.Connection
Set cnn = CurrentProject.Connection
' Get the List Price for the SKU
rstModel.Open sqlmodel, cnn, adOpenStatic, adLockReadOnly
Dim ListPrice As Currency
ListPrice = rstModel("ListPrice")
rstModel.Close
' Open the Sale table and create a new sale
rstSale.Open sqlSale, cnn, adOpenDynamic, adLockOptimistic
Dim SaleID As Long
rstSale.AddNew
rstSale("SaleDate") = Now
rstSale("CustomerID") = CustomerID
rstSale("EmployeeID") = EmployeeID
SaleID = rstSale("SaleID")
rstSale.Update
rstSale.Close
' Add the SKU to the SaleItem table using the new SaleID
rstSaleItem.Open sqlSaleItem, cnn, adOpenDynamic, adLockOptimistic
rstSaleItem("SaleID") = SaleID
rstSaleItem("SKU") = SKU
rstSaleItem("SalePrice") = ListPrice
rstSaleItem("QuantitySold") = 1
rstSaleItem.Update
rstSaleItem.Close
txtSaleID = SaleID
```

Figure 7.23

SalePrice, and a QuantitySold of one unit. Calling the Update method saves the data to the database, and the recordset can be closed. The last step is to display the newly generated SaleID on the form so you can see it. If you open the main Sale form and go to the last record, you should see this new sale with one item having been sold. Of course, you could modify the code to handle multiple items scanned, along with a screen to add the payment data, but they are not needed at this point. This trick of using cursor code to add a row and entering at least one column value is the only way to guarantee that you get the generated key value in Access.

Activity: Compare Pessimistic and Optimistic Locks

The issue of locking records to prevent concurrency errors could be applied to the rental discount form. Think about the possible errors if one clerk enters

Action

Create a new blank form.

Add a combo box to select customers.

Add a text box to enter a new ZIP Code.

Create a button and add the indicated code for it.

Test the form.

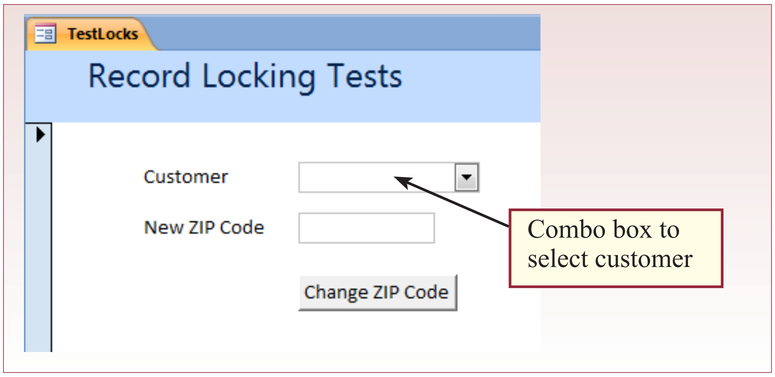


Figure 7.24

new values for damages while a second one is offering a discount. However, the differences between pessimistic and optimistic locking are difficult to understand, and it is better to start with a simple problem that is independent of the other forms. Consider a program that changes ZIP Codes for customer data.

Create a new form in Design view that is not bound to the database. As shown in Figure 7.24, add a combo box to select a customer. Add a text box to enter a new ZIP code. Create a command button that will execute the code to change the ZIP Code for the selected customer. If you use the wizard to add the button, edit the wizard code to leave the error-handling code, but remove the one or two lines of working code.

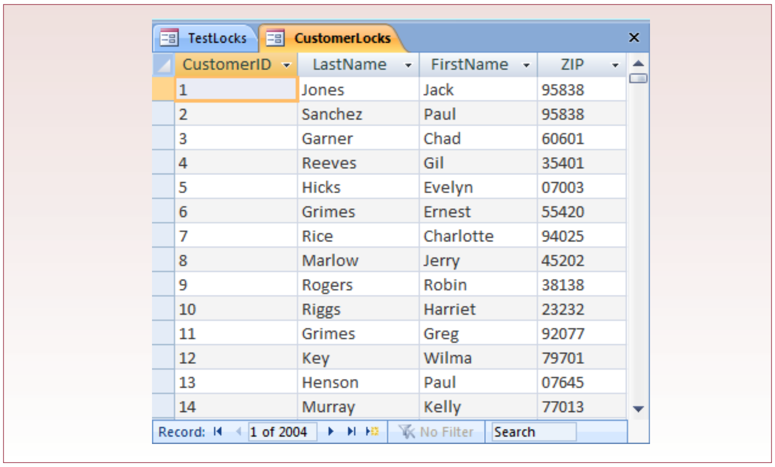
Figure 7.25 shows the code used to update the ZIP code for the selected customer. The SQL statement selects the desired customer through the WHERE clause, and the new ZIP code is assigned using the data cursor.

You need two processes that change the same data to test the data locks and concurrency. To be able to see the effects of locks, create a quick and simple form to view a few of the columns of the Customer table. Use the form wizard to create a form based on the Customer table showing CustomerID, LastName, FirstName, and ZIP. Choose the datasheet view so you can see several rows at one time. Figure 7.26 shows the basic form. To display this form and the TestLock form at the same time, remember that you have to switch to Windows view.

By default, Access forms are set to optimistic locking, and it is the form locking that is easiest to test. Open both forms and position them on the screen so you have easy access to both of them. Click on the ZIP code for the first customer and

Figure 7.25

```
Dim rst As ADODB.Recordset
Set rst = CreateObject("ADODB.Recordset")
Dim SQL As String
SQL = "SELECT CustomerID, ZIP FROM Customer " & _
      " WHERE CustomerID=" & cboCustomerID
Dim cnn As ADODB.Connection
Set cnn = CurrentProject.Connection
rst.Open SQL, cnn, adOpenDynamic, adLockOptimistic
rst("ZIP") = NewZIPCode
rst.Update
rst.Close
```



TestLocks | CustomerLocks

CustomerID	LastName	FirstName	ZIP
1	Jones	Jack	95838
2	Sanchez	Paul	95838
3	Garner	Chad	60601
4	Reeves	Gil	35401
5	Hicks	Evelyn	07003
6	Grimes	Ernest	55420
7	Rice	Charlotte	94025
8	Marlow	Jerry	45202
9	Rogers	Robin	38138
10	Riggs	Harriet	23232
11	Grimes	Greg	92077
12	Key	Wilma	79701
13	Henson	Paul	07645
14	Murray	Kelly	77013

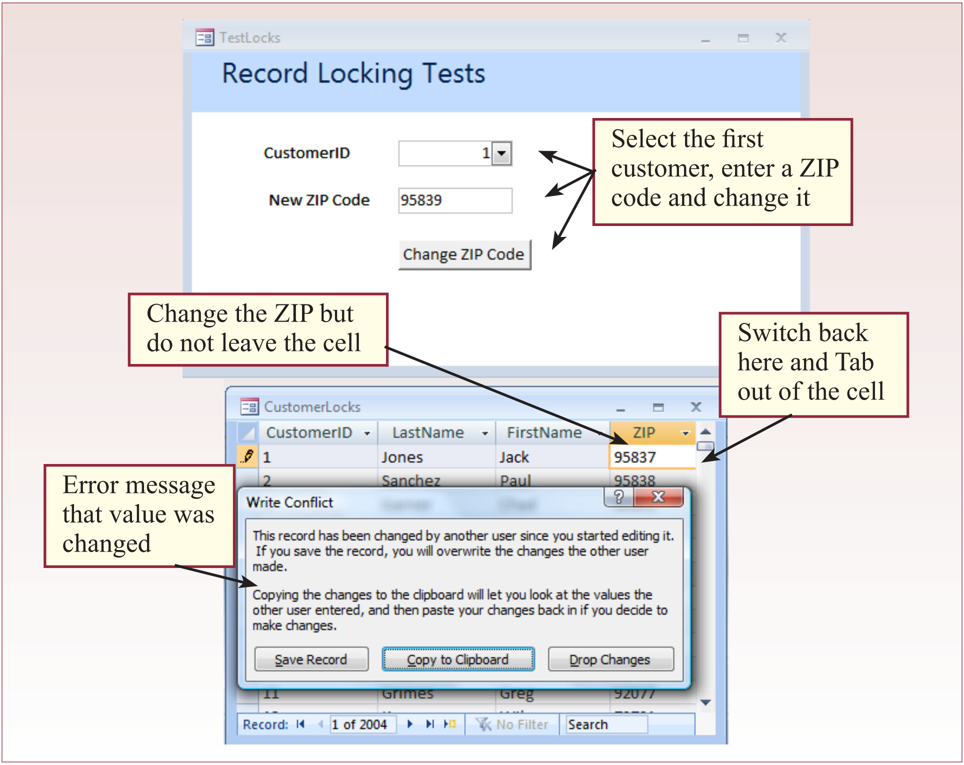
Record: 1 of 2004 No Filter Search

Figure 7.26

change the last digit. But keep the cursor inside the ZIP box; that is, do not press the Tab or Enter keys.

Switch to the CustomerLocks form, select the first customer in the combo box. Enter a new ZIP in the text box. Then click the button to submit the changes. You should not receive any error messages. Return to the CustomerLocks forms and press the Tab key to accept the changes. Figure 7.27 shows the error message you will receive, indicating that another process (the TestLocks form) has changed the same data before you finished editing the value. You are given the choice of ac-

Figure 7.27



TestLocks

Record Locking Tests

CustomerID: 1

New ZIP Code: 95839

Change ZIP Code

Select the first customer, enter a ZIP code and change it

Change the ZIP but do not leave the cell

Switch back here and Tab out of the cell

Error message that value was changed

CustomerLocks

CustomerID	LastName	FirstName	ZIP
1	Jones	Jack	95837
2	Sanchez	Paul	95838
11	Grimes	Greg	92077

Write Conflict

This record has been changed by another user since you started editing it. If you save the record, you will overwrite the changes the other user made.

Copying the changes to the clipboard will let you look at the values the other user entered, and then paste your changes back in if you decide to make changes.

Record: 1 of 2004 No Filter Search

cepting the new value (Drop Changes) or of forcing your changes to overwrite the change made by the TestLocks form (Save Record). The key to understanding that this process uses optimistic locks is to recognize that even though you began by editing the form, Access still allowed the TestLocks program to change the underlying data.

Action

Create a CustomerLocks form as a datasheet into the Customer table.

Run the form.

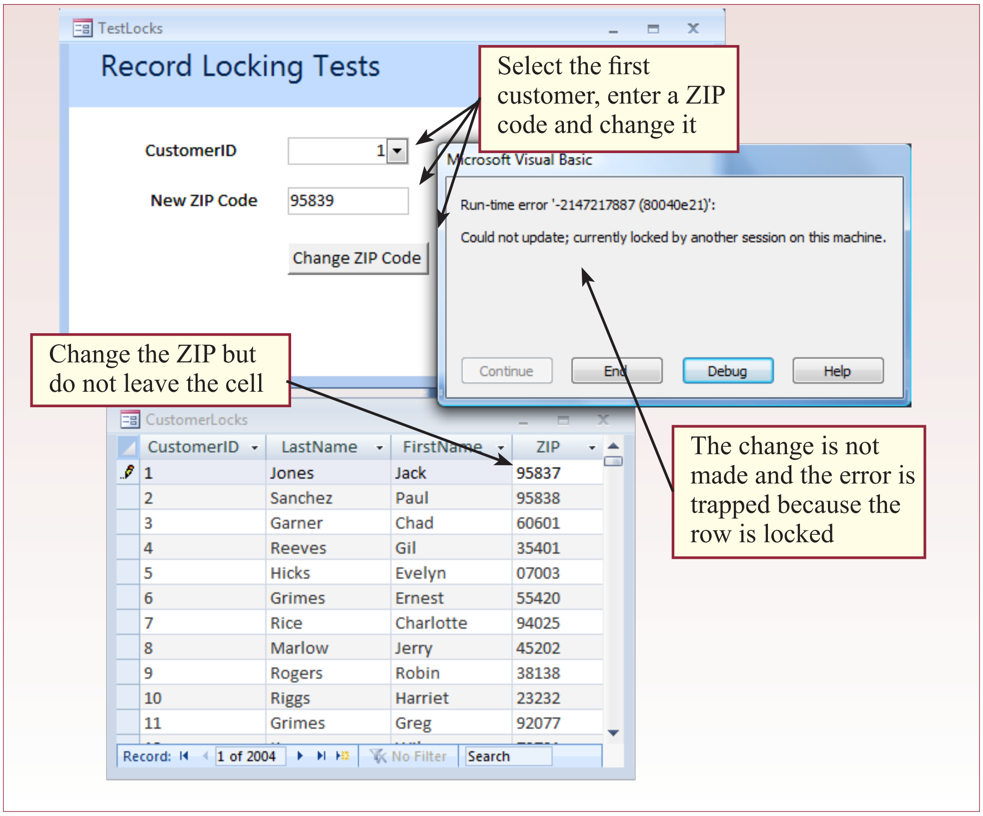
Edit a ZIP code but do not leave the cell.

Switch to the new form and run it for the same CustomerID.

Return to the table and press the Tab key.

You can set pessimistic locks on any form by setting its properties. Switch to the CustomerLocks form in Design view and find the Record Locks property. Currently, it is set to "No Locks." Change this value to "Edited Record," which will assign a pessimistic lock to any row edited on the form. Return to the runtime view of the form. Again, make a change to the ZIP code for the first customer, then click on the TestLocks form. Make sure the first customer is selected and a new ZIP code is specified, then click the Change button. Figure 7.28 shows the result of these steps. The code on the TestLocks form is prevented from making changes to this row of the customer data because the row is locked. The error trapping on the TestLocks form catches the error, skips the update, and displays the error message. If you check the underlying Customer table, you will see that the data has not been changed. Close the error mes-

Figure 7.28


TestLocks
Record Locking Tests
Change ZIP Code
Select the first customer, enter a ZIP code and change it
Microsoft Visual Basic
Continue
End
Debug
Help
Change the ZIP but do not leave the cell
CustomerLocks
The change is not made and the error is trapped because the row is locked
Record: 1 of 2004
No Filter
Search

CustomerID	1
New ZIP Code	95839

Run-time error '-2147217887 (80040e21)':

Could not update; currently locked by another session on this machine.

CustomerID	LastName	FirstName	ZIP
1	Jones	Jack	95837
2	Sanchez	Paul	95838
3	Garner	Chad	60601
4	Reeves	Gil	35401
5	Hicks	Evelyn	07003
6	Grimes	Ernest	55420
7	Rice	Charlotte	94025
8	Marlow	Jerry	45202
9	Rogers	Robin	38138
10	Riggs	Harriet	23232
11	Grimes	Greg	92077

sage box and return to the CustomerLocks form. Press the Tab key to switch to a new row and complete the update.

Action

Switch to Design view on the change form and add the specified error-handling code.

Open the CustomerLocks form in Design view and set the Record Locks property to Edited Record.

Run both forms.

Change a ZIP Code in CustomerLocks.

Enter a new ZIP Code in the change form for the same customer.

The choice of pessimistic or optimistic locking depends on how often you expect collisions to arise and on how you want to handle them. Optimistic locking is considerably more efficient and is the preferred solution. However, all of your update code will need an error-handling section that recognizes an update has occurred and makes a decision of how to proceed. Figure 7.27 shows that forms automatically handle the problem by displaying the two main choices. Essentially your code would have to create the same form. For example, you could create a more sophisticated message box in the TestLocks error section that offers the two main choices. Figure 7.29 shows the foundations of the code. The error handler displays the message and lets the user choose to retry or cancel the operation. The Cancel option discards the current changes and allows the changes made by the other process to remain. The Retry option cancels the current operations to be safe and then restarts the changes which should now succeed (unless some process intervenes again).

This code is difficult to see in operation and hard to test. To trigger the error, you would have to add a delay in the middle of the update code, which would give you enough time to switch to the other form and make an intervening change. Unfortunately, Access does not have an internal Wait function, so you would have to

Figure 7.29

```
RetryUpdate:
   rst.Open SQL, cnn, adOpenDynamic, adLockOptimistic
   rst("ZIP") = NewZIPCode
   rst.Update
   rst.Close

Exit_cmdNewZipCode_Click:
   Exit Sub

Err_cmdNewZipCode_Click:
   If (MsgBox(Err.Description, vbRetryCancel) = vbCancel) Then
      Resume Exit_cmdNewZipCode_Click
   End If
   rst.Cancel
   rst.Requery
   Resume RetryUpdate
```

write one using the Timer function. But if you have time, you should test the code to see that it correctly handles changes under optimistic locking.

Exercises

Many Charms

Inventory control is a critical success factor for determining profitability at Many Charms. Madison and Samantha need to watch the quantity on hand—particularly for the high-cost items. The suppliers are a complicating factor. Some of them are known for being inconsistent in delivering items ordered. As a result, Samantha and Madison have to carefully check every shipment they receive and cross-match it to the orders. Many times the shipment is missing items, and once in a while, the companies send items that were not ordered. These items have to be returned, but the supplier billing is just as bad. Madison has to watch the supplier bills continually to ensure that they are billed only for items they actually ordered and received. As a result of problems, she also wants to track the unordered items that were sent back, so if they show up on a bill, she can provide the details of when the item was returned.

1. Create a form to handle purchase orders to suppliers. Create a second form to handle received shipments. Be sure that it can handle receipt of partial orders and track the day that each partial order arrives. It must also handle receipt of unordered items (which should be stored in a separate table).
2. Add a button to the Received Orders form so that if they receive an interesting unordered item, it can be added to the orders and inventory and paid for. Create it as an entirely new order and be sure to handle optimistic locks and transactions.
3. Create a form that enables Madison to select product and metal categories, and then enter a percentage price increase. Write the SQL update code so that this increase is applied to the list price of the selected categories.
4. The company ships orders to three states, each of which charge different rates of sales tax. Write a function that takes the state code and the amount and returns the tax due.
5. Create a form and write a program that for a given type of charm and type of metal, computes the average of (1) the number of days between sales of that item, and (2) the average number of days between purchase orders for that item.

Standup Foods

While food items and celebrities are important aspects of the business, the day-to-day operations depend on managing the employees. In particular, Laura wants to reward the workers who continue to do well. The evaluation and rating system she has implemented is a major component of this plan. Now, she has to set up the system to make it easy to use so everyone can enter the necessary data. She also needs a way to analyze the data to help managers select the best employees for the next job, and to reward people who do well.

1. Create a form to enter data about an event, with an emphasis on the jobs performed by the employees and their evaluations. Make sure the form includes the revenue received from the event, the costs, and the dates involved. Create a separate form to enter and display data about employee specializations.

2. Create a form for Laura that lets her select a job category and then displays the top-rated employees in that category. (Hint: Create a subform and modify its Record Source query using code.) Create a text box so Laura can enter an average rating as a cutoff value. Create a second text box so Laura can enter a percentage raise increase. Add a button and write the code to give that raise increase to all of the selected employees.

3. Sometimes managers need to hire part-time workers on the spot. Create a simple form that lets managers add basic employee data without allowing them to see or change data for other employees.

4. Workers often want to estimate how much money they will make after all withholdings are deducted. Calculating withholdings is a complex process, but create a simple version to use as an estimate. The function should include number of exemptions, wage rate, and hours worked as inputs. It returns an estimate of the take-home pay. Use sample paychecks or research the Internet to estimate the tax withholding based on the number of exemptions. Create a simple form so employees can plug in these three values and receive the estimate.

5. Laura needs to provide some documentation to the bankers regarding the firm's growth. Create a new table with columns for month, revenue, costs, and percent change for revenue and cost. Write a query to compute the total revenue and costs per month and insert those values into the new table. Write a cursor-based program to compute the percent changes and insert the values into the appropriate columns.

EnviroSpeed

Tracking the knowledge of the workers and experts along with recording the experiences obtained in the many cleanup situations is a primary element of the company. You need to create forms that make it easy for workers to enter the data and knowledge gained. However, for the company to stay in business, you also need to track costs and revenue. Revenue is generally straightforward—the company bills based on the underlying costs, but payments are generally received over time. You will need a form to record receipt of payments by the customers.

1. The company is trying to standardize its fee structure. Write a function that has inputs for the cost of the crews, the cost of expert time, the cost of chemicals, transportation costs, the cost of equipment, and miscellaneous costs. Compute a billing fee based on a percentage profit from each of these costs (crews: 20 percent, experts: 30 percent, chemicals: 15 percent, transportation: 10 percent, equipment: 50 percent, miscellaneous: 15 percent). Also include a $50,000 fixed cost for overhead.

2. Create a form that enables managers to quickly put together a crew in an emergency. The form will have selection boxes for specialty and years of experience (subtract date hired from today). Clicking a button will retrieve a list of crew members meeting the desired conditions. Double-clicking on a name should add that person to the crew required for this disaster.

3. In the middle of an incident, crew members still need to record all of the details so they can be retrieved later. Create a form that enables them to enter the needed information. Be sure to include a way to quickly add a list

of chemicals encountered in the incident. They should primarily be able to select from a known list, but they sometimes encounter new chemicals. Be sure to control for concurrency, since several people may be entering data at the same time.

4. Write a program that evaluates payments by each customer. Assuming payments are due at the end of each month, assesses an interest charge of one-half percent of the outstanding balance. Also, assess a late fee of $200 for each month that a payment is late. Automatically add these values to the customer's balance. Note: You will have to enter several payments and late or missing payments to test the function.
5. Enter enough sample incident data to cover at least a year. Write a cursor-based program to calculate and display the percent increase in revenue per month.

Final Project

The main textbook has an online appendix with several longer case studies. You should be able to work on one of these cases throughout the term. If you or your instructor picks one, perform the following tasks.

1. Make the forms easier to use by automating as many tasks as possible.
2. Examine the case for situations where you can use SQL to update records selected by the users. For example, consider price increases, employee raises, and automated inventory orders.
3. Look for potential reports that require comparing data over time. Write the cursor-based code to generate the necessary change data.

Chapter 8

Applications

Chapter Outline

Objectives

- Build applications that connect forms and reports.
- Add toolbars and menus to forms.
- Add Help files to the database application.

Applications

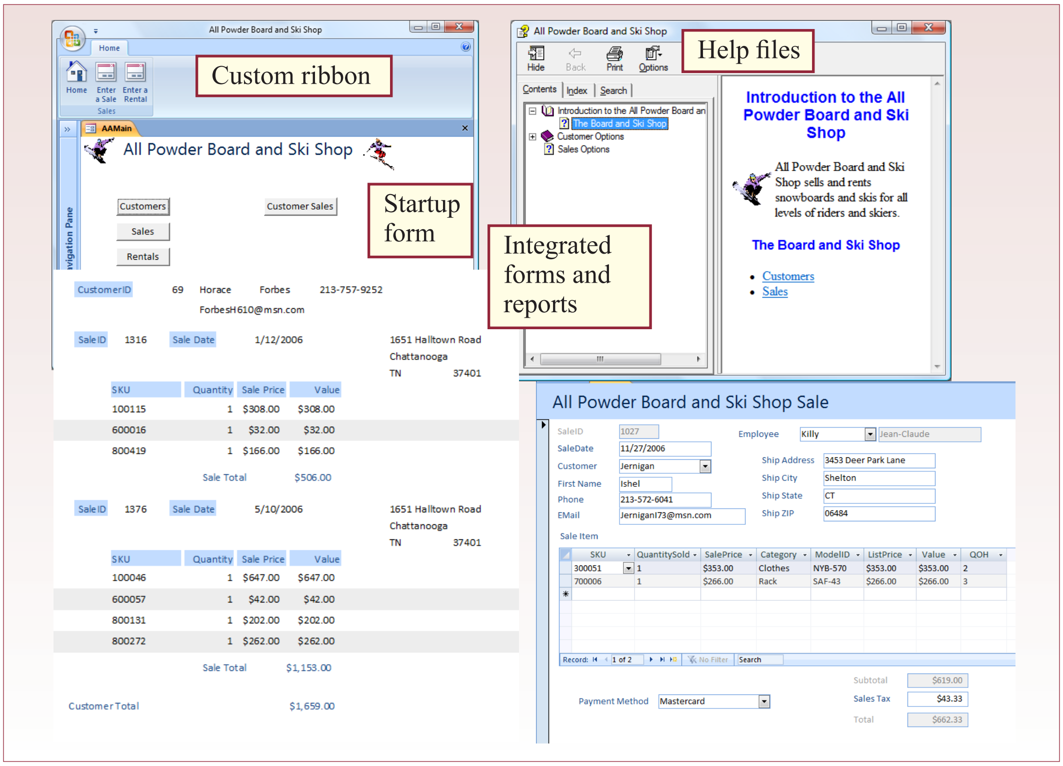
The main purpose of the DBMS is to store data efficiently and provide queries to retrieve data to answer business questions. But from the perspective of businesses, the true value of the DBMS lies in the applications that can be built on top of the database. Chapter 6 shows you how to build the forms and reports that make up the heart of an application. This chapter shows you the additional steps needed to make the application integrated and easy to use.

A finished application contains all of the forms and reports needed to solve a particular problem. It also needs finishing touches such as menus and other navigation links between forms. Additionally, you usually have to create Help files to provide assistance to users when they first learn the system.

Case: All Powder Board and Ski Shop

The primary application at All Powder Board and Ski Shop is the need to track sales and rentals. Of course, these applications also require you to build forms and reports for inventory items and customers as well. As shown in Figure 8.1, these forms and reports are integrated into a common style and structure. A startup form is often used to direct users to the rest of the application. Buttons are used to link to forms and reports. You can also create custom menus to highlight the main operations available to users on a particular form. Finally, you need to build help files to provide additional information or instructions to users.

Figure 8.1


All Powder Board and Ski Shop
Home
Home
Enter a Sale
Enter a Rental
Sales
Custom ribbon
AAMain
Navigation Pane
All Powder Board and Ski Shop
Customers
Customer Sales
Sales
Rentals
Startup form
All Powder Board and Ski Shop
Hide
Back
Print
Options
Help files
Contents
Index
Search
Introduction to the All Powder Board an
The Board and Ski Shop
Customer Options
Sales Options
Integrated forms and reports
Introduction to the All Powder Board and Ski Shop
All Powder Board and Ski Shop sells and rents snowboards and skis for all levels of riders and skiers.
The Board and Ski Shop
Customers
Sales

CustomerID	69	Horace	Forbes	213-757-9252
		ForbesH610@msn.com		

SaleID 1316 Sale Date 1/12/2006 — 1651 Halltown Road, Chattanooga, TN 37401

SKU	Quantity	Sale Price	Value
100115	1	$308.00	$308.00
600016	1	$32.00	$32.00
800419	1	$166.00	$166.00
	Sale Total		$506.00

SaleID 1376 Sale Date 5/10/2006 — 1651 Halltown Road, Chattanooga, TN 37401

SKU	Quantity	Sale Price	Value
100046	1	$647.00	$647.00
600057	1	$42.00	$42.00
800131	1	$202.00	$202.00
800272	1	$262.00	$262.00
	Sale Total		$1,153.00

Customer Total $1,659.00

All Powder Board and Ski Shop Sale

SaleID	1027	Employee	Killy — Jean-Claude
SaleDate	11/27/2006	Ship Address	3453 Deer Park Lane
Customer	Jernigan	Ship City	Shelton
First Name	Ishel	Ship State	CT
Phone	213-572-6041	Ship ZIP	06484
EMail	JerniganI73@msn.com		

Sale Item

SKU	QuantitySold	SalePrice	Category	ModelID	ListPrice	Value	QOH
300051	1	$353.00	Clothes	NYB-570	$353.00	$353.00	2
700006	1	$266.00	Rack	SAF-43	$266.00	$266.00	3

Record: 1 of 2 No Filter Search

Payment Method: Mastercard

Subtotal	$619.00
Sales Tax	$43.33
Total	$662.33

Lab Exercise

Building the All Powder Application

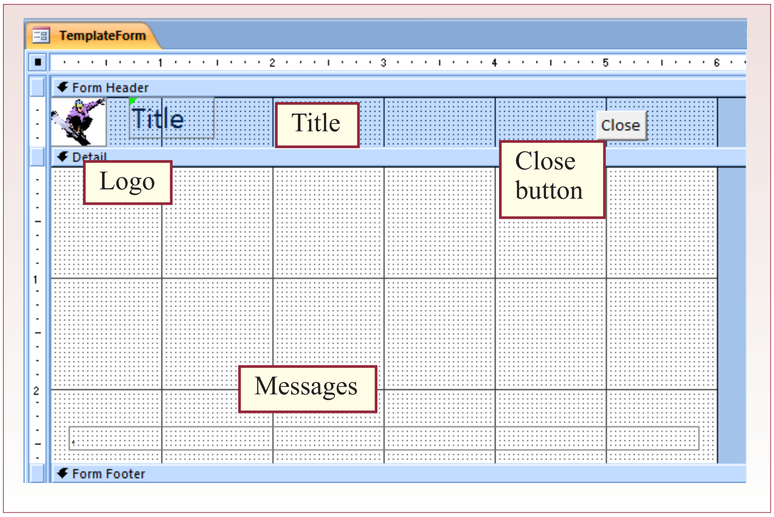
Integrating the forms and reports is the first major step in creating the application. You need to identify the tasks performed by various user groups. With this knowledge, you can sets of forms and reports that match the tasks of each group. While you are integrating the forms and reports, you should also make all of them consistent. Actually, you should create a design template and standard for an application before you begin creating forms and reports. The template contains the primary elements that you want on every form, such as a menu, logo, title, and perhaps a Close button. A design standard spells out details such as the fonts, page sizes, margins, colors, and naming conventions.

Microsoft Access has several predefined formats that you could choose to ensure consistency across your application. Technically, it is possible to edit these styles or create your own. Microsoft even provides an unsupported tool to help you. Search for the AutoFormat Editing Tool or AutoFormat Editor.zip. You would then have to publish the changes to every developer's computer, but it does simplify editing the styles. Once they styles have been created or modified, they are used the same as the existing styles, so the process is not discussed in this chapter.

One of the drawbacks to the styles within Access is that they can be applied only when the form or report is first built. Consequently, they work best if you choose a style and layout before you start building any forms and reports. It is also difficult to include common elements, such as icons and standard buttons. For these items, you need to create a base form (and report) and copy the elements whenever you create a new form.

In some cases, you might want to create a database template (accdt file extension). You can define tables, or connections to a corporate server, along with basic forms and reports. Users can build a new database using this template and simply add the new forms or reports they might need for their specific application. For this level of customization, you should download the free 2007 Access Developer Extensions (ADE) from the Microsoft Web site.

Figure 8.2



Activity: Create a Style

Action

Create a blank form in Design View.

Add a logo and sample title.

Add a button to close the form.

Add a label that can be used to display messages to the user.

Set the message label's Visible property to No.

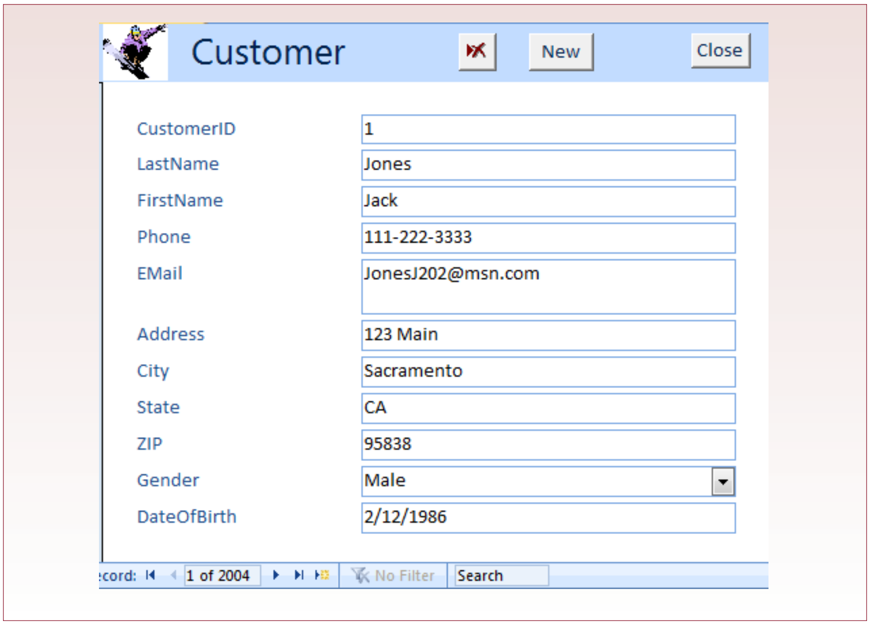
Apply the template form to the Customer form and save it as CustomerD.

When you create a form or a report with the wizard, one of the last steps is choosing an overall design. If one of these designs matches the typical styles for your organization, you can take a big step forward by requiring all developers to use the same style.

On the other hand, if you do want to create a relatively complex style, you could create a blank form that contains the elements you want. You can then copy the data elements from an existing form onto the new template. You will still have to assign properties to the various textboxes and labels, but the paintbrush tool makes this step relatively easy. Figure 8.2 shows the primary elements that you will usually include on any form. You can create this form in design view and save it as TemplateForm.

Unfortunately, the wizard cannot apply this template form when it builds a new form. It also cannot build the data elements onto this form. So, if you want to use it you have two choices: (1) Copy the data elements from an existing form onto this one, or (2) Copy these common elements from the template form onto the form created by the wizard. Your choice of methods depends on the degree of customization you create in the template form. Beyond the basic elements placed on the form, the sample template simply sets the background property to white. In this situation, it is easiest to use the wizard to generate a new form and then paste the common elements onto that form. A related approach is to write a short program that can assign more complex formats automatically. Essentially, the code

Figure 8.3



CustomerID	1
LastName	Jones
FirstName	Jack
Phone	111-222-3333
EMail	JonesJ202@msn.com
Address	123 Main
City	Sacramento
State	CA
ZIP	95838
Gender	Male
DateOfBirth	2/12/1986

loops through all of the text boxes and labels on the form and applies formats in terms of colors and font sizes.

Open the existing Customer form and copy the various template elements onto the form. Set its background color to white. Note that copying a button does not automatically copy the code for the button. You will have to manually copy the code, or use the wizard to rebuild it. Figure 8.3 shows the resulting Customer form with the template objects. You have to follow a similar process for all of the other forms. You should also follow a similar process to set design standards for reports.

This approach is not the most elegant method of establishing styles, but lack of templates is one of limitations in Microsoft Access. Your only other approach is to write a detailed style sheet that specifies all of the background colors, font sizes, and form layout elements. Then every developer will follow the style guide to ensure that all forms have the same appearance.

Activity: Create the Startup Form

Action

Create a blank form in Design view.

Set the properties to remove the scroll bars and navigation buttons.

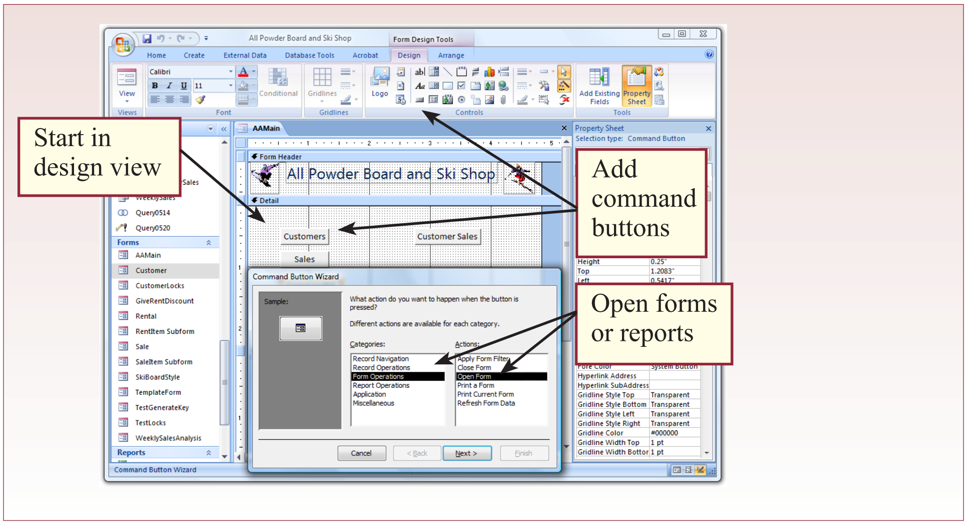
Add command buttons to open other forms (Form Operations/Open Form) and reports.

Use colors and graphics to make the form appealing.

Run the form and test all buttons.

Once you have created the forms and reports, you need to combine them into an application. A Startup or Switchboard form is a key element of an application. It is a form that contains links to the other forms and reports. Generally, it is easy to create; the challenge lies in determining how to organize all of the forms and reports. In most cases, users will see the application only through your forms. They will almost never want to open forms directly from the Access forms list. You have to create a structure, beginning with the startup form that guides them through their tasks. This process will often include links on other forms as well. You will have to test this sequence with the users to make sure that it matches their job workflow.

Figure 8.4



What action do you want to happen when the button is pressed?

Different actions are available for each category.

Categories:	Actions:
Record Navigation	Apply Form Filter
Record Operations	Close Form
Form Operations	Open Form
Report Operations	Print a Form
Application	Print Current Form
Miscellaneous	Refresh Form Data

To create a switchboard form, begin with a new form in Design view. Avoid the wizard because it severely limits your form. Add labels to place a title on the form. Use the form's properties to improve the appearance of the form, removing the navigation buttons and perhaps adding a picture. Then add buttons that open the initial forms and reports.

In large applications, you might want to create additional forms that would function as submenus. For instance, you might have a special menu form just for administrators. This approach reduces clutter on the main form and hides details from other users to make their job easier. Figure 8.4 shows the main steps used to add a command button. In the subsequent steps, you choose the form to be opened (Sales), and give the button a descriptive label instead of an icon. The wizard also gives you the opportunity to name the function to match the button (cmdSales). Giving the functions meaningful names makes it is easier to find the appropriate routines. You should also add command buttons to the other forms so the users have all of the necessary commands readily available. For example, you should include a button on the Sales form to open the Customer form. In this case, use the wizard option to open the form for the matching customer. You can assign the menu form to open when the database starts using the Office icon. Click the Office icon and click the Access Options button. Pick the Current Database options and choose the startup form in the Display Form combo box. You can use the same form to assign an application title and icon if desired. When you close the form and restart, the main menu will open.

Note that Access contains a wizard to build a Switchboard form. This form does approximately the same thing as the startup form. However, you should avoid using it. The wizard provides minimal control over layout and style. More importantly, the resulting form is exceedingly difficult to modify. The approach described in this section is considerably more flexible and it is easy to modify as your application changes.

Activity: Customizing Ribbons or Menus

Startup forms and command button links help users navigate from one form to another, but in complex applications, users might need additional support. Office 2007 replaced standard menus with ribbons. Although the ribbons are difficult to customize by users, application developers can modify or completely replace the ribbon options in Access applications. The ribbons provide a convenient location for quick links to common activities that are needed in any form. For instance, you can include a Print button for all reports, so users always know they can click one button to print whatever report they are viewing. Replacing the existing ribbon also hides common database commands from the user—providing some protection by making it harder for users to change things they should not change, such as table designs. However, remember that if users hold down the shift key while starting your database, all

Action

Create a table with columns: RibbonName (text) and RibbonXML (memo).

Save it as USysRibbons

Add a row of data: Name=Home, XML=Figure 8.5

Close and restart the database

Create code in Module1 to handle events

In VBA Editor use Tools/References to add Microsoft Office 12 Object Library

Use the Office Button to assign the new ribbon.

Close and restart the database.

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
 <ribbon startFromScratch="true">
  <tabs>
   <tab id="tabMain" label="Home">
    <group id="grpSales" label="Sales">
     <button id="cmdHome" label="Home"
       onAction="GoToStartup" size="large"
       imageMso="OpenStartPage" supertip="Return to home page." />
     <button id="cmdSales" label="Enter a Sale"
      imageMso="AccessFormModalDialog" size="large"
      onAction="OpenNewSalesForm" />
     <button id="cmdRentals" label="Enter a Rental"
      imageMso="AccessFormModalDialog" size="large"
      onAction="OpenNewRentalsForm" />
    </group>
   </tab>
  </tabs>
 </ribbon>
</customUI>
```

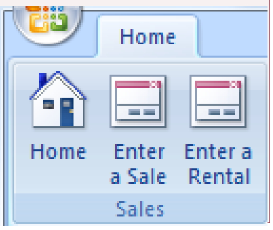


Figure 8.5

startup options will be bypassed, leaving the standard ribbon options. This trick is actually useful if you happen to replace a ribbon and need to get back to the original commands. The challenge is to identify the tasks and options that users need to have available on the toolbar.

Ribbons are relatively powerful and can contain icons, text boxes, checkboxes, toggle buttons, and similar controls. In a sense, each tab on a ribbon represents a miniature form. You add controls and then provide code that responds to various events, such as a click by the user. You can even create the ribbon options dynamically by specifying program code to be called at various stages in the construction of the ribbon. These options are described in the Microsoft documentation on the MSDN Web site. You can find it by searching for "Customizing the 2007 Office Fluent Ribbon for Developers." The example provided in this lab is much simpler and just shows the basics of creating and implementing a new ribbon.

Like many other features in Office, ribbons are based on XML. The exact layout of the XML code is described in the Microsoft documentation, but you can create a simple tab with a few basic items. However, the first question you face is where to store the XML. Access provides a simple method—create a special table to hold the name of the ribbon tab and its XML code. You can also store the XML in a file or build it programmatically and call a special function to load it into the database. The table method is easier.

Figure 8.5 shows the XML code to create a new Home tab and populate it with three buttons. These buttons will eventually be hooked to code that opens the indicated forms. The first step is to create a table to hold the XML that defines the ribbon. Then you can write the short function to make the buttons work. Create a table in design mode. Add two columns: RibbonName and RibbonXML. The name column must be of type text and the XML must be of type memo. You can set RibbonName as a primary key. Save the table with the special name: USysRibbons. Because of the special name (USys), you will not see it in the list of tables. If you want to see it or edit it, click the Office Button, Access Options, Current Database and click the Navigation Options button. Set the check box to Show System Objects.

```
' Use Tools/References to add the Microsoft Office 12 Object Library

Public Sub GoToStartup(control As IRibbonControl)
DoCmd.OpenForm "AAMain"
End Sub

Public Sub OpenNewSalesForm(control As IRibbonControl)
DoCmd.OpenForm "Sale", acNormal, , , acFormAdd
End Sub

Public Sub OpenNewRentalsForm(control As IRibbonControl)
DoCmd.OpenForm "Rental", acNormal, , , acFormAdd
End Sub
```

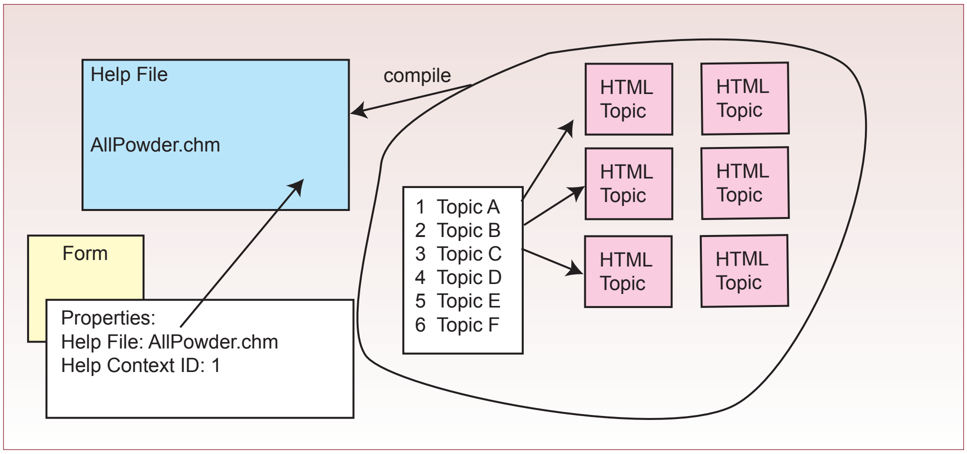
Figure 8.6

Data for each tab is stored in one row of the USysRibbons table. Enter Home as the name of the ribbon and enter the XML code from Figure 8.5 exactly as shown. If you make a mistake in the code, the ribbon will not work and you will not receive an error message.

You can activate the ribbon for the entire application or assign it to a single form. Ribbon buttons generally apply to the entire application. But first you have to close and restart the database so that it rereads the ribbon table. When it restarts, click the Office Button and Access Options. Scroll down the Current Database page until you see the Ribbon Name option. Choose the new Home ribbon in the combo box. Close the form and close and restart the database for the changes to take effect. When you restart the database, your new Home tab should be visible. If it is not, you most likely made a mistake in the XML code.

Your buttons will not work yet because you still have to write the code (or macros) to handle the click events. VBA code is more powerful and easier to edit. Open the Module1 that was created earlier to hold public functions. You will probably have to set a reference to the ribbon objects, so Click Tools/References and scroll down to enable the Microsoft Office 12 Object Library. Figure 8.6 shows the three subroutines you need to add. Their names must match the names you entered in the XML code and they must be declared as Public. Ultimately, you could add any code to the routines, but these buttons simply need to open the specific forms, so you can use the DoCmd object. When the functions are defined, you can

Figure 8.7



test your new ribbon buttons. Simply click the buttons and ensure the matching form opens.

Ribbons can be relatively complex and contain several types of controls. However, be careful to not confuse the users. The icons used in the example are part of the Microsoft Office group and you can find a list on the Internet if you search for imageMso or imgMso. It is also possible to create your own icons, but you have to use a different loading process.

You can also control the Navigation pane so that it highlights your forms and reports as business applications. Microsoft has a Marketing Projects sample database (MarketingProjects.exe) that you can download to see more options with ribbons and the navigation pane.

Activity: Write Help Files

Action

- Create at least three HTML help files for the All Powder forms using an HTML editor or Wordpad.
- If necessary, download and install the HTML Help workshop.
- Create a new project in the workshop.
- Add the HTML files.
- Edit the HTML files to add keywords.
- Create the mapping file with a text editor and add it to the project.
- Set project options to build the TOC and index files.
- Compile and test the help file.
- Edit the database forms and add the help file name and ContentID.
- Run the form and press the F1 key to test the help files.

A finished application also needs customized Help files. Users should be able to press the F1 key or select the Help menu option and receive additional information to help them perform a task or understand the data that needs to be entered. Detailed Help systems can become complex, with large applications requiring hundreds of pages of Help text and instructions. On large projects, companies often hire a special team just to create and edit the Help files. For these situations, you will want to purchase a dedicated Help System editor. However, Microsoft has a free Help Compiler system that can be used to create Help files. You can write the Help text without this system, but you need it to compile the files into a Help package. Search the MSDN site for the htmlhelp.exe file. Figure 8.7 shows the basic steps involved in creating a help system. First you write individual help pages as HTML text files. These pages can have links to each other and to external websites. One of the pages should be the startup page, and each page should contain a list of keywords. You also create a mapping file that assigns a number to each page. The Help Compiler converts all of the pages into a single chm file. Finally, in each Access form, you set the properties to the name of this compiled form, and the number of the topic associated with that page or even a particular control. When the user presses the F1 key, the system looks up the page that matches the number and displays it in the Help viewer. Users can also search by table of contents or by keywords.

Figure 8.8 shows that you can create Help pages using a simple text editor, or you can use most HTML editors. You should create a style sheet to ensure consistency across all of the files. More importantly, use the H1, H2, and H3 heading tags to define the major topics covered in each page. These headings can be used by the Help compiler to generate the table of contents. The keywords are entered in the special <OBJECT> tag. This tag can be created using the Microsoft HTML Help editor, or you can copy, paste, and edit the keyword information. At this

```
<Object type="application/x-oleobject"
classid="clsid:1e2a7bd0-dab9-11d0-b93a-00c04fc99f9e">
   <PARAM name="Keyword" value="Contents">
   <PARAM name="Keyword" value="Introduction">
   <PARAM name="Keyword" value="Start">
   <PARAM name="Keyword" value="Management">
</OBJECT>
<HTML><HEAD>
<TITLE>All Powder Board and Ski Shop</TITLE>
<LINK rel="stylesheet" type="text/css" href="Styles.css">
</HEAD><BODY>
<H1>Introduction to the All Powder Board and Ski Shop</H1>
<TABLE><TR>
<TD><IMG SRC='BoardLogo1.gif' border='0'></TD>
<TD>All Powder Board and Ski Shop sells and rents snowboards and skis for
all levels of riders and skiers.</TD>
</TR></TABLE>
<H2>The Board and Ski Shop</H2>
<UL>
<LI><A HREF='Customers.html'>Customers</A></LI>
<LI><A HREF='Sales.html'>Sales </A></LI>
</UL>
</BODY></HTML>
```

Figure 8.8

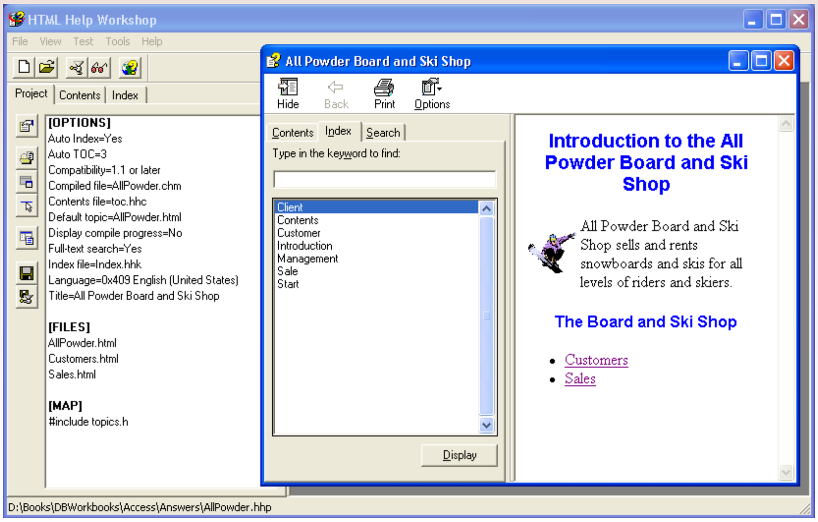
point, you should create two or three HTML files and test the pages and the links to make sure they work together.

Once you have created the individual HTML pages, you should create the mapping file that assigns a number to each topic. In HTML, you refer to each topic by the name of the file, but Access references topics by a number. As shown in the sample in Figure 8.9, you can assign almost any number (it uses a long integer), but it helps if you group the numbers by topic to make them easier for you to find later. This data is stored as a simple text file. It is typically named "topics.h," but that is not a strict requirement. Be careful with the entries in the topics.h file: You must separate the names from the numbers with at least two spaces and you cannot use tabs.

Once the pages have been created, you compile the files into a single chm file that is distributed with the Access mdb file. The Help compiler does most of the work, you simply add the files to the system and set a few options. Begin by starting the HTML Help Workshop and creating a new project file. Use the Menu button to add the html topic files to the project. Use the API menu button to add the topics.h header file to the project under the Map option. Use the Options menu button to set the title and default file for the project. Use the Files tab to tell it to automatically create the contents file and include the keywords from the HTML files to build the index. You might have to click the tabs for Contents and Index to force the compiler to create new versions of these files. Figure 8.10 shows the basic elements of the HTML Help compiler and the resulting compiled Help file.

Figure 8.9

```
#define AllPowder        100
#define Customers        10000
#define Sales            20000
```



```
[OPTIONS]
Auto Index=Yes
Auto TOC=3
Compatibility=1.1 or later
Compiled file=AllPowder.chm
Contents file=toc.hhc
Default topic=AllPowder.html
Display compile progress=No
Full-text search=Yes
Index file=Index.hhk
Language=0x409 English (United States)
Title=All Powder Board and Ski Shop

[FILES]
AllPowder.html
Customers.html
Sales.html

[MAP]
#include topics.h
```

Client
Contents
Customer
Introduction
Management
Sale
Start

Introduction to the All Powder Board and Ski Shop

All Powder Board and Ski Shop sells and rents snowboards and skis for all levels of riders and skiers.

The Board and Ski Shop

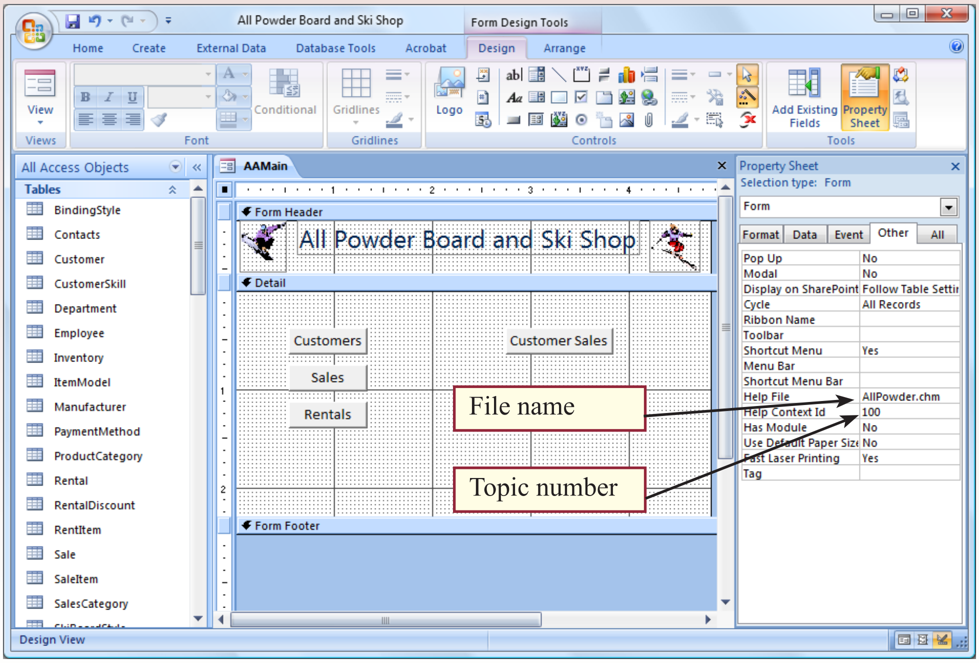
- Customers
- Sales

D:\Books\DBWorkbooks\Access\Answers\AllPowder.hhp

Figure 8.10

The final step is to set up the Access forms so that they open the correct Help file to the correct page. Keep your list of mapping numbers handy, you will have to enter them into the form properties. Figure 8.11 shows the two properties (file-name and topic number) that have to be set. These values have to be entered onto every form. You can also assign different topics to individual controls on a page, but remember that this means you have to write more Help topics. Make sure that you place the Help file in the same folder as the database file. When you have set

Figure 8.11



Tables
BindingStyle
Contacts
Customer
CustomerSkill
Department
Employee
Inventory
ItemModel
Manufacturer
PaymentMethod
ProductCategory
Rental
RentalDiscount
RentItem
Sale
SaleItem
SalesCategory

Property Sheet
Selection type: Form
Form
Format | Data | Event | Other | All

Pop Up	No
Modal	No
Display on SharePoint	Follow Table Settir
Cycle	All Records
Ribbon Name	
Toolbar	
Shortcut Menu	Yes
Menu Bar	
Shortcut Menu Bar	
Help File	AllPowder.chm
Help Context Id	100
Has Module	No
Use Default Paper Size	No
Fast Laser Printing	Yes
Tag	

these properties for all of the forms, you should run the application and test each page to ensure that it brings up the correct topic. Someone should also proofread all of the topics to make sure they contain no errors.

Activity: Deploy an Application

Action

Test your application.

Copy the accdb and chm files to a second computer.

Add a desktop shortcut to the application.

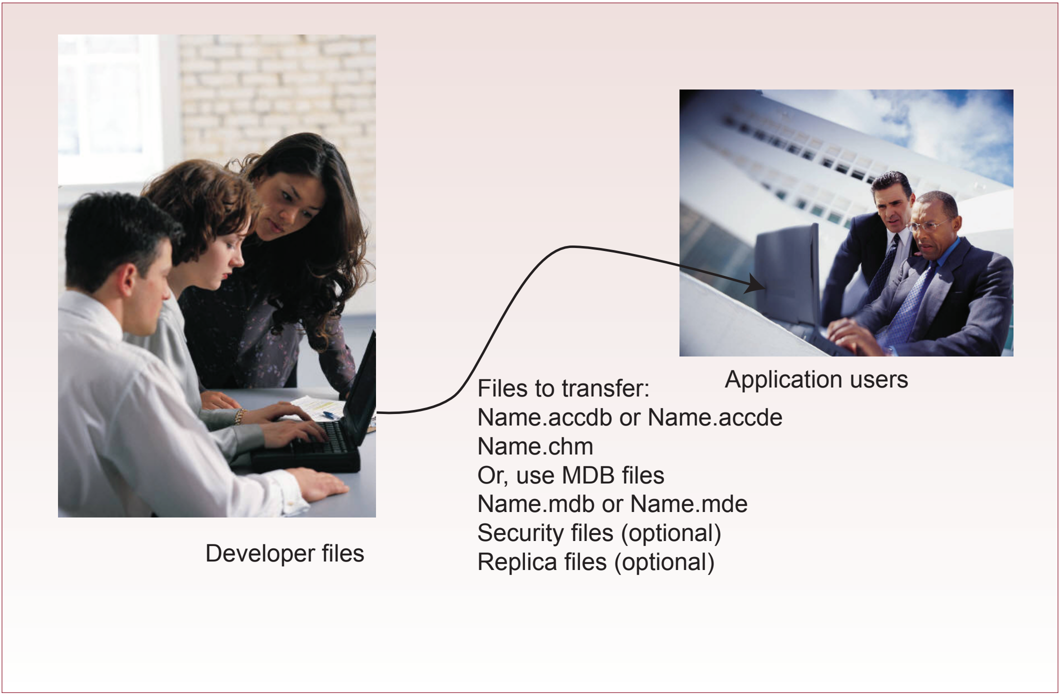
Run the application and test it.

If you have the Access runtime, use its installation software to build an installation file and test it.

Building an application is relatively complicated. You have to create all of the forms and reports with a consistent format, tie them together with a startup form, create menus and toolbars, and write help files. As shown in Figure 8.12, because Access stores most of this work in one file (plus the .chm help file), it is relatively easy to install the application. As long as you do not care about security or need to worry about distributed databases. These topics are covered in Chapters 10 and 11, and both result in the creation of additional files that need to be distributed and installed.

The easiest installation process arises when you do not need to worry about security controls, and you have a single mdb and single chm file. You can just copy the two files into a folder on the user's computer. Be sure the user has read and write permissions to that folder as controlled by the operating system. Add an icon to the desktop, set the desired startup options (Tools/Startup), and you are finished. On the other hand, you should test your application on a second computer before you try to give it to a user. One of the problems you have to watch for is to be sure the help file links point to just the name of the help file—without any drive or path names.

Figure 8.12



Another problem you are likely to encounter is that each computer running your application will need a licensed copy of Microsoft Access installed. Usually you do not want users to run Microsoft Access and alter your application. Is there a way to install the application without providing a full copy of Microsoft Access? Chapter 10 explains how you can create an MDE file so that users cannot alter your forms, reports, and code; even if the user does have a full copy of Access. However, even with an MDE file, users still need a full copy of Microsoft Access. To solve that problem, you can get a copy of the Microsoft Access Runtime system. It is sometimes called the Developer Extensions and is part of the Visual Studio Tools for Microsoft Office. The easiest way to buy the tool is to become a member of the Microsoft Developer's Network (MSDN). This tool includes a set of files that runs your application without needing to install the full version of Access. It also includes a tool to package all of the files into an installation program. For any project that requires multiple files or installation on more than a couple of computers, you should probably get this tool just to help with the installation tasks.

Exercises

Crystal Tigers

The Crystal Tigers club is mostly interested in tracking members and events. The officers who will use the system do not know much about computers, but they can enter data into forms. They are also interested in a few key reports. For instance, they want to be able to get totals for the amount of hours their members devoted to charity events. They also want monthly summaries of the amount of money raised. The vice president also wants to be able to print a simple listing of the officers with their phone numbers and e-mail addresses. Sometimes she also wants a similar list of members who have participated in the initial steps of an event. She wants to be able to carry the list with her when the event starts so she knows who to contact if problems arise.

1. Create a design template and standardize the forms and reports.
2. Build the forms and reports into an application with a start up form.
3. Create the Help files for the system; remember that the users have limited computer experience.
4. Copy your files to a second computer, add a desktop icon, and test the application.

Capitol Artists

Job tracking is the most important aspect of the application needed by Capitol Artists. In particular, the employees need to be able to quickly select a job and enter the time and expenses for the task performed. This data is then used to create a monthly billing report for the client. Consequently, you need to focus on creating the forms to capture this data. You need to make sure they are fast and easy to use. The managers also want weekly reports showing the hours and money generated by each employee so they can use the data for personnel evaluations.

1. Create a design template and standardize the forms and reports.
2. Build the forms and reports into an application with a start up form.

3. Create Help files for the system.
4. Copy your files to a second computer, add a desktop icon, and test the application.

Offshore Speed

Special orders have always been a complex problem for the Offshore Speed managers. Customers come to the shop because it is one of the few that can obtain the custom parts they want. But the company has always had problems training employees to collect all of the order data and to keep track of getting the orders placed and delivered in a timely manner. Some of these orders include contracts with other local firms to perform customization and finish work on the boats. Although these firms do excellent work, most are terrible at keeping records. Consequently, the managers want to use the system to generate reports on individual boats for each contract shop that can be used to remind the other owners of the details. The company also needs reports on the inventory status of the specialized parts. They have trouble keeping some items in stock, and other items seem to sit on the shelves forever. They have no good way of keeping track at the moment.

1. Create a design template and standardize the forms and reports.
2. Build the forms and reports into an application with a start up form.
3. Create Help files for the system.
4. Copy your files to a second computer, add a desktop icon, and test the application.

Final Project

The main textbook has an online appendix with several longer case studies. You should be able to work on one of these cases throughout the term. If you or your instructor picks one, perform the following tasks.

1. Define a form template and standards for consistency.
2. Build the forms and reports into an application with a start up form.
3. Build a toolbar that makes the application easier to use.
4. Create Help files for the system.
5. Copy your files to a second computer, add a desktop icon, and test the application.

Chapter 9

Data Warehouses and Data Mining

Chapter Outline

Objectives

- Extract data from spreadsheets and import it into a data warehouse.
- Create and browse an OLAP cube.
- Analyze time series data.
- Analyze geographic data.
- Analyze data with statistical tools.

Data Warehouse

Data warehouses have evolved because of the need for online analytical processing (OLAP) and its conflicts with online transaction processing (OLTP). The goal of a data warehouse is to hold consistent data, possibly obtained from several sources, which can be quickly searched and analyzed. Unfortunately, Access was never designed to function as a data warehouse, and it does not include much in the way of analysis functions. However, it is extremely flexible at importing and exporting data, which makes it easy to extract and transform data from other sources. In particular, it can interact with Microsoft Excel, which does have many tools for analyzing data. Also, either as a standalone tool or in conjunction with Excel, the PivotTable is a powerful OLAP cube browser.

Case: All Powder Board and Ski Shop

Like most businesses, the managers of All Powder need to analyze data to spot trends and to solve problems. One of the most challenging aspects of a board and ski shop is the huge variety of inventory needed. As vendors produce even more styles and variations, it becomes difficult to stock all of the items in a collection of sizes. Yet, if the store does not have the desired items in stock, it will lose sales. This balancing act between inventory costs and sales revenue has destroyed many other firms. The owners of All Powder are committed to running a large enough shop so that they can afford to carry a good selection of snowboards and skis. However, managers need to constantly evaluate styles and products so items can be cleared out if needed. For that analysis, one of the main tools they need is an OLAP cube browser or PivotTable that shows sales split by several features and categories. Figure 9.1 lists some of the main dimensions that managers want to examine in terms of sales. Because they are not certain about the validity of the last three, they are displayed with question marks.

Managers also occasionally raise some more challenging statistical questions, such as whether customers who rent equipment are likely to buy that equipment, and whether skiers buy specific types of poles or boots with their skis. They also need to forecast sales by categories. In particular, they often argue about whether certain styles are increasing or decreasing in popularity. Some of these analyses might require the help of a statistician to build a formal model, but the managers would at least like to see some rough analyses.

Figure 9.1

Sales Dimensions
State (ship)
Month
Category
Style
SkillLevel
Size
Color
Manufacturer
BindingStyle
WeightMax?
ItemMaterial?
WaistWidth?

Lab Exercise

All Powder Board and Ski Shop

As organizations grow over time, the internal processes undergo changes, data also changes, systems improve, and number systems rarely stay the same. Consequently, most information systems consist of a mix of technologies and databases. Rarely is the data consistent across all of these systems. Before the database was created for the All Powder shop, the managers kept limited records in Microsoft Excel. These records are not perfect. They are organized by Sales and by Rentals, and the data is not normalized. Also, the records focused primarily on the equipment and did not keep data on customers. From our modern database perspective, the records are a pain—but at least they are electronic and not paper so you do not have to enter all of the data by hand.

Nonnormalized data is common in business, and you will often be asked to convert this data into a relational database. Fortunately, you can use the power of SQL as a magical “super tool” to impress mere mortals with your skills.

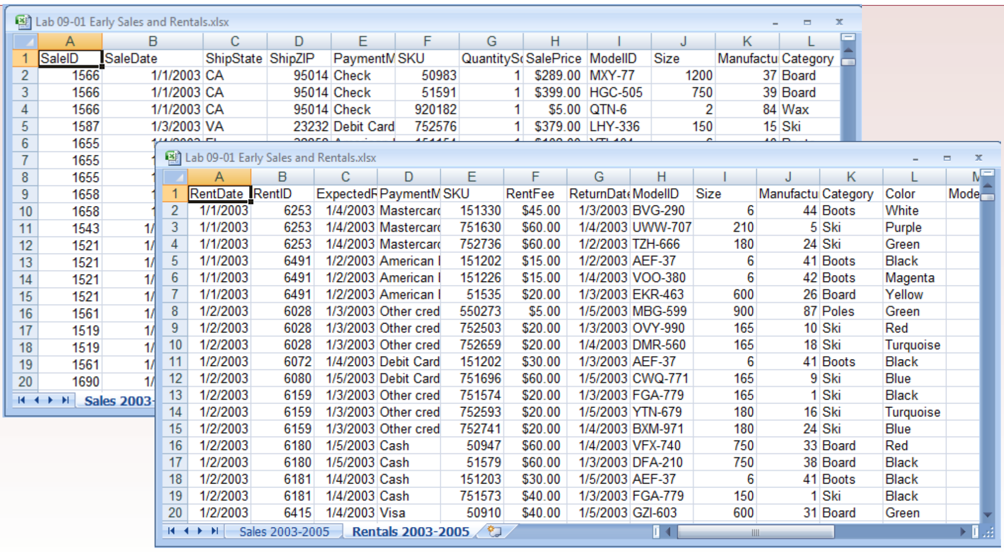
Activity: Extract and Transform Data

> **Action**
>
> Choose External Data/Import/Excel.
>
> Read the Sales spreadsheet into a new table.
>
> Read the Rental data into another new table.
>
> Create new entries for CustomerID and EmployeeID equal to zero.

If you need to save time in the lab, you can skip the first section and use the sample Lab9 database that already has the old data loaded. But, remember that with data warehouse projects, a great deal of time is spent on cleaning and importing data

Figure 9.2 shows the layout of the data in the two worksheets. Again, notice that lack of normalization. Each row represents an item being sold or rented. Fortunately, the worksheets repeat the SalesID and RentalID so you can still recover which items are grouped onto

Figure 9.2



Lab 09-01 Early Sales and Rentals.xlsx

	A	B	C	D	E	F	G	H	I	J	K	L
1	SaleID	SaleDate	ShipState	ShipZIP	PaymentM	SKU	QuantitySc	SalePrice	ModelID	Size	Manufactu	Category
2	1566	1/1/2003	CA	95014	Check	50983	1	$289.00	MXY-77	1200	37	Board
3	1566	1/1/2003	CA	95014	Check	51591	1	$399.00	HGC-505	750	39	Board
4	1566	1/1/2003	CA	95014	Check	920182	1	$5.00	QTN-6	2	84	Wax
5	1587	1/3/2003	VA	23232	Debit Card	752576	1	$379.00	LHY-336	150	15	Ski
6	1655											
7	1655											
8	1655											
9	1658											
10	1658											
11	1543											
12	1521											
13	1521											
14	1521											
15	1521											
16	1561											
17	1519											
18	1519											
19	1561											
20	1690											

Lab 09-01 Early Sales and Rentals.xlsx

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	RentDate	RentID	ExpectedF	PaymentM	SKU	RentFee	ReturnDate	ModelID	Size	Manufactu	Category	Color	Mode
2	1/1/2003	6253	1/4/2003	Mastercard	151330	$45.00	1/3/2003	BVG-290	6	44	Boots	White	
3	1/1/2003	6253	1/4/2003	Mastercard	751630	$60.00	1/4/2003	UWW-707	210	5	Ski	Purple	
4	1/1/2003	6253	1/4/2003	Mastercard	752736	$60.00	1/2/2003	TZH-666	180	24	Ski	Green	
5	1/1/2003	6491	1/2/2003	American I	151202	$15.00	1/2/2003	AEF-37	6	41	Boots	Black	
6	1/1/2003	6491	1/2/2003	American I	151226	$15.00	1/4/2003	VOO-380	6	42	Boots	Magenta	
7	1/1/2003	6491	1/2/2003	American I	51535	$20.00	1/3/2003	EKR-463	600	26	Board	Yellow	
8	1/2/2003	6028	1/3/2003	Other cred	550273	$5.00	1/5/2003	MBG-599	900	87	Poles	Green	
9	1/2/2003	6028	1/3/2003	Other cred	752503	$20.00	1/3/2003	OVY-990	165	10	Ski	Red	
10	1/2/2003	6028	1/3/2003	Other cred	752659	$20.00	1/4/2003	DMR-560	165	18	Ski	Turquoise	
11	1/2/2003	6072	1/4/2003	Debit Card	151202	$30.00	1/3/2003	AEF-37	6	41	Boots	Black	
12	1/2/2003	6080	1/5/2003	Debit Card	751696	$60.00	1/5/2003	CWQ-771	165	9	Ski	Blue	
13	1/2/2003	6159	1/3/2003	Other cred	751574	$20.00	1/3/2003	FGA-779	165	1	Ski	Black	
14	1/2/2003	6159	1/3/2003	Other cred	752593	$20.00	1/5/2003	YTN-679	180	16	Ski	Turquoise	
15	1/2/2003	6159	1/3/2003	Other cred	752741	$20.00	1/4/2003	BXM-971	180	24	Ski	Blue	
16	1/2/2003	6180	1/5/2003	Cash	50947	$60.00	1/4/2003	VFX-740	750	33	Board	Red	
17	1/2/2003	6180	1/5/2003	Cash	51579	$60.00	1/3/2003	DFA-210	750	38	Board	Black	
18	1/2/2003	6181	1/4/2003	Cash	151203	$30.00	1/5/2003	AEF-37	6	41	Boots	Black	
19	1/2/2003	6181	1/4/2003	Cash	751573	$40.00	1/3/2003	FGA-779	150	1	Ski	Black	
20	1/2/2003	6415	1/4/2003	Visa	50910	$40.00	1/5/2003	GZI-603	600	31	Board	Green	

a single sale or rental. Likewise, they repeated the descriptive item data for each time the model was sold. To be really safe, you should eventually check to see that they were consistent in recording this data. For example, ModelID BVG-290 might have been given different descriptions at different times. If there are many inconsistencies of this type, it will be difficult and time-consuming to clean up this data. Most of the corrections would have to be handled manually, unless you have a third source of data that you know is correct. These are the types of problems you often face when extracting data from diverse systems.

The first step in extracting and transforming this data is to get it into the database where you can use SQL to work on it. Fortunately, Access can quickly read a worksheet into a new table. Use File/Get External Data/Import to find the spreadsheet, select the worksheet, and read it into a new table. You should give the table a simple name, such as OldSales because spaces and hyphens will make it harder to type later. Read the sales and rental worksheet data into separate temporary tables.

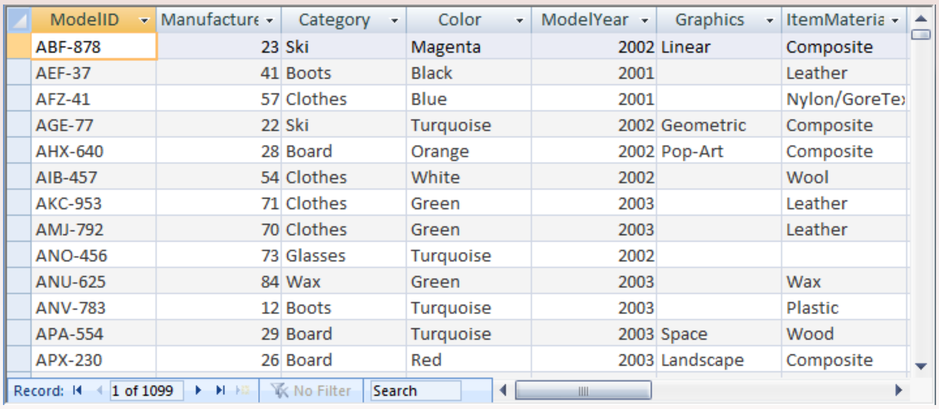
Looking through the temporary Sales table, you will see that the data needs to be split into four tables: SaleItem, Sale, Inventory, and ItemModel. Go back and examine the relationships for those tables, and you will see that because of the dependencies, you will have to enter data first into the tables for ItemModel, Inventory, Sale, and finally SaleItem. The relationships and foreign keys require that data be entered in that order. You must also be careful with the Customer and Employee data. If you try to create a row in the Sale table, the system will automatically try to set a value of zero for the CustomerID and EmployeeID. But there is no matching data for a zero ID in either of these tables. So either you try to force a blank CustomerID and EmployeeID, or you create a new Customer and new Employee called “walk-in” and “staff.” This latter approach is slightly better than relying on blank data. Your first task, therefore, is to create these new entries in the respective tables. As shown in Figure 9.3, if you use SQL in a Design View query, you can force the ID values to zero, which makes these anonymous entries easy to find.

Before moving on to the next steps that will alter the data in your main tables, it is critical that you make a backup copy of the database. Go to Windows now and copy the database file so you can return to this point if anything major goes wrong.

SQL makes it relatively easy to extract the new model data and copy it to the ItemModel table. The first step is to create a SELECT query that retrieves the model data from the temporary tables and removes the duplicates. This process is slightly complicated because of the two tables. It is possible that a product has been sold but not rented and vice versa. The easiest way to handle this problem is to write two queries and use UNION to combine the results. Figure 9.4 shows the basic query to retrieve the model attributes from the OldSale table. Recall that you have to add the DISTINCT keyword by hand. Move this query to the side and

Figure 9.3

```
INSERT INTO Customer (CustomerID, LastName)
Values (0,'Walk-in')

INSERT INTO Employee (EmployeeID, LastName)
Values (0,'Staff')
```



ModelID	Manufacture	Category	Color	ModelYear	Graphics	ItemMateria
ABF-878	23	Ski	Magenta	2002	Linear	Composite
AEF-37	41	Boots	Black	2001		Leather
AFZ-41	57	Clothes	Blue	2001		Nylon/GoreTe
AGE-77	22	Ski	Turquoise	2002	Geometric	Composite
AHX-640	28	Board	Orange	2002	Pop-Art	Composite
AIB-457	54	Clothes	White	2002		Wool
AKC-953	71	Clothes	Green	2003		Leather
AMJ-792	70	Clothes	Green	2003		Leather
ANO-456	73	Glasses	Turquoise	2002		
ANU-625	84	Wax	Green	2003		Wax
ANV-783	12	Boots	Turquoise	2003		Plastic
APA-554	29	Board	Turquoise	2003	Space	Wood
APX-230	26	Board	Red	2003	Landscape	Composite

Record: 1 of 1099 No Filter Search

SELECT **DISTINCT** OldSales.ModelID, OldSales.ManufacturerID, OldSales.Category, OldSales.Color, OldSales.ModelYear, OldSales.Graphics, OldSales.ItemMaterial, OldSales.ListPrice, OldSales.Style, OldSales.SkillLevel, OldSales.WeightMax, OldSales.WaistWidth, OldSales.BindingStyle
FROM OldSales;

Figure 9.4

build a similar one from the OldRentals table. Be extremely careful to list the columns in exactly the same sequence.

Add the data rows from the two queries with the UNION statement. Figure 9.5 shows the basic structure of the query, but yours will contain several more columns. Save this query as qryOldModels so you can use it as one set of data.

Action

Create a new query in Design view.

Retrieve DISTINCT data from the new sales table.

Retrieve DISTINCT data from the new rental table.

Connect the two queries with a UNION.

Save the query.

Now that you can retrieve the new model data, it is relatively easy to write a query to insert these rows into the base ItemModel table. Build a new SELECT query using the qryOldModels query with all of its columns. Add the DISTINCT keyword to be absolutely certain that all duplicates are removed. Run the query to make sure it retrieves the data. As shown in Figure 9.6, switch to SQL view and at the top of the query, add the phrase: INSERT INTO Item Model (ModelID, …). Because you do not have data for all of the columns, you must list them in the parentheses, in the order of the columns being selected. Run the query and all of the new models will be added to the ItemModel table.

Follow a similar process to add the SKU, ModelID, and Size columns to the Inventory table. Note that you should set the QuantityOnHand to zero for each of these items since the store probably does not have any of the old models in stock. If they do happen to have a few of the older items around, the quantity can be

Figure 9.5

```
SELECT DISTINCT ModelID, ManufacturerID, Category, …
FROM OldSales
UNION
SELECT DISTINCT ModelID, ManufacturerID, Category, …
FROM OldRentals
```

```
INSERT INTO ItemModel (ModelID, ManufacturerID, Category, Color,
ModelYear, Graphics, ItemMaterial, ListPrice, Style, SkillLevel, WeightMax,
WaistWidth, BindingStyle)
SELECT DISTINCT qryOldModels.ModelID, qryOldModels.ManufacturerID,
qryOldModels.Category, qryOldModels.Color, qryOldModels.ModelYear,
qryOldModels.Graphics, qryOldModels.ItemMaterial, qryOldModels.ListPrice,
qryOldModels.Style, qryOldModels.SkillLevel, qryOldModels.WeightMax,
qryOldModels.WaistWidth, qryOldModels.BindingStyle
FROM qryOldModels;
```

Figure 9.6

entered by hand later. Figure 9.7 shows the final step that inserts the data into the Inventory table. Notice the use of the column alias to force a zero value into the QuantityOnHand column for each row.

The Sale and Rental data is considerably easier because they are separate, and

```
INSERT INTO Inventory (ModelID, SKU, Size, QuantityOnHand)
SELECT DISTINCT qryOldInventory.ModelID, qryOldInventory.SKU,
qryOldInventory.Size, 0 As QuantityOnHand
FROM qryOldInventory;
```

Figure 9.7

you will not need the UNION command to merge the two sets of data. In fact, you can copy the Sales (or Rental) data with one SQL command. First, build a query to retrieve the distinct sales data from the OldSales table. Be sure to include the DISTINCT keyword in the SELECT statement. After you test the SELECT statement, add the INSERT INTO line above it. Figure 9.8 shows an additional trick that is often helpful. Remember that the SaleID column is AutoNumber, so if you added new rows of data to your Sale table, it might have generated values that would conflict with the values from this older dataset. To avoid this problem, you can add an offset number to the old SaleID (+5000 in this example). If you choose a large enough offset, this step will ensure that all of the new ID values will be safe. However, you must also remember to add the same calculation in the final step of transferring the SaleItem rows.

Action

Create a new query that retrieves DISTINCT values from the saved UNION query.

Verify that it works.

Add an INSERT INTO statement above the SELECT statement to copy the data to the ItemModel table.

Run the query.

Use a similar process to add SKU, ModelID, and Size to the Inventory table.

Follow a similar process to copy the Sales, Rental, SalesItem, and RentalItems tables.

Figure 9.8

```
INSERT INTO Sales (SaleID, SaleDate, ShipState, ShipZIP, PaymentMethod)
SELECT DISTINCT OldSales.SaleID+5000, OldSales.SaleDate, OldSales.
ShipState, OldSales.ShipZIP, OldSales.PaymentMethod
FROM OldSales;
```

```
INSERT INTO SaleItem (SaleID, SKU, QuantitySold, SalePrice)
SELECT DISTINCT OldSales.SaleID+5000, OldSales.SKU, OldSales.
QuantitySold, OldSales.SalePrice
FROM OldSales;
```

Figure 9.9

Figure 9.9 shows that the query for the SaleItem table is almost identical to the query that copied the sale data, but with slightly different columns. Remember that if you transform the SaleID in the Sale table, you must make exactly the same transformation for the SaleItem table. Otherwise, the data will never match and cannot be joined. If you forget, you will most likely receive several error messages. But some of the data might be joined to your existing Sales data, making it difficult to reverse the query. Finally, you need to do the same two steps for the Rental and RentalItem tables. At this point, you have successfully imported the old data and cleaned it up so it can be used within your database.

Activity: Create PivotTable to Browse the Sales Cube

Action

Create a new query in Design view.

Tables: Sale, SaleItem, Inventory, ItemModel, Manufacturer.

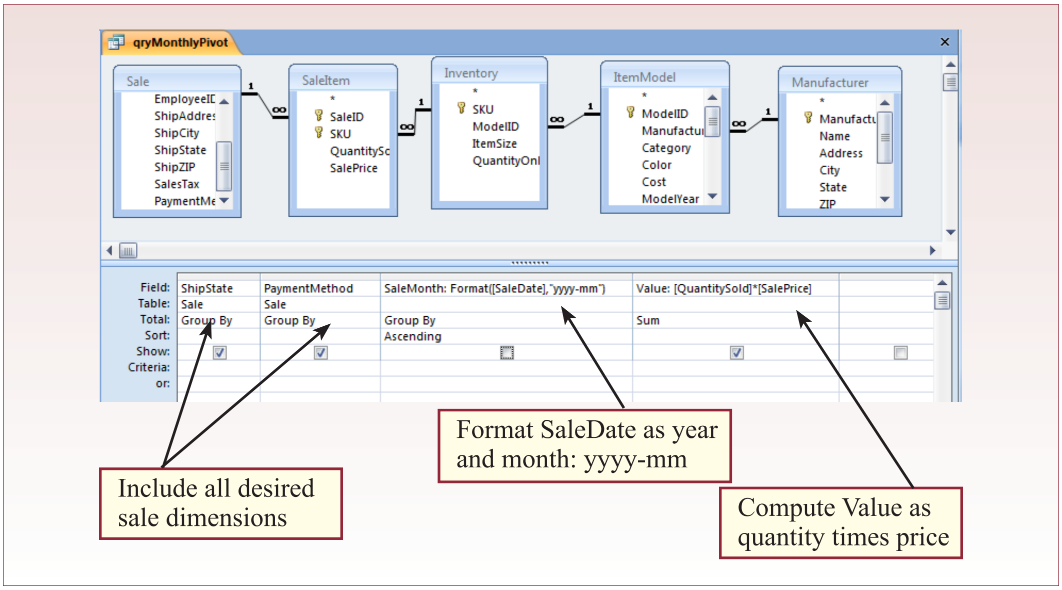
Columns: ShipState, PaymentMethod, SaleMonth with a format of "yyyy-mm", and Value = QuantitySold*SalePrice.

Sum the Value column.

Test the query and save it.

Investigating sales by a variety of dimensions is an important task for the managers and owners of All Powder. It would be difficult to train all of them to build queries to examine all of the items that might be of interest. A faster and more flexible solution is to create an OLAP cube that contains the sales value (Price * Quantity) as the factor, along with the dimensions. Using Microsoft's PivotTable technology, the cube can be manipulated to see subtotals and sort or filter the dimensions. PivotTables can be displayed and manipulat-

Figure 9.10



Field:	ShipState	PaymentMethod	SaleMonth: Format([SaleDate],"yyyy-mm")	Value: [QuantitySold]*[SalePrice]	
Table:	Sale	Sale			
Total:	Group By	Group By	Group By	Sum	
Sort:			Ascending		
Show:	✓	✓	☐	✓	☐
Criteria:					
or:					

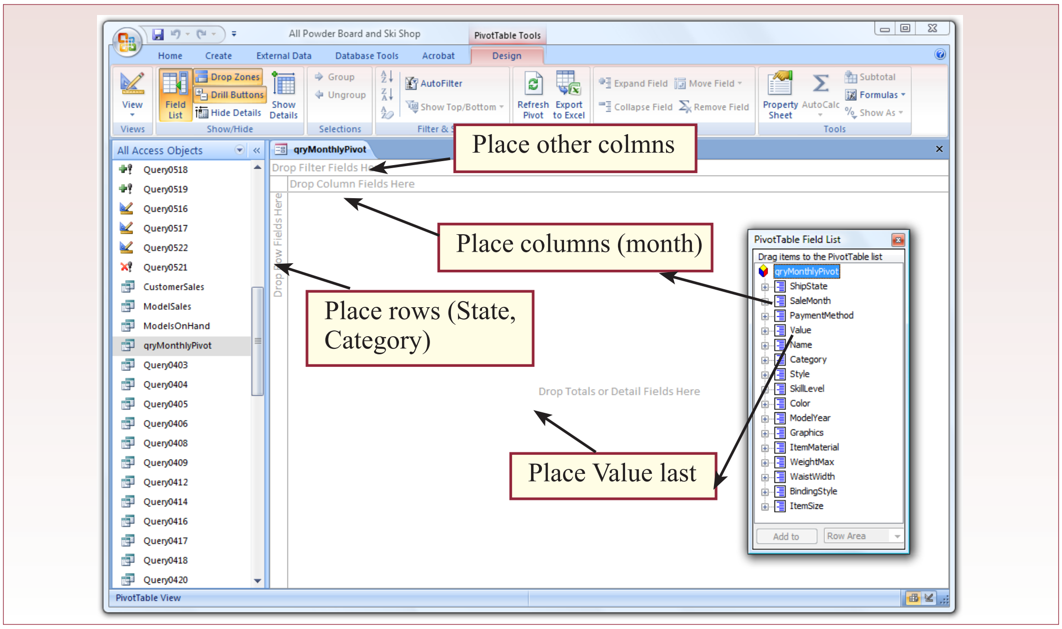


Figure 9.11

ed on Access forms, in Excel spreadsheets, or even on Web pages generated from within Access. You can also create PivotCharts that let managers select the data to be displayed.

Begin the process by creating a query that includes all of sale and model dimensions needed by the managers. Then add the value column. As shown in Figure 9.10, the SaleDate column needs to be formatted to display just the year and the month. The format "yyyy-mm" displays the month as a number so that it sorts correctly. The PivotTable does not directly support hierarchies of data (such as year, quarter, and month) so you have to format the date as the lowest level that you want displayed, then build groups later to provide rollup capabilities. Compute the sum of the value column and leave everything else as Group By columns. You should sort the data by year and month, but it can also be sorted later. Save the query as "qryMonthlyPivot."

Action

Select the qryMonthlyPivot in the Navigation pane.

On the ribbon, find Create/Forms/More Forms/PivotTable.

Drag the SaleMonth and drop it on the column fields position.

Drag the Category and ShipState fields and drop them on the row fields position.

Drag the Value field and drop it in the middle Detail section.

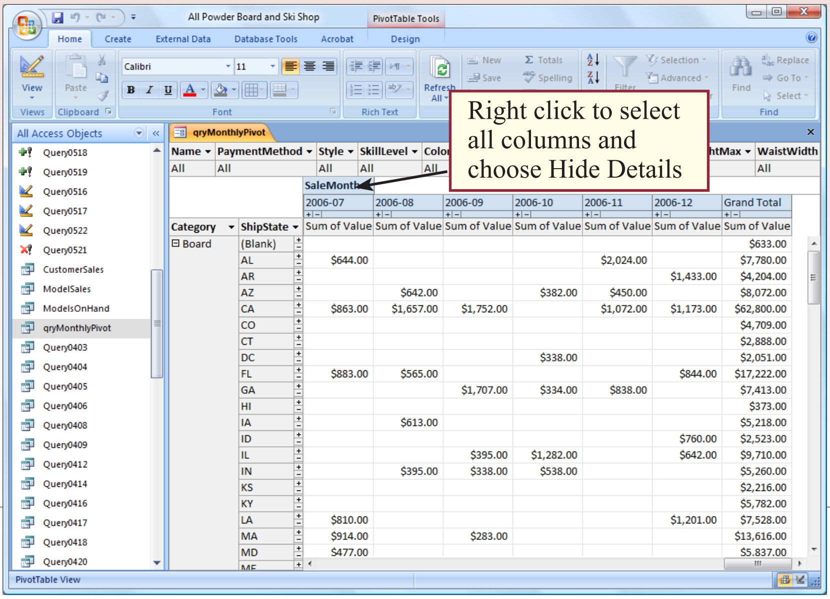
Drag the other fields and drop them at the top as filter fields for future use.

Scroll the main grid to the far right.

Drop the Value field in the Totals column.

With the query in place, you can create the PivotTable form. The easiest approach is to use the PivotTable form wizard. First select the qryMonthlyPivot in the Navigator pane. On the ribbon, select Create/Forms/More Forms/PivotTable.

As shown in Figure 9.11, the PivotTable wizard creates a blank screen with four sections and a list of the columns from the query. The four sections are (1) filter fields, (2) column fields, (3) row fields, and (4) the value field. The first three represent the dimension elements; the fourth is the fact data that you wish to sum.



Name	PaymentMethod	Style	SkillLevel	Color	… htMax	WaistWidth
All	All	All	All	All		All

Category	ShipState	2006-07	2006-08	2006-09	2006-10	2006-11	2006-12	Grand Total
		Sum of Value	Sum of Value	Sum of Value	Sum of Value	Sum of Value	Sum of Value	Sum of Value
Board	(Blank)							$633.00
	AL	$644.00				$2,024.00		$7,780.00
	AR						$1,433.00	$4,204.00
	AZ		$642.00		$382.00	$450.00		$8,072.00
	CA	$863.00	$1,657.00	$1,752.00		$1,072.00	$1,173.00	$62,800.00
	CO							$4,709.00
	CT							$2,888.00
	DC				$338.00			$2,051.00
	FL	$883.00	$565.00				$844.00	$17,222.00
	GA			$1,707.00	$334.00	$838.00		$7,413.00
	HI							$373.00
	IA		$613.00					$5,218.00
	ID						$760.00	$2,523.00
	IL			$395.00	$1,282.00		$642.00	$9,710.00
	IN		$395.00	$338.00	$538.00			$5,260.00
	KS							$2,216.00
	KY							$5,782.00
	LA	$810.00					$1,201.00	$7,528.00
	MA	$914.00		$283.00				$13,616.00
	MD	$477.00						$5,837.00

Figure 9.12

Because of the way the table dynamically adjusts to changes, be sure to place the fact field last.

In this example, begin by placing the SaleMonth field to create the columns. Simply drag it from the list and drop it on the area indicated for columns. The managers primarily want to see the Category and ShipState as rows of the cube. Drag each of those fields to the left side of the screen. If you drag them one at a time, simply drop the remaining fields on top of the first ones you placed. If they are not displayed in the order you want, you can drag them to rearrange the sequence. The other fields, such as Color and Style, might also be used by the managers, but including them would make the table too large. Place all of the other dimensions (not Value) onto the filter location at the top of the table. From here, managers can choose to restrict the display based on various conditions. They can also change the composition of the table by replacing Row or Column fields with a Filter field. Finally, drag the Value field onto the middle of the table.

The next step is to scroll to the right edge of the PivotTable and notice that the Total column is empty. You also have to drag the Value field and drop it onto this column to be able to see subtotals. However, by default, the PivotTable shows too much detail—it tends to compute subtotals for every single item. To hide this unnecessary detail, right click on the SaleMonth heading to select all of the columns, then choose the Hide Details option. Figure 9.12 shows the initial structure of the PivotTable.

The next step is to create the time hierarchy (year-quarter-month) to enable managers to roll up the data or drill down, depending on the detail they want to see. The PivotTable uses Groups to establish the year and quarter elements. You highlight the columns that belong together, right click the highlighted area and choose the group option. It is also important that you use the properties to rename the individual cells so they are meaningful to the managers. To begin the process, you must build the lowest-level groups first (quarters). Click the cell 2003-01, then shift-click the cell for 2003-03 to highlight the first three months.

Right click the highlighted area and choose the Group Items option. A new cell labeled SaleMonth1 will be added to the left of the original SaleMonth cell. Select this new cell and click the Property Sheet option in the Tools tab of the Design menu.Click the Captions tab, and rename the caption to Quarter. Immediately below that cell, you will see a new cell labeled Group 1. Change its caption to 2003-Q1 because it represents the group you just created. Repeat the same process for each three-month interval. Be careful to label the quarters correctly. It is easy to make mistakes: The process becomes tedious because you have to do it for all four years. Note that the last quarter will already be grouped under the "Other" label, you only have to change the label to 2006-Q4. Once the quarters are defined, you can aggregate them into years using the same process. Click the 2003 Q1 cell and right-click the 2003-Q4 cell to highlight all four quarters. Right click to group the items. Notice that a third grouping cell has been added at the top of the form. Rename it to "Year," then rename the Group 1 cell to "2003." Repeat the grouping process for 2004 and 2005. The 2006 data is already grouped; you need only to rename the cell. A faster way to create these date groups is to use the Format function within the query for each level that you need to create. But, it is helpful to know that you can create your own groups as needed.

Action

Create quarterly groups.

Click on the first month and right click on the third month to highlight them.

Right click and select the Group/Group Items option.

Right click on the new cell SaleMonth1 and change its Caption property to Quarter.

Right click on the Group1 cell and set its caption to 2001-Q1.

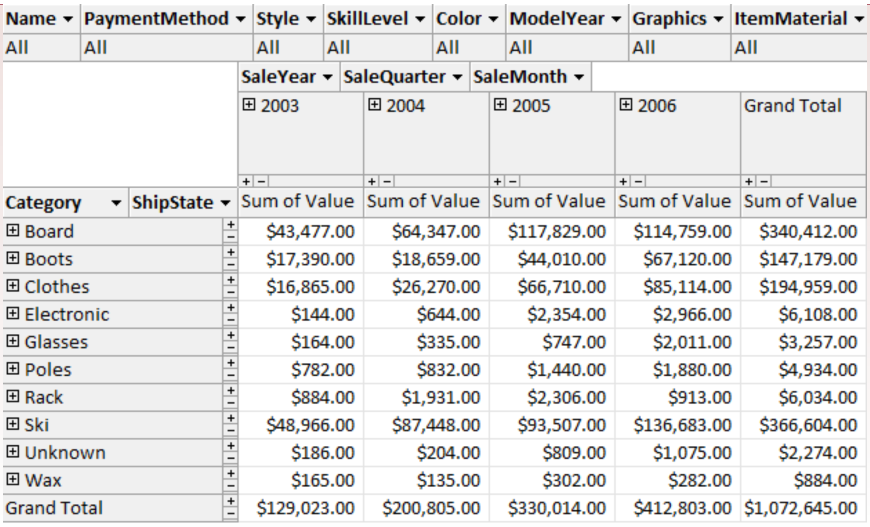
Repeat the process for all quarters.

Use a similar process to group the four years.

Use the Collapse option to reduce the grid display.

To see the purpose of the groups, scroll back to the left side and click the Year cell. On the PivotTable toolbar, click on the Collapse icon (red minus sign), or right-click the year cell and choose Collapse from the menu. The Collapse option rolls up the data and displays the group (year) totals instead of the details

Figure 9.13



Name	PaymentMethod	Style	SkillLevel	Color	ModelYear	Graphics	ItemMaterial
All	All	All	All	All	All	All	All

SaleYear ▾ SaleQuarter ▾ SaleMonth ▾

Category	ShipState	⊞ 2003	⊞ 2004	⊞ 2005	⊞ 2006	Grand Total
		Sum of Value	Sum of Value	Sum of Value	Sum of Value	Sum of Value
⊞ Board		$43,477.00	$64,347.00	$117,829.00	$114,759.00	$340,412.00
⊞ Boots		$17,390.00	$18,659.00	$44,010.00	$67,120.00	$147,179.00
⊞ Clothes		$16,865.00	$26,270.00	$66,710.00	$85,114.00	$194,959.00
⊞ Electronic		$144.00	$644.00	$2,354.00	$2,966.00	$6,108.00
⊞ Glasses		$164.00	$335.00	$747.00	$2,011.00	$3,257.00
⊞ Poles		$782.00	$832.00	$1,440.00	$1,880.00	$4,934.00
⊞ Rack		$884.00	$1,931.00	$2,306.00	$913.00	$6,034.00
⊞ Ski		$48,966.00	$87,448.00	$93,507.00	$136,683.00	$366,604.00
⊞ Unknown		$186.00	$204.00	$809.00	$1,075.00	$2,274.00
⊞ Wax		$165.00	$135.00	$302.00	$282.00	$884.00
Grand Total		$129,023.00	$200,805.00	$330,014.00	$412,803.00	$1,072,645.00

(months). Figure 9.13 shows the PivotTable collapsed on the years and on the Category row dimension. Managers can expand or contract individual years and quarters. They can also use the drop-down boxes on any of the dimensions to quickly display the subtotals for combinations of attributes.

If the managers prefer graphical displays, you can use a similar process to create a PivotChart. Just be careful not to put too many items on the chart at one time—it quickly becomes unreadable. Instead, focus it to one or two of the major dimensions and place the rest into the filter area. From there, managers can move the dimensions onto the chart to see exactly the combinations of data that interest them. For even more flexibility, you can build the same PivotTable in Excel. Managers can use familiar Excel tools to format, chart, or manipulate the data. You can even include a button on a form to open the Excel file directly from the database application.

Also, you should always remember the power of SQL when analyzing data. Although the queries are difficult for managers to create, as a developer you can build some standard queries for them to run. For instance, managers might be interested in seeing a list of customers who have rented but not purchased any skis.

Introductory Data Analysis

Activity: Analyze Time-Series Data

Action

- Create a new Excel spreadsheet.
- Choose Insert/PivotTable.
- Select External data source.
- Click Choose Connection button.
- Browse to your database and select the query built in the previous section.
- Create a PivotTable with the SaleMonth field as rows, the Category field as columns, and Value as the Detail.

Most database systems do not provide sophisticated statistical analysis and data mining as part of the base system. In general, you will have to purchase specialized software to perform the analysis that you want. Some companies sell data mining tools that perform a variety of analyses. Most of these tools are stand-alone and can extract data from various sources. In addition to performing a wide variety of functions, they are also optimized to handle large sets of data.

However, if you do not have these tools available, you can still use Microsoft Excel to perform some fundamental analysis. Excel is somewhat limited by the amount of data it can handle, but it is relatively easy to use, and managers can use typical Excel tools to interact with the results. For other problems, you could write your own data mining tools within the Visual Basic programming language, but you should also consider hiring an expert in statistics and numerical analysis.

The data for the All Powder case is small enough that it can be used within Excel. To illustrate the basic capabilities available, the following analyses rely on the same query that powers the PivotTable (qryMonthlyPivot). Begin with a time-series analysis that examines total monthly sales for some of the main product categories. Ultimately, managers want a chart with time on the horizontal axis, sales on the vertical axis, and lines for two or three product categories. To create this chart, it is best to have a table of data that shows the months as rows, and the columns as the product categories. This table is similar to the PivotTable built in the previous section, but with the rows and columns switched. To create this chart, start Excel and create a new PivotTable, using the same qryMontlyPivot query already saved in the database. On the Excel menu, choose Data/PivotTable. In the wizard, select the option for External data source, then click the Get Data button. Choose the MS Access Database, click the OK button, and then search for the All Powder database. Find the qryMonthlyPivot and move it with all of its columns to the right window. Do not worry about filtering or sorting the data, just return it to Excel. Create the PivotTable on the main page, and you will see a layout screen similar to the one in the previous section. For the time-series analysis, drag the SaleMonth field to the row location, drag the Category field to the columns, and finish by dragging the Value field onto the detail section.

Action

Select the PivotChart option in the ribbon.

In the PivotChart worksheet remove all categories except Boards, Skis, Boots, and Clothes.

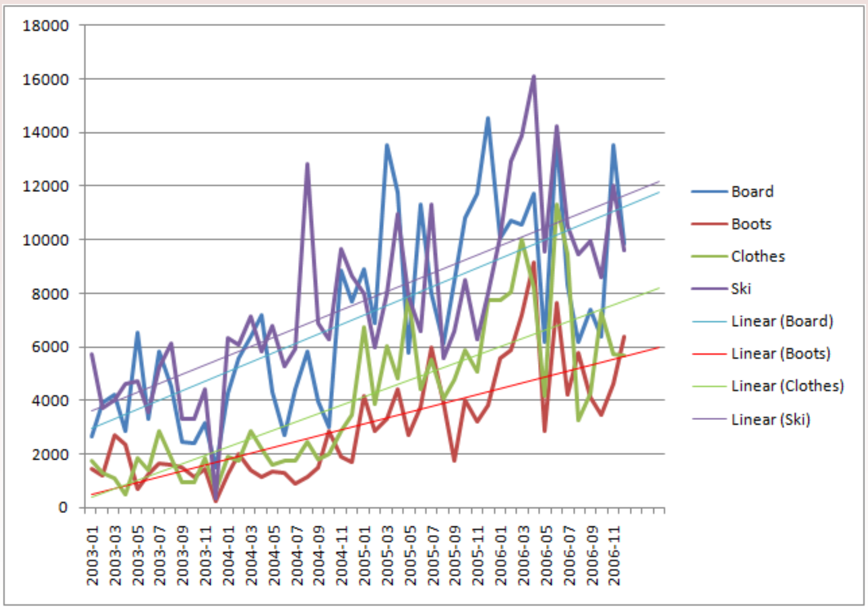
Right click each series and Add Trendline.

Choose a linear trend and forecast it for three periods ahead.

Once the data is in Excel, you can use Excel tools to analyze it. For the time-series analysis of sales, select the PivotChart option in the ribbon and pick a line chart. Excel will create a chart on a new worksheet. You need to modify several properties to make this chart useful. First, remove some of the categories. Click the drop-down arrow on the legend box and uncheck the minor categories, leaving only Boards, Skis, Boots, and Clothes. Each of the categories appears to show an upward trend. To verify that, right-click on one of the data lines and select the Add Trendline option. Choose the linear method and forecast it ahead for three periods. Follow the same procedure for each of the four data series. You should also go back and set the properties of each trend line to match the color of the underlying series.

Figure 9.14 shows the time-series chart with the properties modified to make it readable on paper. The trend lines provide some interesting insight into the sales. First, notice that the overall trends for snowboards and skis are almost identical.

Figure 9.14



Second, observe that although all four categories exhibit increasing sales over time, the sales for boots are not increasing as fast as sales for snowboards and skis. In looking at the properties for the trend lines, notice that you can also display some of the statistics used to generate them. Excel provides several statistical tools to analyze data. For example, managers might be interested to compare the sales of boards and skis more carefully. You can use a T-test to see if there is a significant difference in the means of the sales. Note that you have to install several data analysis add-ins when you install Office. Also, you have to activate the add-ins by selecting them from the Tools/Add-Ins menu option. Once the tools are installed, you can use Tools/Data Analysis to perform the T-test on the two data series. Try it and you will find that because of the large variance in sales, the means are not significantly different. The Data Analysis add-in also provides tools for regression computations and some time-series analyses, such as moving average smoothing.

Action

Make sure the Data Analysis Add-Ins are installed in Excel (Tools/Add-Ins).

Choose Tools/Data Analysis.

Select T-Test Two-sample with unequal variances.

Compare the Board and Ski sales.

Activity: Analyze Geographic Data

The Office 2000 version of Excel contained a mapping tool that made it possible to display some geographic relationships. Its main advantage is that it was bundled with Office. Unfortunately, Office XP does not contain that tool, but Microsoft does ship the MapPoint software that performs even more sophisticated graphs (for a fee). Other companies sell even more sophisticated geographic information system software (notably ArcInfo from ESRI). Although you might not have this software available in your lab, it is worth seeing the steps and the maps to learn some of the capabilities and how it can be used to analyze and explore data.

Action

If you have Microsoft MapPoint start a new Excel worksheet.

Create a PivotTable with rows for ShipState and columns for Board, Boots, Clothes, and Skis.

Create a second worksheet that copies the main data and titles from the PivotTable without the first title rows.

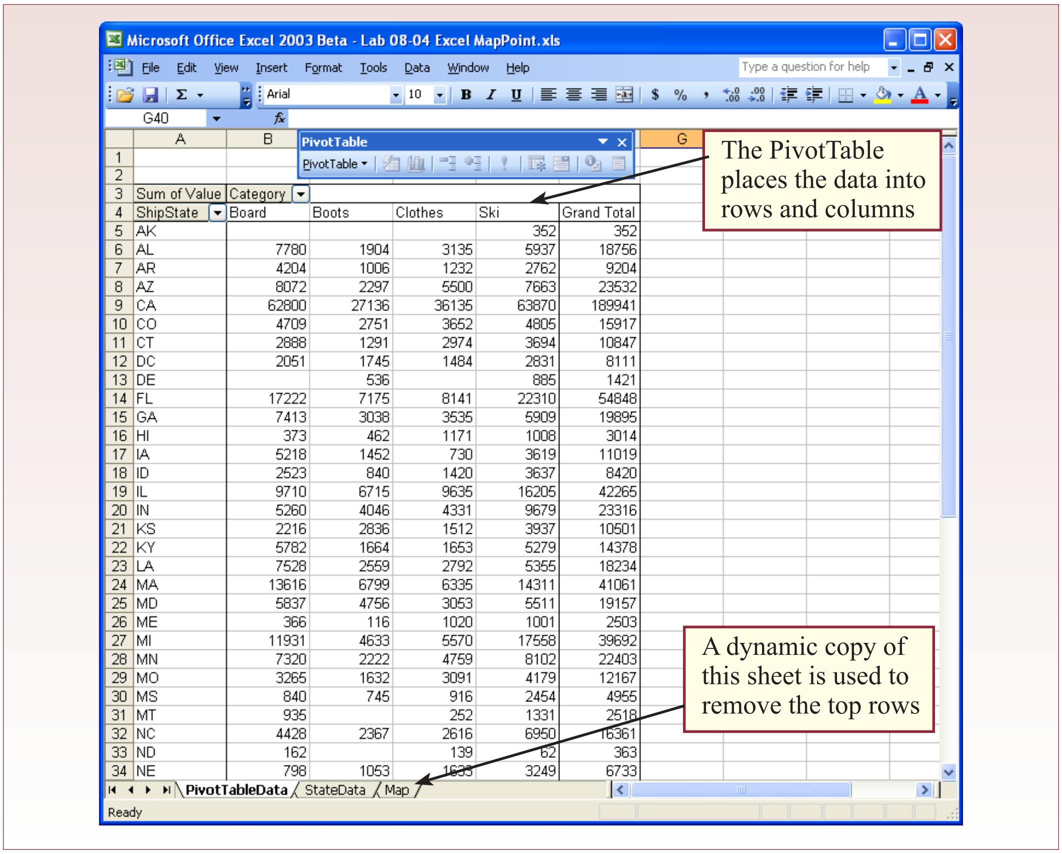
Save the file.

Insert a MapPoint chart.

Follow the wizard instructions to add a North American map with pie charts.

Figure 9.15 shows that a PivotTable is used to retrieve the desired data and format it into rows and columns. Recall that a simple SQL query would not separate the categories into separate columns. Also, the PivotTable data is copied onto a second worksheet that does not include the first row of the PivotTable. This row tends to throw off the MapPoint data wizard and needs to be removed, but the PivotTable has no mechanism to remove it. Be sure you save the entire worksheet file.

To create the map, you insert a MapPoint chart onto a third worksheet file, and start the Data Mapping Wizard from the toolbar. Figure 9.16 shows that the wizard enables you to dynamically link to the worksheet file. You are then asked to browse and find the spreadsheet and select the worksheet and columns you need. The next step is to select the type of chart you want on the map. To display only a single series at a time, choose the Shaded Area option. This choice draws the

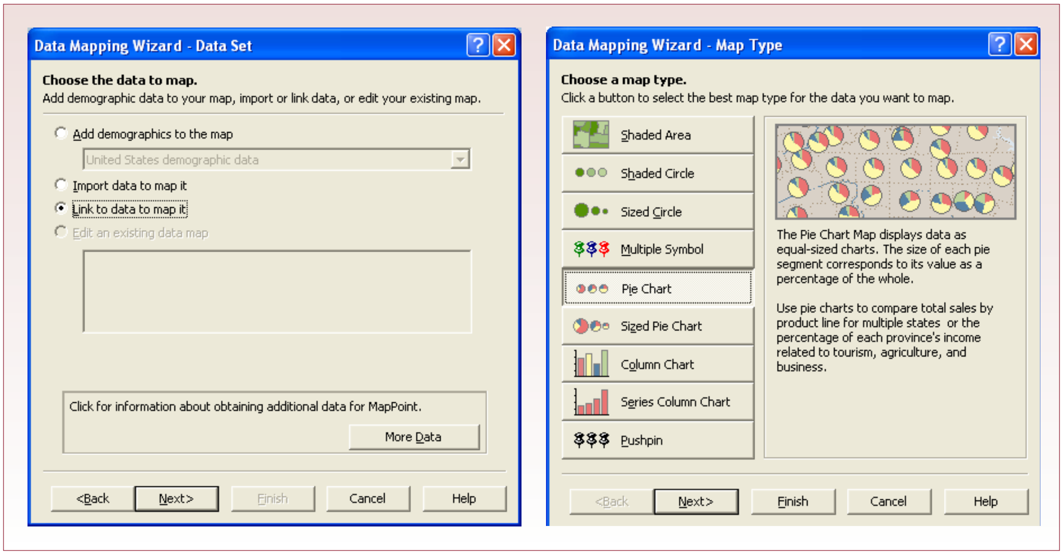

Microsoft Office Excel 2003 Beta - Lab 08-04 Excel MapPoint.xls
File Edit View Insert Format Tools Data Window Help
Type a question for help
Arial 10
G40
PivotTable
The PivotTable places the data into rows and columns
A dynamic copy of this sheet is used to remove the top rows
PivotTableData StateData Map
Ready

Sum of Value	Category				
ShipState	Board	Boots	Clothes	Ski	Grand Total
AK				352	352
AL	7780	1904	3135	5937	18756
AR	4204	1006	1232	2762	9204
AZ	8072	2297	5500	7663	23532
CA	62800	27136	36135	63870	189941
CO	4709	2751	3652	4805	15917
CT	2888	1291	2974	3694	10847
DC	2051	1745	1484	2831	8111
DE		536		885	1421
FL	17222	7175	8141	22310	54848
GA	7413	3038	3535	5909	19895
HI	373	462	1171	1008	3014
IA	5218	1452	730	3619	11019
ID	2523	840	1420	3637	8420
IL	9710	6715	9635	16205	42265
IN	5260	4046	4331	9679	23316
KS	2216	2836	1512	3937	10501
KY	5782	1664	1653	5279	14378
LA	7528	2559	2792	5355	18234
MA	13616	6799	6335	14311	41061
MD	5837	4756	3053	5511	19157
ME	366	116	1020	1001	2503
MI	11931	4633	5570	17558	39692
MN	7320	2222	4759	8102	22403
MO	3265	1632	3091	4179	12167
MS	840	745	916	2454	4955
MT	935		252	1331	2518
NC	4428	2367	2616	6950	16361
ND	162		139	62	363
NE	798	1053	1633	3249	6733

Figure 9.15

states in darker shades that represent higher levels of sales. To show multiple series on the same state, you need either the pie or column chart.

Figure 9.16



Choose the data to map.
Add demographic data to your map, import or link data, or edit your existing map.

( ) Add demographics to the map
( ) Import data to map it
[✓ Link to data to map it]
( ) Edit an existing data map

Click for information about obtaining additional data for MapPoint.

Choose a map type.
Click a button to select the best map type for the data you want to map.

The Pie Chart Map displays data as equal-sized charts. The size of each pie segment corresponds to its value as a percentage of the whole.

Use pie charts to compare total sales by product line for multiple states or the percentage of each province's income related to tourism, agriculture, and business.

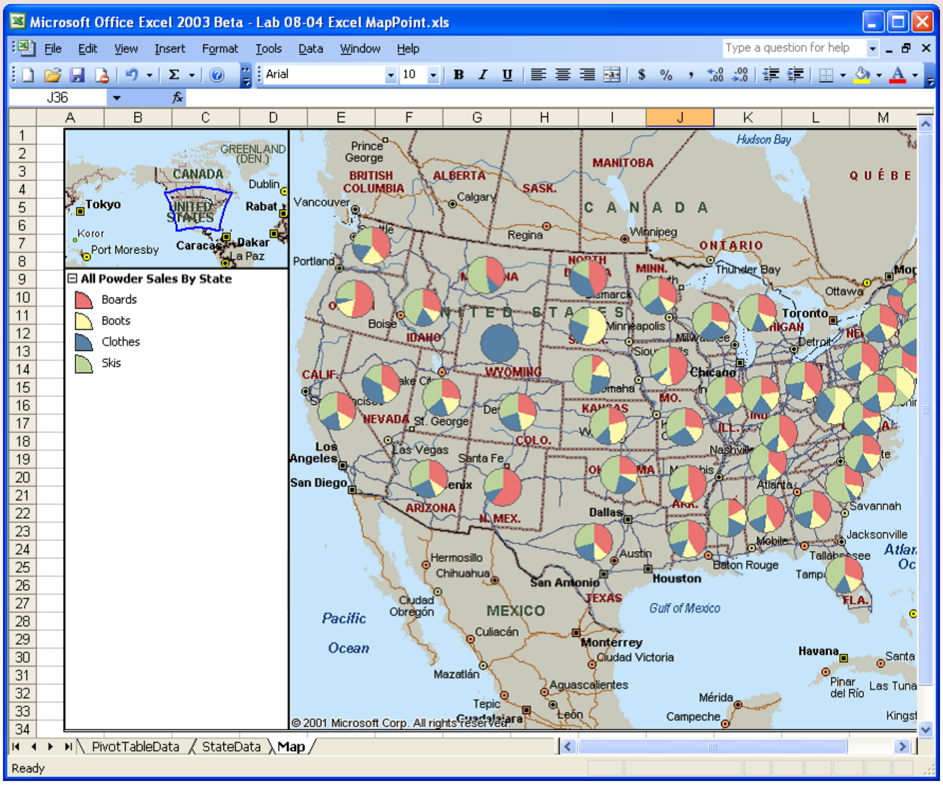

Microsoft Office Excel 2003 Beta - Lab 08-04 Excel MapPoint.xls
All Powder Sales By State
Boards
Boots
Clothes
Skis
PivotTableData
StateData
Map
Ready

Figure 9.17

Figure 9.17 shows the resulting chart. The shop managers are probably most interested in looking for differences between sales of skis and boards. With more realistic data, perhaps there would be a geographic pattern, but no regional trends leap out of this chart. A few states seem to have a substantially higher portion of sales in snowboards, but they are randomly scattered across the country. These cases might be important, or they might be due to random fluctuations. Either way, it would be worthwhile to talk with customers in these states.

Wyoming stands out as unique because only clothes have been shipped there. This piece of information could be highly important, or it might be due to a limited number of customers from Wyoming. Observations of this type often arise in data mining applications. One of the goals of data mining is to highlight patterns and unusual situations. It is up to the analyst to determine if the relationships are important. It is up to the managers to decide how to use the results to increase sales and profits.

Activity: Analyze Data with Regression

Linear regression is a tool that is relatively easy to use and is supported by a variety of platforms, including Excel. Again, with Excel, you need to install and select the Data Analysis tools. Linear regression can be used for many things, and many options and features exist in high-end tools. Its primary purpose is to compare sets of data in terms of closeness. The classic two-dimensional example is illustrated in Figure 9.14 where the best line is found that fits the set of points. In multidimensional terms, it is commonly used to determine how a set of exogenous (independent) variables influence an endogenous (dependent) variable.

In the All Powder case, the managers would like to analyze the state data and see if the total sales within a state are heavily determined by the population or income level of the state. In this case, the population and income are exogenous variables (predictors), and the sales total is the endogenous variable to be predicted. As a data mining tool, regression has some strengths and weaknesses. Its main strength is that it has been heavily analyzed and applied for many years, and the results are relatively easy to understand and interpret. Its main drawback is that the results are largely determined by averages, so the conclusions apply to the average or general group, but not necessarily to the outliers. Sometimes the most valuable insights come from understanding the outliers—such as the people who do not buy certain items, or the few leading edge customers who pursue new sports before the crowd arrives.

Action

Use the database command External Data/Excel to read the demographic spreadsheet data.

Create a query to combine the sales data by state with the demographic data.

Copy and paste the three columns of data into a new worksheet.

In Excel, choose Tools/Data Analysis/Regression.

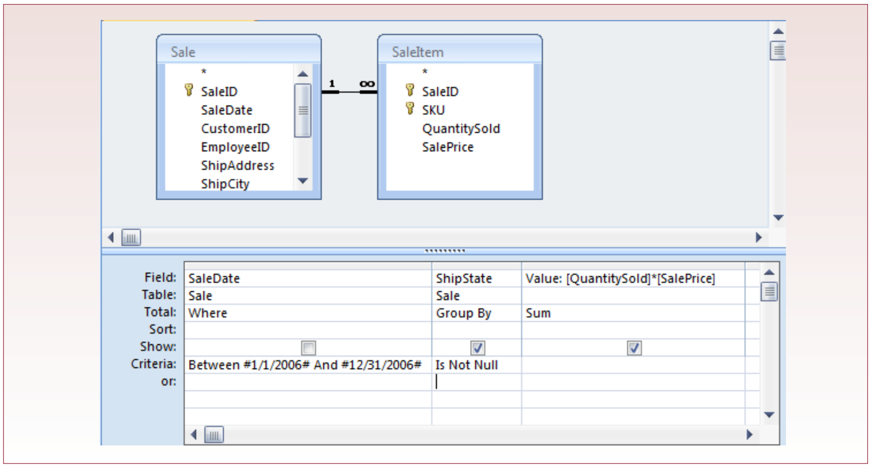
Select the Value column as the Y-range and the population and income columns as the X-range.

Check the top row as label option

Run the regression.

Demographics and economic data on states and counties can be found in the federal government publications. The http://www.fedstats.gov site contains links and search engines to an enormous amount of data. For this exercise, the 2002 population and 2001 total personal income by state have been saved in an Excel spreadsheet. As shown in Figure 9.18, a basic query can be used to compute the total sales by state for the year 2006. But if you look closely at the results of the query, you will spot a problem. Only 47 states are represented in the sales data. The spreadsheet file contains data on 51 states (counting the District of Columbia as a state). You will need to match the rows from the query with the data in the spreadsheet and discard the states for which your sales query has no data. Fortunately, SQL makes this process easy. Simply import the spreadsheet data into a temporary table, join it to the sales query rows, then save the resulting data into a

Figure 9.18



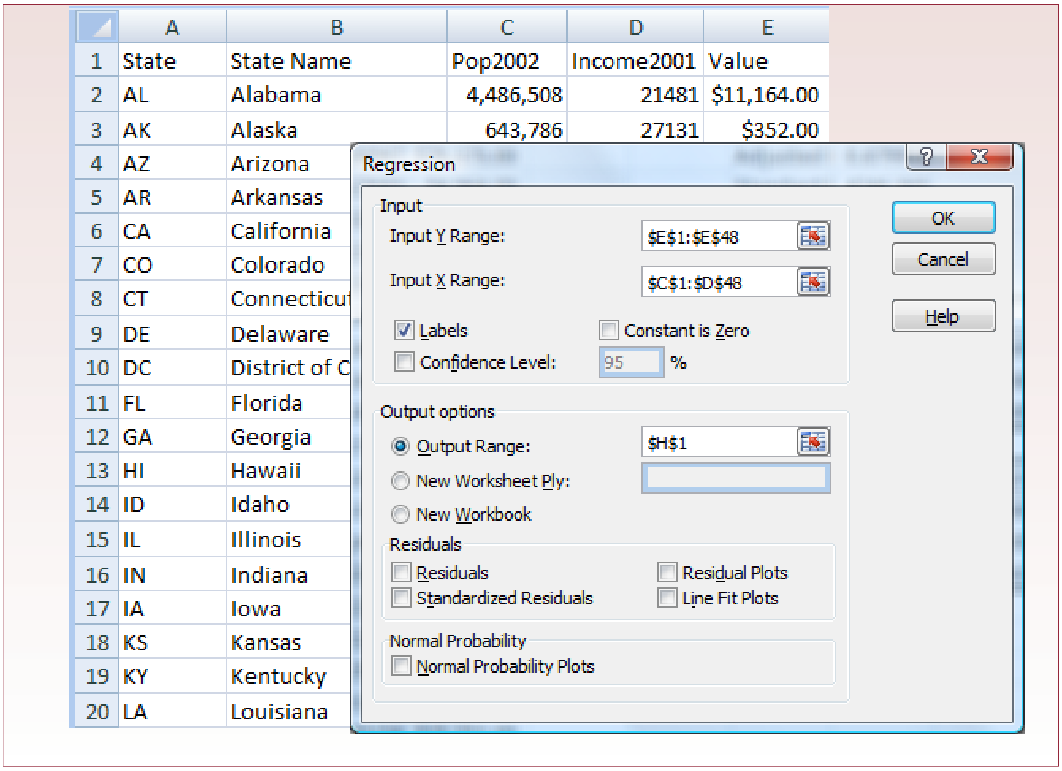
Field:	SaleDate	ShipState	Value: [QuantitySold]*[SalePrice]
Table:	Sale	Sale	
Total:	Where	Group By	Sum
Sort:			
Show:	☐	☑	☑
Criteria:	Between #1/1/2006# And #12/31/2006#	Is Not Null	
or:			

new worksheet. In Access, use the File/Get External Data/Link Tables command to open the spreadsheet. This way, you can link directly to the worksheet without copying the data into a new table. Create a new query that joins the SalesByState query and the new StateDemographics table. Run the query, highlight all of the rows, copy and paste them into a new Excel spreadsheet.

Start the Excel regresison wizard with Data/Analysis/Data Analysis/Regression. As shown in Figure 9.19, you will have to select the Y (dependent) variable as the Value column, as well as the X (independent) variables as the Population and Income columns. To make the results easier to read, when you select the data, you should also include the top row that contains the label for the column. Then make sure you check the Labels box in the wizard so it knows the first row is not part of the numeric data. Most of the time you will want to accept the default option to place the results in a new worksheet.

Figure 9.20 shows the results of the regression for this example. Notice that the R-square value is relatively high, indicating that this simple model describes a little under 90 percent of the variance in the sales. Second, notice that only the population coefficient is significantly different from zero at a 5 percent level of significance (P-value less than 0.05). The coefficient value is positive, indicating that states with higher populations tend to purchase more items from All Powder. In fact, the company receives about $1.76 in sales for every thousand people. In general, this population result is not too surprising. Larger states with more people mean that there will be more boarders and skiers. The lack of significance on the income term is a little more interesting. There is a question of whether or not it really is insignificant, or if there is simply too little data or too few variables in

Figure 9.19



	A	B	C	D	E
1	State	State Name	Pop2002	Income2001	Value
2	AL	Alabama	4,486,508	21481	$11,164.00
3	AK	Alaska	643,786	27131	$352.00
4	AZ	Arizona			
5	AR	Arkansas			
6	CA	California			
7	CO	Colorado			
8	CT	Connecticu			
9	DE	Delaware			
10	DC	District of C			
11	FL	Florida			
12	GA	Georgia			
13	HI	Hawaii			
14	ID	Idaho			
15	IL	Illinois			
16	IN	Indiana			
17	IA	Iowa			
18	KS	Kansas			
19	KY	Kentucky			
20	LA	Louisiana			

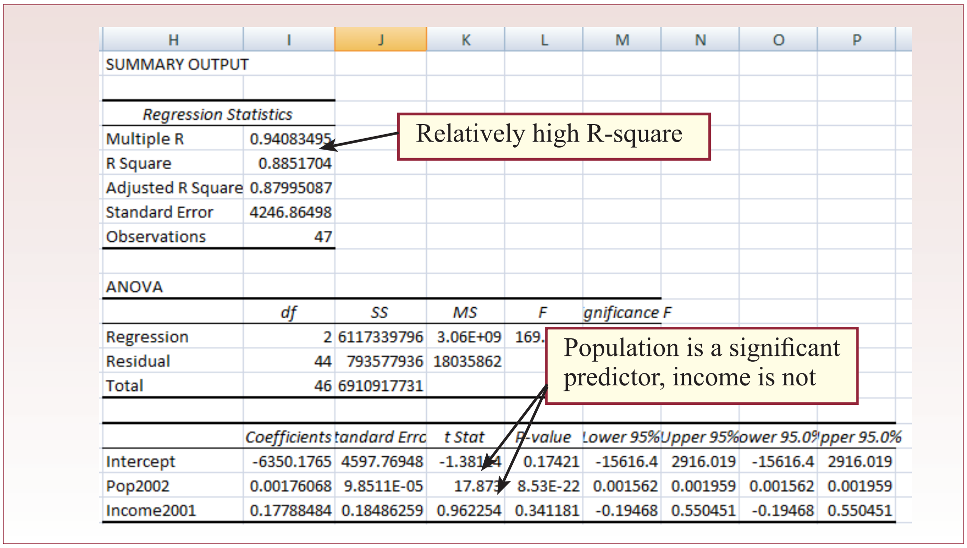
Regression

Input
Input Y Range: $E$1:$E$48
Input X Range: $C$1:$D$48
[✓] Labels
[ ] Constant is Zero
[ ] Confidence Level: 95 %

Output options
(●) Output Range: $H$1
( ) New Worksheet Ply:
( ) New Workbook

Residuals
[ ] Residuals
[ ] Residual Plots
[ ] Standardized Residuals
[ ] Line Fit Plots

Normal Probability
[ ] Normal Probability Plots



H	I	J	K	L	M	N	O	P
SUMMARY OUTPUT								
Regression Statistics								
Multiple R	0.94083495							
R Square	0.8851704							
Adjusted R Square	0.87995087							
Standard Error	4246.86498							
Observations	47							
ANOVA								
	df	*SS*	*MS*	*F*	*gnificance F*			
Regression	2	6117339796	3.06E+09	169.				
Residual	44	793577936	18035862					
Total	46	6910917731						
	Coefficients	*tandard Errc*	*t Stat*	*P-value*	*Lower 95%*	*Upper 95%*	*ower 95.0%*	*pper 95.0%*
Intercept	-6350.1765	4597.76948	-1.38144	0.17421	-15616.4	2916.019	-15616.4	2916.019
Pop2002	0.00176068	9.8511E-05	17.873	8.53E-22	0.001562	0.001959	0.001562	0.001959
Income2001	0.17788484	0.18486259	0.962254	0.341181	-0.19468	0.550451	-0.19468	0.550451

Figure 9.20

the model. These questions should always be examined for insignificant variables. But, if it is true, it means that the sport has changed. Historically, because of the costs, snow sports have generally appealed more to wealthier people. In terms of the business and marketing, it would mean the company should consider a major shift in advertising and promotion if the sports now appeal to all income levels. Of course, considerably more data would be needed before making this commitment, but it is worth investigating. In truth, for this case, the customers were generated based on city populations, so the statistical results match the underlying data model. Actual sales data would provide more interesting results, but the process is the same.

Exercises

Crystal Tigers

The Crystal Tigers club does not have a huge amount of data to analyze within the organization. However, they are interested in comparing their service data and the organizations they work with to see if they are serving the needs of the community. Periodically, they survey people in the surrounding areas to determine if they have heard of the club, if they know what charities the club supports, and their overall opinion of the club. In the process, they also ask citizens about the events and problems that most affect their lives. A substantial part of the survey is a listing of support organizations with which the club is considering partnering. Crystal Tigers has collected this survey data every six months for the last three years, and they get several hundred responses each time. All of the data is stored in Excel spreadsheets.

1. Create two sample spreadsheets with the survey data. Create tables in Access to hold the normalized data. Write the SQL statements to transfer the data. Build this code into a form and button that will automate the transfer.

2. Create a query and a PivotTable that will enable managers to analyze the survey data.

3. Create a PivotTable that will enable managers to analyze the existing club service data. Use two possible fact fields: hours worked and money raised. Include all of the dimensions you think managers might need.

4. Do a time series analysis of the money raised. Managers are particularly interested in trends and in identifying the months that raise the most money.

5. Assume you have data on money raised for several years (make up monthly totals if necessary). Obtain personal income data for your state or metropolitan area over those years and see if the income level is correlated with the money raised.

Capitol Artists

The managers of Capitol Artists are primarily interested in identifying the best employees and the most profitable customers. The job tracking system ultimately generates a considerable amount of data—at the hourly and daily level. Note that all employee tasks are supposed to be recorded in the system based on client, job, and the task involved. The firm has considerable information on clients, including a size classification (tiny, small, medium, and large), and type of company (such as print shop, marketing, retail, and medical). This additional client information is currently stored in a spreadsheet, with one page devoted to each client.

1. Create three sample client worksheets with sample data. Modify the Access tables as needed to handle this new data. Create a form that will enable a clerk to find the worksheet and transfer the data to Access.

2. Create a PivotTable that will enable managers to analyze the hours worked and revenue generated by employees, day of week, client, client size, and so on.

3. Create a PivotChart that compares employees based on billable hours by day during the past month.

4. Assume that you have approximate sales numbers representing the size of each of the clients (make up the data). Create a categorical variable for the client industry (for example, 1= print shop, 2=marketing, and so on). Perform a regression to see if the client size or industry influences the amount of sales revenue Capitol Artists generates.

5. If you have mapping software available, analyze the client revenue by location.

Offshore Speed

Inventory control is critical for Offshore Speed because it has to stock thousands of small parts for different engines and drives. All of these parts are grouped into categories in terms of the manufacturer, and the location within the engine or boat. Lately, the owners think there has been an increased demand for oil pump impellers, but they are not certain because there are several different brands. They also suspect that sales of electronic navigation devices have tapered off. Although they have the sales data available, they are not sure how to analyze and compare it. Of course, the sales data for the past three years is stored in Excel spreadsheets. One sheet for each month of sales, and each line contains a sale number, date, part number, quantity, and price. Unfortunately, the part numbers do not match the new ones entered into the database. However, there is a separate spreadsheet that maps the two numbers. The first column lists the old number and the second column contains the new number.

1. Create at least two sample spreadsheets for the older sales, and the spreadsheet that maps the old numbers to the new ones. Create a form that can be used by a clerk to pick a spreadsheet and import the data into the new database.
2. Create a PivotTable that will enable managers to analyze sales by category, manufacturer, and time. Note that category should be a hierarchy. For example, managers might want to see detailed parts, or just the parts that are used in engines (or drives, or steering, and so on).
3. Create a PivotChart that analyzes sales of the major categories over time based on monthly sales.
4. For some reason, an employee of the company has kept records of the weather for the last three years. She has a spreadsheet that contains the date, the amount of rain on that day, and the high temperature for the day. Create a regression to see if there is a relationship between the weather and your sales. (Make up some sample weather data, or find it on the Internet for your area.)
5. If you have mapping software available, analyze the client revenue by location.
6. If you have access to software that performs association or market basket analysis, this case would be a good application to see what types of parts might be purchased together.

Final Project

The main textbook has an online appendix with several longer case studies. You should be able to work on one of these cases throughout the term. If you or your instructor picks one, perform the following tasks.

1. Identify at least one primary fact attribute that managers would want to track, along with several dimensions. Create the query and PivotTable to analyze the data.
2. Identify any data that could be analyzed over time, and create a PivotChart and an Excel spreadsheet to forecast the data.
3. Identify any data that could benefit from market basket or association analysis. If you have access to the software, create the queries and analyze sample data.
4. Identify any data that could benefit from geographic analysis. If you have access to the software, create the queries and analyze sample data.
5. Identify any correlations or regression analysis that might help managers better understand the operations and effects of various attributes. If possible, collect sample data and analyze it.

Chapter 10

Database Administration

Chapter Outline

Objectives

- Evaluate and improve the application performance.
- Establish backup and recovery methods and plans.
- Install simple security controls to provide basic protection of the data.
- Protect the forms, reports, and code from unauthorized changes.
- Protect the data with user-level security controls.

Database Administration Tasks in Access

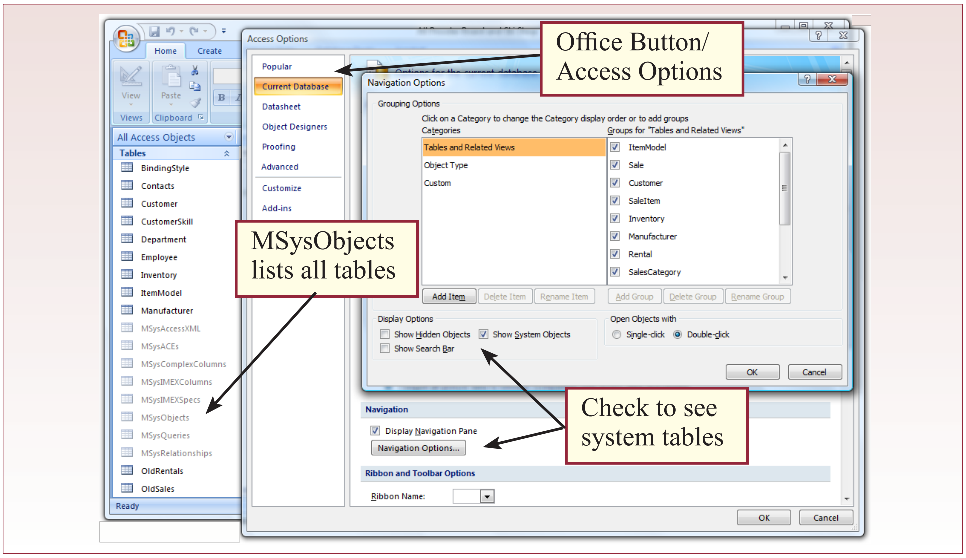
One of the appealing features of Microsoft Access is that it is easy to use and does not require complex administration chores. On the other hand, the lack of sophisticated administration tools makes it less valuable for large-scale applications. Although Access does not have the powerful tools to support thousands of users, it does have some useful features to help developers and database administrators.

Every DBMS maintains an internal list of all of the database objects, such as table, query, and report names. Only recently has the SQL standard proposed a common method to obtain these names. Consequently, most systems have proprietary tables and columns for the metadata tables. In Access, you rarely need to deal with these tables directly. Instead, it is easier to use the visual administrator and simply select the items you want to modify. Someday you might need to examine a list of the tables from within your code, however, so you should learn about the system tables.

For safety and to reduce clutter, the metadata tables in Access are normally hidden. Figure 10.1 shows that you can use the Office Button/Access Options to make the system tables visible. All system table names begin with MSys, and the MSysObjects table contains a list of all tables in the database. You can open the table to see its structure, but do not change any of the data. The Type column is set to 1 to indicate a table object, so you could run a query against the system table to see a list of all tables in the database. Microsoft also provides the Active Data Object Extensions (ADOX) control to provide programmatic access to the metadata in an underlying DBMS, and you can read the Microsoft documentation to learn how to use ADOX to read the list of tables in a DBMS. For now, return to Access Options and uncheck the box to show the system objects.

Performance is always a tricky issue in a DBMS. Small tables with a limited number of joins and a handful of simultaneous users rarely encounter performance problems. Also, with hardware improvements, Access has been able to handle

Figure 10.1



even larger applications without experience performance problems. However, you will encounter some databases that will need changes to improve performance.

Backup and recovery are relatively simple in Access since it does not write a transaction log or journal. You can back up an Access database simply by copying the file. Also, Access does not really provide recovery facilities, so if the system crashes in the middle of an application, few steps are needed because most of the data in transition will be lost.

Security is more complex. By default, Access is completely open and everyone has full access to the data, forms, reports, and code. The assumption is that generally only trusted users will be given access to the database, and then they have the ability to use any part of the database. Microsoft now provides several security features to provide different levels of protection. It is even possible to set up user accounts and assign separate permissions to access individual tables, queries, forms, and reports. Once you set up user-level security, someone needs to be assigned to handle the user accounts.

Case: All Powder Board and Ski Shop

Ultimately, the owners of All Powder want to assign individual user permissions. Although the shop trusts its employees, it often hires students to work as clerks, and the owners would like to limit what the clerks can do with the application. The issue is only partly a matter of trust: It is also useful to protect the database so clerks and other users cannot change form layouts or accidentally delete items.

The managers are also somewhat concerned about performance, particularly at the checkout machines. Sometimes the store's checkout lines get hectic, so the application has to be fast. Some of the issues can be handled by installing more computers. The salesperson can enter the basic customer data immediately, and the checkout clerk simply selects the customer and enters the product numbers. Of course, having more computers means that the company will need a network. It also means that more people will be simultaneously accessing the data, increasing the risk of collisions and locks.

Lab Exercise

All Powder Board and Ski Shop

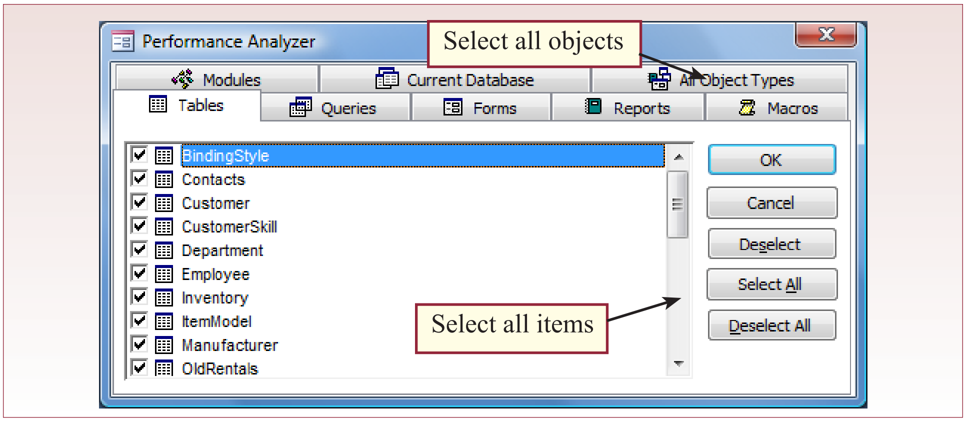
DBMS developers learned early that indexes can significantly improve the performance of a relational DBMS. Key columns and join columns are almost always indexed because they often represent single-item lookups. Without an index, the computer has to search each row sequentially to find a match. An index can exponentially reduce the number of lookups in a search. On the other hand, indexes have to be updated whenever data is changed, deleted, or added. As a result, placing too many indexes on a table can result in even worse performance.

Activity: Evaluate the Application Performance

Action

Choose Tools/Analyze/Performance.
Select all objects.
Run the analysis.
Add indexes if they are reasonable.

Your job is to find the balance with enough indexes to improve performance for key tasks, but not so many that other portions become too slow. This balance is unique to each application and can be difficult to find. Ultimately, you will have to fine-tune the



BindingStyle
Contacts
Customer
CustomerSkill
Department
Employee
Inventory
ItemModel
Manufacturer
OldRentals

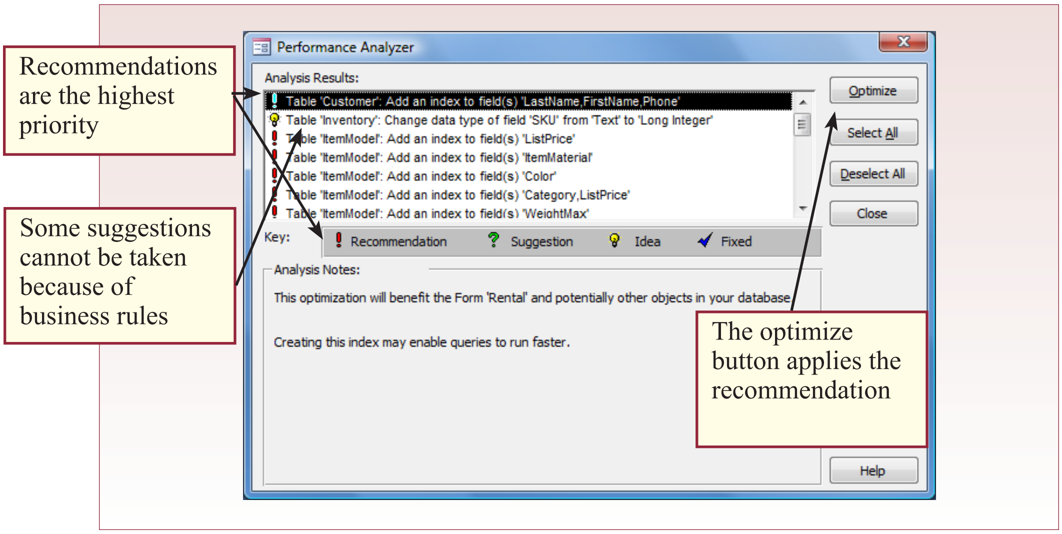
Figure 10.2

application over time. A few simple rules help you begin: (1) All primary keys should be indexed, (2) Join columns should be indexed—particularly in large tables, (3) Heavily searched or sorted columns should be indexed, and (4) Transaction tables that are constantly changed (such as SaleItem) should have few indexes. The first rule is so strong that Access automatically builds the index for every column in a primary key.

Access provides a tool to help you determine which columns should be indexed and other ways to improve performance. Use the Database Tools/Analyze Performance menu option to start the analyzer wizard. Figure 10.2 shows the first step of the wizard. You can select the items that you want to examine for performance. You can select individual items, groups of objects (such as all tables), or choose the tab for All Object Types to select everything in the database. Selecting all objects is a good way to cover all possibilities, but it might be overwhelming. Try it first, if there are too many suggestions to scroll through, start over and do one group at a time. At a minimum, you need to analyze the queries.

Figure 10.3 shows sample analyzer results. It attaches a level of importance to each recommendation, and you should seriously consider the higher-level recom-

Figure 10.3


Performance Analyzer
Recommendations are the highest priority
Some suggestions cannot be taken because of business rules
Optimize
Select All
Deselect All
Close
Help
Key:
Recommendation
Suggestion
Idea
Fixed
The optimize button applies the recommendation

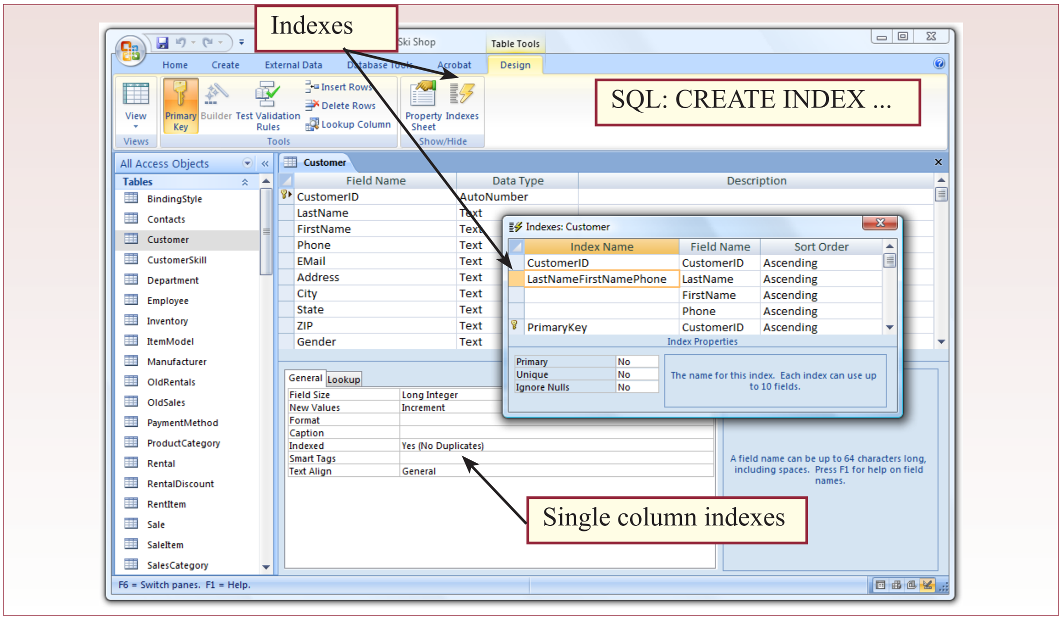
Analysis Results:

Table 'Customer': Add an index to field(s) 'LastName,FirstName,Phone'
Table 'Inventory': Change data type of field 'SKU' from 'Text' to 'Long Integer'
Table 'ItemModel': Add an index to field(s) 'ListPrice'
Table 'ItemModel': Add an index to field(s) 'ItemMaterial'
Table 'ItemModel': Add an index to field(s) 'Color'
Table 'ItemModel': Add an index to field(s) 'Category,ListPrice'
Table 'ItemModel': Add an index to field(s) 'WeightMax'

Analysis Notes:

This optimization will benefit the Form 'Rental' and potentially other objects in your database.

Creating this index may enable queries to run faster.



BindingStyle
Contacts
Customer
CustomerSkill
Department
Employee
Inventory
ItemModel
Manufacturer
OldRentals
OldSales
PaymentMethod
ProductCategory
Rental
RentalDiscount
RentItem
Sale
SaleItem
SalesCategory

Customer

Field Name	Data Type	Description
CustomerID	AutoNumber	
LastName	Text	
FirstName	Text	
Phone	Text	
EMail	Text	
Address	Text	
City	Text	
State	Text	
ZIP	Text	
Gender	Text	

General Lookup

Field Size	Long Integer
New Values	Increment
Format	
Caption	
Indexed	Yes (No Duplicates)
Smart Tags	
Text Align	General

A field name can be up to 64 characters long, including spaces. Press F1 for help on field names.

Indexes: Customer

Index Name	Field Name	Sort Order
CustomerID	CustomerID	Ascending
LastNameFirstNamePhone	LastName	Ascending
	FirstName	Ascending
	Phone	Ascending
PrimaryKey	CustomerID	Ascending

Index Properties

Primary	No
Unique	No
Ignore Nulls	No

The name for this index. Each index can use up to 10 fields.

Figure 10.4

mendations. Notice that many of them involve indexes. The index for LastName, FirstName makes sense because customer data is often sorted that way.

At this point, you have three methods to create the recommended index. (1) Click the Optimize button and the wizard builds the index for you. This is by far the easiest method. (2) Open the table in Design view and choose the Indexes menu option. Add an index name along with the columns to be indexed. Generally, you will use this method to check to see that the wizard correctly built the requested index. Figure 10.4 shows the index list with the newly created index. (3) You can use the SQL command: CREATE INDEX LastFirstName ON Customer (LastName, FirstName). While this method might seem cumbersome, it is a useful way to write down the indexes and store them in a file. This list could be used to re-create the database on a different machine and to document database definition in case of a disaster.

Again, remember that you will rarely want to accept all of the analyzer's recommendations. If you know that a table is heavily involved in transactions, you want to avoid creating too many indexes on it. If in doubt, test the application both ways for a few days and carefully observe the performance. If an application is too slow, remove the associated indexes and test again. Also, some of the recommendations cannot be taken because it would violate the business rules. In the example, SKUs are currently numeric, so the wizard will suggest converting that column to a Long Integer instead of text. But the managers are planning to implement new SKUs with alphabetic characters, so you should avoid the wizard's recommendation—unless the performance gain is so large that it becomes necessary to talk the managers into always using simple numbers.

Activity: Backup and Recovery

Access does not really provide backup and recovery tools. About the only solution is to use Windows to back up the database file. The Office Button/Manage/Backup link essentially makes a copy for you from within the database. Several tools are available to make backup copies of regular files, and all of these work on the Access database file. One useful technique is to compress the file using a Zip format. Windows XP and above have a built-in capability to create a compressed (zipped) folder. Just create the folder and drag the database file into it to compress it. The compressed folder is often small enough to e-mail or copy to a secure tape or drive that can be moved offsite.

Action

Choose Office Button/Manage/ Compact and Repair Database.

Close the file but not Access.

Run Compact and Repair again, this time enter a new name for the compacted database to use as a backup copy.

One catch with the Windows approach is that you should make sure that no one else is using the database when you make the backup copy. For small businesses that shut down over night, this requirement is easy. For operations that run 24 hours a day, it is more complicated, and a key aspect of backup is setting a schedule and notifying everyone of the backup time. Although it might slightly interfere with the business, a full copy provides the advantage of ensuring that you have a clean, consistent copy of the database.

There is an important tool that you need to run on a regular basis in Access databases. To see the effect of this utility, first go to Windows, browse to the folder that holds the main database file, and write down the size of the file. Then select the Office Button/Manage/ Compact and Repair menu option. After a few moments, the database will close and restart. Return to the Windows folder and check the size of the database. It should be substantially smaller. The main effect of this process is to completely remove the items that have been deleted. Remember that to save time Access simply flags an item as deleted. It is not actually removed from the database until you run the compact and repair utility. This utility is also useful if the system crashes while the database is open. It fixes some basic index issues and can repair common damage issues.

The compact and repair utility cannot repair databases with large errors—particularly those caused when disk drives fail. Keeping full backup copies is critical. However, in an emergency, it is sometimes possible to recover data and perhaps even forms and reports from a damaged copy of Access. The trick is to create a completely new, blank Access database. Then use the External Data/Import utility to import whatever pieces you can obtain from the damaged file.

Security and Privacy

Activity: Simple Access Security Controls

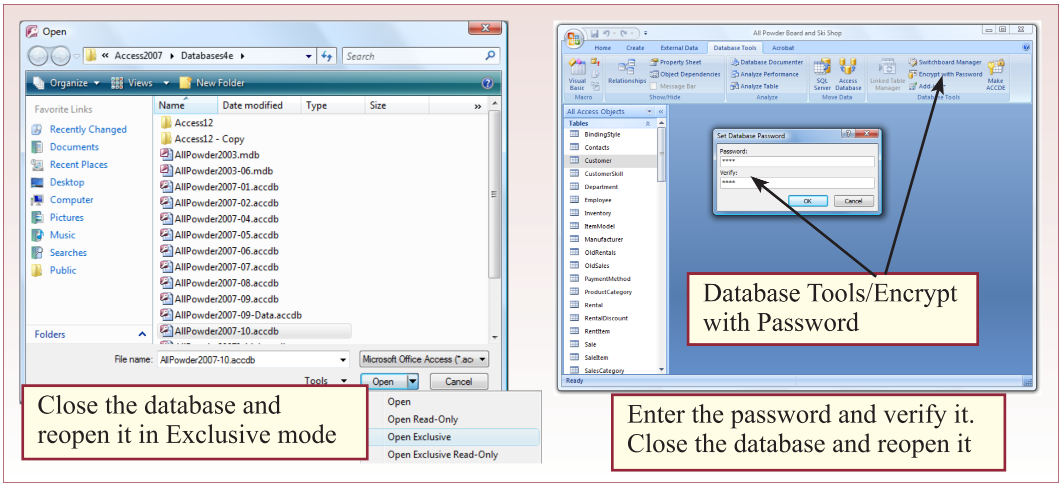
Setting and maintaining detailed lists of users and access rights can be a time-consuming job. In many small applications, it is not necessary to provide this level of control. In these cases only a few people may need to use the database, and you simply need to restrict access to those selected people and prevent others from using it at all. You can use the simple security controls provided within Access and avoid the management task of setting full security controls.

Action

Choose Database Tools/Encrypt with Password.

Enter test as a password.

Close the database and reopen it.



Close the database and reopen it in Exclusive mode

Database Tools/Encrypt with Password

Enter the password and verify it. Close the database and reopen it

Figure 10.5

Figure 10.5 shows the steps needed to assign a database password. The one trick is that you must open the database in Exclusive mode—which means that no one else can be using the database when you set the password. To open a database in exclusive mode, you have to use the Open command under the Office Button and browse to the location of the database file. Then carefully select the Exclusive option at the bottom right of the browser screen. Once the file is open, use the Database Tools/Encrypt with Password ribbon option, and use the pop-up box to enter the new password. When you close the database and reopen it (it no longer needs to be opened exclusively), you will be asked for the password. Without the password, no one else can open the database. You can use the same process to remove or "unset" the password. Keep in mind that the database password is all or nothing. Anyone who knows the password has full control over everything in the database. Many times, this level of protection is sufficient, but make sure you have backup copies in case things are accidentally deleted.

This password also encrypts the database, solving a second problem. Without encryption, someone could use a utility program to open the database file and examine or change the data without ever starting Microsoft Access. Encryption protects the database from this attack, but you should also be careful to set network security rights so that no one else can find or read the file. Attackers will then have to circumvent two security blocks instead of one to get to the file. The network access rights are set within Windows by right clicking on the filename and following the options on the security tab.

The option to encrypt the database is straightforward, and you do not have to open the file in exclusive mode. Choose Tools/Security/Encrypt/Decrypt Database on the menu. Then browse to a folder and enter a new filename for the secured database. The encrypted file is essentially gibberish so no one can find the specific data. However, they can still destroy data by writing random bytes into the file, or even delete the entire file. You still need the Windows security controls to remove write and delete access to the file to prevent these problems.

A different issue with security involves Trojan Horses and code embedded in an Access document. An attacker (or phisher) might send a small database to someone and when it opens, the malicious code could install spyware or cause damage to computer files. To reduce this threat, Office initially treats all code as untrusted and blocks it from running. You have undoubtedly seen the startup messages asking if you want to enable your database to run. Although this process is probably slightly safer, it is a nuisance within a company application. Access 2007 provides two ways to signify that your database code is safe and should be trusted—avoiding the continual notification. The first, and older method, is to purchase a developer security certificate and sign your code. For code that you wish to sell or distribute, this approach is the best, but you have to pay an annual fee. The second approach is easier, and useful for applications used within a single company. You can use the Trust Center to define a location on each computer, or on the network, that contains trusted files. Placing the Access database file in that folder then enables it to run in full mode without warning messages. Use the Office Button/Access Options to open the Trust Center.

Activity: Create an ACCDE File to Protect Forms and Reports

Action

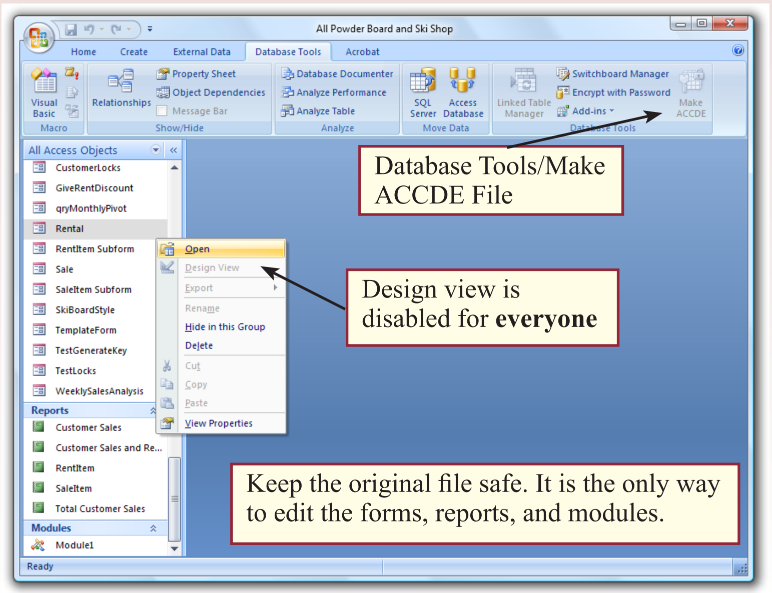
Choose Database Tools/Make ACCDE File.

Close the database and open the new ACCDE database.

Try to edit a form.

Access provides another useful utility that you can use to protect your forms, reports, and code. This option is particularly useful when you are hired to create a database for a client and you do not want the client to see or modify your proprietary code. The client still needs access to the data and to use the forms, so securing the database is not enough. You need to lock down your forms and reports. Simply choose the Database Tools/Make ACCDE File from the menu. This option creates a completely new file with an mde suffix. Figure 10.6 shows that when you distribute this file to the client, no one can open the forms (or reports or modules) in Design view. They will not be able to change the layouts or to see or modify your code. It is a convenient way to protect your trade secrets and to prevent accidental changes.

Figure 10.6



Database Tools/Make ACCDE File

Design view is disabled for **everyone**

Keep the original file safe. It is the only way to edit the forms, reports, and modules.

When you create ACCDE files, be extremely careful. No one can edit forms, reports, or modules in an ACCDE file—including you! Make sure that you keep your original mdb file in several safe locations. When you need to ship a new version to the client, you have to make the modifications in the original accdb file, and then rebuild a new accde file.

If you are building Access databases for clients, you should also download the Microsoft Developer's tools, which are free with the Access 2007 version. In particular, you want the runtime module which enables you to package your database and distribute it to clients who can run it on their computers without needing to purchase and install Microsoft Access. They will not be able to modify or create new forms and reports, but they will be able to run you application as a standalone program.

Activity: Setting User-Level Security Controls

Sometimes a business application requires you to control access to individual tables, forms, and reports for specific users. In this situation, you need to convert the database to implement full user-level security controls. Unfortunately, Microsoft removed user-level security controls in Access 2007. However, Access 2007 can still read (and create) the older MDB file format which does support user-level security. Consequently, the first thing you have to do is create a blank database in the older MDB format and import all of the objects from your 2007 version. Then you can run the security wizard. The steps are a little tricky, but the security wizard handles most of the work for you. To understand the steps and questions asked in the wizard, you need to understand

Action

Create a new blank database in the older MDB format.

Open the new database and use External Data/Import to copy everything from your original database.

Database Tools/Users and Permissions/User-Level Security Wizard.

Create a new workgroup.

Select the Backup Operators group.

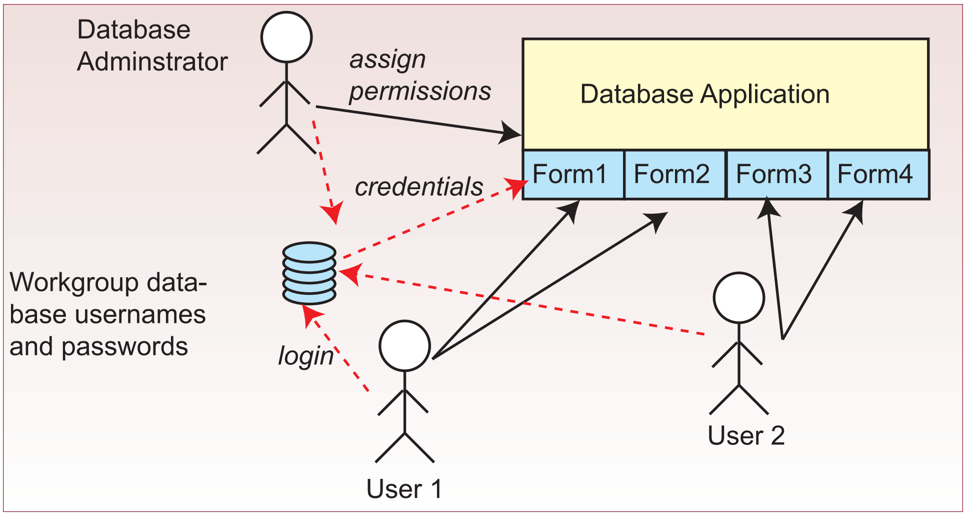
Set a password for your username.

Add three users.

Make sure you are in the Admins group.

Close the database.

Figure 10.7



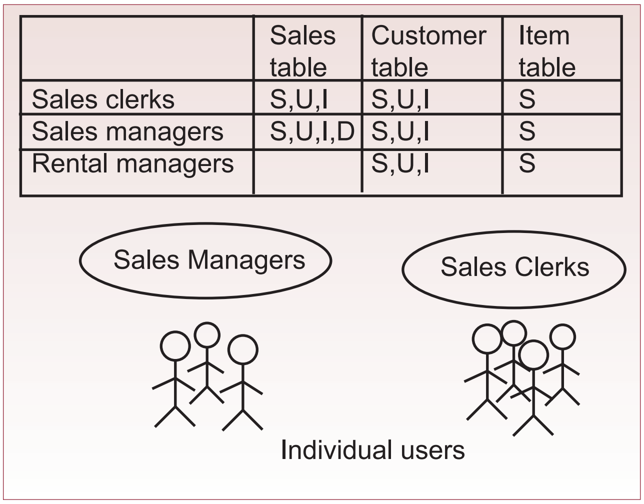
the overall process. Ultimately, if you need user-level security, you should switch to a SQL Server (or Oracle) database instead of Access.

The first issue to face is that Access needs to be able to identify the individual users. Figure 10.7 outlines the basic process. The main database application contains forms, reports, and tables. As the database administrator (DBA), you want to assign individual permissions to separate users for each object. For instance, sales clerks would be able to read some supplier data, but not change it, and probably would not need access to the main supplier form. But, before you can assign any permissions, the database application needs to be able to identify the user. Access identifies users by setting up a separate workgroup database that contains usernames and passwords. The main application is associated to this workgroup database. When the application is opened, users must enter a correct username and password. Obviously, this workgroup database must be secure so that no one can alter data. Access handles this issue by automatically encrypting the workgroup database. Of course, someone has to maintain the workgroup database and handle problems such as users forgetting their passwords. From an enterprise perspective, the workgroup database approach has some problems. Mainly, it requires companies to maintain separate lists of users and passwords for applications. Although workgroup databases can be reused in other applications, they rarely are configured that way. Consequently, users end up needing to know dozens of usernames and passwords. For small companies with one or two applications, this issue is less important.

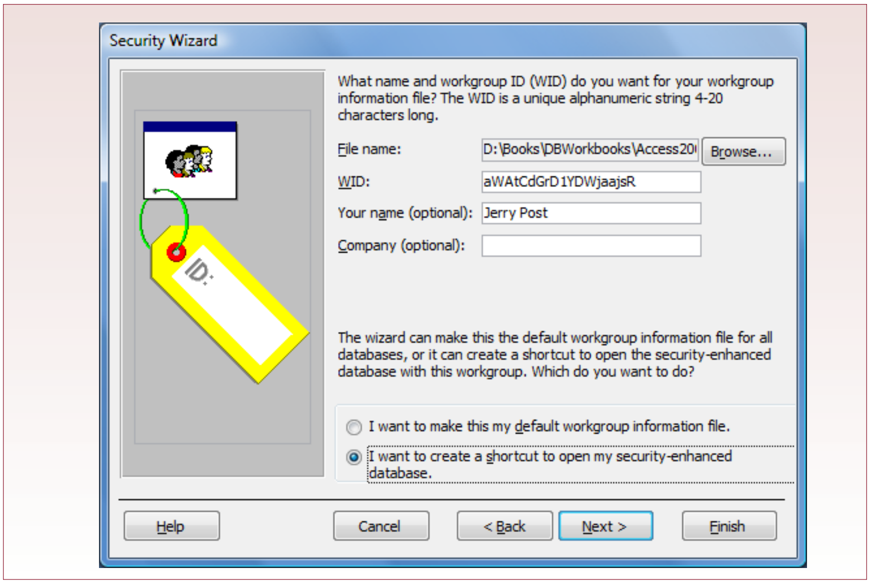
Before running the security wizard, you should write down a list of usernames and initial passwords that will be asked to enter into the workgroup database. While you are identifying users, you should also classify them in terms of tasks or groups. You almost never want to assign permissions to individual users. Instead, you place users into groups and assign database permissions to the groups. Figure 10.8 illustrates the main concept. By assigning permissions to the groups, you should only have to set permissions one time. As individual users are added to or removed from groups, their permissions automatically change.

Before running the security wizard, you should have a list of groups and a list of users and the groups they should be assigned to. You should also have an initial start on the permissions matrix that lists the various resources and the group per-

Figure 10.8



	Sales table	Customer table	Item table
Sales clerks	S,U,I	S,U,I	S
Sales managers	S,U,I,D	S,U,I	S
Rental managers		S,U,I	S


Security Wizard

What name and workgroup ID (WID) do you want for your workgroup information file? The WID is a unique alphanumeric string 4-20 characters long.

File name: D:\Books\DBWorkbooks\Access20
WID: aWAtCdGrD1YDWjaajsR
Your name (optional): Jerry Post
Company (optional):

The wizard can make this the default workgroup information file for all databases, or it can create a shortcut to open the security-enhanced database with this workgroup. Which do you want to do?

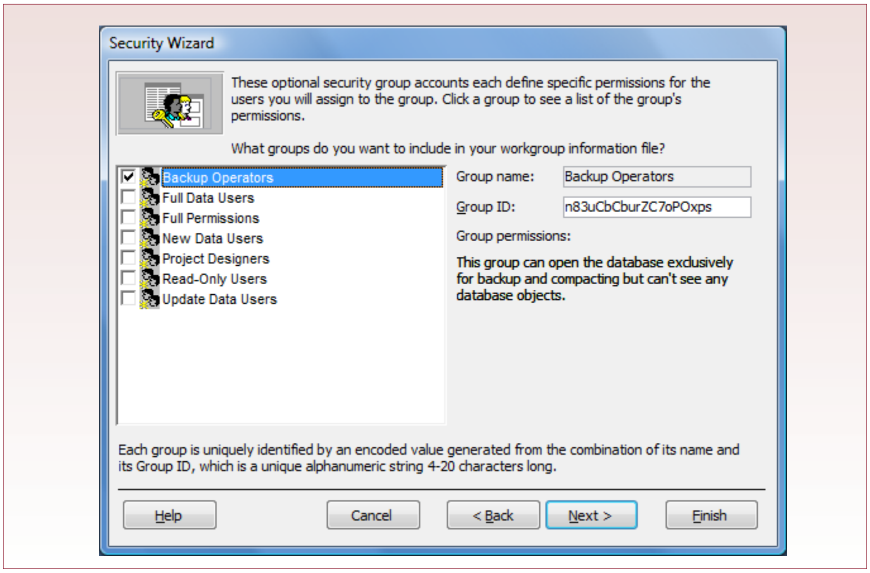
I want to make this my default workgroup information file. [✓ I want to create a shortcut to open my security-enhanced database.]

Figure 10.9

missions for each item. Start the wizard with the Tools/Security/User-Level Security Wizard menu option. Most times, you will create a new workgroup—unless you have already developed similar projects and want to use the same list of users. Figure 10.9 shows an important step in the wizard that sets up the workgroup file. Generally, you should avoid making the workgroup file the default. If you do, you will be asked to log in every time you open an Access database. You should certainly never pick this option in a shared computer lab.

For the most part, you can select the default options in the security wizard. However, Figure 10.10 shows some options that you should carefully consider. The wizard can create some predefined user groups that might be useful. In particular, the backup user group is a good idea, since it automatically sets permissions

Figure 10.10


Security Wizard

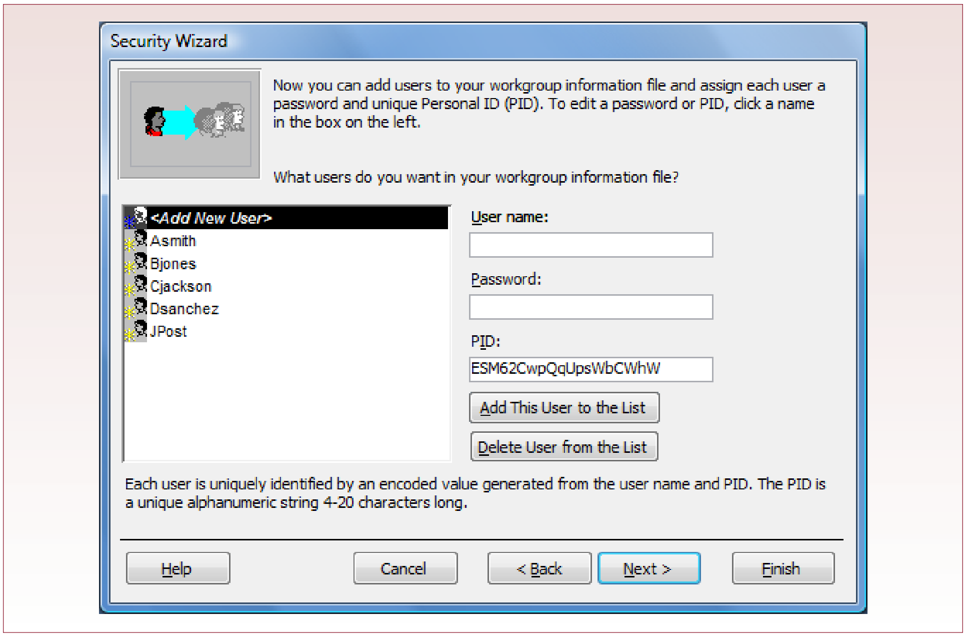
These optional security group accounts each define specific permissions for the users you will assign to the group. Click a group to see a list of the group's permissions.

What groups do you want to include in your workgroup information file?

- [✓] Backup Operators
- [ ] Full Data Users
- [ ] Full Permissions
- [ ] New Data Users
- [ ] Project Designers
- [ ] Read-Only Users
- [ ] Update Data Users

Group name: Backup Operators
Group ID: n83uCbCburZC7oPOxps
Group permissions:
This group can open the database exclusively for backup and compacting but can't see any database objects.

Each group is uniquely identified by an encoded value generated from the combination of its name and its Group ID, which is a unique alphanumeric string 4-20 characters long.



Security Wizard

Now you can add users to your workgroup information file and assign each user a password and unique Personal ID (PID). To edit a password or PID, click a name in the box on the left.

What users do you want in your workgroup information file?

<Add New User>
Asmith
Bjones
Cjackson
Dsanchez
JPost

User name:

Password:

PID: ESM62CwpQqUpsWbCWhW

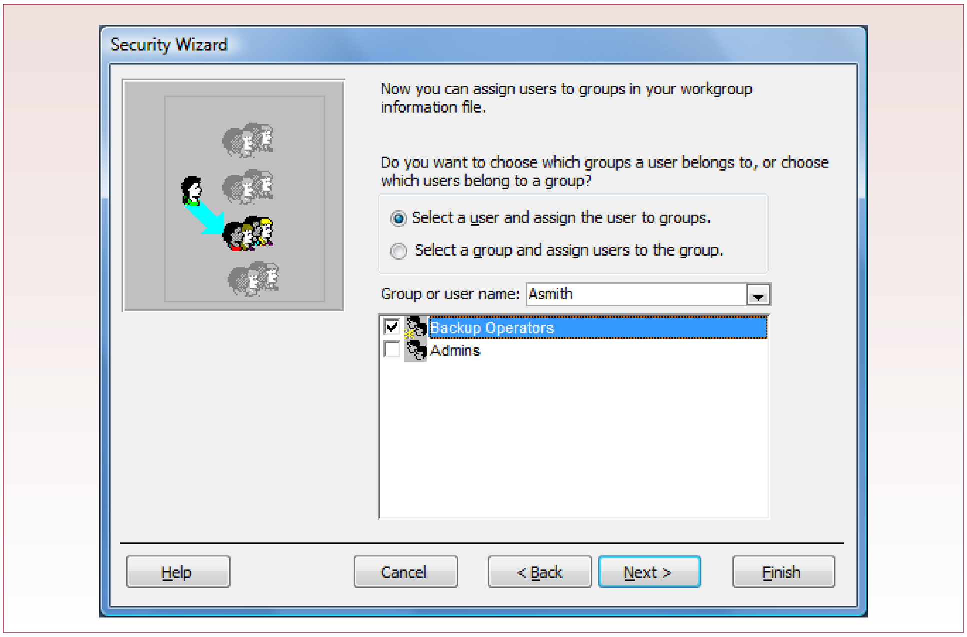
Each user is uniquely identified by an encoded value generated from the user name and PID. The PID is a unique alphanumeric string 4-20 characters long.

Figure 10.11

properly for someone to make copies of the data, but still prevents them from modifying anything. You might also want to use some of the other groups—such as the Full Permissions group. Keep in mind that there is also an internal Administrators group that you will want to use for the DBA, but it is always created automatically.

Figure 10.11 shows that eventually you reach another important screen in the wizard, where you need to enter your list of users. By default, the wizard picks up

Figure 10.12



Security Wizard

Now you can assign users to groups in your workgroup information file.

Do you want to choose which groups a user belongs to, or choose which users belong to a group?

[✓ Select a user and assign the user to groups.]
Select a group and assign users to the group.

Group or user name: Asmith

☑ Backup Operators
☐ Admins

Action

Open the database using the desktop link.

Log in with your username and password.

Choose Database Tools/Users and Permissions/User

and Group Accounts.

Select the Groups tab and add SalesClerks and SalesManagers groups.

Add one of your new users to each group.

Choose Database Tools/Users and Permissions/User and Group Permissions.

Grant specified permissions for the two groups to the Sale, SaleItem, Customer, and Inventory tables.

Assign appropriate permissions to the database and the forms.

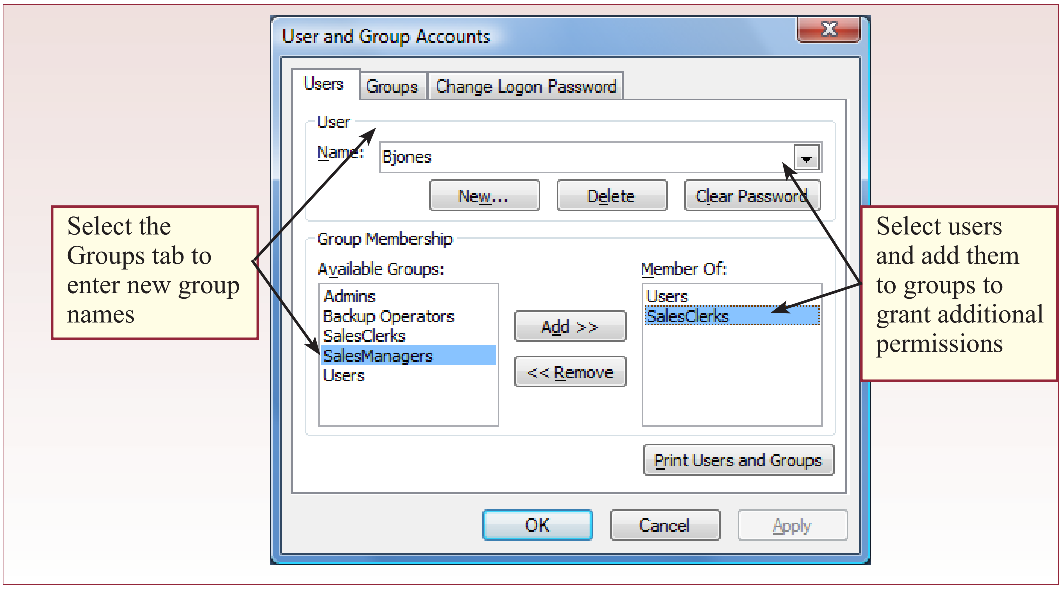
Close the database.

Log in as a sales clerk and test the permissions.

the current user on your system (you). Make sure that you assign a password to your username. If you forget, the password will be blank, so you can still log in, but so could any attacker. You will have to add all of the users and passwords from your list. Note that it is possible to add users later, but it is slightly easier to enter them now. Note that you are entering users, not groups.

Figure 10.12 shows the screen that enables you to assign users to groups. Again, this task can be performed later. And you will have to add your application groups later and assign people to those groups. However, it is critical that you assign at least one user to the Admins group now. By default, the wizard will assign the current user (you) to the Admins group. To be safe, select the option to see the list by group and double-check that (1) there is at least one name in the Admins group, and (2) only people who need this high-level permission are in the group. The Admins group always has full control over all objects in the database, including the ability to reassign permissions and change group memberships. You should keep this list as small as possible. But keep in mind that you could use two people in this group in case one person is on vacation or sick and someone needs a password changed. When the wizard finishes, it

Figure 10.13



Name: Bjones

Available Groups	Member Of
Admins	Users
Backup Operators	SalesClerks
SalesClerks	
SalesManagers	
Users	

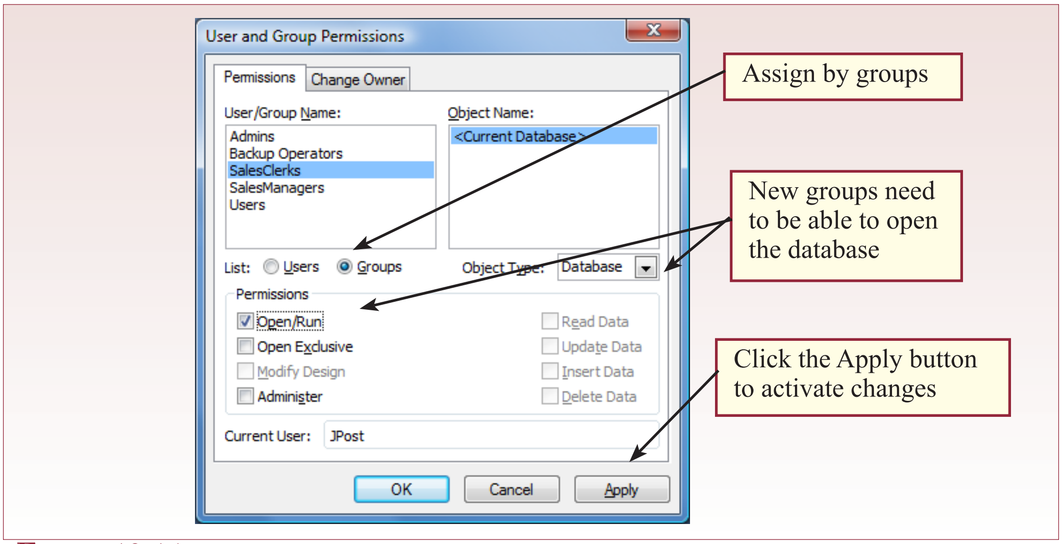


Figure 10.14

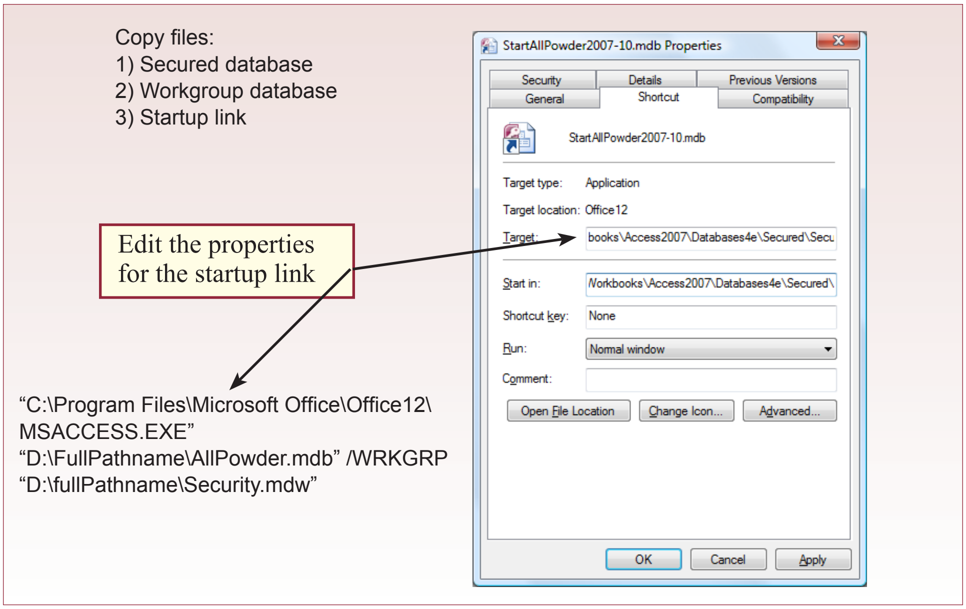
displays a report that lists the filenames and the workgroup ID. Keep this report in a safe place: It can be used to re-create the workgroup file if it is damaged.

When the wizard is finished, close the database. Because it is encrypted and tied to the workgroup file, you cannot open the database directly. Instead, click on the link that the wizard created. If you examine the properties of the link, you will see that it associates the workgroup file with the database application. When the database starts, you will be asked to log in using the username/password pair that you created earlier. Since you still have some administrative tasks to perform, log in using an account that is a member of the Admins group.

The next step is to create the customized groups and assign users to them. Use the Database Tools/users and Permissions/User and Group Accounts menu option. Use the Groups tab to enter the custom group names. Simply click the New button and enter the group name and an ID value. The ID value should be random so that no one can guess it. If you ever need to re-create the workgroup file, you should write down the personal IDs that are created. But in general, it is easier and safer to keep backup copies of the workgroup file. For now, at least create the Sales-Clerks and SalesManagers groups. Figure 10.13 shows the screen used to assign users to groups. Simply select the user, then add the desired group to the right-side window. You can also add new users from this screen with the New button.

Once the groups and users have been created, you need to assign the desired permissions within the database. Note that unless you changed the access for the Users group during setup, users will have no access to any of the database objects. So you have to grant all of the permissions that each group will need. Use the Database Tools/Users and Permissions/User and Group Permissions menu option to assign permissions. One of the first permissions that you will have to grant to all of the groups is the ability to open the database. Figure 10.14 shows the basic process. Make sure you select the Groups option to assign permissions to groups and not to individual users. Selecting the object type (Database) displays a list of permissions available for that object. In this case, you need to assign Open/Run to the groups for SalesClerks and SalesManagers.

Other permissions are assigned using the same method. Examine your permissions matrix for the SalesClerks group and customers. Sales clerks need to be



StartAllPowder2007-10.mdb

Target type: Application

Target location: Office12

Target: books\Access2007\Databases4e\Secured\Secu

Start in: Workbooks\Access2007\Databases4e\Secured\

Shortcut key: None

Run: Normal window

Comment:

"C:\Program Files\Microsoft Office\Office12\MSACCESS.EXE" "D:\FullPathname\AllPowder.mdb" /WRKGRP "D:\fullPathname\Security.mdw"

Figure 10.15

able to read, update, and insert customer data. But they should not have the ability to delete customers; that permission is reserved for the sales managers. Select Table as the object type, then click the Customer table in the right-side window. Set the check-boxes to give the appropriate permissions for the SalesClerks and SalesManagers groups. Then switch the object type to Form, and make sure both groups have Open/Run permission for the Customer form and for the startup form. Eventually you can assign the other permissions needed by each group.

For now, reopen the database using the shortcut link. This time, log in as one of the sales clerks and make sure the user can open the customer form and update the data but not delete any rows. Close the database and reopen it as an administrator. Now you can set all of the access rights listed in your permissions matrix. Keep in mind that you need to test all of the permissions. Check to ensure that users can perform their jobs without receiving security notices, and that the users do not have too many permissions. It is time-consuming to test all of the permissions, and you should eventually enlist the help of the actual users to do final testing and adjustments. To speed up the testing process, you should open two copies of the database—one logged in as a user for testing, and one logged in as an administrator to change the permissions.

To implement a secured database, you need to copy the actual database file (AllPowder.mdb), the workgroup file (Security.mdw), and the startup link (renamed to be StartAllPowder.mdb). Moving the databases means that you will have to edit the properties in the startup link so it points to the correct locations on the new machine. Right-click the link and select the Properties option. As shown in Figure 10.15, you will have to set the Target and Start-In folder. The target contains the names and full pathname location of three files: (1) The Microsoft Access executable file, usually installed to its default location; (2) the full pathname of the secured database (this one you will usually have to change); and (3) the full pathname of the workgroup database (you will also have to change this path). You should generally set the start-in folder to be the same as the location of the main database.

When you find yourself performing these steps on a real-world database, stop and think for a few minutes. If the application truly needs this higher level of security, you will be better off using SQL Server (or Oracle) for the database and Visual Studio for the forms. These tools have security built in from the ground up along with better management tools.

Exercises

Many Charms

Samantha and Madison do not believe that security will be a critical issue at Many Charms. The database will run on one machine, and rarely be used by anyone except the two of them. On the other hand, they do need a system for which it is easy to create backup copies. For some security, they are willing to use the single database password. On the other hand, they are concerned about performance. Although they do not expect too many orders to arrive at one time, they do want to examine some lengthy reports to evaluate sales trends.

1. Run the performance wizard to improve the performance of the database—particularly in terms of the reports and queries needed.
2. Create a backup option that makes it easy for the managers to create a backup copy. As much as possible, keep it down to one button. But provide some notices about moving the backup copy offsite in case of fire.
3. Add the security provisions needed by Samantha and Madison.

Standup Foods

Security is a serious concern for Laura. The database contains a large amount of data about employees—and movie star preferences. Managerial employees will need access to the database to enter a considerable amount of information regarding other employees and the status of the event. Consequently, employee access has to be carefully thought out. Managers should have the ability to enter data on employees who report to them, but should not be able to even see most data on other employees. You will have to use queries to provide this level of security. Assigning access to the entire employee table would give managers too much permission. Instead, you will have to set up queries that retrieve the data for specific managers and then give the managers access to the data through that query.

1. Run the performance wizard to improve the performance of the database.
2. Create a backup option and a written set of procedures that Laura can follow to ensure the data is protected.
3. Create the security provisions needed by Laura. Concentrate on the permissions needed to handle evaluation of employees by a manager—without allowing the manager full access to data for all employees.

EnviroSpeed

The knowledge in the EnviroSpeed database is a major strategic asset to the company. This data represents experience gained over several years and enables the company to be considerably more productive and profitable than its competitors. Tyler and Brennan believe it is critical to protect this asset. On the other hand, it is also critical that employees and hired experts have immediate access to all of

the knowledge during a disaster cleanup. Security controls need to be set carefully to protect the database from outside hackers. Fortunately, Brennan and Tyler can trust all of the employees and experts, and do not believe it is necessary to track the exact usage by each person to prevent theft.
Run the performance wizard to improve the response times of the database.

1. Create a backup option and a written set of procedures to follow to protect the database.
2. Create the security provisions needed. Concentrate on protecting the data from external attacks.

Final Project

The main textbook has an appendix with several longer case studies. You should be able to work on one of these cases throughout the term. If you or your instructor picks one, perform the following tasks.

1. Run the performance wizard and improve the responsiveness of the application. Identify the main areas that will be stressed as loads increase.
2. Create a backup option and a written set of procedures to protect the database.
3. Identify the main risk factors and implement the security provisions needed to protect the data, but still ensure users have the access needed to perform their jobs efficiently.

Chapter 11

Distributed Databases

Chapter Outline

Objectives

- Split a database and link the parts for use on a LAN.
- Replicate a database and synchronize the changes.
- Create Web pages to edit data over the Internet.
- Export and import data as XML files.

Location, Location, Location

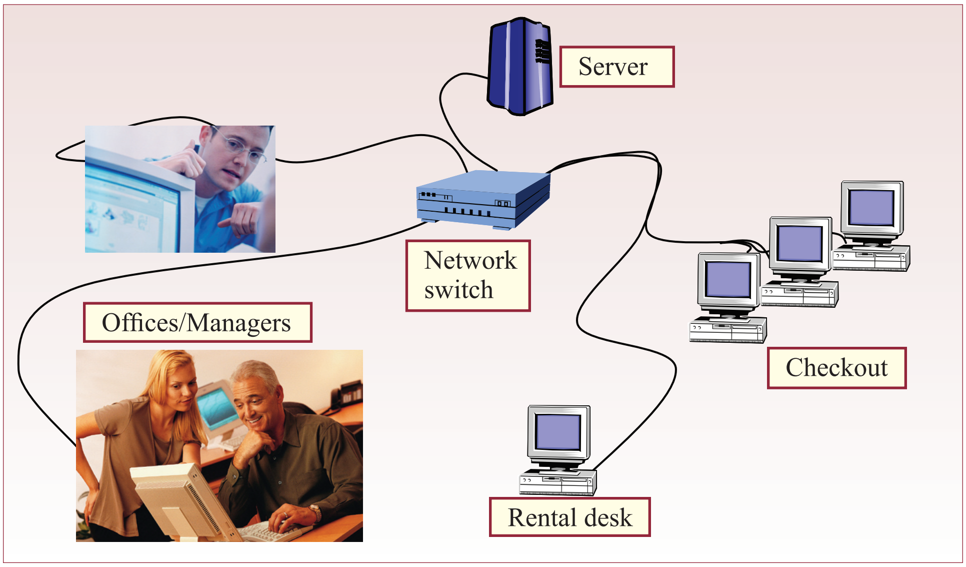
Even small companies often need to access data in multiple locations. This distributed access generates several issues in database management. The most important question you will face is where to store the data. The answer depends on how the database is used, how fast the connections are, and whether everyone needs 24-hour access to immediately current data. The first step in designing a distributed system is to answer these questions and determine the most efficient method for handling data updates in the various locations. Note that efficiency also includes cost issues.

Microsoft Access provides some support for distributed databases, but does not have as many options as SQL Server or the other big systems. The two main techniques supported are linked files, and data replication. However, support for replication was dropped with the new Access 2007 accdb file format. You can still use replication if you use the older mdb file format. Also, with some additional tools Access can also be used as a database server for small Web sites. Internet connections are rapidly becoming a useful way to handle distributed access to data. Interestingly, with most Internet database approaches, the data itself is actually centralized in one location. The browser and Internet protocols provide users access to the database from any location with an Internet connection—including wireless devices.

Case: All Powder Board and Ski Shop

Initially, you might think that All Powder, having only one store, would not care much about distributed databases. Certainly if the owners consider adding a second store, the issues become more complex. This situation will be examined in a second lab exercise. In the meantime, even with one store some distributed issues need to be addressed. The distributed aspect arises because there will be several locations within the store that need access to the database—the checkout stations, the rental desk, and a couple of offices. Figure 11.1 shows that each of these locations will have a computer that needs to run the forms and share the data.

Figure 11.1



Distributed questions within a single building are much easier to solve than those spreading across wide geographic areas. The reason is because of the speed of local area networks (LANs). Within the store, it is relatively easy to install a high-speed LAN that can transfer data as quickly as a typical computer can transfer data to an internal hard drive. Consequently, it is possible to store the database in one location and share it with all of the other computers—with no noticeable delays.

Lab Exercise

All Powder Board and Ski Shop

Consider the in-store situation with a high-speed LAN and the need to run the database on several computers at the same time. Currently, you have a single accdb file (plus the security workgroup file if you are using the secured copy). How would you use this single file on six different computers at the same time? You could copy the file to all six machines, but remember that the file contains all of the forms, reports, and data. Any changes to the data on one machine are not going to be available to the other computers.

With a LAN, you could put the single copy of the database on the server on a shared folder. Each workstation would open the same copy of the database. This way, everyone is using the same copy of the data, so all changes are available to everyone. The drawback to using a single file for the database is that all of the forms and reports are also shared—which places a load on the network. Each time a form is opened, it has to be transferred across the network. Another problem is that you cannot make design changes to forms or reports until everyone has closed the database. Even if you simply need to tweak a report that only one person needs, you have to wait until everyone closes the database—because all users are sharing the same forms and reports file.

Activity: Split and Link the Database

Action

Choose Database Tools/Move Data/Access Database.

Choose the name and location for the back-end database.

Examine the contents of the two databases.

One solution to these issues is to split the database into two components: (1) the data tables with relationships, and (2) the forms, reports, queries, and modules. The main data file will reside on the shared server. The file containing the forms and queries can be copied and placed on each individual machine. With this approach, forms and reports load quickly—even if they contain graphics. Minor changes can be made to the files on individual workstations, although at some point you have to copy the changes to all computers. Figure 11.2 shows the concept of the split database. The key to making it work is that the forms database contains links to the actual data tables stored on the shared server. These links are shown as virtual tables in the forms copy of the database, and can be referenced the same as a native table.

Splitting the database into these elements has an additional impact that can be useful later as the company expands. At some point, the data can be transferred to a more powerful DBMS. The forms and reports will still work! You generally just have to change the links to point to the new, larger DBMS.

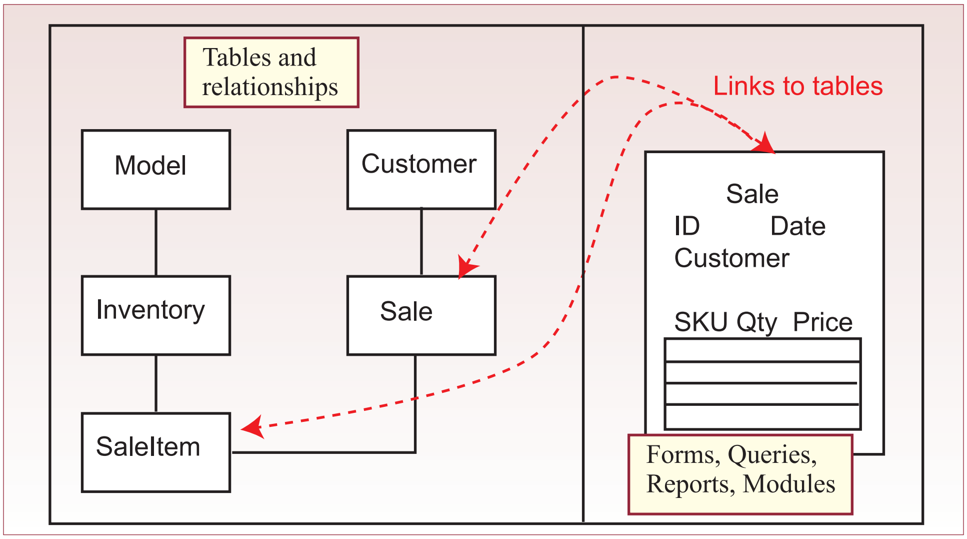
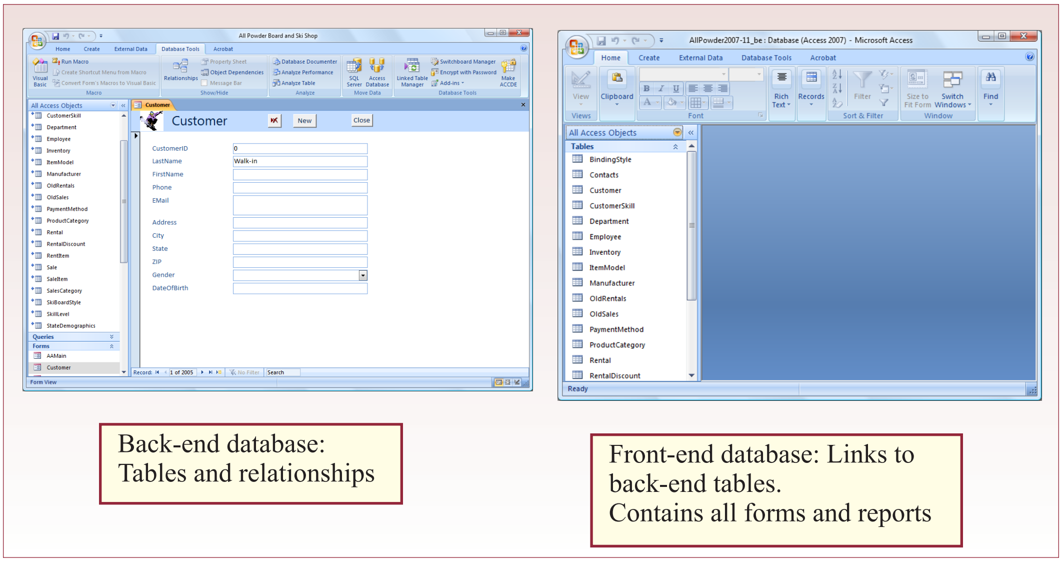


Figure 11.2

To illustrate the process, it is best to use the unsecured database. The process will work on the secured database, but either way, you will have to rebuild the security conditions. In practice, you would build the split database first and then set the security conditions. You could split a database manually; the process is not too difficult. You simply create a new database and import all of the tables from the original. Then you delete the tables from the original database and build links into the new back-end database. To make the process even easier, Access contains a database splitter wizard. To use this wizard, open the main database and choose the Database Tools/Move Data/Access Database menu option. There are no options, but you will be asked to select a name and location for the new back-end database that will contain the tables. Figure 11.3 shows the results. You end up with two accdb files, one for the front end, one for the back end. You should

Figure 11.3

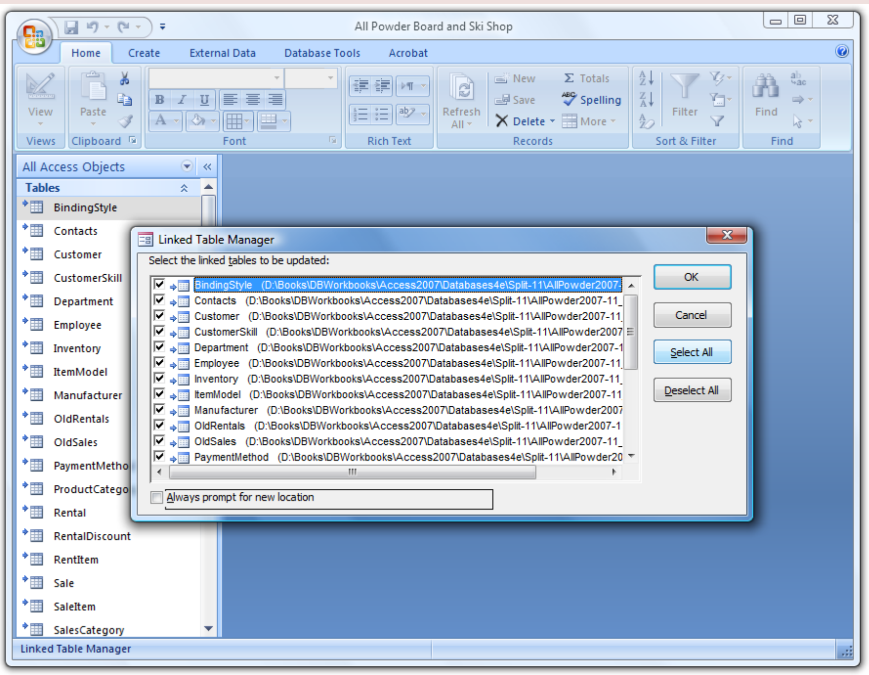


CustomerSkill
Department
Employee
Inventory
ItemModel
Manufacturer
OldRentals
OldSales
PaymentMethod
ProductCategory
Rental
RentalDiscount
RentItem
Sale
SaleItem
SalesCategory
SkiBoardStyle
SkillLevel
StateDemographics
AAMain
Customer

Customer

CustomerID	0
LastName	Walk-in
FirstName	
Phone	
EMail	
Address	
City	
State	
ZIP	
Gender	
DateOfBirth	

BindingStyle
Contacts
Customer
CustomerSkill
Department
Employee
Inventory
ItemModel
Manufacturer
OldRentals
OldSales
PaymentMethod
ProductCategory
Rental
RentalDiscount



BindingStyle
Contacts
Customer
CustomerSkill
Department
Employee
Inventory
ItemModel
Manufacturer
OldRentals
OldSales
PaymentMetho
ProductCatego
Rental
RentalDiscount
RentItem
Sale
SaleItem
SalesCategory

Linked Table Manager

Select the linked tables to be updated:

```
☑ BindingStyle   (D:\Books\DBWorkbooks\Access2007\Databases4e\Split-11\AllPowder2007-
☑ Contacts   (D:\Books\DBWorkbooks\Access2007\Databases4e\Split-11\AllPowder2007-11_
☑ Customer   (D:\Books\DBWorkbooks\Access2007\Databases4e\Split-11\AllPowder2007-11_
☑ CustomerSkill   (D:\Books\DBWorkbooks\Access2007\Databases4e\Split-11\AllPowder2007
☑ Department   (D:\Books\DBWorkbooks\Access2007\Databases4e\Split-11\AllPowder2007-1
☑ Employee   (D:\Books\DBWorkbooks\Access2007\Databases4e\Split-11\AllPowder2007-11_
☑ Inventory   (D:\Books\DBWorkbooks\Access2007\Databases4e\Split-11\AllPowder2007-11_
☑ ItemModel   (D:\Books\DBWorkbooks\Access2007\Databases4e\Split-11\AllPowder2007-11
☑ Manufacturer   (D:\Books\DBWorkbooks\Access2007\Databases4e\Split-11\AllPowder2007
☑ OldRentals   (D:\Books\DBWorkbooks\Access2007\Databases4e\Split-11\AllPowder2007-1
☑ OldSales   (D:\Books\DBWorkbooks\Access2007\Databases4e\Split-11\AllPowder2007-11_
☑ PaymentMethod   (D:\Books\DBWorkbooks\Access2007\Databases4e\Split-11\AllPowder20
```

☐ Always prompt for new location

Figure 11.4

open both files and see what pieces exist. The back-end database contains only tables. The front-end database contains links to the tables, but not the actual data. It also contains all of the forms, reports, queries, and modules. Run the forms in the front-end database to verify that they work.

Once the tables are linked, you can treat them as if they were local tables. When Access needs data, it automatically goes across the network and retrieves it. In the All Powder store, you can place the back-end database on the server, and then put copies of the front-end database on each machine. Be sure you set security permissions in both the back-end and front-end databases first. One issue does arise with linked databases: What happens if you move the back-end database? The front-end forms will not be able to find the tables to establish the link. Access contains a Linked Table Manager to solve this problem. As shown in Figure 11.4, right-click on the linked tables in the front-end database and select the Linked Table Manager. Then select (check) all of the links and click the OK button. At this point, Access will prompt you for the new location of the back-end database. Simply browse to its location and select it. The Linked Table Manager will rebuild all of the links.

Activity: Replicate and Synchronize a Database

Linked tables are an efficient and easy solution when all of the computers are connected by a high-speed LAN. If the company decides to open a second store, even if it is in the same city, it is usually too expensive to link the stores with a high-speed network. Within each store, the LAN probably runs at 100 megabits per second (Mbps), but connections between stores are expensive and pricing is based on the transmission capacity. Even relatively low-speed simple T1 connections at 1.544 Mbps can cost $500 to $1,000 per month. No one will be happy with linked tables at that speed, and faster links can be substantially more expensive. With slower connection speeds, you need to distribute the data so that most of the accesses remain within each store. In this case, you will have two separate copies of the data tables. Then you need to synchronize the data so the two copies of the database exchange updates until they both contain the same data. How often the databases need to be synchronized depends on the use of the data. If sales clerks routinely need to know the inventory status of items in the other store, the synchronization will have to occur several times throughout the data. However, in most cases, daily synchronization will be sufficient, and the databases can be exchanged over night when there is less traffic on the network.

Action

Create a new MDB file and import all objects.

Choose Database Tools/Replication Options/Create Replica.

Make a change to a customer name in the master database.

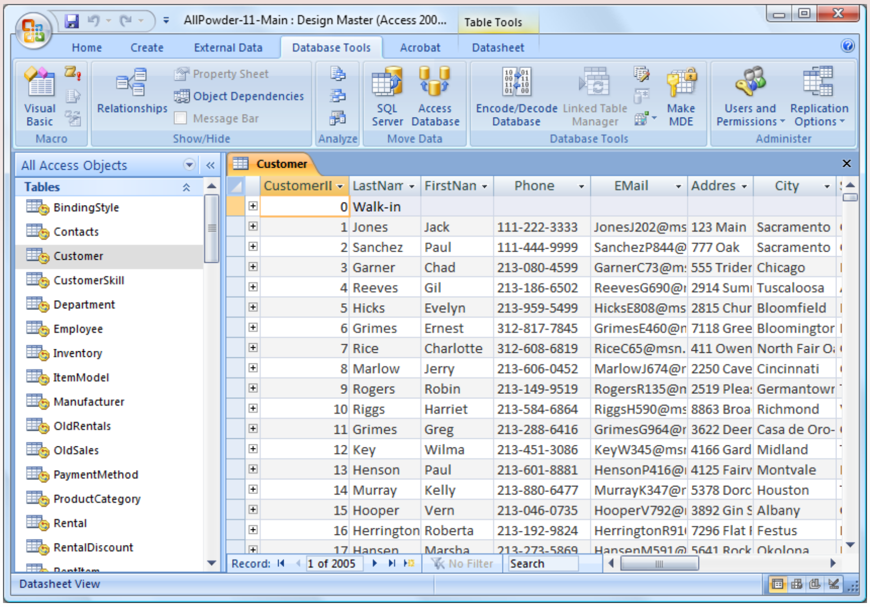
Add a new customer in the replica.

In the master database, choose Database Tools/Replication Options/Synchronize Now.

Verify that both databases now contain the same data.

The Access 2007 file format does not support replication, so you first have to create a new blank MDB file and copy all of the objects from the original ACCDB file. To create a replica, you cannot just copy the database file. Instead, you run the replication program which changes ID values to random numbers and tags all data with times so Access knows how to merge changes from multiple copies. For the case of the All Powder database, each store would run split copies of the database. Each store would contain a back-end database that contains only the tables. This database is the one that would be replicated and later synchronized. The front-end database could simply be copied and loaded onto the individual user machines.

Figure 11.5



CustomerII	LastNam	FirstNan	Phone	EMail	Addres	City
0	Walk-in					
1	Jones	Jack	111-222-3333	JonesJ202@ms	123 Main	Sacramento
2	Sanchez	Paul	111-444-9999	SanchezP844@	777 Oak	Sacramento
3	Garner	Chad	213-080-4599	GarnerC73@m:	555 Trider	Chicago
4	Reeves	Gil	213-186-6502	ReevesG690@	2914 Sumi	Tuscaloosa
5	Hicks	Evelyn	213-959-5499	HicksE808@ms	2815 Chur	Bloomfield
6	Grimes	Ernest	312-817-7845	GrimesE460@n	7118 Gree	Bloomingtor
7	Rice	Charlotte	312-608-6819	RiceC65@msn.	411 Owen	North Fair O:
8	Marlow	Jerry	213-606-0452	MarlowJ674@r	2250 Cave	Cincinnati
9	Rogers	Robin	213-149-9519	RogersR135@n	2519 Plea:	Germantowr
10	Riggs	Harriet	213-584-6864	RiggsH590@ms	8863 Broa	Richmond
11	Grimes	Greg	213-288-6416	GrimesG964@r	3622 Deer	Casa de Oro-
12	Key	Wilma	213-451-3086	KeyW345@msı	4166 Gard	Midland
13	Henson	Paul	213-601-8881	HensonP416@	4125 Fairv	Montvale
14	Murray	Kelly	213-880-6477	MurrayK347@r	5378 Dorc	Houston
15	Hooper	Vern	213-046-0735	HooperV792@	3892 Gin S	Albany
16	Herrington	Roberta	213-192-9824	HerringtonR91	7296 Flat I	Festus
17	Hansen	Marsha	213-273-5869	HansenM591@	5641 Rock	Okolona

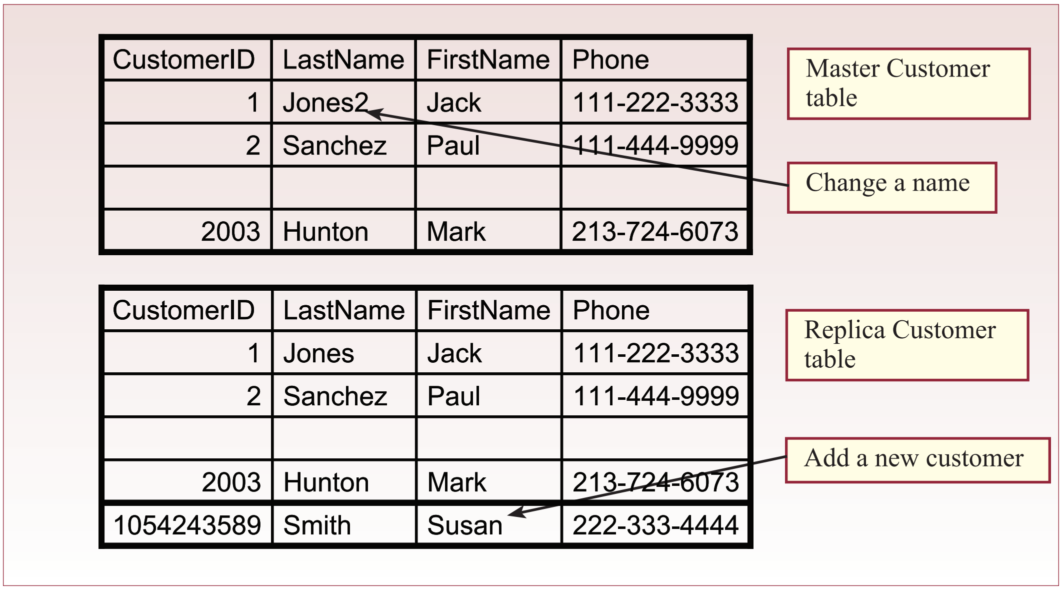
To create a replica, open the MDB database and choose Database Tools/Replication Options/Create Replica on the ribbon menu. The process is almost automatic. Stick with the default choices and make sure you include the backup copy of the original database. When the wizard finishes, you will have the original master copy of the database and a replica. If desired, you can create additional replicas from the master. Figure 11.5 shows the result of the process. Initially, it appears that little has changed except for the new icon. However, several changes were made to the master database internally. Check the file size and you will see that it has doubled.

AutoNumber keys present one of the more visible issues in replicated databases. When managers are simultaneously adding new customers in different stores, how does the database ensure that the generated keys are unique? It can no longer use a simple increment approach. Instead, Access essentially randomly assigns values for AutoNumber keys. A more sophisticated approach would be to assign the values with a component added for each copy. Purely random numbers are not guaranteed to be unique—but with over 2 billion possibilities, the chance of simultaneously generating exactly the same value is fairly small.

To see how the replicated database works, open the Customer table in the master database and change one of the names slightly (for example, Jones2). Close the database and open the Customer table in the replica. This copy still has the original name because the databases have not been synchronized yet. Add a new customer in the replica database, and observe the generated AutoNumber key. Again, this new row exists only within the replica database until it is synchronized to the master. Figure 11.6 shows a partial version of the two Customer tables and the changes.

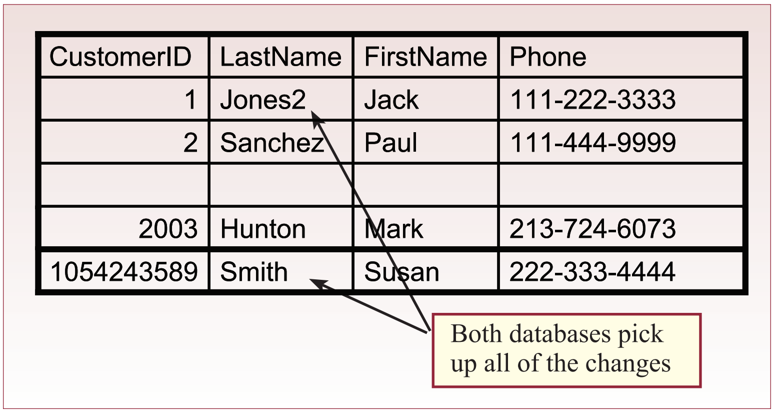
To synchronize the replica to the master database, close the replica and open the master database. Choose the Tools/Replication/Synchronize Now menu option. If necessary, browse to find the replica. The process is largely automatic. Open the Customer table in the master database. The modified name is still intact. Scroll to the end of the table and notice that the row added to the replica has been

Figure 11.6



CustomerID	LastName	FirstName	Phone
1	Jones2	Jack	111-222-3333
2	Sanchez	Paul	111-444-9999
2003	Hunton	Mark	213-724-6073

CustomerID	LastName	FirstName	Phone
1	Jones	Jack	111-222-3333
2	Sanchez	Paul	111-444-9999
2003	Hunton	Mark	213-724-6073
1054243589	Smith	Susan	222-333-4444



CustomerID	LastName	FirstName	Phone
1	Jones2	Jack	111-222-3333
2	Sanchez	Paul	111-444-9999
2003	Hunton	Mark	213-724-6073
1054243589	Smith	Susan	222-333-4444

Both databases pick up all of the changes

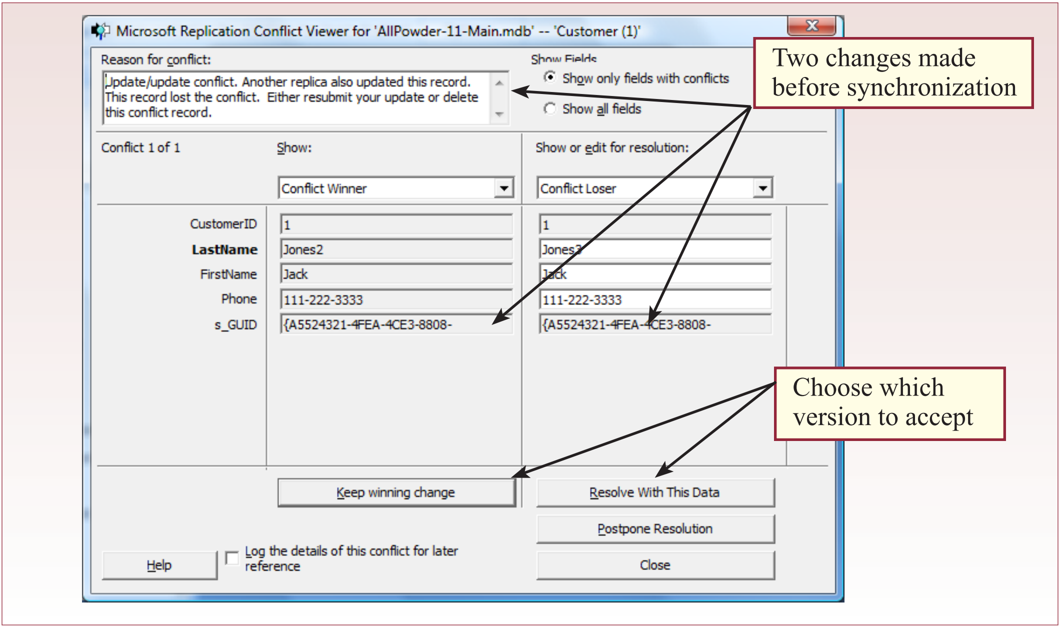
Figure 11.7

transferred to the master. Open the Customer table in the replica database. Notice that the modified name has been updated in the replica. Figure 11.7 shows that all changes are made in each table.

What about concurrency? With replicated databases, it is possible for two people in different locations to modify the same piece of data at essentially the same time. For instance, both could be changing an address for the first customer. Try it by modifying the address for the first customer in the master database and then making a different change for the same address in the replica. Then reopen the master database and synchronize it with the replica. You will receive a message that a conflict exists and an option to resolve the conflict.

Figure 11.8 shows the analysis by the Resolve Conflicts wizard. It indicates that the same row was changed by two different operations and shows the two values that were entered for the Address. In this case, the values on the left were entered into the master database. By clicking the button beneath the desired set of values,

Figure 11.8



Microsoft Replication Conflict Viewer for 'AllPowder-11-Main.mdb' -- 'Customer (1)'

Reason for conflict:

Update/update conflict. Another replica also updated this record. This record lost the conflict. Either resubmit your update or delete this conflict record.

Show Fields: [✓ Show only fields with conflicts] Show all fields

Conflict 1 of 1

	Show: Conflict Winner	Show or edit for resolution: Conflict Loser
CustomerID	1	1
LastName	Jones2	Jones3
FirstName	Jack	Jack
Phone	111-222-3333	111-222-3333
s_GUID	{A5524321-4FEA-4CE3-8808-	{A5524321-4FEA-4CE3-8808-

Keep winning change
Resolve With This Data
Postpone Resolution
Help
Log the details of this conflict for later reference
Close
Two changes made before synchronization
Choose which version to accept

you can indicate which data should be kept and which discarded. You can also postpone the decision, while you talk with others to find out which change should be kept. With replicated databases, this manual process cannot be avoided. There is no good way for the computer to decide which change to accept when two changes occur at the same time. You could make an argument that the most recent change should be kept, but that decision might not be the best in every case.

The Internet

Activity: Web Pages with Access

Web-based applications are becoming the preferred method of building systems—particularly when users need distributed access. With relatively simple browsers, users can access the data and the applications from anyplace that has an Internet connection. With the increasing availability and features of wireless devices such as cell phones, managers will want even more access to data remotely. Unfortunately, the technologies and infrastructure are still immature, and Web-enabled applications can require intricate coding and time-consuming debugging.

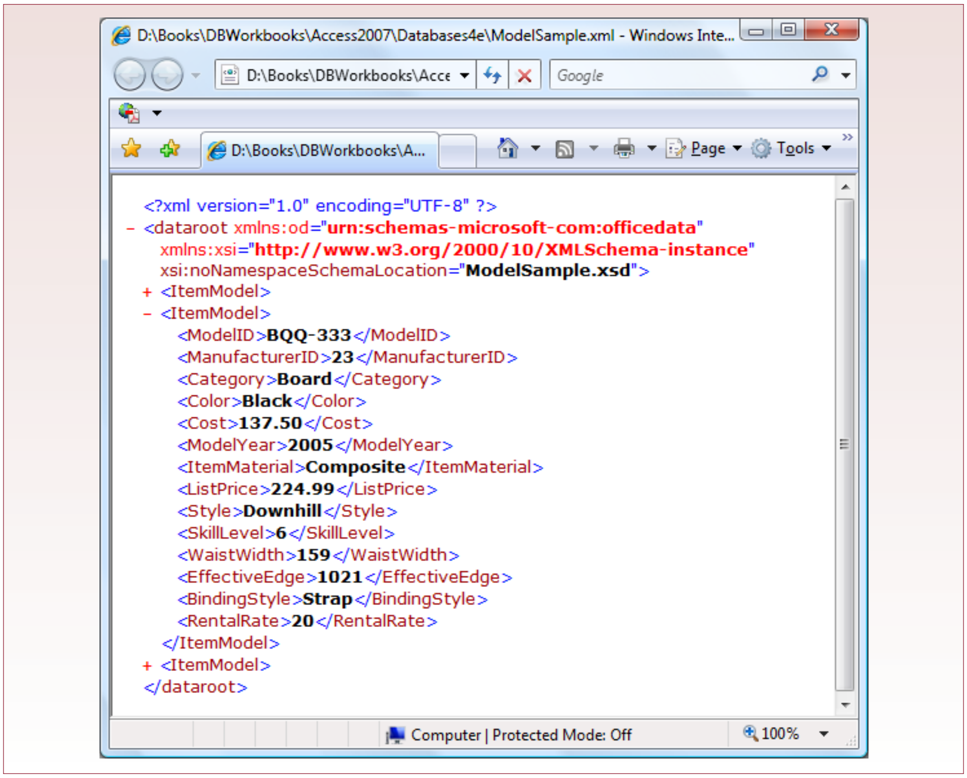
With Access 2007, Microsoft has discontinued the tool to create and modify Web data access pages. Pages can still be imported from earlier versions and data can be entered using Internet Explorer. However, there is no way to create or modify Web pages within Access 2007. On the other hand, this tool was not particularly useful and seldom used.

To create data-enabled Web pages, you need a more powerful tool. Microsoft Visual Studio is one method that uses ASP .NET to create dynamic Web pages. It contains wizards to build data-bound forms relatively easily and quickly. It does require a certain level of programming skills to create complex pages, but simple forms can be built with minimal programming. There is even a free version available to create Web sites. However, it is much better to use SQL Server instead of Access with the current version of these tools. SQL Server Express is a free version that can be used on Web sites.

Activity: Transferring Data with XML

Action
Highlight the Employee table name.
Right-click and choose Export/XML.
Save the file and close the database.
Open the file in Explorer.

One issue you will face with distributed databases is the need to transfer data among differing database systems. For example, a supplier might send you product information electronically. Since the supplier does not know what type of database system you have or how your database is organized, it can be difficult to provide the data in a format that your system can read. The process is complicated when suppliers have thousands of customers just like your shop. Suppliers have no desire to create thousands of different electronic files. Instead, they should be able to send one file in a standard format, and your system should be able to identify the necessary data, select it, and import it into your database. This dream is not quite reality, but XML (eXtensible Markup Language) was created to make it easier to exchange data among disparate systems. If you need to transfer data to someone outside the company, you can use File/Export and select XML as the file type. An XML file is a text file that contains tags that describe the data. You should select the option to create a schema file (xsd) that defines the structure of the data file.


D:\Books\DBWorkbooks\Access2007\Databases4e\ModelSample.xml - Windows Inte...
D:\Books\DBWorkbooks\Acce
Google
D:\Books\DBWorkbooks\A...
Page
Tools
Computer | Protected Mode: Off
100%

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <dataroot xmlns:od="urn:schemas-microsoft-com:officedata"
    xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="ModelSample.xsd">
  + <ItemModel>
  - <ItemModel>
      <ModelID>BQQ-333</ModelID>
      <ManufacturerID>23</ManufacturerID>
      <Category>Board</Category>
      <Color>Black</Color>
      <Cost>137.50</Cost>
      <ModelYear>2005</ModelYear>
      <ItemMaterial>Composite</ItemMaterial>
      <ListPrice>224.99</ListPrice>
      <Style>Downhill</Style>
      <SkillLevel>6</SkillLevel>
      <WaistWidth>159</WaistWidth>
      <EffectiveEdge>1021</EffectiveEdge>
      <BindingStyle>Strap</BindingStyle>
      <RentalRate>20</RentalRate>
    </ItemModel>
  + <ItemModel>
  </dataroot>
```

Figure 11.9

Importing data is generally not much more difficult, but your original source file might not have all of the data that you need. You might have to edit the file to add or modify some of the data. For example, a vendor might send you a file that lists some new skis and snowboards. This data can be imported into the ItemModel table, but this table has a column for ManufacturerID. You will have to add that column to the XML file because your vendor is not going to know what number you use. Alternatively, if you are careful, you can import the data with a blank ManufacturerID and use SQL to update those new rows. You have to be careful with this method, because you need a way to identify which rows should be updated.

Figure 11.9 shows that you can open XML files using the Internet Explorer browser. This approach highlights the individual data records and makes it easy to see the structure of the data. You can expand or contract individual segments to focus on individual areas. This example has three new items, with basic data about the products. Notice the tags used to indicate the purpose of the data items. These tags could have any names, but choosing names that are understandable to humans makes the file easier to read and to edit. This file was generated by an Access export, so it uses the existing column names.

Action

Open the ModelSample.xml file with Wordpad and verify that the fields match your column names.

Edit the file if needed and save it.

In the database, choose External Data/XML

Select the file and append the data to your ItemModel table.

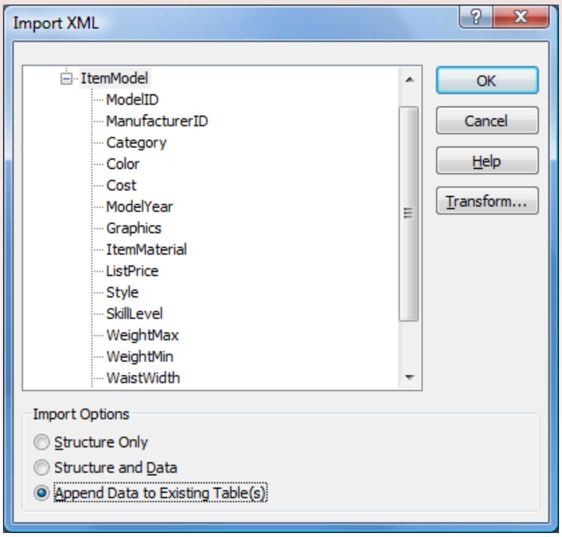
```xml
<ItemModel>
<ModelID>BQQ-333</ModelID>
<ManufacturerID>23</ManufacturerID>
<Category>Board</Category>
<Color>Black</Color>
<Cost>137.50</Cost>
<ModelYear>2005</ModelYear>
<ItemMaterial>Composite</ItemMaterial>
<ListPrice>224.99</ListPrice>
<Style>Downhill</Style>
<SkillLevel>6</SkillLevel>
<WaistWidth>159</WaistWidth>
<EffectiveEdge>1021</EffectiveEdge>
<BindingStyle>Strap</BindingStyle>
<RentalRate>20</RentalRate>
</ItemModel>
```

Figure 11.10

Figure 11.10 shows a portion of the same XML file as it would appear in a text editor. Notice the tags are there to help you, but the text editor lacks the formatting and highlighting of Internet Explorer. Nonetheless, the text-based data is easy to edit, and you could quickly change the ManufacturerID data. However, be careful if you have to add new tags. It is easy to add the tags into an XML file, but most files are also transferred with a schema file that lists the tags that can be included in the XML file. If you add sections or tags to an XML file, you will also have to modify the accompanying XSD file.

Once the file is acceptable, you can import the data into Access. Use the File/Get External Data/Import menu option. Figure 11.11 shows that you are given the option to append the data directly to the end of an existing table. If the data and tags in the XML file match the table definition, you can use this option. It will still work if columns are in a different order. On the other hand, if the XML file is missing key columns, or needs additional cleanup, it would be wiser to import the

Figure 11.11


Import XML

- ItemModel
  - ModelID
  - ManufacturerID
  - Category
  - Color
  - Cost
  - ModelYear
  - Graphics
  - ItemMaterial
  - ListPrice
  - Style
  - SkillLevel
  - WeightMax
  - WeightMin
  - WaistWidth

Import Options
- ○ Structure Only
- ○ Structure and Data
- ● Append Data to Existing Table(s)

file into a new table. Then use SQL to extract and clean up the data and transfer it to the desired database table. If you look closely at the data being imported, you will see that there are many opportunities for problems. For instance, what happens if the manufacturer does not use exactly the same style and binding descriptions that are in the main database? The inconsistent rows will not be appended because they would violate the integrity constraints. When there are many rows of data, and many of these situations, you will need to import the data into a separate table and clean it up first. Many times you can write a sequence of SQL statements to do the cleanup, making the process easier next month when you receive a new file from the same vendor. For these reasons, even XML cannot solve all of the data transfer issues.

Exercises

Crystal Tigers

Most of the information for the Crystal Tigers club can be maintained on one computer run by the club secretary. However, the secretary sometimes needs assistance entering all of the data during special events. Although he brings the database on his laptop, it would probably be easier if two or three people brought laptops and handled specific tasks. At the end of the day, the data could be synchronized and available for analysis. It would at least speed up the data entry and give more people access to the critical information needed during the day.

1. Replicate the database and test it on three separate computers, then synchronize the changes a few times to see if this approach will work for the club.
2. The club has talked about making some data available to members over the Internet. Although many of the members do not have Microsoft Office installed, the club would prefer to provide read-only access. Set up a page that generates activity lists for an upcoming event so members can check the schedule.
3. One of the charitable organizations the club works with is impressed with the database and would like some of the data. Create a query and export an XML file that lists the members and the hours worked for a particular event.

Capitol Artists

Because the system for Capitol Artists collects data from many employees at the same time, the main database needs to run on a central server. All of the computers are connected by a high-speed LAN, and based on the company growth rates, you need to plan ahead for possibly moving to a larger back-end DBMS. It makes sense to split the database and link the forms to the data tables. The company is unlikely to open a second office; however, many of the employees have suggested that they would be more productive if they worked from home. The managers have suggested testing this idea by using the database work-tracking system. A few employees would install the database on their home systems. As they completed client tasks, they would fill out the work table as usual. This data could then be synchronized with the company database at the end of the day. After a month, the managers could see if the employee productivity declined or improved.

1. To handle the internal LAN, split the database. Install the back-end database on one machine, and install at least two copies of the front-end database on other machines. Get someone to help you test the system for performance.
2. Create a replica of the database that could be checked out by telecommuting employees. Test the work-progress forms and synchronization.
3. One of the owners travels often and wants to check on daily progress reports over the Internet using her laptop. Create a Web page that displays the work done for the current day and lists the hours and expense of the employees for each project.

Offshore Speed

The Offshore Speed company has some aspects in common with All Powder. In particular, the store needs several computers running the application to handle sales, orders, and management reports. For some reason, the company owner is leery about splitting the database. To satisfy him, you should test the application both ways, so you will need two or three people to assist you in loading and running the forms at exactly the same time. Because of the huge number of parts, the company is particularly interested in obtaining XML product files from vendors. With 20 primary vendors and changes every month, you need to find a way to automate the imports of this data.

1. Install the single copy of the database on one computer. Get two or three other students to simultaneously open the application across the network and enter some basic sales data. Time the process. Split the database, putting copies of the forms on each computer. Repeat the test and see if there is a difference in performance. Make a recommendation to the owner.
2. Create at least three different XML files that might arrive from vendors with product details. These files should not contain VendorID and other specific information that exists only in your database. Create a system that will automate these imports whenever a file arrives.
3. One of the managers wants to bring a copy of the database home to look through reports and the PivotTables at night. Create a replica and set up a synchronization process that is easy for the manager to use at the end of the day, so that his laptop will have up-to-date data.

Final Project

The main textbook has an online appendix with several longer case studies. You should be able to work on one of these cases throughout the term. If you or your instructor picks one, perform the following tasks.

1. Split the database into a back-end and front-end database. Test the applications.
2. Create a replica, test all of the forms and reports on both copies. Test the synchronization.
3. Export at least one table into an XML file that could be sent to an outside firm such as a customer or supplier.
4. Create a basic Web page to enable workers to enter data using a browser.
5. Create a more complex Web page to handle a main form/subform situation such as Sales. Be sure to create drop down lists for foreign key columns. *Hint*: you will have to first create a query for the data displayed in the drop-down list because a Web page combo box can use at most two columns.

Chapter 12

Physical Database Design

Chapter Outline

Objectives

- Understand limitations on Microsoft Access storage methods.
- Identify files used by Microsoft Access.

Storing Data

Microsoft Access is designed to handle small databases with a limited number of users. It is also designed to be easy to use with easy-to-create forms and reports. Consequently, you have almost no control over how the data is stored in data files. As a result, there is not much to cover in this chapter of the workbook. If you need more control over the data storage, you should look at the bigger systems such as SQL Server, Oracle, or DB2.

On the other hand, you will probably face some issues involving the files used by Microsoft Access, so you need to understand the role of the various files and the underlying data engine. So, this chapter looks at some of the technical issues in how Microsoft Access interacts with the operating system.

Access is effective and inexpensive for small to mid-size applications. For several years, Access was an inexpensive way to create database applications. Switching to high-powered systems carried a high price tag. Today, you can implement powerful tools inexpensively with the free copies of database systems (SQL Server Express and Oracle XE) plus the open source tools such as MySQL and PostgreSQL. Because of its flexibility and ease of use, Access is also a useful tool for prototyping systems. You can quickly build a small application in a day or two and users can play with it to identify problems and additional needs. Once you and the users are comfortable with the design, you can transfer to one of the bigger systems.

Case: All Powder Board and Ski Shop

Microsoft Access can be used effectively within one store. Even then, you will have to split the database into a back-end server and front-end forms so everyone uses a single copy of the database. As long as you use a fast network and relatively fast disk drives, Access should be able to handle a typical small store. But, if All Powder managers want to expand to additional stores or set up Web sites that will get thousands of hits an hour, the Access database system will probably not keep up. Keep in mind that Access also does not support transaction logs, so if something goes wrong, you are likely to lose some transactions.

Lab Exercise

Activity: Data Storage and Partitions

Internally, Microsoft Access uses B+trees to store indexes, so it is relatively efficient at handling typical queries and joins. The only control you have over the storage is to add or remove indexes. By default, Access creates indexes for all columns in the primary key. There is no provision for hashed or direct access to rows of data. Access does not directly support clusters or partitions. Remember that all data within a database is stored in a single file.

Action

Create a new blank database.

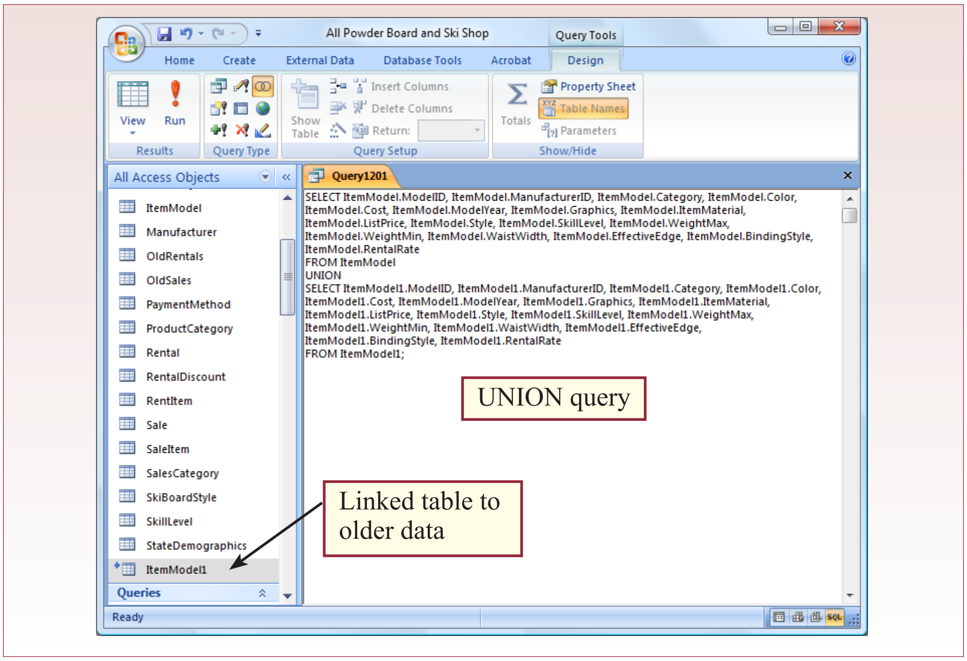
Create a copy of the ItemModel table.

Load it with all data before 2006.

In the ItemModel table in the main database, remove the data before 2006.

Use a linked table in the main database to connect to the older data.

Create a UNION query that retrieves all rows of data for the ItemModel table.



```
SELECT ItemModel.ModelID, ItemModel.ManufacturerID, ItemModel.Category, ItemModel.Color,
ItemModel.Cost, ItemModel.ModelYear, ItemModel.Graphics, ItemModel.ItemMaterial,
ItemModel.ListPrice, ItemModel.Style, ItemModel.SkillLevel, ItemModel.WeightMax,
ItemModel.WeightMin, ItemModel.WaistWidth, ItemModel.EffectiveEdge, ItemModel.BindingStyle,
ItemModel.RentalRate
FROM ItemModel
UNION
SELECT ItemModel1.ModelID, ItemModel1.ManufacturerID, ItemModel1.Category, ItemModel1.Color,
ItemModel1.Cost, ItemModel1.ModelYear, ItemModel1.Graphics, ItemModel1.ItemMaterial,
ItemModel1.ListPrice, ItemModel1.Style, ItemModel1.SkillLevel, ItemModel1.WeightMax,
ItemModel1.WeightMin, ItemModel1.WaistWidth, ItemModel1.EffectiveEdge,
ItemModel1.BindingStyle, ItemModel1.RentalRate
FROM ItemModel1;
```

Figure 12.1

On the other hand, you could use a tricky method to create your own partitions. It is relatively easy to create linked tables. Recall that a linked table stores the data in a separate database file, but through a link, the table can be accessed just like any other table in the original file. The partition trick is to create a new database and store its accdb file on a different disk drive. Transfer the older data to a table in this new database. In the original database, create a link to the new table. Finally, build a UNION query to retrieve all of the rows from both tables. Figure 12.1 shows the link and the UNION query. The older data could even be stored on a different computer attached to the network. This process is somewhat cumbersome. Your forms will have to write new data to just the table in the main database. Your reports will use the UNION query to retrieve and analyze data from both the old and new data. You have to split and copy the data by hand, and within your application forms and reports, you have to choose which table or query to use. However, in exchange for this additional work, you end up with a system that enables you to move large amounts of old data to a different computer or disk drive. The smaller tables of new data improve overall application performance. Consequently, this trick is useful when the database has been running for a long time and you no longer need immediate access to the older data.

Activity: Identifying Access Files

By default, Microsoft Access stores everything in a single accdb file. This approach makes it easy to transfer the application to another computer. However, if you have worked through the labs in this book, you have probably noticed that Access uses several other files as well. The purpose of this sec-

Action

Use "My Computer" to locate the folder with the MDB file.

Open the database file.

Identify the LACCDB file.

Suffix	Old Suffix	Purpose
ACCDB	MDB	Primary database file to hold data, forms, and reports.
LACCDB	LDB	Holds lock data to prevent multi-user collisions.
ACCDE	MDE	Stripped down MDB file that removes source code and prevents users from editing forms and reports.
ACCDT		Template file. Download the Access Extensions to create template files from your database.
ACCDR		Run-time version that prevents changes to forms and reports. You can restore it by renaming the extension.
	MDW	Workgroup file that contains user and group login information. The default is System.MDW.
	ADP	Project file that contains code to connect to a SQL Server database.

Figure 12.2

tion is to summarize the main operating system files created and used by Access. You need to know the various types because you will have to set operating system access permissions or transfer the correct file to other computers and users. Figure 12.2 lists the various file types based on the suffix.

All data and forms are stored within the accdb file that is created when you create a new database. However, Access automatically uses a second file when accessing the accdb file. Open an Access database and use your computer browser to find the folder that holds that file. You will see a second file that has an laccdb suffix. This file holds the locking information that Access uses to track which tables are opened by each user. When you share an Access data file on a network folder, you have to remember to give write access to the folder itself so the data engine can create the lock file if it does not already exist. When everyone closes the Access database file, the lock file is automatically deleted. This lock file is a main reason why you cannot open an Access database that is stored on a CD-ROM drive.

An ACCDE file is similar to an ACCDB file but it removes the VBA source code and disables the edit commands so users cannot alter the forms and reports. It is a convenient way to distribute an application and prevent casual users from altering the design and application itself. You create it with the menu command: Database Tools/ Make ACCDE File. Be careful to keep the original file around. You need the original to make changes, because even you will not be able to alter the designs or code in the ACCDE file.

If you use the older MDB file format for user-level security, Microsoft Access stores security information in a separate MDW workgroup file. When you create users and groups, Access needs someplace to store the username and password login information. The original MDB file would not be a secure place to store this information. Consequently, Access creates the MDW file, which is a specialized database that is encrypted automatically to protect its data. By default, Access uses the System.MDW file. When you run the User Security Wizard, you will create a new MDW file to hold the login information. To deploy the database, you will need to install the MDW file along with the MDE (or MDB) file.

The ADP file is a specialized file that is rarely used. It contains login information when you want to connect your Access database to a SQL Server database. It will be created automatically if you build a client-server system that uses Access forms as the front end and linked tables into a SQL Server backend.

The LACCDB and MDW files are the trickiest, since you need to remember to deploy them when the application is installed. Since they are operating system files, you also have to set operating system (network) security permissions. Both files require read/write permissions or the database will not start. It is not possible to move the LACCDB lock file to a different disk drive, so there is minimal opportunity for performance gains. You can move the MDW file, but you will not see any performance gain. Since your startup link needs to point to the MDW file, moving it also does not increase security.

Activity: Altering Registry Parameters for the Jet Engine

Action

Start the registry editor with Start/Run/regedit.

Expand the Local Machine keys to find the entries for the Jet engine.

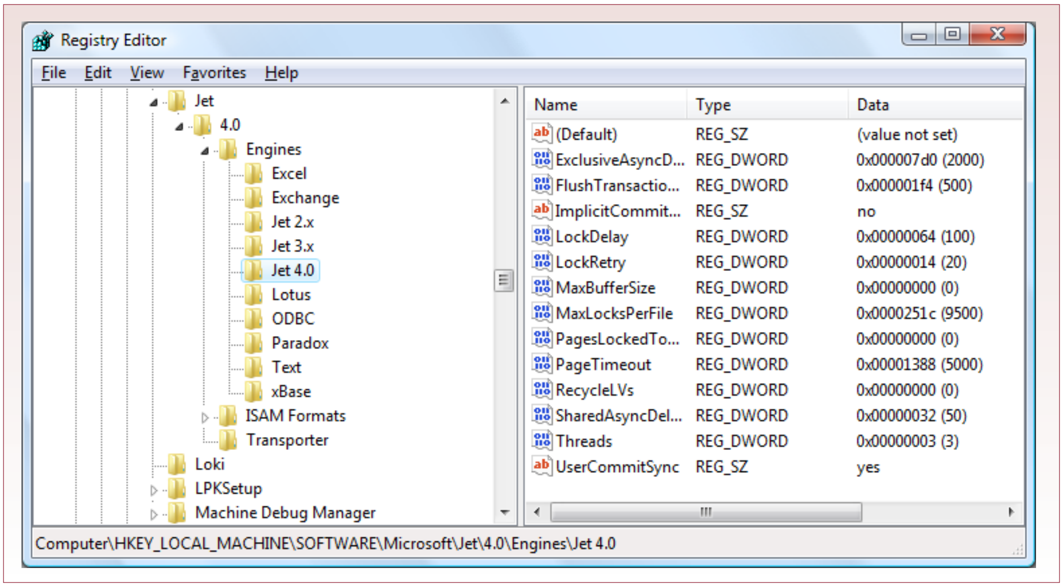
Change the Threads value from 3 to 5.

Test some queries and forms to see if performance changes.

The data, forms, reports, and VBA code are stored in the MDB file. However, the actual program code that connects to the database and handles the queries and updates is stored in the Access Jet Engine—which is installed as a separate component in the operating system. The Jet Engine also uses the Microsoft Data Access Components (MDAC) to handle data exchanges between client programs and the database. Microsoft provides some control over the Jet Engine through registry entries. You should avoid changing any registry entries, but sometimes you can improve performance with a few minor changes.

The registry is a simple hierarchical Windows database that the operating system and programs use to store configuration information. Windows contains a registry editor to help you find and alter these settings. However, you must be extremely careful about editing registry settings—you could crash the entire system.

Figure 12.3



Name	Type	Data
(Default)	REG_SZ	(value not set)
ExclusiveAsyncD...	REG_DWORD	0x000007d0 (2000)
FlushTransactio...	REG_DWORD	0x000001f4 (500)
ImplicitCommit...	REG_SZ	no
LockDelay	REG_DWORD	0x00000064 (100)
LockRetry	REG_DWORD	0x00000014 (20)
MaxBufferSize	REG_DWORD	0x00000000 (0)
MaxLocksPerFile	REG_DWORD	0x0000251c (9500)
PagesLockedTo...	REG_DWORD	0x00000000 (0)
PageTimeout	REG_DWORD	0x00001388 (5000)
RecycleLVs	REG_DWORD	0x00000000 (0)
SharedAsyncDel...	REG_DWORD	0x00000032 (50)
Threads	REG_DWORD	0x00000003 (3)
UserCommitSync	REG_SZ	yes

Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Jet\4.0\Engines\Jet 4.0

It is possible to back up portions of the registry, but you must still be careful or you might not even be able to restart the registry editor.

On Windows XP, or similar versions, use Start/Run/regedit to start the editor. The main keys affecting the Jet Engine are stored in:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Jet\4.0\Engines\Jet 4.0.

In the registry editor, you expand the node for HKEY_LOCAL_MACHINE. Under there, you expand SOFTWARE, then Microsoft, and so on until you click the Jet 4.0 entry. Figure 12.3 shows the parameters for the Jet 4.0 entry. You can double-click an item displayed in the right-hand window to edit it. The Help system in Microsoft Access contains explanations for all of the entries. In general, you should leave the default settings. Two settings that you might need to change are the LockDelay and LockRetry. These settings are used to determine how often and how many times a form should try to access a locked page before returning an error message. The LockDelay is the number of milliseconds between tries. You might want to change the MaxBufferSize for really large databases. It is the number of kilobytes used in the engine's internal cache. Increasing the size might help increase throughput in data transfers. The Threads entry controls the number of background threads available to the engine. If you have dozens of simultaneous users and a fast processor, you might be able to improve performance by increasing this number.

Note that changes made to registry items are saved as soon as click the OK button, and there is no undo capability. When you edit the registry, you should write down the keys that you are changing along with the original and new values. Then test the system for a few days to see if you should go back to the original values.

As a simple test, you can try changing the Threads entry. The default value is 3, which is sufficient for a single user. For multiple simultaneous users, you might want to try a larger number such as 5 or perhaps 10. Each thread takes up processor cache space and time, so be cautious and make small changes. In addition to looking at whether the change improved performance for Access, you will want to observe overall system performance and performance of other applications when several programs are running at the same time. With a relatively fast computer, you are unlikely to see any major performance changes. You would need a large database and multiple users to be able to discern performance changes from these small tweaks. And, if you have a database that large with that many users, you might as well upgrade to SQL Server.

Exercises

Many Charms

The database for Many Charms is likely to remain relatively small and performance should not be a serious issue. Nonetheless, you should look for possible ways to improve performance by controlling the data storage.

1. Create a list of signs that you would watch to indicate that data storage is becoming a performance problem.
2. Do some research to identify tools that you could use to transfer the existing database to a more powerful DBMS.
3. Partition the Production table into two sections based on the ProductionDate.

Standup Foods

Standup Foods has the potential to grow to a relatively large company over the next couple of years. It is possible that performance will become an issue with some of the tables. The client list is particularly interesting, because studios are continually creating new companies and partnerships. As a result, many of the older companies in the list no longer exist. On the other hand, the contact list is important, since it contains data on individual people. Similarly, the Employee list changes on an almost daily basis. Laura is reluctant to delete the older employees because many of them come back for special projects every couple of years.

1. Create a list of signs that you would watch to indicate that data storage is becoming a performance problem.
2. Do some research to identify tools that you could use to transfer the existing database to a more powerful DBMS.
3. Partition the project table into two sections based on the contract date.

EnviroSpeed

The database for EnviroSpeed could eventually become quite large. Because the system contains valuable knowledge, the company does not want to delete anything. The company also benefits by keeping all of the data in one large database. Although much of the data becomes dated, employees still want the ability to search through older cases. However, the older data does not change so it could be moved to different disk drives.

1. Create a list of signs that you would watch to indicate that data storage is becoming a performance problem.
2. Do some research to identify tools that you could use to transfer the existing database to a more powerful DBMS.
3. Partition the Situation and ProposedSolution tables into two segments based on the date.

Final Project

The main textbook has an online appendix with several longer case studies. You should be able to work on one of these cases throughout the term. If you pick one or your instructor picks one, perform the following tasks.

1. Create a list of signs that you would watch to indicate that data storage is becoming a performance problem.
2. Do some research to identify tools that you could use to transfer the existing database to a more powerful DBMS.
3. Identify the tables that could be improved using partitions or clusters. Explain your reasoning.